Internet Engineering Task Force (IETF)
Request for Comments: 7873
Category: Standards Track
ISSN: 2070-1721

D. Eastlake 3rd
Huawei
M. Andrews
ISC

May 2016

Domain Name System (DNS) Cookies

Abstract

DNS Cookies are a lightweight DNS transaction security mechanism that provides limited protection to DNS servers and clients against a variety of increasingly common denial-of-service and amplification/forgery or cache poisoning attacks by off-path attackers. DNS Cookies are tolerant of NAT, NAT-PT (Network Address Translation - Protocol Translation), and anycast and can be incrementally deployed. (Since DNS Cookies are only returned to the IP address from which they were originally received, they cannot be used to generally track Internet users.)

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7873.

Table of Contents

1. Introduction 4
1.1. Contents of This Document 4
1.2. Definitions 5
2. Threats Considered 5
2.1. Denial-of-Service Attacks 6
2.1.1. DNS Amplification Attacks 6
2.1.2. DNS Server Denial of Service 6
2.2. Cache Poisoning and Answer Forgery Attacks 7
3. Comments on Existing DNS Security 7
3.1. Existing DNS Data Security 7
3.2. DNS Message/Transaction Security 8
3.3. Conclusions on Existing DNS Security 8
4. DNS COOKIE Option 8
4.1. Client Cookie 10
4.2. Server Cookie 10
5. DNS Cookies Protocol Specification 11
5.1. Originating a Request 11
5.2. Responding to a Request 11
5.2.1. No OPT RR or No COOKIE Option 12
5.2.2. Malformed COOKIE Option 12
5.2.3. Only a Client Cookie 12
5.2.4. A Client Cookie and an Invalid Server Cookie 13
5.2.5. A Client Cookie and a Valid Server Cookie 13
5.3. Processing Responses 14
5.4. Querying for a Server Cookie 14
6. NAT Considerations and Anycast Server Considerations 15
7. Operational and Deployment Considerations 17
7.1. Client and Server Secret Rollover 17
7.2. Counters 18
8. IANA Considerations 18
9. Security Considerations 19
9.1. Cookie Algorithm Considerations 20
10. Implementation Considerations 20
11. References 20
11.1. Normative References 20
11.2. Informative References 21
Appendix A. Example Client Cookie Algorithms 23
A.1. A Simple Algorithm 23
A.2. A More Complex Algorithm 23
Appendix B. Example Server Cookie Algorithms 23
B.1. A Simple Algorithm 23
B.2. A More Complex Algorithm 24
Acknowledgments 25
Authors' Addresses 25

1. Introduction

As with many core Internet protocols, the Domain Name System (DNS) was originally designed at a time when the Internet had only a small pool of trusted users. As the Internet has grown exponentially to a global information utility, the DNS has increasingly been subject to abuse.

This document describes DNS Cookies, a lightweight DNS transaction security mechanism specified as an OPT [RFC6891] option. The DNS Cookie mechanism provides limited protection to DNS servers and clients against a variety of increasingly common abuses by off-path attackers. It is compatible with, and can be used in conjunction with, other DNS transaction forgery resistance measures such as those in [RFC5452]. (Since DNS Cookies are only returned to the IP address from which they were originally received, they cannot be used to generally track Internet users.)

The protection provided by DNS Cookies is similar to that provided by using TCP for DNS transactions. Bypassing the weak protection provided by using TCP requires, among other things, that an off-path attacker guess the 32-bit TCP sequence number in use. Bypassing the weak protection provided by DNS Cookies requires such an attacker to guess a 64-bit pseudorandom "cookie" quantity. Where DNS Cookies are not available but TCP is, falling back to using TCP is reasonable.

If only one party to a DNS transaction supports DNS Cookies, the mechanism does not provide a benefit or significantly interfere, but if both support it, the additional security provided is automatically available.

The DNS Cookie mechanism is designed to work in the presence of NAT and NAT-PT (Network Address Translation - Protocol Translation) boxes, and guidance is provided herein on supporting the DNS Cookie mechanism in anycast servers.

1.1. Contents of This Document

In Section 2, we discuss the threats against which the DNS Cookie mechanism provides some protection.

Section 3 describes existing DNS security mechanisms and why they are not adequate substitutes for DNS Cookies.

Section 4 describes the COOKIE option.

Section 5 provides a protocol description.

Section 6 discusses some NAT considerations and anycast-related DNS Cookies design considerations.

Section 7 discusses incremental deployment considerations.

Sections 8 and 9 describe IANA considerations and security considerations, respectively.

1.2. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

"Off-path attacker", for a particular DNS client and server, is defined as an attacker who cannot observe the DNS request and response messages between that client and server.

"Soft state" indicates information that is learned or derived by a host and that may be discarded when indicated by the policies of that host but can be re-instantiated later if needed. For example, it could be discarded after a period of time or when storage for caching such data becomes full. If operations that require soft state continue after the information has been discarded, the information will be automatically regenerated, albeit at some cost.

"Silently discarded" indicates that there are no DNS protocol message consequences.

"IP address" is used herein as a length-independent term and includes both IPv4 and IPv6 addresses.

2. Threats Considered

DNS Cookies are intended to provide significant but limited protection against certain attacks by off-path attackers, as described below. These attacks include denial of service, cache poisoning, and answer forgery.

2.1. Denial-of-Service Attacks

The typical form of the denial-of-service attacks considered herein is to send DNS requests with forged source IP addresses to a server. The intent can be to attack that server or some other selected host, as described below.

There are also on-path denial-of-service attacks that attempt to saturate a server with DNS requests having correct source addresses. Cookies do not protect against such attacks, but successful cookie validation improves the probability that the correct source IP address for the requests is known. This facilitates contacting the managers of the networks from which the requests originate or taking other actions for those networks.

2.1.1. DNS Amplification Attacks

A request with a forged source IP address generally causes a response to be sent to that forged IP address. Thus, the forging of many such requests with a particular source IP address can result in enough traffic being sent to the forged IP address to interfere with service to the host at the IP address. Furthermore, it is generally easy in the DNS to create short requests that produce much longer responses, thus amplifying the attack.

The DNS Cookie mechanism can severely limit the traffic amplification obtained by requests from an attacker that is off the path between the server and the request's source address. Enforced DNS Cookies would make it hard for an off-path attacker to cause any more than rate-limited short error responses to be sent to a forged IP address, so the attack would be attenuated rather than amplified. DNS Cookies make it more effective to implement a rate-limiting scheme for error responses from the server. Such a scheme would further restrict selected host denial-of-service traffic from that server.

2.1.2. DNS Server Denial of Service

DNS requests that are accepted cause work on the part of DNS servers. This is particularly true for recursive servers that may issue one or more requests and process the responses thereto, in order to determine their response to the initial request; the situation can be even worse for recursive servers implementing DNSSEC [RFC4033] [RFC4034] [RFC4035], because they may be induced to perform burdensome cryptographic computations in attempts to verify the authenticity of data they retrieve in trying to answer the request.

The computational or communications burden caused by such requests may not depend on a forged source IP address, but the use of such addresses makes

+ the source of the requests causing the denial-of-service attack harder to find and

+ restriction of the IP addresses from which such requests should be honored hard or impossible to specify or verify.

The use of DNS Cookies should enable a server to reject forged requests from an off-path attacker with relative ease and before any recursive queries or public key cryptographic operations are performed.

2.2. Cache Poisoning and Answer Forgery Attacks

The form of the cache poisoning attacks considered is to send forged replies to a resolver. Modern network speeds for well-connected hosts are such that, by forging replies from the IP addresses of a DNS server to a resolver for names that resolver has been induced to resolve or for common names whose resource records have short time-to-live values, there can be an unacceptably high probability of randomly coming up with a reply that will be accepted and cause false DNS information to be cached by that resolver (the Dan Kaminsky attack [Kaminsky]). This can be used to facilitate phishing attacks and other diversions of legitimate traffic to a compromised or malicious host such as a web server.

With the use of DNS Cookies, a resolver can generally reject such forged replies.

3. Comments on Existing DNS Security

Two forms of security have been added to DNS: data security and message/transaction security.

3.1. Existing DNS Data Security

DNS data security is one part of DNSSEC and is described in [RFC4033], [RFC4034], [RFC4035], and updates thereto. It provides data origin authentication and authenticated denial of existence. DNSSEC is being deployed and can provide strong protection against forged data and cache poisoning; however, it has the unintended effect of making some denial-of-service attacks worse because of the cryptographic computational load it can require and the increased size in DNS response packets that it tends to produce.

3.2. DNS Message/Transaction Security

The second form of security that has been added to DNS provides "transaction" security through TSIG [RFC2845] or SIG(0) [RFC2931]. TSIG could provide strong protection against the attacks for which the DNS Cookie mechanism provides weaker protection; however, TSIG is non-trivial to deploy in the general Internet because of the burdens it imposes. Among these burdens are pre-agreement and key distribution between client and server, keeping track of server-side key state, and required time synchronization between client and server.

TKEY [RFC2930] can solve the problem of key distribution for TSIG, but some modes of TKEY impose a substantial cryptographic computation load and can be dependent on the deployment of DNS data security (see Section 3.1).

SIG(0) [RFC2931] provides less denial-of-service protection than TSIG or, in one way, even DNS Cookies, because it authenticates complete transactions but does not authenticate requests. In any case, it also depends on the deployment of DNS data security and requires computationally burdensome public key cryptographic operations.

3.3. Conclusions on Existing DNS Security

The existing DNS security mechanisms do not provide the services provided by the DNS Cookie mechanism: lightweight message authentication of DNS requests and responses with no requirement for pre-configuration or per-client server-side state.

4. DNS COOKIE Option

The DNS COOKIE option is an OPT RR [RFC6891] option that can be included in the RDATA portion of an OPT RR in DNS requests and responses. The option length varies, depending on the circumstances in which it is being used. There are two cases, as described below. Both use the same OPTION-CODE; they are distinguished by their length.

In a request sent by a client to a server when the client does not know the server's cookie, its length is 8, consisting of an 8-byte Client Cookie, as shown in Figure 1.

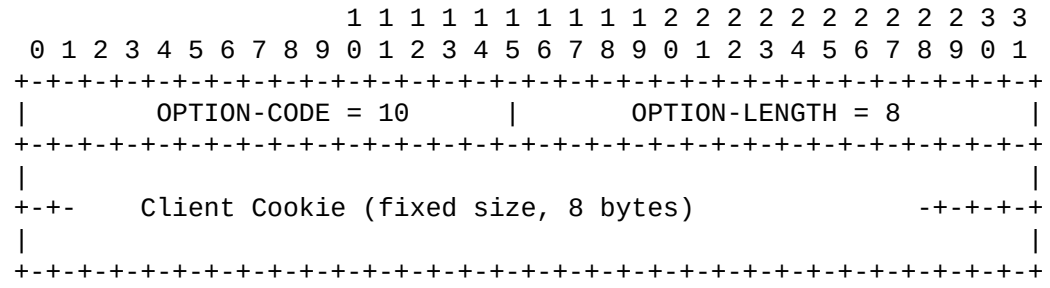

Figure 1: COOKIE Option, Unknown Server Cookie

In a request sent by a client when a Server Cookie is known, and in all responses to such a request, the length is variable -- from 16 to 40 bytes, consisting of an 8-byte Client Cookie followed by the variable-length (8 bytes to 32 bytes) Server Cookie, as shown in Figure 2. The variability of the option length stems from the variable-length Server Cookie. The Server Cookie is an integer number of bytes, with a minimum size of 8 bytes for security and a maximum size of 32 bytes for convenience of implementation.

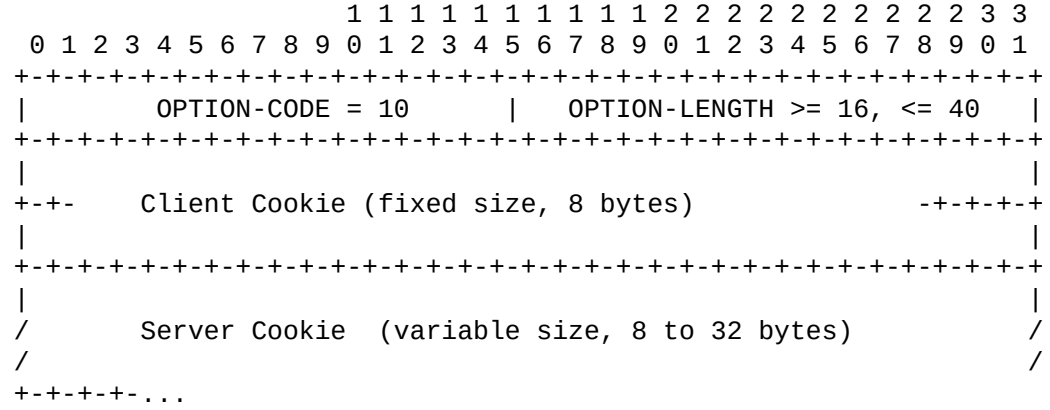

Figure 2: COOKIE Option, Known Server Cookie

4.1. Client Cookie

The Client Cookie SHOULD be a pseudorandom function of the Client IP Address, the Server IP Address, and a secret quantity known only to the client. This Client Secret SHOULD have at least 64 bits of entropy [RFC4086] and be changed periodically (see Section 7.1). The selection of the pseudorandom function is a matter private to the client, as only the client needs to recognize its own DNS Cookies.

The Client IP Address is included so that the Client Cookie cannot be used to (1) track a client if the Client IP Address changes due to privacy mechanisms or (2) impersonate the client by some network device that was formerly on path but is no longer on path when the Client IP Address changes due to mobility. However, if the Client IP Address is being changed very often, it may be necessary to fix the Client Cookie for a particular server for several requests, to avoid undue inefficiency due to retries caused by that server not recognizing the Client Cookie.

For further discussion of the Client Cookie field, see Section 5.1. For example methods of determining a Client Cookie, see Appendix A.

In order to provide minimal authentication, a client MUST send Client Cookies that will usually be different for any two servers at different IP addresses.

4.2. Server Cookie

The Server Cookie SHOULD consist of or include a 64-bit or larger pseudorandom function of the request source (client) IP address, a secret quantity known only to the server, and the request Client Cookie. (See Section 6 for a discussion of why the Client Cookie is used as input to the Server Cookie but the Server Cookie is not used as an input to the Client Cookie.) This Server Secret SHOULD have at least 64 bits of entropy [RFC4086] and be changed periodically (see Section 7.1). The selection of the pseudorandom function is a matter private to the server, as only the server needs to recognize its own DNS Cookies.

For further discussion of the Server Cookie field, see Section 5.2. For example methods of determining a Server Cookie, see Appendix B. When implemented as recommended, the server need not maintain any cookie-related per-client state.

In order to provide minimal authentication, a server MUST send Server Cookies that will usually be different for clients at any two different IP addresses or with different Client Cookies.

5. DNS Cookies Protocol Specification

This section discusses using DNS Cookies in the DNS protocol. The cycle of originating a request, responding to that request, and processing responses is covered in Sections 5.1, 5.2, and 5.3. A de facto extension to QUERY to allow the prefetching of a Server Cookie is specified in Section 5.4. Rollover of the Client Secrets and Server Secrets, and transient retention of the old cookie or secret, are covered in Section 7.1.

DNS clients and servers SHOULD implement DNS Cookies to decrease their vulnerability to the threats discussed in Section 2.

5.1. Originating a Request

A DNS client that implements DNS Cookies includes one DNS COOKIE option containing a Client Cookie in every DNS request it sends, unless DNS Cookies are disabled.

If the client has a cached Server Cookie for the server against its IP address, it uses the longer cookie form and includes that Server Cookie in the option along with the Client Cookie (Figure 2). Otherwise, it just sends the shorter-form option with a Client Cookie (Figure 1).

5.2. Responding to a Request

The Server Cookie, when it occurs in a COOKIE option in a request, is intended to weakly assure the server that the request came from a client that is both at the source IP address of the request and using the Client Cookie included in the option. This assurance is provided by the Server Cookie that server sent to that client in an earlier response appearing as the Server Cookie field in the request.

At a server where DNS Cookies are not implemented and enabled, the presence of a COOKIE option is ignored and the server responds as if no COOKIE option had been included in the request.

When DNS Cookies are implemented and enabled, there are five possibilities:

(1) There is no OPT RR at all in the request, or there is an OPT RR but the COOKIE option is absent from the OPT RR.

(2) A COOKIE option is present but is not a legal length or is otherwise malformed.

(3) There is a COOKIE option of valid length in the request with no Server Cookie.

(4) There is a COOKIE option of valid length in the request with a Server Cookie, but that Server Cookie is invalid.

(5) There is a COOKIE option of valid length in the request with a correct Server Cookie.

These five possibilities are discussed in the subsections below.

In all cases of multiple COOKIE options in a request, only the first (the one closest to the DNS header) is considered. All others are ignored.

5.2.1. No OPT RR or No COOKIE Option

If there is no OPT record or no COOKIE option present in the request, then the server responds to the request as if the server doesn't implement the COOKIE option.

5.2.2. Malformed COOKIE Option

If the COOKIE option is too short to contain a Client Cookie, then FORMERR is generated. If the COOKIE option is longer than that required to hold a COOKIE option with just a Client Cookie (8 bytes) but is shorter than the minimum COOKIE option with both a Client Cookie and a Server Cookie (16 bytes), then FORMERR is generated. If the COOKIE option is longer than the maximum valid COOKIE option (40 bytes), then FORMERR is generated.

In summary, valid cookie lengths are 8 and 16 to 40 inclusive.

5.2.3. Only a Client Cookie

Based on server policy, including rate limiting, the server chooses one of the following:

(1) Silently discard the request.

(2) Send a BADCOOKIE error response.

(3) Process the request and provide a normal response. The RCODE is NOERROR, unless some non-cookie error occurs in processing the request.

If the server responds choosing (2) or (3) above, it SHALL generate its own COOKIE option containing both the Client Cookie copied from the request and a Server Cookie it has generated, and it will add this COOKIE option to the response's OPT record. Servers MUST, at least occasionally, respond to such requests to inform the client of the correct Server Cookie. This is necessary so that such a client can bootstrap to the more secure state where requests and responses have recognized Server Cookies and Client Cookies. A server is not expected to maintain per-client state to achieve this. For example, it could respond to every Nth request across all clients.

If the request was received over TCP, the server SHOULD take the authentication provided by the use of TCP into account and SHOULD choose (3). In this case, if the server is not willing to accept the security provided by TCP as a substitute for the security provided by DNS Cookies but instead chooses (2), there is some danger of an indefinite loop of retries (see Section 5.3).

5.2.4. A Client Cookie and an Invalid Server Cookie

The server examines the Server Cookie to determine if it is a valid Server Cookie that it had generated previously. This determination normally involves recalculating the Server Cookie (or the Hash part thereof) based on the Server Secret (or the previous Server Secret, if it has just changed); the received Client Cookie; the Client IP Address; and, possibly, other fields. See Appendix B.2 for an example. If the cookie is invalid, it could be because

+ it is too old

+ a client's IP address or Client Cookie changed, and the DNS server is not aware of the change

+ an anycast cluster of servers is not consistently configured, or

+ an attempt to spoof the client has occurred

The server SHALL process the request as if the invalid Server Cookie was not present, as described in Section 5.2.3.

5.2.5. A Client Cookie and a Valid Server Cookie

When a valid Server Cookie is present in the request, the server can assume that the request is from a client that it has talked to before and defensive measures for spoofed UDP requests, if any, are no longer required.

The server SHALL process the request and include a COOKIE option in the response by (a) copying the complete COOKIE option from the request or (b) generating a new COOKIE option containing both the Client Cookie copied from the request and a valid Server Cookie it has generated.

5.3. Processing Responses

The Client Cookie, when it occurs in a COOKIE option in a DNS reply, is intended to weakly assure the client that the reply came from a server at the source IP address used in the response packet, because the Client Cookie value is the value that client would send to that server in a request. In a DNS reply with multiple COOKIE options, all but the first (the one closest to the DNS header) are ignored.

A DNS client where DNS Cookies are implemented and enabled examines the response for DNS Cookies and MUST discard the response if it contains an illegal COOKIE option length or an incorrect Client Cookie value. If the client is expecting the response to contain a COOKIE option and it is missing, the response MUST be discarded. If the COOKIE option Client Cookie is correct, the client caches the Server Cookie provided, even if the response is an error response (RCODE non-zero).

If the extended RCODE in the reply is BADCOOKIE and the Client Cookie in the reply matches what was sent, it means that the server was unwilling to process the request because it did not have the correct Server Cookie in it. The client SHOULD retry the request using the new Server Cookie from the response. Repeated BADCOOKIE responses to requests that use the Server Cookie provided in the previous response may be an indication that either the shared secrets or the method for generating secrets in an anycast cluster of servers is inconsistent. If the reply to a retried request with a fresh Server Cookie is BADCOOKIE, the client SHOULD retry using TCP as the transport, since the server will likely process the request normally based on the security provided by TCP (see Section 5.2.3).

If the RCODE is some value other than BADCOOKIE, including zero, the further processing of the response proceeds normally.

5.4. Querying for a Server Cookie

In many cases, a client will learn the Server Cookie for a server as the "side effect" of another transaction; however, there may be times when this is not desirable. Therefore, a means is provided for obtaining a Server Cookie through an extension to the QUERY opcode for which opcode most existing implementations require that QDCOUNT be one (1) (see Section 4.1.2 of [RFC1035]).

For servers with DNS Cookies enabled, the QUERY opcode behavior is extended to support queries with an empty Question Section (a QDCOUNT of zero (0)), provided that an OPT record is present with a COOKIE option. Such servers will send a reply that has an empty Answer Section and has a COOKIE option containing the Client Cookie and a valid Server Cookie.

If such a query provided just a Client Cookie and no Server Cookie, the response SHALL have the RCODE NOERROR.

This mechanism can also be used to confirm/re-establish an existing Server Cookie by sending a cached Server Cookie with the Client Cookie. In this case, the response SHALL have the RCODE BADCOOKIE if the Server Cookie sent with the query was invalid and the RCODE NOERROR if it was valid.

Servers that don't support the COOKIE option will normally send FORMERR in response to such a query, though REFUSED, NOTIMP, and NOERROR without a COOKIE option are also possible in such responses.

6. NAT Considerations and Anycast Server Considerations

In the classic Internet, DNS Cookies could simply be a pseudorandom function of the Client IP Address and a Server Secret or the Server IP Address and a Client Secret. You would want to compute the Server Cookie that way, so a client could cache its Server Cookie for a particular server for an indefinite amount of time and the server could easily regenerate and check it. You could consider the Client Cookie to be a weak client signature over the Server IP Address that the client checks in replies, and you could extend this signature to cover the request ID, for example, or any other information that is returned unchanged in the reply.

But we have this reality called "NAT" [RFC3022] (including, for the purposes of this document, NAT-PT, which has been declared Historic [RFC4966]). There is no problem with DNS transactions between clients and servers behind a NAT box using local IP addresses. Nor is there a problem with NAT translation of internal addresses to external addresses or translations between IPv4 and IPv6 addresses, as long as the address mapping is relatively stable. Should the external IP address to which an internal client is being mapped change occasionally, the disruption is little more than when a client rolls over its COOKIE secret. Also, external access to a DNS server behind a NAT box is normally handled by a fixed mapping that forwards externally received DNS requests to a specific host.

However, NAT devices sometimes also map ports. This can cause multiple DNS requests and responses from multiple internal hosts to be mapped to a smaller number of external IP addresses, such as one address. Thus, there could be many clients behind a NAT box that appear to come from the same source IP address to a server outside that NAT box. If one of these were an attacker (think "zombie" or "botnet") behind a NAT box, that attacker could get the Server Cookie for some server for the outgoing IP address by just making some random request to that server. It could then include that Server Cookie in the COOKIE option of requests to the server with the forged local IP address of some other host and/or client behind the NAT box. (An attacker's possession of this Server Cookie will not help in forging responses to cause cache poisoning, as such responses are protected by the required Client Cookie.)

To fix this potential defect, it is necessary to distinguish different clients behind a NAT box from the point of view of the server. This is why the Server Cookie is specified as a pseudorandom function of both the request source IP address and the Client Cookie. From this inclusion of the Client Cookie in the calculation of the Server Cookie, it follows that, for any particular server, a stable Client Cookie is needed. If, for example, the request ID was included in the calculation of the Client Cookie, it would normally change with each request to a particular server. This would mean that each request would have to be sent twice: first, to learn the new Server Cookie based on this new Client Cookie based on the new ID, and then again using this new Client Cookie to actually get an answer. Thus, the input to the Client Cookie computation must be limited to the Server IP Address and one or more things that change slowly, such as the Client Secret.

In principle, there could be a similar problem for servers, not due to NAT but due to mechanisms like anycast that may cause requests to a DNS server at an IP address to be delivered to any one of several machines. (External requests to a DNS server behind a NAT box usually occur via port forwarding such that all such requests go to one host.) However, it is impossible to solve this in the way that the similar problem was solved for NATed clients; if the Server Cookie was included in the calculation of the Client Cookie in the same way that the Client Cookie is included in the Server Cookie, you would just get an almost infinite series of errors as a request was repeatedly retried.

For servers accessed via anycast, to successfully support DNS Cookies, either (1) the server clones must all use the same Server Secret or (2) the mechanism that distributes requests to the server clones must cause the requests from a particular client to go to a particular server for a sufficiently long period of time that

extra requests due to changes in Server Cookies resulting from accessing different server machines are not unduly burdensome. (When such anycast-accessed servers act as recursive servers or otherwise act as clients, they normally use a different unique address to source their requests, to avoid confusion in the delivery of responses.)

For simplicity, it is RECOMMENDED that the same Server Secret be used by each DNS server in a set of anycast servers. If there is limited time skew in updating this secret in different anycast servers, this can be handled by a server accepting requests containing a Server Cookie based on either its old or new secret for the maximum likely time period of such time skew (see also Section 7.1).

7. Operational and Deployment Considerations

The DNS Cookie mechanism is designed for incremental deployment and to complement the orthogonal techniques in [RFC5452]. Either or both techniques can be deployed independently at each DNS server and client. Thus, installation at the client and server end need not be synchronized.

In particular, a DNS server or client that implements the DNS Cookie mechanism can interoperate successfully with a DNS client or server that does not implement this mechanism, although, of course, in this case it will not get the benefit of the mechanism and the server involved might choose to severely rate-limit responses. When such a server or client interoperates with a client or server that also implements the DNS Cookie mechanism, these servers and clients get the security benefits of the DNS Cookie mechanism.

7.1. Client and Server Secret Rollover

The longer a secret is used, the higher the probability that it has been compromised. Thus, clients and servers are configured with a lifetime setting for their secret, and they roll over to a new secret when that lifetime expires, or earlier due to deliberate jitter as described below. The default lifetime is one day, and the maximum permitted is one month. To be precise and to make it practical to stay within limits despite long holiday weekends, daylight saving time shifts, and the like, clients and servers MUST NOT continue to use the same secret in new requests and responses for more than 36 days and SHOULD NOT continue to do so for more than 26 hours.

Many clients rolling over their secret at the same time could briefly increase server traffic, and exactly predictable rollover times for clients or servers might facilitate guessing attacks. For example, an attacker might increase the priority of attacking secrets they

believe will be in effect for an extended period of time. To avoid rollover synchronization and predictability, it is RECOMMENDED that pseudorandom jitter in the range of plus zero to minus at least 40% be applied to the time until a scheduled rollover of a COOKIE secret.

It is RECOMMENDED that a client keep the Client Cookie it is expecting in a reply until there is no longer an outstanding request associated with that Client Cookie that the client is tracking. This avoids rejection of replies due to a bad Client Cookie right after a change in the Client Secret.

It is RECOMMENDED that a server retain its previous secret after a rollover to a new secret for a configurable period of time not less than 1 second or more than 300 seconds, with a default configuration of 150 seconds. Requests with Server Cookies based on its previous secret are treated as a correct Server Cookie during that time. When a server responds to a request containing an old Server Cookie that the server is treating as correct, the server MUST include a new Server Cookie in its response.

7.2. Counters

It is RECOMMENDED that implementations include counters of the occurrences of the various types of requests and responses described in Section 5.

8. IANA Considerations

IANA has assigned the following DNS EDNS0 option code:

Value	Name	Status	Reference
10	COOKIE	Standard	RFC 7873

IANA has assigned the following DNS error code as an early allocation per [RFC7120]:

RCODE	Name	Description	Reference
23	BADCOOKIE	Bad/missing Server Cookie	RFC 7873

9. Security Considerations

DNS Cookies provide a weak form of authentication of DNS requests and responses. In particular, they provide no protection against "on-path" adversaries; that is, they provide no protection against any adversary that can observe the plaintext DNS traffic, such as an on-path router, bridge, or any device on an on-path shared link (unless the DNS traffic in question on that path is encrypted).

For example, if a host is connected via an unsecured IEEE Std. 802.11 link (Wi-Fi), any device in the vicinity that could receive and decode the 802.11 transmissions must be considered "on path". On the other hand, in a similar situation but one where 802.11 Robust Security (WPA2, also called "Wi-Fi Protected Access 2") is appropriately deployed on the Wi-Fi network nodes, only the Access Point via which the host is connecting is "on path" as far as the 802.11 link is concerned.

Despite these limitations, deployment of DNS Cookies on the global Internet is expected to provide a significant reduction in the available launch points for the traffic amplification and denial-of-service forgery attacks described in Section 2 above.

Work is underway in the IETF DPRIVE working group to provide confidentiality for DNS requests and responses that would be compatible with DNS Cookies.

Should stronger message/transaction security be desired, it is suggested that TSIG or SIG(0) security be used (see Section 3.2); however, it may be useful to use DNS Cookies in conjunction with these features. In particular, DNS Cookies could screen out many DNS messages before the cryptographic computations of TSIG or SIG(0) are required, and if SIG(0) is in use, DNS Cookies could usefully screen out many requests given that SIG(0) does not screen requests but only authenticates the response of complete transactions.

An attacker that does not know the Server Cookie could do a variety of things, such as omitting the COOKIE option or sending a random Server Cookie. In general, DNS servers need to take other measures, including rate-limiting responses, to protect from abuse in such cases. See further information in Section 5.2.

When a server or client starts receiving an increased level of requests with bad Server Cookies or replies with bad Client Cookies, it would be reasonable for it to believe that it is likely under attack, and it should consider a more frequent rollover of its secret. More rapid rollover decreases the benefit to a cookie-guessing attacker if they succeed in guessing a cookie.

9.1. Cookie Algorithm Considerations

The cookie computation algorithm for use in DNS Cookies SHOULD be based on a pseudorandom function at least as strong as 64-bit FNV (Fowler/Noll/Vo [FNV]), because an excessively weak or trivial algorithm could enable adversaries to guess cookies. However, in light of the lightweight plaintext token security provided by DNS Cookies, a strong cryptography hash algorithm may not be warranted in many cases and would cause an increased computational burden. Nevertheless, there is nothing wrong with using something stronger -- for example, HMAC-SHA-256 [RFC6234] truncated to 64 bits, assuming that a DNS processor has adequate computational resources available. DNS implementations or applications that need somewhat stronger security without a significant increase in computational load should consider more frequent changes in their client and/or Server Secret; however, this does require more frequent generation of a cryptographically strong random number [RFC4086]. See Appendices A and B for specific examples of cookie computation algorithms.

10. Implementation Considerations

The DNS COOKIE option specified herein is implemented in BIND 9.10 using an experimental option code. BIND 9.10.3 (and later) use the allocated option code.

11. References

11.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <http://www.rfc-editor.org/info/rfc6891>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <http://www.rfc-editor.org/info/rfc7120>.

11.2. Informative References

[FNV] Fowler, G., Noll, L., Vo, K., and D. Eastlake 3rd, "The FNV Non-Cryptographic Hash Algorithm", Work in Progress, draft-eastlake-fnv-10, October 2015.

[Kaminsky] Olney, M., Mullen, P., and K. Miklavcic, "Dan Kaminsky's 2008 DNS Vulnerability", July 2008, <https://www.ietf.org/mail-archive/web/dnsop/current/pdf2jgx6rzxN4.pdf>.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, DOI 10.17487/RFC2845, May 2000, <http://www.rfc-editor.org/info/rfc2845>.

[RFC2930] Eastlake 3rd, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, DOI 10.17487/RFC2930, September 2000, <http://www.rfc-editor.org/info/rfc2930>.

[RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, DOI 10.17487/RFC2931, September 2000, <http://www.rfc-editor.org/info/rfc2931>.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <http://www.rfc-editor.org/info/rfc3022>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <http://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <http://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <http://www.rfc-editor.org/info/rfc4035>.

[RFC4966] Aoun, C. and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", RFC 4966, DOI 10.17487/RFC4966, July 2007, <http://www.rfc-editor.org/info/rfc4966>.

[RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers", RFC 5452, DOI 10.17487/RFC5452, January 2009, <http://www.rfc-editor.org/info/rfc5452>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

Appendix A. Example Client Cookie Algorithms

A.1. A Simple Algorithm

A simple example method to compute Client Cookies is the FNV64 [FNV] of the Client IP Address, the Server IP Address, and the Client Secret:

```
Client Cookie =
   FNV64( Client IP Address | Server IP Address | Client Secret )
```

where "|" indicates concatenation. Some computational resources may be saved by pre-computing FNV64 through the Client IP Address. (If the order of the items concatenated above is changed to put the Server IP Address last, it might be possible to further reduce the computational effort by pre-computing FNV64 through the bytes of both the Client IP Address and the Client Secret, but this would reduce the strength of the Client Cookie and is NOT RECOMMENDED.)

A.2. A More Complex Algorithm

A more complex algorithm to calculate Client Cookies is given below. It uses more computational resources than the simpler algorithm shown in Appendix A.1.

```
Client Cookie =
   HMAC-SHA256-64( Client IP Address | Server IP Address,
                   Client Secret )
```

Appendix B. Example Server Cookie Algorithms

B.1. A Simple Algorithm

An example of a simple method producing a 64-bit Server Cookie is the FNV64 [FNV] of the request IP address, the Client Cookie, and the Server Secret.

```
Server Cookie =
   FNV64( Client IP Address | Client Cookie | Server Secret )
```

where "|" represents concatenation. (If the order of the items concatenated was changed, it might be possible to reduce the computational effort by pre-computing FNV64 through the bytes of the Server Secret and Client Cookie, but this would reduce the strength of the Server Cookie and is NOT RECOMMENDED.)

B.2. A More Complex Algorithm

Since the Server Cookie has a variable size, the server can store various information in that field as long as it is hard for an adversary to guess the entire quantity used for authentication. There should be 64 bits of entropy in the Server Cookie; for example, it could have a sub-field of 64 bits computed pseudorandomly with the Server Secret as one of the inputs to the pseudorandom function. Types of additional information that could be stored include a timestamp and/or a nonce.

The example below is one variation of the Server Cookie that has been implemented in BIND 9.10.3 (and later) releases, where the Server Cookie is 128 bits, composed as follows:

Sub-field	Size
Nonce	32 bits
Time	32 bits
Hash	64 bits

With this algorithm, the server sends a new 128-bit cookie back with every request. The Nonce field assures a low probability that there would be a duplicate.

The Time field gives the server time and makes it easy to reject old cookies.

The Hash part of the Server Cookie is the part that is hard to guess. In BIND 9.10.3 (and later), its computation can be configured to use AES, HMAC-SHA-1, or, as shown below, HMAC-SHA-256:

```
hash =
    HMAC-SHA256-64( Server Secret,
        (Client Cookie | Nonce | Time | Client IP Address) )
```

where "|" represents concatenation.

Acknowledgments

The suggestions and contributions of the following are gratefully acknowledged:

Alissa Cooper, Bob Harold, Paul Hoffman, David Malone, Yoav Nir, Gayle Noble, Dan Romascanu, Tim Wicinski, and Peter Yee

Authors' Addresses

Donald E. Eastlake 3rd
Huawei Technologies
155 Beaver Street
Milford, MA 01757
United States

Phone: +1-508-333-2270
Email: d3e3e3@gmail.com

Mark Andrews
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
United States

Email: marka@isc.org