Reducing DNS Cache Poisoning Attacks

Jayashree Mohan
Department of Computer Science
and Engineering
National Institute of Technology
Karnataka, Surathkal, India
Email: jayashree2912@gmail.com

Shruthi Puranik
Department of Computer Science
and Engineering
National Institute of Technology
Karnataka, Surathkal , India
Email: shruthi3093@gmail.com

K Chandrasekaran
Department of Computer Science
and Engineering
National Institute of Technology
Karnataka, Surathkal , India
Email: kch@nitk.ac.in

***Abstract*—The increasing attacks on The Domain Name System (DNS) and the problems faced in deploying Domain Name System Security Extensions (DNSSEC) on a large scale, result in the need of a simple, and a practical approach to safeguard the DNS. In this paper, we present an efficient approach to significantly reduce the success rate of DNS cache poisoning attacks. The proposed Shift Key(S-Key) based domain name encoding scheme considerably raises the entropy of the DNS packet by encoding the domain name, using the randomly generated 4 bit S-Keys. To successfully poison a DNS cache, the attacker must now guess the 4 bit S-key as well as the encoded domain name, in addition to the port number and the transaction ID. The Bi-Query scheme captures the malicious reply packets by initiating a re-query or pairing up two consecutive requests to resolve the same domain name, thereby validating the Internet Protocol (IP) address retrieved for each domain name, before caching it. The first method proposed makes it difficult for the attacker to guess the DNS packet fields, while the latter detects and discards any packet that has been forged.**

Keywords*—*DNS Cache poisoning, Domain name encoding, DNS query, malicious IP, Bi-Query, Re-Query.

I. Introduction

The Domain Name System, also called the phonebook of the Internet, is the key element that maps the human-friendly domain names to their corresponding IP addresses. DNS is one of the services without which it would be genuinely hard to graze the internet. The initial design of DNS did not focus on the security; rather the goal was to make it scalable with the rising demands. Thus, on that point is a great deal of scope for attacker intervention and as a result, DNS can fail in the very purpose of its inception (i.e. successful domain name resolving). DNS is inherently vulnerable to attacks [1], [2], some of them being DNS hijacking [4], cache poisoning [1], and Birthday attack [3]. The principal cause for this could be accounted to the 16 bit transaction ID being the only unique identifier for each DNS response packet. Cache poisoning is one of the persistent attacks on the DNS [5], which allows an attacker to inject fake records into the DNS cache. If a forged reply packet, targeted at a particular domain contains the same transaction ID as that of the query packet sent, then the server inappropriately validates the response which is not from an authoritative server and thus, caches wrong entries and also serves them to other users who make the request to the same domain name. This could be a serious threat if the attacker hijacks and manipulates services like emails that make use of DNS [6], [7].

In this paper we discuss two algorithms, namely S-Key based domain name encoding and DNS Bi-Query to lower the probability of cache poisoning attacks. The advantages of the proposed scheme are:

- These algorithms decrease the chances of spoofing attacks by a large extent.
- No change is commanded in the DNS architecture to put through these algorithms.
- The Bi-Query algorithm effectively identifies most of the spoofed reply packets and therefore prevents it from getting cached.
- The S-Key based domain name encoding scheme increases the entropy of the DNS, thus making it hard for the attacker to formulate a forged reply.

The remainder of the paper is formed as follows: In section III. A, the working of DNS in general and the defects that provide a scope for cache poisoning are depicted. In section III.B we discuss a simple cache poisoning attack. We further go along to propose algorithms to prevent such kind of attacks, in section IV. In section V, we discuss the possible improvements that could be pulled into the proposed methods.

II. Motivation and Background

A. Motivation

DNS attacks being very popular in hacking community could be used not only to divert Internet traffic, but also to hijack emails, which was put to light recently by the CERT/CC team [6]. Let's take a step backwards and find out what comes about when an organization attempts to send an e-mail. Fig. 1 portrays the usual course at a higher degree of generalization. A DNS search is made by the organization's mail server, to get the IP address of the target mail server and the mail is sent along this route.

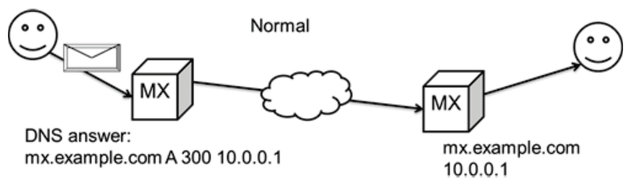

Fig. 1. A usual mail handling path (Adapted from [6])

In Fig. 2 we show how this usual process can be foiled if the DNS answer for the IP address of the destination mail server is modified. The mail unintentionally stops at the malicious server and the server can then direct the mail to its designated destination. Since mail transmission is asynchronous, the sender and the would-be-recipient are highly unlikely to detect anything strange. The sending IP address in the header of the message might indicate the diversion, but since mail management often has some exchanges before the end target, it is not really striking to anyone in the way that the diversionary attack was unusual.

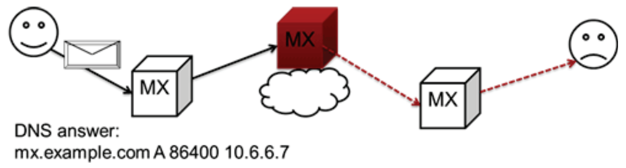

Fig. 2. A mail handling path that is hijacked through DNS cache poisoning (Adapted from [6])

If the message is not encrypted, the attacker's server will be able to read and change the contents of the message. Also, some minor changes such as modifying the bank account number and manipulating their accounts can result in huge losses to the account holders. Officially, deploying DNSSEC should prevent this issue. End-to-end encryption, or even just signing mail cryptographically, would also facilitate. Still, neither of these is deployed on a large scale, to be an immediate solution. This prompted us to offer a safer method that could not only prevent cache poisoning attacks, but also find and discard corrupted replies before taking hold of them.

B. A brief look into DNS

The DNS acts as follows:
Whenever a client requests for the resolving of a domain name, a piece of software called the Stub resolver which is built into the network operating system, first contacts the Recursive DNS Server (Usually at the ISP), if there is no matching domain name in its cache. The recursive DNS server sends a UDP query down the pecking order in a sequence: Root level domain, top level domain and finally the authoritative server. The response is cached, so as to better the efficiency of lookup. For each of these queries the recursive server waits for an answer from a remote server: either the address of the name server to be queried next or the address of the requested domain name.

C. DNS Cache poisoning attack

The basic DNS protocol does not authenticate the recursive response packets except for:

- 16 bit transaction ID.
- The source and destination port in the response packets.

An attacker's task is to guess the 16-bit transaction ID. Fig. 3 shows an attacker sending many DNS replies (spoofed) to a recursive server. If the attacker succeeds in guessing the right ID, and his reply arrives before that of the authoritative server's , the recursive server will store the wrong entry in the cache. This time window, for which the attacker can send malicious packets, [10] is known as the attacker's time window.

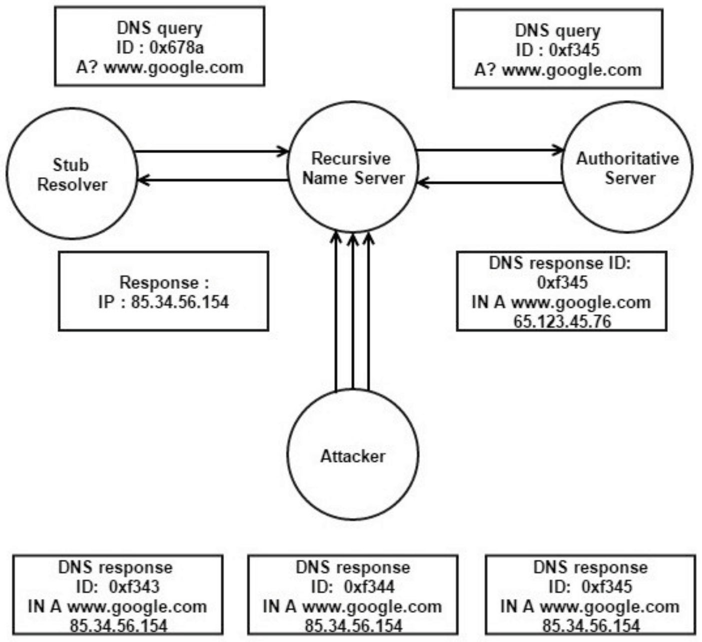

Fig. 3. A simple cache poisoning attack

III. Related Work

The well known approach to prevent DNS cache poisoning is DNSSEC [20], which is a suite of Internet Engineering Task Force (IETF) specifications. In this method, the zone administrators use public key cryptography to sign their zone data digitally so that we can be reassured it is valid. The primary drawback of DNSSEC is its execution. DNSSEC has to be implemented on all the servers (i.e. from the root servers to the authoritative servers). Besides, the encryption process would consume a heap of time and keep the clients waiting which is not satisfactory. The Domain Name Cross Referencing (DoX) [10] method takes advantage of the fact that it would be real difficult to attack more than two recursive servers simultaneously. The recursive servers are linked to each other by P2P (peer to peer) connections and they are permitted to share the cached data and update the entries accordingly. In this method, a lot of network bandwidth is wasted in the sharing of cached data. Besides, it would be really costly to establish P2P connections between the recursive servers. The 0x20-bit encoding [9] method changes the cases of the characters in the domain name, which is maintained in the reply packet as well, making it harder for the attackers to guess the correct cases of the characters in the domain name. The success of this encoding scheme is purely based on the length of the domain name, i.e. It would be easy to guess the mixed-case domain name if it is short. Consequently, we also associate a 4 bit S-Key to every label in the field name and improve the randomness of the DNS packets significantly irrespective of the length of the domain name. Hence we see the necessity of an effective approach which not just increases the entropy of the DNS packet, but as well brings down the profanity of caching malicious entries. In this paper, we put forth two methods to fetch down the chances of successful DNS attacks and transform DNS into a more fair system.

IV. The Proposed Method

A. S-key based Domain name encoding

In a DNS query or response packet, domain names such as google.com are encoded as a serial publication of length-value couples, where the first byte in every pair represents the length of the value, followed by the value (the label) itself. For instance, the domain google.com is expressed as 0x06google0x03com0x00, where the hex digits indicate the length of each label and the ASCII characters indicate the individual labels, and the zero at the end suggests the end of the domain name. Besides, it turns out that, the domain name field is replicated into the response packet, in the way it is obtained in the query packet. This structure permits the attacker to easily create a forged reply packet. To address this issue, we propose an algorithm to encode the domain name at the recursive servers and decode the same at the authoritative servers.

The proposed algorithm is as follows:

At the Recursive Server:

1) The domain name request for resolution, which is the input to the algorithm is converted to a standard form, say all lower cases.
2) Using an encryption scheme, for e.g. AES, [19] generate a random bit array for each label in the domain name with size equal to the length of the label.
3) Generate a random 4-bit number >0, called the S-Key, for each label.
4) Scan the array label by label. Let array [] represent the generated bit array for the label and label [] represent the label in the domain name to be encoded. The pseudo code for this procedure is described in Algorithm IV.1

Algorithm IV.1: ENCODING($label(), bitArray, S-Key$)

```
procedure SHIFT($value, key$)
  $k \leftarrow value + key$
  if $k > 122$
    then $k = k - 26$
  return $(k)$

main
  for $i \leftarrow 1$ to $arraySize$
        if bitArray(i)=1
          then output (label(i), SHIFT(label(i),S-Key))
    do
          else output (label(i))
        $i \leftarrow i + 1$
```

5) Express each label in its encoded form, prefixed by the S-key and the bit array used to generate them, i.e. label Length followed by S-key, Bit Array and the label itself. This encoded domain name is sent over the DNS query packet.

At the Authoritative Server:

1) The encoded domain name received in the query packet from the recursive server serves as the input to the algorithm.
2) The one byte length field in every label, say x is interpreted as TotalLength=4+x+8*x bits. The first 4 bits represent the S-Key, the next x bits represent the bit array generated for the label in question and the rest x bytes is the label itself.
3) To resolve the domain name, the encoded label has to be decoded using the S-Key and the bit array extracted.
4) Let array [] denote the extracted bit array for a label, label [] represent the encoded label. The pseudo code for this procedure is described in Algorithm IV.2

Algorithm IV.2: DECODING($label(), bitArray, S-Key$)

```
procedure SHIFT($value, key$)
  $k \leftarrow value - key$
  if $k < 97$
    then $k = k + 26$
  return $(k)$

main
  for $i \leftarrow 1$ to $arraySize$
        if bitArray(i)=1
          then output (label(i), SHIFT(label(i),S-Key))
    do
          else output (label(i))
        $i \leftarrow i + 1$
```

5) The sequence of decoded labels, separated by a period represents the original domain name, which can now be looked up to find its corresponding IP address.

Consider, for e.g. the domain name "google.com " which is to be resolved. At the recursive server, suppose the S-Key and the Bit Array for each label are as in Table I, then the encoded labels are as shown in the last column.

TABLE I. An example for domain name encoding

Label	Label Length	S-Key	Bit Array	Encoded Label
google	0x06	0001	101011	hopgmf
com	0x03	0101	110	htm

Thus the encoded domain name, as in the DNS query packet is depicted in Table II

B. DNS Bi-Query

We propose an algorithm to efficiently validate a RR reply, before caching it at the Recursive Server. In this method, for

TABLE II. THE ENCODED DOMAIN NAME

0x06 0001 101011 hopgmf	0x03 0101 110 htm	0x00

a record to be cached, a minimum of two lookups have to be made for the same domain name. The IP addresses received in the two replies are compared and cached only if they are identical. Else at least one of these IP addresses is malicious and hence we rightly discard them. Since we do not have criteria to decide which of these a corrupt reply is, we ignore both of them. If this Bi-Query strategy is not used, then the malicious IPs will be cached and served to numerous users until the TTL for the entry expires.

Algorithm to prevent Cache Poisoning using the Bi-Query strategy :

Consider a queue Q to store the incoming resolve-requests for a particular domain, a variable count to keep track of the number of DNS query packets already sent for the same domain. Whenever a client requests a name resolution, the request is enqueued into Q, and when the request is processed i.e. a query is sent to that particular request, then the count corresponding to the domain is incremented and the request is de-queued. Whenever count is equal to two, further requests are simply enqueued.

1) The input to the algorithm is the DNS request packet sent by the client to the recursive server.
2) If the entry for the requested domain name exists in the cache, it is immediately returned to the Client. Else,
   a) If the request received for the resolution of the domain name is the first one (i.e. count=0 and the queue has no requests), initiate a DNS query for the same, de-queue it and increment the count.
      i) If the queue is empty when the DNS reply packet arrives i.e. no intermediate requests have been received to resolve the same domain name, then the IP address received is delivered to the client and this IP address (say IP1) is stored temporarily without caching it. Now that the queue is empty and count=1, a re-query is issued to the same domain name and the count is incremented. Also, any further request to resolve the same domain name is not processed as the count is already equal to two. Instead, it is simply stored in the queue. The IP address obtained as a result of the re-query (say IP2) is compared with IP1, and count is made zero, as in Fig. 4.
         A) If IP1=IP2, then cache the entry for the domain. If the queue is not empty, all the pending requests are serviced with the cached IP address.
         B) Else, (i.e. IP1 != IP2) discard both the IP addresses obtained. Any requests in the queue to resolve this domain name will start as explained in step 2.
      ii) If the queue is non-empty at the time of arrival of IP1 (i.e. A request was made to resolve the domain name), since the count was equal to one, the request was sent for the resolution and the count was incremented. However, remaining requests were simply enqueued. The reply to the second request, say IP2, is now compared with IP1, as in Fig. 5.
         A) If IP1= IP2, the entry is cached. If the queue is not empty, all the pending requests are serviced with the cached IP address.
         B) Else both the IP addresses are discarded, but the requested clients still receive the respective IP addresses. If the queue is not empty, a new request is initiated as in step 2. This situation is illustrated in Fig. 6.
   b) If count is not equal to 0 i.e. the request for that particular domain is not the first one to arrive, then:
      i) If count = 1, then the behavior of the algorithm is as explained in step 2.A.ii, i.e. a second query is sent for resolution of the same domain name.
      ii) Else if count= 2, then the behavior of the algorithm is again described in step 2.A.ii i.e. the request is simply enqueued until IP2 is received.

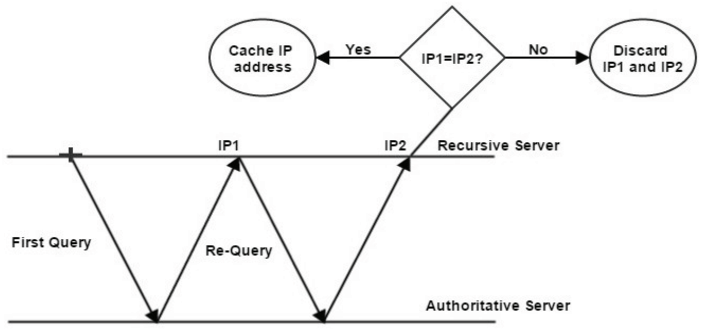

Fig. 4. Illustration of Re-Query

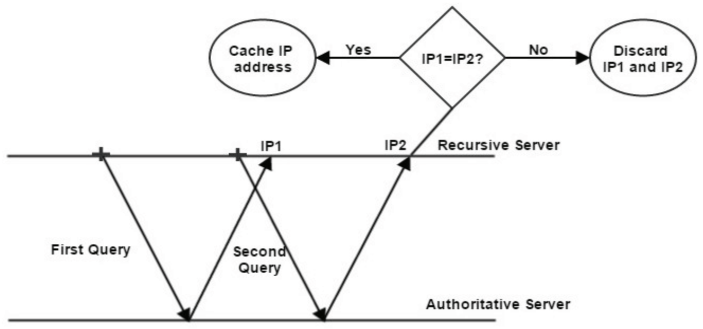

Fig. 5. Second request before the arrival of IP1

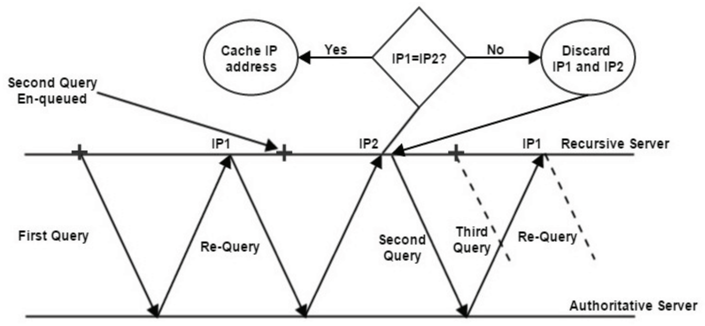

Fig. 6. Processing the third query, when both IP1 and IP2 are discarded

V. Comparative Analysis

To demonstrate the strength of the suggested method, we study the performance of the algorithms, i.e. we give the probability of the success of DNS attacks. We also analyze the already suggested/implemented methods for preventing cache poisoning so that an easy comparison can be made between them.
We introduce some variables in Table III which we would be using in our analysis [15].

TABLE III. Introduction of variables used in the analysis

ID	Number of transaction IDs, 65536
PN	Number of port numbers , maximum around 64000 as the well known ports (<1024) are not available always
NS	Total count of authoritative servers for the given domain, around 2.5 on average
NL	Number of labels in the domain name
LD	Length of the domain name excluding periods
RT	Response time of the DNS queries, around 0.1 sec
NF	Number of spoofed replies from the attacker per second
NR	Number of spoofed replies during response time of the DNS queries
T	Time for which attacker persists
NA	Number of attempts that can be made by the attacker, one per each TTL

The number of forged reply packets sent by the attacker in one window time NR is given by (1) [15].

$$NR = NF * RT \tag{1}$$

The number of attempts that can be made by the attacker during his persistence time NA is given by (2) [15].

$$NA = \frac{T}{TTL} \tag{2}$$

TTL (Time To Live) gives the time for which an entry remains in the cache.
The probability P1 of success of cache poisoning made by the attacker when the basic method is used is given in (3) . Here, the attacker becomes successful if he is able to guess the transaction ID and the address of the authoritative server rightly.

$$P1 = \frac{NR}{NS * ID} \tag{3}$$

The probability P2 of success of cache poisoning made by the attacker when the source port randomization method is used is given in (4) . In addition to the transaction ID and authoritative server's address, the attacker should also guess the source port number here.

$$P2 = \frac{NR}{NS * ID * PN} \tag{4}$$

The probability P3 of success of cache poisoning made by the attacker when the 0x20 encoding scheme is applied along with source port randomization is given in (5) . Since the cases of the characters in the domain name are changed, the attacker succeeds in poisoning the cache only if he is able to guess the mixed-case encoding of the domain name.

$$P3 = \frac{NR}{NS * ID * PN * 2^{LD}} \tag{5}$$

The probability P4 of success of cache poisoning made by the attacker when the above proposed method would be used is presented in (6) . The proposed Bi-query algorithm ensures that any domain name is resolved twice before it is cached. However, the decision as whether to cache the resolved domain name or not is completely based on the equality of the IP addresses obtained in the two replies. The S-key based domain name encoding scheme increases the randomness of the DNS packets by performing some shifting operations on the individual characters in the field name. The S-key can range from 1 to 15. i.e. 4 bits are used to represent the randomly generated S-Keys for each label in the domain.

$$P4 = \left(\frac{NR}{NS * ID * PN * 2^{LD} * 16^{NL}}\right)^2 \tag{6}$$

The probability PA of success of the attack after NA attempts is given in (7) .

$$PA = 1 - (1 - P)^{NA} \tag{7}$$

Fig. 7 portrays the graph of probability of a successful attack (P4) vs. the number of fake packets sent per second (NF), to achieve it. It is plotted using equation (6) and (1) assuming parameters [15]:
RT = 0.1s
NS = 2.5
ID = 65536
PN = 64000
LD = 12 [14]
NL = 3(average) (for e.g. abc.google.com).

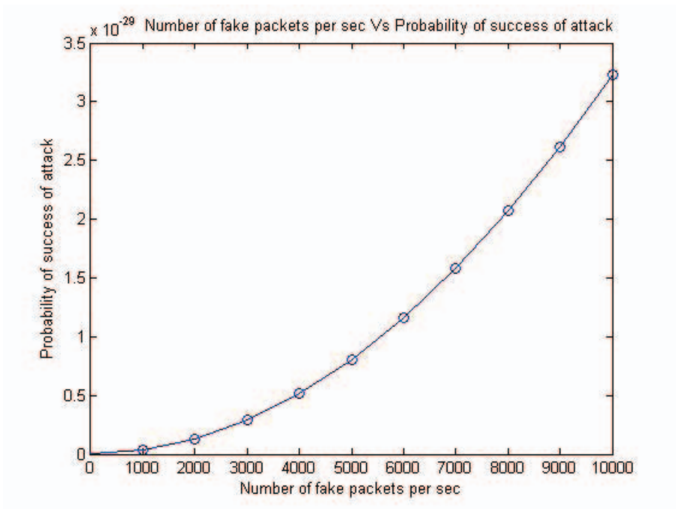

Fig. 7. Graph of Probability of Success vs. Number of fake packets.

From (3) (4) (5) (6) we see that the chance of success of a cache poisoning attack is the least in our method in comparison with 0x20, port randomization and the basic method, thereby showing that the method we proposed is more efficient than the ones already existing. This has also been illustrated in the Table IV and Fig. 8

VI. Conclusion And Future Work

Our immediate objective is to implement the algorithms proposed in this paper, for the existing DNS servers. In the DNS Bi-Query Scheme, even though a correct IP

TABLE IV. COMPARISON OF PROBABILITY OF SUCCESSFUL ATTACK USING VARIOUS METHODS

Number of packets sent per sec	Probability of success of attack			
	Basic method	Source port randomization	0x20 bit encoding	S-key based encoding + Bi-query scheme
1000	$6.1035 * 10^{-4}$	$9.5367 * 10^{-9}$	$2.3283 * 10^{-12}$	$3.2312 * 10^{-31}$
4000	0.0024	$3.8147 * 10^{-8}$	$9.3132 * 10^{-12}$	$5.1698*10^{-30}$
7000	0.0043	$6.6757 * 10^{-8}$	$1.6298 * 10^{-11}$	$1.5832*10^{-29}$
10000	0.0061	$9.5367 * 10^{-8}$	$2.3283 * 10^{-11}$	$3.2312*10^{-29}$

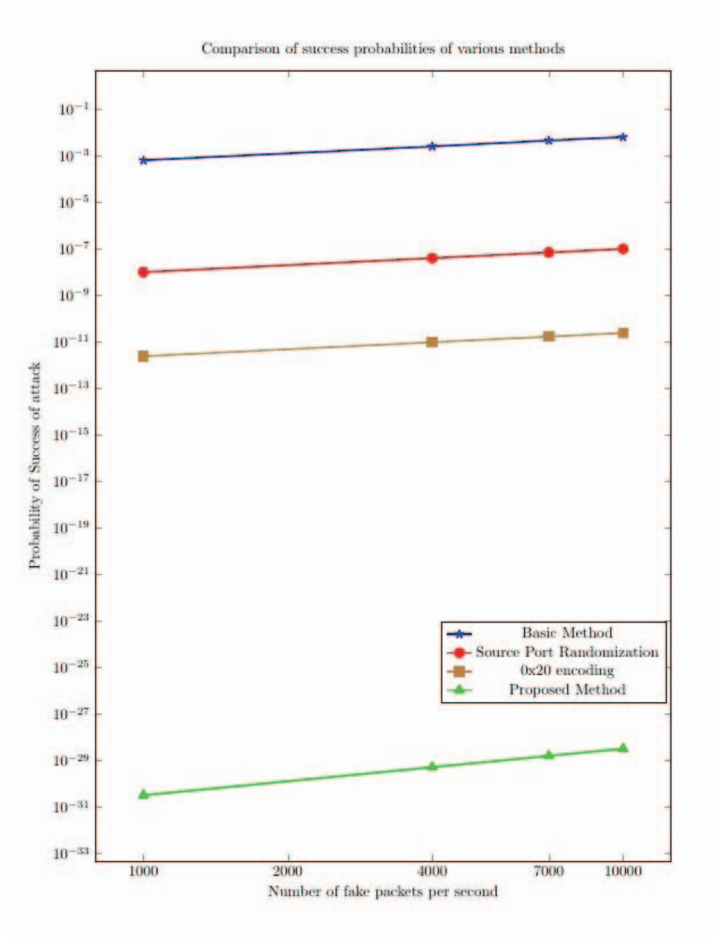

Fig. 8. Comparison of probability of success of various methods

address is retrieved, a malicious reply to the Re-query can still result in the IP address not being cached. Therefore, there is scope to improve the algorithm to identify which, among the two replies is malicious. This paper presents an efficient and easy-to-implement model to detect and prevent DNS cache poisoning, which is shaking the roots of the Internet in today's world. The proposed algorithms can be implemented without any change to the architecture of the existing servers, unlike DNSSEC. The two embedded schemes: DNS Bi-Query and S-Key based domain name encoding, make our strategy more efficient than the source port randomization and 0x20 coding schemes in terms of the success rate of attacks. The proposed scheme greatly decreases the probability of cache poisoning attacks.

REFERENCES

[1] Steve Friedl, *An Illustrated Guide to the Kaminsky DNS Vulnerability*, 3rd ed. http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html , 2008

[2] Son and V Shmatikov, *The hitchhiker's guide to DNS cache poisoning*, In: 6th International ICST Conference, SecureComm , Singapore, September 7-9, 2010.

[3] Nicholas A. Plante ,*Practical Domain Name System Security: A Survey of Common Hazards and Preventative Measures*

[4] Schuba, Christoph. *Addressing Weaknesses in the Domain Name System Protocol*, Aug. 1993: http://ftp.cerias.purdue.edu/pub/papers/christoph-schuba/schuba-DNSmsthesis.pdf

[5] J. Stewart, *DNS Cache Poisoning-The Next Generation*, LURHQ Threat Intelli, January 2003.

[6] Probable Cache Poisoning of Mail Handling Domains https://www.cert.org/blogs/certcc/post.cfm?EntryID=206

[7] DNS cache poisoning attacks to steal emails are a reality: http://securityaffairs.co/wordpress/28283/cyber-crime/dns-cache-poisoning-emails.html

[8] Email Hijacking :New Research Shows Why We Need DNSSEC Now! http://www.internetsociety.org/deploy360/blog/2014/09/email-hijacking-new-research-shows-why-we-need-dnssec-now/

[9] D. Dagon, M. Antonakakis, P. Vixie, J. Tatuya, and W. Lee, *Increased DNS forgery resistance through 0x20-bit encoding* In ACM ACM CCS, 2008.

[10] L. Yuan, K. Kant, P. Mohapatra, C.N. Chuah, *DoX: A Peer-to-Peer Antidote for DNS Cache Poisoning Attacks*, Istanbul, IEEE International Conference on Communications, 2006.

[11] Perdisci, R. Antonakakis, M. Damballa, *WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks*, Lisbon, Dependable Systems Networks, 2009.

[12] Y.W. Ju, K.H. Song, E.J. Lee, Y.T. Shin, *Cache Poisoning Detection Method for Improving Security of Recursive DNS*, Gangwon-Do, The 9th Int. Conference on Advanced Communication Technology, 2007

[13] Alexiou, N., et al.: *Formal Analysis of the Kaminsky DNS Cache- Poisoning Attack Using Probabilistic Model Checking* In: High- Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium, pages. 94-103.

[14] Lejun Fan, Yuanzhuo Wang, Xueqi Cheng and Jinming Li, *Prevent DNS cache poisoning using security proxy*, In: Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference, pages 387-393.

[15] Y Xi, C Xiaochen, X Fangqin, *Recovering and protecting against DNS cache poisoning attacks* In: Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on (Volume:2) pages 120-123

[16] C Gutierrez, R Krishnan and R Sundaram *HARD-DNS: Highly-available redundantly-distributed DNS* In: Military Communications Conference, 2010 - Milcom 2010, pages 1343-1348.

[17] J Trostle, B Van Besien, A Pujari *Protecting against DNS cache poisoning attacks* In: Secure Network Protocols (NPSec), 2010 6th IEEE Workshop, pages 25-30.

[18] V Ramasubramanian and EG Sirer, *Perils of transitive trust in the domain name system*, In: IMC '05 Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement. Pages 35-35.

[19] NIST. Announcing the advanced encryption standard (aes). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[20] D. Eastlake, *Domain Name System Security Extensions*, RFC 2535,1999