

Spoof Detection for Preventing DoS Attacks against DNS Servers

Fanglu Guo Jiawu Chen Tzi-cker Chiueh
Computer Science Department
Stony Brook University, NY 11794
{fanglu, jiawu, chiueh}@cs.sunysb.edu

Keywords: DNS spoof detection, Defense against spoofing-based DNS DoS attacks

Abstract

The Domain Name System (DNS) is a critical element of the Internet infrastructure. Even a small part of the DNS infrastructure being unavailable for a very short period of time could potentially upset the entire Internet and is thus totally unacceptable. Unfortunately, because DNS queries and responses are mostly UDP-based, it is vulnerable to spoofing-based denial of service (DoS) attacks, which are difficult to defeat without incurring significant collateral damage. The key to thwart this type of DoS attacks is spoof detection, which enables selective discarding of spoofed DNS requests without jeopardizing the quality of service to legitimate requests. This paper presents a comprehensive study on spoof detection strategies for protecting DNS servers from DoS attacks. These strategies all create some form of cookies for a DNS server to check if each incoming request is indeed from where the request packet says it is from, but vary in performance overhead, transparency and deployment complexity. We have implemented all of them as a firewall module called DNS guard. Measurements on the current DNS guard prototype show that it can deliver up to 80K requests/sec to legitimate users in the presence of DoS attacks at the rate of 250K requests/sec.

I. INTRODUCTION

The Domain Name System (DNS) is a critical component of the Internet infrastructure, because most network services and applications require a translation step from domain name to IP address to just send the packets out. As a result, even a small part of the DNS infrastructure being unavailable for a short period of time could have a significant rippling effect on the rest of the Internet. However, common DNS queries and responses use UDP as their transport protocol. The combination of the simplicity of the DNS protocol and its use of UDP makes DNS extremely vulnerable to spoofing-based Denial of Service (DoS) attack. Unlike TCP, UDP does not use three-way handshake procedure to start a connection and therefore has no way to be sure that a UDP packet indeed comes from where the packet's source address indicates. Worse yet, a DNS server only sees one UDP query and replies with one UDP response for most DNS interactions. Therefore it is not possible for a DNS server to ascertain the identity of the requesting host at the DNS level, either. As suggested in a CERT report [1], an attacker could take advantage of this vulnerability and launch a DoS attack against a DNS server by spoofing the source address of the requests. Indeed, several spoofing-based DoS attacks against DNS have been reported in the past. ComputerWire

[2] reported that seven of the Internet's thirteen DNS root servers became inaccessible for an hour. Hu [3] reported that an attack aiming at Akamai's DNS servers blocked nearly all accesses to Apple Computer, Google, Microsoft and Yahoo's Web sites for two hours. The importance of effectively stopping spoofing-based DoS attacks against DNS is pointedly summarized in [2], "If you could take down .com, what would be the cost in billions of dollars?"

There are two possible DoS attack strategies against DNS servers. The first is to send a large number of requests to a DNS server to overload it. Because a standard DNS server cannot distinguish between spoofed and non-spoofed requests, it has no choice but to handle all of them when it can, and starts to drop requests indiscriminately when it becomes overloaded. However, legitimate requesters interpret request drops as a sign of congestion and back off its timer for retransmission, thus drastically decreasing the amount of legitimate requests served by the overloaded servers.

The other attack strategy is to exploit DNS servers to amplify attack traffic. The attacker crafts a DNS request that gets a response significantly larger than the request itself, e.g., a 50-byte request for a 500-byte response. The amplified response is replied to a spoofed third-party victim machine. Under this attack, both the amplifying DNS server's upstream bandwidth and the third-party machine's downstream bandwidth could be exhausted. Due to traffic amplification, an attacker can starve the bandwidth of its victims even if his bandwidth is 10 times smaller.

An effective defense against spoofing-based DoS attacks on DNS servers requires source address spoof detection. Assuming a DNS server can distinguish between spoofed requests from real ones, it can selectively drop those spoofed ones with little collateral damage. If a DNS server is sure that the incoming requests use a genuine source IP address, it can use a rate-limiting strategy to drop packets in a fair way.

This paper presents a comprehensive study on spoof detection strategies for DNS requests. In all these strategies, a DNS server sends a distinct cookie to each requesting host, and the requester associates each request it sends to the DNS server with the server's corresponding cookie. By checking the cookie that comes with each incoming request, it is possible to determine if a DNS request indeed originates from source address indicated in the packet. However, how to introduce these cookies in a way that is transparent to the existing Internet infrastructure and incurs minimal performance overhead is the design challenge.

We have examined several spoof detection strategies and implemented all of them in a firewall module called DNS guard, which is designed to be deployed without modifying protected DNS servers or DNS requesters. It can even be deployed only when a DoS attack arises and contains the DoS attack without lengthy training or tuning. Though generic spoof detection methods exist as reviewed in Section II, these methods invariably have certain drawbacks such as false positives/negatives, deployment difficulties, etc. Among the three most popular Internet protocols used by network applications, TCP, UDP and ICMP, only UDP-spoofing attacks do not have an adequate countermeasure yet. TCP SYN flooding was addressed by SYN cookie. ICMP spoofing can be contained by limiting its rate. Among UDP-based network services, only DNS is truly indispensable and needs to be open to the whole public. Therefore, it is worthwhile to develop spoof detection techniques specifically tailored to DNS traffic.

The remaining part is organized as follows. Section II surveys previous research related to DNS and spoofing based DoS. Section III describes our three strategies to detect source spoofing in DNS requests and compares their weaknesses and strengths. Section IV reports the run-time latency and throughput of the spoof detection strategies studied. Section V concludes the paper with a summary of its major contributions.

II. RELATED WORK

1) *Protecting DNS from DoS attack*: Paxson [4] provides a thorough analysis about reflection-based DoS attack. Two attack strategies against DNS are analyzed. Fortunately, these two attacks can be controlled by filtering out replies to spoofed request at the victim site and restricting recursive servers to serve local machines only. Rikitake [5] proposes to use T/TCP as the DNS transport protocol. But T/TCP cannot hold its promise when it is attacked. As Vivo et al. [6] pointed out, the key point for T/TCP to detect spoofing is the TAO test. Unfortunately, TAO test only requires a subsequent request to use a CC value larger than its current one. Thus an attacker can just use any larger CC value to defeat the TAO test. The DNS Security Extensions (DNSSEC) [7] is designed to provide data integrity and authentication instead of authenticating the requester. It has no protection against DoS attacks.

Ramasubramanian and Sirer [8] propose a next generation name service called CoDoNS. Building on a peer-to-peer overlay network, the resistance against DoS is achieved through a decentralized architecture and dynamic load balancing. Yang et al. [9] propose a hierarchical overlay architecture, HOURS for hierarchical open services such

as DNS. Overlay is used to get rich and unpredictable connectivity. Queries are forwarded via sibling and nephew pointers in the overlay as a detour in case that DoS attack stops an intermediate node.

2) *Victim-based DoS Protection*: We classify the general DoS protection mechanisms based on which components of the system needs to be modified. It can be either at the victim side or in the network. Jin et al. [10] propose a hop count based mechanism called HCF to detect spoofed packets. Servers can learn the normal distance (hop count) of their clients. It is assumed that there is no easy way for an attacker to learn the distance between a server and its clients, thus the server can detect spoofed packets because of the incorrect hop count in the packet. In parallel, Templeton [11] also propose to use TTL to detect spoofing. Overall this is a good solution. But some drawbacks may hinder its application to DNS because of the false negative ratio and hop count learning issue.

Instead of using path information to differentiate packets, papers [12], [13] propose to profile the source IP addresses of traffic it receives under normal conditions to create a normal client model. When DoS attack is suspected, packets whose source IP address is not consistent with the model are filtered out. DNS may not use this mechanism because of the limited number of local recursive servers. An attacker may set up an authoritative name server to collect the IP address of these local recursive servers, then spoof requests with the IPs of these local recursive servers. These spoofed requests cannot be filtered out because they are consistent with the model of the authoritative name servers. Collins and Reiter [14] did a good comparison among the solutions in [10], [12], [13], [15] and pointed out several important issues such as learning time, change in network topology, etc. SYN cookie [16] is specific to protect TCP from spoofing based attacks. There is no state for a potential spoofed request with cookie. The DNS guard uses the same cookie idea to avoid maintaining states.

3) *Network-based DoS Protection*: Ingress filtering [17] deploys filters at the edge of the network to discard packets whose source IP is not in the edge network. Unfortunately, its effectiveness depends on the universal deployment. In reality, the deployment is slow due to administrative burden, lack of incentive and so on. To facilitate ingress filtering, Li et al. [18] propose a protocol called Save to allow routers to construct an incoming table to validate the source IP address of packets.

The Pushback mechanism [19] allow routers to limit attack traffic to some destinations. Unfortunately, the filter

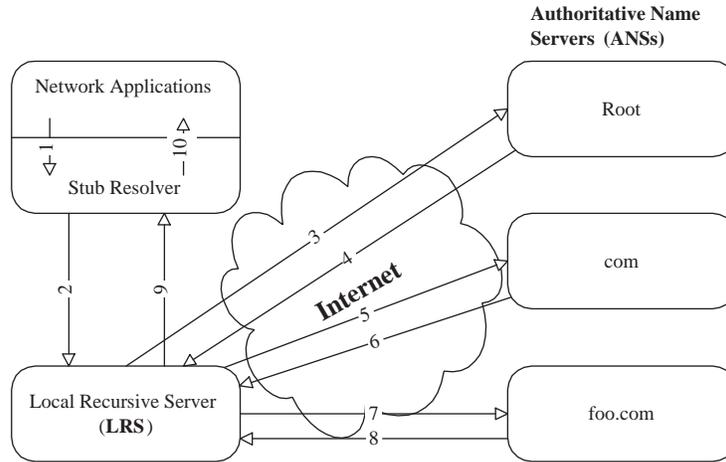


Fig. 1. The architecture of DNS

may block some legitimate packets because there is no way to differentiate attack traffic from other traffic. Yaar et al. [20] propose SIFF, a Stateless Internet Flow Filter to allow a receiver to notify a router to selectively stop individual flows from reaching its network. Similarly, Yang et al. [21] propose a network architecture called TVA which uses capabilities to allow a receiver to grant permissions to senders and enhance the weakness of [20]. In [20], [21], individual packet is marked thus there is little collateral damage. Yaar et al. [15] propose a mechanism called Pi (Path identifier). Routers are modified to mark packets. Network-based solutions are in a better position to suppress attacks at early stage. The key issue in network-based solutions is the difficulties in changing the network.

III. DESIGN AND IMPLEMENTATION

A. Overview

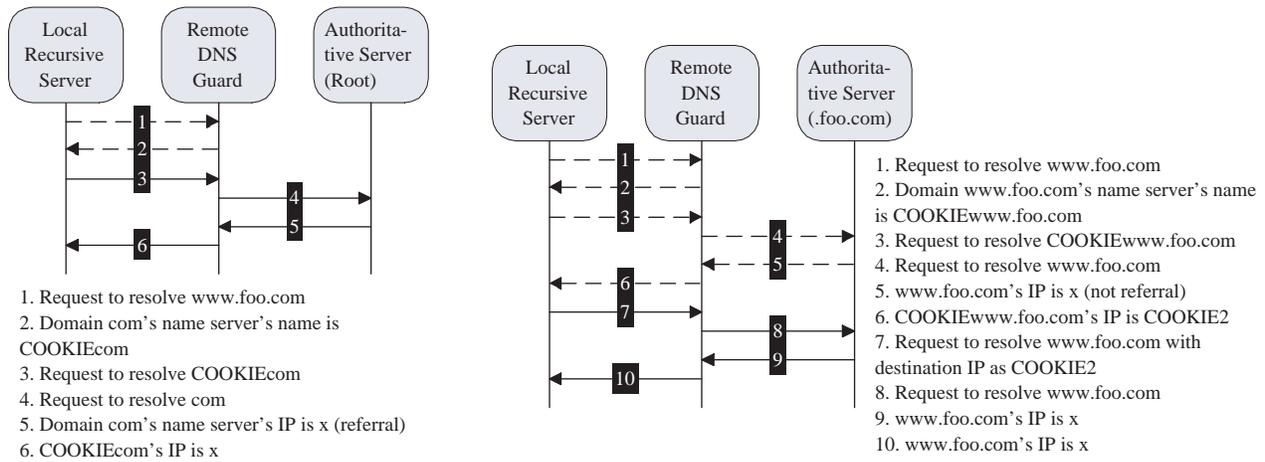
As shown in Figure 1, the DNS infrastructure comprises three types of components: the *stub resolver*, *local recursive server* (LRS), and *authoritative name server* (ANS). The stub resolver is typically implemented as a library on an end-user machine. It is not sophisticated enough to do everything that a local recursive server can. Whenever a network application requests a name resolution, the stub resolver simply sends a recursive DNS request to the local recursive server (message 2 in Figure 1).

The local recursive server (LRS) is usually set up for an organization, e.g., a department in a university. LRS provides two main functionalities. First, it is capable of serving recursive requests. To answer a recursive request, LRS sends one or multiple iterative requests (message 3, 5, and 7) to multiple ANSs. Second, LRS can cache the answers from ANSs, and queries ANSs only when it cannot answer with its cache.

The authoritative name servers (ANSs) maintain a name-address mapping database. In Figure 1, three example ANSs are shown: *root*, *com*, and *foo.com*. For example, to resolve the address of *www.foo.com*, one of the root DNS servers is queried. Currently there are thirteen well-known IP addresses for root DNS servers world-wide. The root DNS server returns the name and IP address of the *com* domain's name server using an NS (Name Server) record and A (Address) record in message 4. Then the LRS queries the *com* domain name server for the IP address of the *foo.com* domain name server, which in turn answers the IP address of the host *www.foo.com* using an A (Address) record. In the above process, the root server and the *com* domain name server only provide referral information (NS records and A records for the ANSs of the next level of domain). The *foo.com* name server provides the final authoritative answer.

In most cases, an LRS sends one request packet to an ANS and gets one response. Additionally, neither UDP/IP nor DNS provides any authentication about a request's source. These two factors make it easy to mount spoofing-based DoS attacks against DNS servers. Because ANSs need to be open to the whole Internet and are likely to be the target of a DoS attack, we focus this project on spoof detection schemes that protect ANSs without requiring modifications on LRSs. An effective spoof detection algorithm should be able to detect spoofed DNS requests, minimizes traffic amplification, requires minimal infrastructure changes to facilitate deployment, and incurs low run-time overhead.

The general strategy to ascertain the source of a DNS request is to send a cookie to the requesting host after receiving the first request, and require the requesting client to attach the cookie to all subsequent requests. There are two design issues: (1) How to return a cookie to an LRS? (2) How to trick an LRS into embedding a cookie in every requests. We explore three schemes in this section. The first scheme is to embed cookies into legitimate DNS messages, where the cookie could be represented by a *referral's name* or a *part of an IP address*. The second scheme is TCP-based DNS, where the cookie is represented by TCP's sequence number. The third scheme is to modify DNS by explicitly introducing a cookie exchange procedure. The first two schemes do not require modifications to LRS, whereas the third scheme does.



(a) Embedding the cookie in the NS name for referral answers. (b) Embedding the cookie in fabricated NS name and IP for non-referral answers.

Fig. 2. Using NS name and IP to embed cookie in traditional DNS. Messages presented as dashed lines will be skipped after first access.

B. DNS-based: Embedding Cookies in DNS Messages

The ANS can return two kinds of answers: a referral answer or a non-referral answer. A referral answer provides information about the ANSs in the next level of the domain name hierarchy. A non-referral answer is any answer that is not a referral.

1) *Referral Answer: Embedding Cookie in NS Name:* The referral information in DNS is represented in two types of resource records. The first type is the NS (Name Server) record, which provides the name of an ANS. The second type is the A (Address) record, which provides the IP addresses of an ANS. If an LRS only receives the name of an ANS, it issues another query to find out the ANS's IP address and query the ANS.

The key idea here is to exploit the fact that an LRS is capable of executing further queries when the LRS only receives the name of an ANS. Basically this algorithm replaces the real name of an ANS with a fabricated name in which the cookie is embedded. That is, an LRS never sees the real NS records. Instead, a fabricated NS record is received for each domain.

Figure 2(a) illustrates an example on how the DNS guard protects a root DNS server. The DNS guard is deployed in front of an ANS. The LRS and ANS are traditional DNS servers. In message 1, LRS sends a DNS request to the root server to resolve the host name *www.foo.com*. The DNS guard intercepts this message and returns the name of the ANS for the *com* domain as *COOKIEcom* in message 2. The name in the NS record in message 2 is fabricated.

In this step, ANS is not involved. *COOKIE* is a secret generated for each distinct requester based on the source IP address in the DNS request. The suffix *com* is used to restore the original request information. Because the name *COOKIEcom* is in the root domain, the LRS requests the root DNS server again to resolve name *COOKIEcom* in message 3. Upon receiving message 3, the DNS guard realizes this request has a cookie embedded and checks the validity of the cookie. Only request packets that passes this check are forwarded to the root server. In message 4, the original request information, *com*, is restored and the request is modified to resolve the name *com*. Although it is not the original request *www.foo.com*, it is sufficient for now because the root DNS server only cares about the first-level domain. Message 5 returns the IP address of the ANS for the *com* domain to the DNS guard, which then returns this IP address as the IP address of *COOKIEcom* in message 6. The above scheme can detect source address spoofing because a successful exchange of cookie information between the requesting LRS and the DNS guard requires both parties to use a real IP address.

There are several issues in the above design. First, instead of two packets and one round trip time (RTT), there are four packets and two RTTs for each interaction. Fortunately, this overhead can be minimized by setting a large Time To Live (TTL) value for the fabricated NS record (i.e., *COOKIEcom* in the above example). This doesn't break the TTL design of the original referral information because the TTL of the ANS's IP address does not change. In normal operations, the fabricated NS records rarely expire. So the LRS can query an ANS with a cookie embedded NS name directly when the ANS's IP address expires. That is, an LRS's cache could eliminate message 1 and 2 during most operations.

Second, in each domain, there are usually multiple ANSs each with a distinct name and IP address. In the above scheme, the LRS never sees the real names of the ANS but fabricated names with cookies embedded. Fortunately, one name can be mapped to multiple IP addresses. By returning multiple IP addresses for a fabricated domain name, the LRS can still use multiple ANSs.

Third, the current scheme assumes that the ANS will return the IP address of the next-level ANSs in message 5. But this is not always true. Sometimes an ANS only returns the name of the next-level ANSs. Fortunately, standard DNS delegation practice requires each next-level domain to provide both the name and IP address of its ANS. By

adding the IP address to the ANS's zone file, all ANSs can return the IP address of the next-level ANSs in message 5.

Fourth, the cookie is encoded in the form of *COOKIEcom* in the above example. This means both the cookie and the original question are encoded in one *label* whose length is limited to 64 bytes according to RFC 1035. Fortunately, legitimate domain names are much shorter than 64 bytes. As a result, there is plenty of room to embed cookies.

2) *Non-Referral Answer: Embedding Cookie in NS Name and IP:* The above scheme does not work if the ANS returns non-referral information, e.g., an A (Address) resource record for the queried name. As in Figure 2(b), the LRS contacts the ANS for the domain *foo.com*. Message 5 is not a referral's IP address but the IP address corresponding to the name *www.foo.com*. We cannot return this IP address in message 6 because LRS thinks this is the IP address of the ANS for the domain *www.foo.com*.

One choice is that the DNS guard returns this name server's IP address and sets up a small time window to allow the LRS to issue another request. If the same requester sends another request again within the window, it is a legitimate request and will be served in message 7 to 10. However, there are several disadvantages. First, if an attacker spoofs requests with the IP address of a legitimate LRS, the time window allows an attacker to get in when it is opened by some legitimate LRSs. Second, we must set the TTL of message 6 as zero so that the LRS will send message 3 again to notify the DNS guard to setup time window for message 7. The best-case of this solution still requires 2 RTTs.

Instead, we introduce a second cookie to achieve 1 RTT for the best case. The key idea is to fabricate an ANS for each non-referral answer. For each fabricated ANS, two records are faked: an NS record and an A record. Each record embeds one cookie.

As the example in Figure 2(b), the DNS guard creates an artificial ANS for a nonexistent domain *www.foo.com*. The fabricated NS record says that domain *www.foo.com*'s ANS name is *COOKIEwww.foo.com*. The fabricated A record says that the ANS *COOKIEwww.foo.com*'s IP is *COOKIE2*. Assuming the DNS guard module is part of a firewall and is able to intercept all traffic to 1.2.3.0/24, *COOKIE2* will be 1.2.3.y. Then LRS will send message

7 to 1.2.3.y to query again. The DNS guard intercepts this query and checks if y is consistent with its source IP address. If it is a correct cookie, the DNS guard queries the ANS (or use previous result got from message 5) and replies the result (e.g., *www.foo.com*'s IP is x) in message 10 to LRS. In this solution, the TTL of COOKIE2 can be as large as COOKIE. There is no need to setup time window because COOKIE2 itself serves as a cookie. Thus when COOKIE and COOKIE2 are not expired, it only requires 2 packets (message 7 and 10) and 1 RTT.

The security strength of this solution depends on the range of y, say R_y . If R_y is small, an attacker can simply spread attack packets randomly in the range of y and successfully penetrate the DNS guard with a probability of $\frac{1}{R_y}$.

3) *Summary:* In summary, this DNS-based scheme embeds cookies in fabricated NS records for referral answers. For non-referral answers, a fabricated ANS (an NS record and an A record) is created for each non-referral request. The scheme can be implemented as a firewall module and is totally transparent to both ANS and LRS. Neither ANS nor LRS needs to be modified. This transparency is the key advantage of this scheme. However, it pays its price by creating more state and/or latency overhead. The solution for referral answers doesn't have extra state or latency overhead. The cookie embedded NS records need to be cached by LRS anyway. The maximum latency is 2 RTT and only happens when an LRS contacts an ANS for the first time. The solution for non-referral answers is less good. Encoding cookies as the IP address of a fabricated ANS limits the security strength to the size of the subnet where the DNS guard is deployed. The maximum latency is 3 RTT when the LRS contacts an ANS for the first time. Moreover, each name requires a fabricated ANS (an NS record and an A record) being cached in the LRS, which means multiple cookies are stored duplicately when there are multiple names cached from the same ANS.

C. TCP-based: Transparently Fall Back to TCP

The key idea of this scheme is to exploit the DNS truncation notification mechanism to notify a requesting LRS to use TCP as the transport protocol. Because TCP uses three-way handshake to establish a connection, it is difficult to spoof the source IP address for DNS requests. Normally DNS uses UDP and limits the UDP message size to be 512 bytes or less. For message size larger than 512 bytes, ANS replies with a truncation flag. Then the LRS will

automatically initiate a TCP connection and send the request again on the TCP connection.

The DNS guard exploits this feature and replies with a truncation flag on behalf of its protected ANS when spoofing-based DoS is suspected. Furthermore, the DNS guard uses a TCP proxy module to mitigate the performance penalties of TCP on an ANS. The TCP proxy transparently terminates the TCP connection to offload the protected ANS from TCP processing. It is implemented in the Linux kernel to eliminate context switches. From the LRS's point of view, it still thinks it is interacting with the protected ANS. TCP packets from LRSs are intercepted by the DNS guard and the destination IP address is transparently changed to the IP address of the DNS guard (The DNAT mechanism in Linux netfilter). Thus the TCP proxy can transparently terminate the TCP connection. The request is converted to UDP request and passed to the ANS. UDP responses from the ANS are converted back to the corresponding TCP connection.

The TCP proxy module improves the security of TCP itself by introducing the following techniques. First, the SYN cookie mechanism is enabled to protect TCP from spoof-based SYN flooding. Second, TCP connections are closely monitored and controlled. An attacker may initiate too many concurrent TCP connections or initiate connections too frequently. The TCP proxy module traces each connection's duration. When it is 5 times longer than the RTT of the connection, the connection state is removed. The TCP proxy further uses token buckets to limit the rate that a TCP client can initiate new connections to it.

The advantage of this scheme is that it is completely compatible with the existing DNS protocol. Furthermore, this scheme doesn't require any changes to the LRSs. The downside is that it always takes 3 RTTs to complete a query: one for redirecting LRS to use TCP, one for TCP 3-way handshake and the third one for the traditional DNS request and response. Its throughput is lower than UDP-based schemes as we'll see in Section IV.

D. Modified DNS: Extending DNS Protocol

The idea of this scheme is to extend DNS protocol to explicitly support cookies so that best efficiency can be achieved. Meanwhile, it serves as the optimal baseline to benchmark the other two schemes. The extension is backward compatible with traditional DNS. It uses a similar method as DNSSEC (DNS Security, RFC 2535) to extend the DNS protocol.

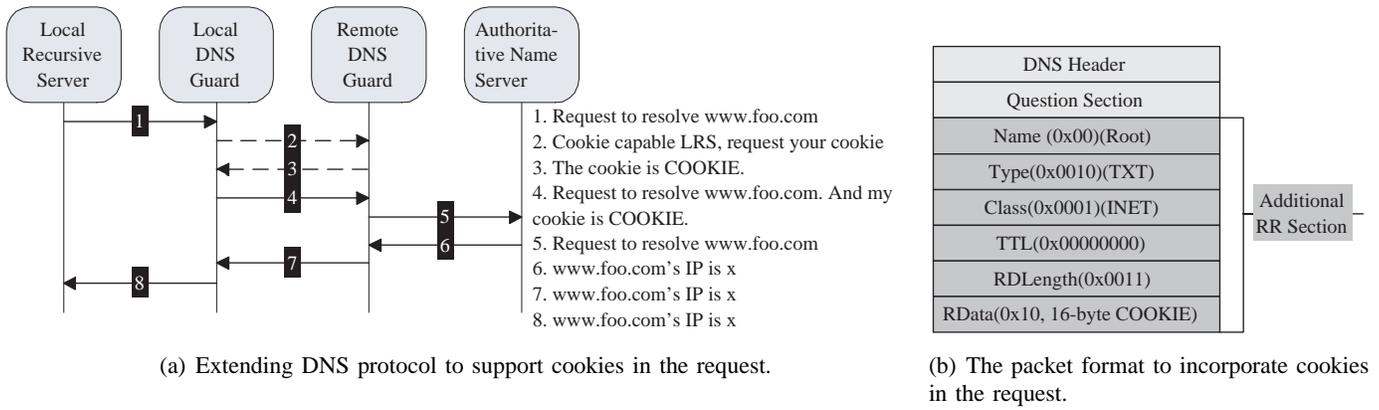


Fig. 3. The method to extend DNS protocol to incorporate cookies in the request.

As in Figure 3(a), the DNS protocol is extended and both ends need to be modified to understand the new protocol. To facilitate deployment, we developed two firewall modules which can be deployed independently of the DNS software being used. The local DNS guard is deployed in front of LRS and the remote DNS guard is deployed in front of an ANS. The LRS is not changed so it sends out traditional DNS requests (message 1). The local DNS guard intercepts the request and checks if it knows the cookie to the destination ANS. If it has cached the cookie, message 2 is skipped and message 4 is sent directly. Otherwise, message 2 is sent to request a cookie from the ANS. The remote DNS guard intercepts message 2 and returns message 3 with a cookie that is a function of the source IP address of the DNS request. The local DNS guard then sends message 4 with the cookie. The remote DNS guard checks the validity of the cookie. Only request with valid cookie is passed to the ANS. In message 5, the cookie information is removed, so the ANS doesn't see any cookie extension. Message 6, 7 and 8 are forwarded without any change. In Figure 3(a), message 2 and message 3 are only needed when LRS contacts ANS for the first time. After the first contact, the cookie is cached by the local DNS guard. Subsequent requests can use the cached cookie without using message 2 and 3.

A cookie is added to DNS request by extending the DNS message format as shown in Figure 3(b). Traditional DNS requests only have question section. This extension adds another section, *additional resource record section*. In this section, a TXT (text) resource record is included. The format of additional resource record section and TXT resource record is the same as defined in RFC 1035. The cookie is stored in the RData field.

Message 2, 3 and 4 use the same format as in Figure 3(b). When a local DNS guard doesn't know the cookie

for a new ANS, it just leaves the COOKIE field as 0 in message 2. When the remote DNS guard receives a request with an all-zero COOKIE, it returns the correct cookie for the requester in message 3. Upon receiving message 3, the local DNS guard caches the cookie based on its TTL. Message 2 and message 3 are designed to have the same size so that there is no traffic amplification.

The advantage of this extension is that it eliminates all the drawbacks of the two schemes above. First, it doesn't overload any existing DNS mechanism, so there is no any potential confliction with existing usage. Second, the cookie size is extensible and can be as long as 255 bytes, so its security strength is not a problem. Finally, it is efficient in both state and latency. For each ANS, the LRS only needs to cache one cookie as its state. In the worst case, each request requires at most 2 RTTs for the first interaction with the ANS to complete, while most of the requests are serviced in 1 RTT. Compared with the other two schemes, the drawback of this scheme is that it requires modifications to LRS side by adding a DNS guard module.

E. Cookie Design

The cookie is computed as follows. For each DNS request whose source IP address is `source_ip`, its cookie is: $c = \text{MD5}(\text{source_ip}, \text{key})$

Each DNS guard holds a 76-byte secret key. No key distribution is needed because only the DNS guard needs to know it. The 76-byte key is concatenated with the 4-byte source IP address and the resulting 80-byte plain text is fed to the MD5 hash function whose minimum input is 80 bytes. The MD5 function generates a 16-byte hash value as the cookie c for `source_ip`. For an attacker to attack an ANS, he needs to know the correct cookie c for each spoofed `source_ip`. This requires the attacker to know the ANS's key, whose large size makes it difficult to be compromised.

When a cookie is embedded in an NS name, the COOKIE is encoded in 10 bytes. The first 2 bytes serve as a prefix to differentiate COOKIE from other normal names. The remaining 8 bytes use 0 – 9, a – f to encode the first 4 bytes of the cookie c . For instance, COOKIE can be something like "PRa1b2c3d4". The prefix is "PR". The remaining part encodes a 4-byte value in hex format. In this case, the COOKIE range is 4 billion. We think this range is large enough to make random guess infeasible. Different DNS guards can also choose to use different

number of bytes for COOKIE because the COOKIE format is independent of other ANSs or LRSs. When a cookie is embedded in a destination IP address, y is the first 4 bytes of cookie c modulo R_y . The cookie computation and encoding/decoding is straightforward and efficient. This ensures that cookie checking is fast enough to sustain high attack rate.

The COOKIE size is a tradeoff between the security strength and traffic amplification. In Figure 2, message 1 may be a spoofed request. Message 2 will reply with referral name (NS record). The NS record causes message 2 longer than message 1 thus traffic amplification. The NS record is similar to the TXT record shown in Figure 3(b). The total increase is 24 bytes. Because the minimum size of a DNS request is around 50 bytes (IP packet size), the traffic amplification effect is at most 50% for DNS-based scheme. For the modified DNS scheme, the whole cookie c is stored directly in the message. Since the request and the reply has the same size, there is no traffic amplification.

If a DNS guard wants to change its key periodically, the first bit of cookie c can be overwritten and used as an indicator about the generation of cookies cached at LRS. For instance, if the TTL of COOKIE and key change interval is one week, the cookie obtained in week 0 will have its first bit as 0. In week 1, the DNS guard changes its key from key0 to key1. The DNS guard will use key0 to verify cookie that has a start bit as 0 even if its current key is key1. Thus each cookie only requires one MD5 computation. Since the TTL of cookie is one week, stale cookie of week 0 will expire when the DNS guard use key2 in week 2.

F. The Big Picture

Three spoof detection schemes are presented in this section. They all require some form of initial handshaking to ascertain the IP address of the requesting client and generate a cookie that serves as a credential in subsequent interactions. The first scheme (DNS-based) embeds cookies into existing DNS protocol messages. The second scheme (TCP-based) uses TCP's sequence numbers as cookies. The third scheme (modified DNS) introduces a cookie exchange process explicitly and requires extending the DNS protocol.

Figure 4 shows the system architecture for the DNS guard that implements all the spoof detection schemes described in this section. Each scheme works independently. The cookie checker handles all incoming UDP-based

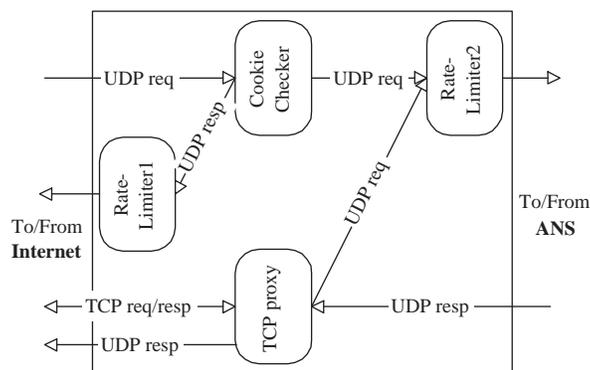


Fig. 4. The big picture of how all schemes work together in the remote DNS guard.

DNS requests. If a DNS request carries a valid cookie (supporting the modified DNS scheme) and the cookie is all 0, it generates a new cookie and returns it to the requesting LRS. This newly generated cookie response needs to pass the check of a rate limiter (*Rate-Limiter1*), which prevents the ANS from being used as a traffic reflector. *Rate-Limiter1* tracks the top requesters and limits the rate of cookie response to them. The DNS guard doesn't limit the rate of incoming DNS requests because doing so could drop requests from legitimate requesters when their addresses are used in spoofed requests. If the cookie that comes with a DNS request is valid, the request is passed to another rate limiter (*Rate-Limiter2*), which protects the ANS from non-spoofing-based DoS attacks. That is, an attacker can mount a DoS attack without address spoofing. In this case, the DNS guard simply limits the request rate of each interacting host to a nominal rate, which is usually very low.

If the requesting LRS is not cookie-capable, the DNS guard generates a cookie using the DNS-based scheme or the TCP-based scheme. In either case, the response goes through *Rate-Limiter1*. For DNS-based scheme, the subsequent DNS request goes to the cookie checker. For TCP-based scheme, subsequently a TCP connection is established with the TCP proxy module, which converts incoming DNS requests to UDP-based DNS requests and passes them to *Rate-Limiter2*. When the TCP proxy module receives a DNS response from the ANS, it either puts it in its corresponding TCP connection or sends it to the cookie checker.

Table I compares the three spoof detection schemes. The DNS-based scheme has two variants. The first variant embeds cookies in NS names and is almost as good as the modified DNS scheme. Because every NS record is modified to embed a cookie, the cookie storage at LRS side is not very efficient. The second variant fabricates NS names and IPs to convey cookies, and its effectiveness depends on the available public IP address range. In this

TABLE I. Comparison among spoof detection schemes

	DNS-based		TCP-based	Modified DNS
	NS Name	Fabricated NS Name and IP		
Worst Latency (RTT)	2	3	3	2
Best Latency (RTT)	1	1	3	1
Cookie Storage	1 cookie per NS record	2 cookies per non-referral record	0	1 cookie per ANS
Cookie Range	2^{32}	2^{32} and R_y	2^{24}	2^{128}
Traffic Amplification	< 50% (24 Bytes)	< 50% (24 Bytes)	0	0
Deployment Transparency	ANS side only		ANS side only	LRS side and ANS side

case, a DNS request's latency is 3 RTTs for the first access and 1 RTT for subsequent accesses. The DNS-based scheme could amplify DNS traffic, but at most by 50% (24 bytes) for each request. In contrast, unprotected DNS servers could amplify traffic by a factor of 10.

The TCP-based scheme uses TCP to transport DNS requests and responses. The main disadvantages of this approach are long latency and large processing overhead. Neither the DNS-based scheme nor the TCP-based scheme requires modification to LRSs. The modified DNS scheme extends the DNS protocol and thus needs to extend LRSs. But it is secure and efficient in cookie storage, and incurs small request latency without complicating protocols or amplifying traffic.

G. Attack Analysis

The goal of spoof detection is to protect an ANS from being bombarded by a spoofing-based DoS attack or becoming a traffic amplifier. One prevents traffic amplification by designing short response packets when a DNS request's source IP address is not verified yet. In the DNS-based scheme, the traffic amplification ratio is less than 50%. For the TCP-based and modified DNS scheme, the truncation response and cookie response are of the same size as the DNS request, so there is no traffic amplification at all. Moreover, Rate-Limiter1 could control the DNS response rate to top requesters, and thus makes it difficult to use ANS as a traffic amplifier.

Another attack is to guess the value of a cookie. The first way is to brute-force all possible values of a victim host's cookie. The cookie range for NS name can be easily larger than 4 billion, and the cookie range for the modified DNS scheme is 16 bytes. The fabricated NS and IP variant of the DNS-based scheme has the smallest

cookie range. For small networks, the range of R_y may be less than 254, and therefore it is not difficult to enumerate all possible values of y . One attack strategy is to send an attack request to the ANS with a guessed y value. While the attack traffic is going on, the attacker does a normal DNS query to the ANS to probe its performance and see if the guessed value is correct. For this attack to succeed, the attack rate must be sufficiently high to saturate the ANS. Fortunately, Rate-Limiter2 can control the attack request rate and make it difficult to check if a guessed y value is correct or not. As a last resort, an attacker can distribute his attack requests randomly in the cookie range, e.g., R_y . Then $\frac{1}{R_y}$ of the attack requests will have a correct cookie value and thus succeed. This is the worst false negative ratio for DNS guard.

Yet another attack is to brute-force the key used in generating the cookie. We use the MD5 hash function with a 76-byte key. Up to now, it is generally believed that it is very difficult to obtain the input to an MD5 hash function from its output. Explicitly enumerating all possible keys also seems impractical.

One can also obtain a host's cookie with respect to an ANS by sending a DNS request to the ANS and sniffing the network for the corresponding response. However, this requires the attacker to be on the same subnet as the spoofed host. Even when an attacker successfully obtains a host's cookie, not much damage can be done because Rate-Limiter2 can throttle an individual host's request rate. This is the same case as when the attacker mounts a DoS attack using public or zombie computers.

An attacker could act as a stub resolver and send attack requests to an LRS, which then forwards it to the ANS on behalf of the attacker. In this case, the LRS is a legitimate requester and there is no spoofing. But most LRSs limit the set of nodes that they are willing to serve. For example, the LRS for a computer science department may be configured to only serve computers in that department. As a result, it is difficult to recruit a large number of legitimate LRSs to attack an ANS using this method.

Another attack is to overload the cookie checker on the DNS guard. The current cookie checker uses the MD5 hash algorithm and simple encoding/decoding. As shown in Section IV, the cookie checker sustains large attack rates and cannot be easily overwhelmed.

IV. PERFORMANCE EVALUATION

A. Testbed Setup

We set up a testbed to evaluate the performance of the three DNS spoof detection schemes. The testbed consists of six nodes: one remote DNS guard, one local DNS guard, one ANS and three LRSs. The ANS and the three LRSs are connected via the DNS guards, which handle cookies in DNS requests and deliver only valid ones to the ANS. The local DNS guard is only used when testing the modified DNS scheme. The DNS guards are DELL 600SC with 2.4 GHz CPU, 512MB memory, and a dual-port Intel gigabit Ethernet card with 33-MHz 64-bit PCI interface. The ANS and 3 LRSs are DELL 400SC machines with 2.26 GHz CPU, 512MB memory, and an Intel gigabit Ethernet card with 33-MHz 32-bit PCI interface. All machines run Linux 2.4.31. The DNS guards work in the router mode and the spoof detection mechanisms are implemented in the iptable module. The ANS and LRSs run BIND (version 9.3.1) or a DNS simulator program to test the throughput of the DNS guard. Unless specified otherwise, each reported number is an average of 20 measurements. The average round-trip time (RTT) between the LRSs and the ANS on the testbed is 0.4 msec.

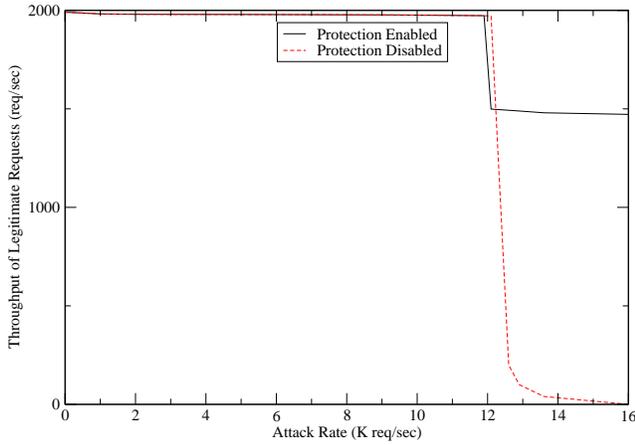
B. DNS Request Latency

In this test, the ANS is on a university campus network and the LRS is on a cable-modem network that is connected to ANS through the Internet, and the average round-trip time (RTT) between the ANS and the LRS is 10.9 msec.

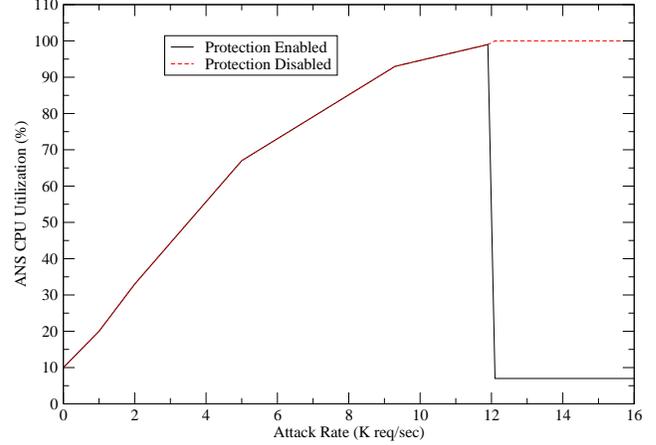
Table II shows the average DNS request latency under different spoof protection schemes for the first access (cache miss) and subsequent accesses (cache hit). When an LRS interacts with the DNS guard for the first time, it needs to get a cookie, which takes multiple RTTs. After the first access, the LRS can cache the cookie and reuse it in subsequent accesses, which need only one RTT to complete. Therefore, the latency of most DNS requests is mainly limited to RTT. By capturing packets at the ANS, the DNS guards and the requesting LRS, we find that the combined processing time inside the DNS guards, the ANS and the requesting LRS is less than 1 msec even for cases that need 3 RTTs. Among the spoof detection schemes studied, the TCP-based scheme incurs the worst latency, and the DNS-based schemes are comparable to the modified DNS scheme.

TABLE II. Average DNS request latency (msec) for different spoof detection schemes. The average RTT between the requesting LRS and the ANS is 10.9 msec.

	DNS-based		TCP-based	Modified DNS
	NS Name	Fabricated NS Name/IP		
Cache Miss	21.0	32.1	34.5	22.4
Cache Hit	11.1	11.3	33.7	10.8



(a) The throughput of legitimate requests.



(b) The CPU utilization of the ANS

Fig. 5. Throughput and CPU utilization of an ANS running BIND 9 when the DNS guard is turned on and off. The DNS guard can protect the legitimate request throughput of a BIND server and reduce its CPU utilization when it is being attacked.

C. BIND Throughput under Attack

In this subsection, we report the DNS request throughput of a BIND-based ANS with and without the DNS guard when it is being attacked. All machines run BIND version 9.3.1. First we measure the maximum throughput of BIND, which is 14K requests/sec when UDP is used, and 2.2K requests/sec when TCP is used.

Then, we measure the throughput of the ANS when it is attacked. In this experiment, one ANS and three LRSs are used. The TTL of each DNS response is configured to be 0 to disable DNS caching. The DNS guard uses UDP-based cookies with the first LRS, and TCP redirection with the second LRS. The third LRS sends UDP-based attack requests. By default the first and the second LRS each send UDP-based legitimate requests at a constant rate of 1K requests/sec. The sending rate of the third LRS is varied in the experiment. The DNS guard is configured to use the NS name mechanism for spoof detection. Other UDP-based spoof detection schemes show similar performance results.

Figure 5 shows the throughput of the ANS (the BIND server) and its CPU utilization when the DNS guard is turned on and off. When the DNS guard is completely turned off, the ANS’s CPU utilization keeps on increasing

with the attack request rate. As the attack request rate becomes greater than 12K requests/sec, the ANS starts to saturate and the throughput of legitimate requests drops dramatically, because the BIND-based LRS uses a large time-out value of 2 seconds. With a large time-out value, the legitimate request rate decreases very quickly even with very small loss rate. This test shows that BIND is extremely vulnerable to DoS attack. People may wonder why normal operation rarely shows this vulnerability. As reported in [22], the peak request rate at a root server is only 5 K request/sec, which is well below BIND's capacity.

Because spoof detection requires additional computation overhead, it is advisable to enable the DNS guard's spoof detection mechanism only when the input request rate exceeds a threshold. Since the ANS's capacity is around 14K requests/sec, we set the threshold at 14K requests/sec in this test. When the attack request rate is below 12K requests/sec, there is no spoof detection even if the DNS guard is enabled and all incoming packets go to the ANS. As soon as the attack request rate exceeds 12K requests/sec, the DNS guard's spoof detection mechanisms kick in, and the ANS's CPU utilization drops immediately because the DNS guard filters out all attack requests, as shown in Figure 5(b). At the same time, the DNS guard is able to maintain a fixed legitimate request throughput regardless of the increase in attack request rate. But the legitimate request suffers a little bit because the second LRS is redirected to use TCP and LRS's TCP maximum throughput is only 0.5K requests/sec.

D. DNS Guard Throughput Without Attack

We measured the DNS Guard throughput of different spoof detection schemes using an ANS simulator and an LRS simulator because the throughput of BIND is too low to stress the DNS guard prototype. The ANS simulator responds to each DNS request with the same answer. Its maximum throughput can reach around 110K requests/second on our testbed. The LRS simulator repeatedly submits requests to resolve the same domain name, and is able to handle DNS responses containing NS records, A records, and truncation flag. After submitting a request, the LRS simulator waits for the associated response for 10 msec, and sends in the next request if it receives a response or the timer expires.

When an LRS contacts an ANS for the first time, it needs to get the cookie. This is the worst-case scenario for the DNS guard. We measure this worst-case throughput by disabling cookie caching in LRS. For each request, LRS

TABLE III. Average DNS request throughput (requests/sec) for different spoof detection schemes.

	DNS-based		TCP-based	Modified DNS
	NS Name	Fabricated NS Name/IP		
Cache Miss	84.2K	60.1K	22.7K	84.3K
Cache Hit	110.1K	109.7K	22.7K	110.3K

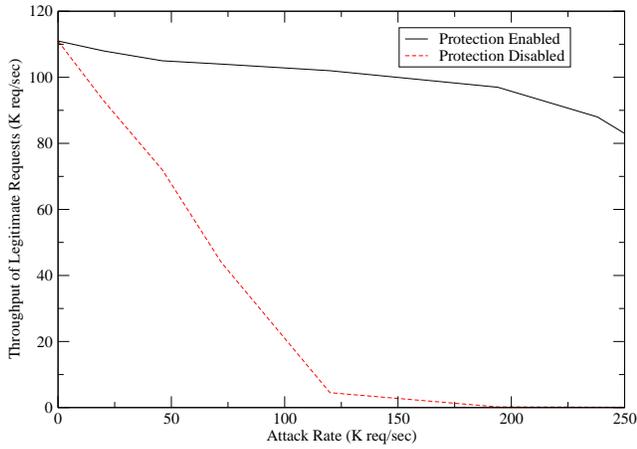
takes multiple RTTs to finish one request. Table III shows the DNS request throughput between an ANS simulator and an LRS simulator under different spoof detection schemes when the CPU of the DNS guard is fully utilized.

Because the TCP-based scheme needs to establish a TCP connection and transfer 10 to 12 packets just to service one DNS request, its throughput is the lowest at around 22K requests/sec. The fabricated NS name/IP scheme needs to compute the cookie three times and transfer 8 packets to service one DNS request, its throughput is around 60K requests/sec and is reasonably good compared with the modified DNS scheme. The modified DNS scheme and the NS name scheme need to compute the cookie only twice and transfer 6 packets to service one DNS request, their throughput is around 84K requests/sec. Theoretically, their throughput should be between $\frac{3}{2}$ (cookie computation) and $\frac{8}{6}$ (packet processing) times of that of the fabricated NS name/IP scheme. The experimental results are consistent with the theoretical predictions.

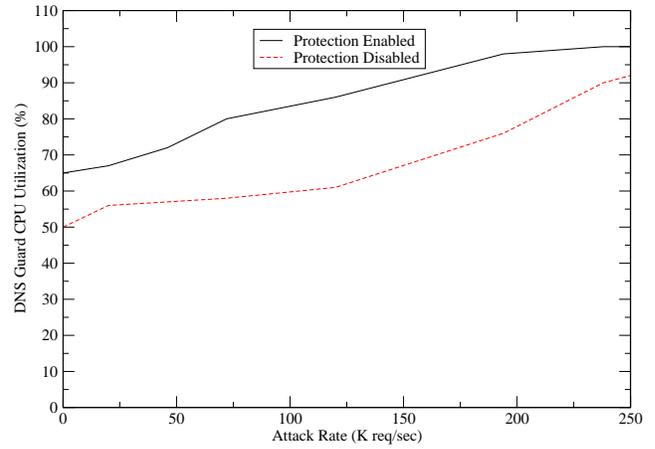
For non-TCP-based spoof detection schemes, the best throughput occurs when the cookie is cached (cache hit) after the first access. The DNS guard computes the cookie once and transfer just 4 packets to service one DNS request. In this cache hit case, the throughput of non-TCP-based schemes is around 110K requests/sec. Theoretically, its throughput should be between $\frac{3}{1}$ (cookie computation) and $\frac{8}{4}$ (packet processing) times of the fabricated NS name/IP scheme, i.e., between 120K and 180K rather than 110K. The discrepancy arises because the ANS becomes the bottleneck: the ANS's CPU is fully utilized whereas the DNS guard's CPU is only 65% utilized.

E. DNS Guard Throughput Under Attack

In this experiment, we use two LRSs, one as a legitimate LRS that already has the correct cookie for the ANS, and the other as an attacker that spoofs requests and does not have the right cookie. The legitimate LRS sends requests to the ANS as fast as possible. So the ANS is always saturated by the requests from the legitimate LRS. We then vary the attacker's DNS request rate to evaluate how effectively the DNS guard can prevent degradation of the legitimate LRS's received throughput in the presence of attacks. Only the results for the modified DNS scheme



(a) The throughput of legitimate requests.



(b) The CPU utilization of the remote DNS guard.

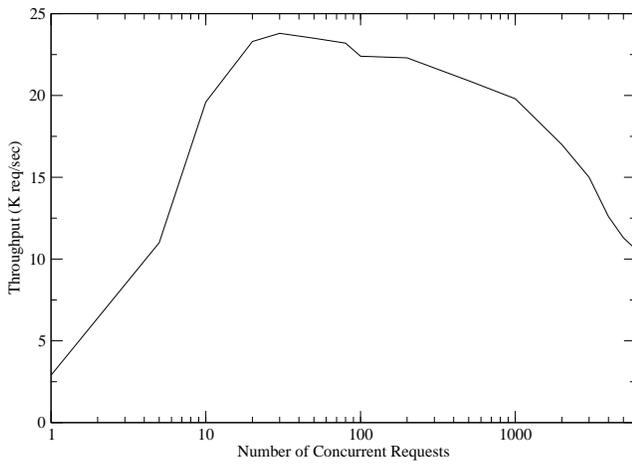
Fig. 6. The modified DNS scheme and the DNS-based schemes can protect the throughput of legitimate DNS requests at a CPU overhead of 15% to 25%.

are reported, as the DNS-based schemes are almost identical.

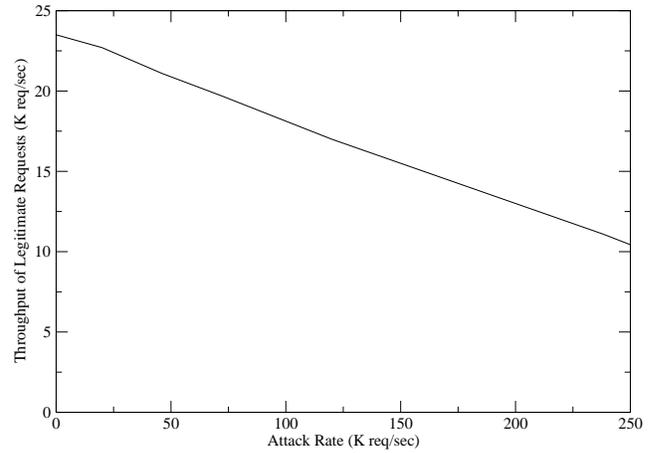
Figure 6(a) shows the throughput of legitimate requests. When the DNS guard is disabled, the throughput of legitimate requests decreases almost linearly with the increase in the attack rate, because legitimate requests can only take the remaining ANS capacity left by the attack requests. The reason for this behavior is that the legitimate LRS waits for 10 msec if requests are lost, whereas the attacker LRS never waits. When the attack request rate reaches around 110K requests/sec, the legitimate LRS is effectively halted. In practice, BIND use 2 seconds for the wait timer, which leaves legitimate LRSs at a even more vulnerable position.

When the DNS guard is enabled, the throughput of legitimate requests also decreases with the increase in attack request rate, but at a much slower rate. Consequently, the DNS guard is able to maintain the legitimate request throughput at no less than 100K requests/sec even when the attack request rate reaches 200K requests/sec. The reason for this graceful degradation behavior is that the DNS guard can recognize and drop attack requests so that they can never reach the ANS. That is, legitimate requests don't need to compete for the ANS's capacity with attack requests. When the attack request rate is above 200K requests/sec, the CPU utilization of the DNS guard is nearly 100%, as shown in Figure 6(b). When the DNS guard's CPU becomes the performance bottleneck, it has no choice but to drop legitimate requests thus the legitimate request throughput starts to degrade quickly.

Figure 6(b) shows the CPU utilization of the DNS guard machine when the DNS guard is turned on and off. The difference between these two curves represents the CPU overhead of spoof detection algorithms, which is generally



(a) Throughput of the kernel-level TCP proxy under varying numbers of concurrent requests



(b) Throughput of the kernel-level TCP proxy under varying attack rate. The number of concurrent requests is 50.

Fig. 7. The throughput of the kernel-level TCP proxy is much better than BIND 9 and degrades gracefully in the face of high attack request rate.

between 15% and 25%. In general, the DNS guard machine’s CPU utilization increases almost linearly with the increase in attack request rate. For a Pentium 4 2.4 GHz CPU, the DNS guard can sustain an attack rate of 200K requests/sec and supports a legitimate request rate of 100K requests/sec before becoming a bottleneck. Even when its CPU is fully saturated, it can still support a legitimate request rate of 80K requests/sec under an attack load of 250K requests/sec. Without the DNS guard, the same capability requires 18 ANS machines if BIND 9 is used.

It is long believed that the performance of TCP-based DNS is poor. As RFC 3225 stated, TCP connection management increases network traffic, latency and the load on DNS servers. To mitigate this performance problem, the DNS guard prototype implements a kernel-level TCP proxy to reduce the associated overhead to the minimum. In the test, the DNS guard instructs the LRS simulator to use TCP for each DNS request. The LRS simulator can be configured to maintain a particular number of concurrent requests during the test. It first starts up the specified number of TCP connections, and whenever an old connection is terminated, it initiates a new connection.

Figure 7(a) shows the throughput of the kernel-level TCP proxy under varying numbers of concurrent requests. In a LAN environment, with around 20 concurrent requests, the kernel-level TCP proxy can support around 22K requests/sec. However, when the number of concurrent requests is around 6000, the TCP proxy’s throughput decreases to around 11K requests/sec because of the management overhead associated with the large number of TCP connections.

Figure 7(b) shows the kernel-level TCP proxy's throughput when the DNS guard is under attack. In this test, there are always 50 concurrent TCP requests as legitimate requests, but the attack requests are UDP packets. The throughput of the TCP proxy decreases linearly with the increase in the attack request rate. When the attack request rate reaches 250K requests/sec, the TCP proxy's throughput decreases to 10K requests/sec. During the test, the CPU of the DNS guard is always fully utilized. UDP-based attack requests compete for the CPU with TCP-based legitimate requests. When attack requests consume more and more CPU, less CPU resource is left to legitimate requests.

V. CONCLUSION

DNS is one of the most critical components of the Internet infrastructure, but is vulnerable to spoofing based DoS attack. A DNS server cannot tell if a request packet comes from the IP address as indicated in the request. Spoofed attack requests result in DoS attack by overloading a DNS server or by saturating a victim's bandwidth via amplified DNS response. The key technology to protect DNS from DoS attacks is spoof detection. Once spoofed requests are identified, a DNS server can safely drop the spoofed requests without any collateral damage. Attack requests using real IP addresses can be rate-limited.

The key contribution of this paper is that it provides a comprehensive study on the use of cookies in DNS spoof detection. Each DNS requester needs to obtain a unique cookie from a guarded ANS and accompanies all subsequent requests to the ANS with the corresponding cookie. Spoofed requests cannot present correct cookie thus can be detected. This paper designed and implemented three schemes to embed cookies to DNS. The DNS-based scheme exploits mechanisms in the current DNS protocol by embedding cookies in NS name and NS IP address. The TCP-based scheme redirects LRS to use TCP and uses TCP sequence number as cookies. A kernel-level transparent TCP proxy is proposed to offload ANS from processing TCP connections. The modified DNS scheme extends the DNS protocol with cookies and achieves optimal performance. The first two schemes only need to add a DNS guard at ANS side while the third one needs add DNS guards at both the ANS side and the LRS side.

These schemes are implemented as a Linux kernel module called DNS guard. The measurements on the DNS guard prototype demonstrates that the DNS guard can indeed protect ANS from DoS attacks by maintaining 80K

requests/sec throughput in the presence of 250K requests/sec attacks. The TCP-based scheme can achieve 22K and 10K requests/sec throughput in normal operation and under attack, respectively. None of the three spoof detection schemes generates false positives, and there is no obvious way to evade the detection either. Finally the DNS guard can be deployed incrementally and transparently as traditional firewall.

REFERENCES

- [1] CERT, "Denial of Service Attacks using Nameservers," 2000. http://www.cert.org/incident_notes/IN-2000-04.html
- [2] ComputerWire. (2002) DDOS attack 'really, really tested' UltraDNS. http://www.theregister.co.uk/2002/12/14/ddos_attack_really_really_tested/
- [3] J. Hu, "'Zombie' PCs caused Web outage, Akamai says," 2004. http://news.com.com/Zombie+PCs+caused+Web+outage,+Akamai+says/2100-1038_3-5236403.html
- [4] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 3, pp. 38–47, 2001.
- [5] K. Rikitake, "A Study of DNS Transport Protocol for Improving the Reliability," Ph.D. dissertation, Graduate School of Information Science and Technology, Osaka University, 2005.
- [6] M. de Vivo, G. O. de Vivo, R. Koenke, and G. Isern, "Internet vulnerabilities related to TCP/IP and T/TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 1, pp. 81–85, 1999.
- [7] D. E. Eastlake, "Domain name system security extensions," RFC 2535, 1999.
- [8] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 331–342, 2004.
- [9] H. Yang, H. Luo, Y. Yang, S. Lu, and L. Zhang, "HOURS: Achieving DoS Resilience in an Open Service Hierarchy," in *Proc. IEEE DSN'04*, 2004.
- [10] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: An effective defense against spoofed DDoS traffic," in *Proc. of the 10th ACM Conference on Computer and Communications Security*, 2003.
- [11] S. Templeton and K. Levitt, "Detecting spoofed packets," in *Proc. of The Third DARPA Information Survivability Conference and Exposition (DISCEX III)2003*, 2003.
- [12] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites," in *Proc. of the 11th International World Wide Web Conference*, 2002.
- [13] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *Proc. of the 2000 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2000.
- [14] M. Collins and M. K. Reiter, "An Empirical Analysis of Target-Resident DoS Filters (Extended Abstract)," in *Proc. of the 2004 IEEE Symposium on Security and Privacy*, 2004.
- [15] A. Yaar, A. Perrig, and D. Song, "Pi: A path identification mechanism to defend against ddos attacks," in *Proc. of IEEE Symposium on Security and Privacy*, 2003.
- [16] D. J. Bernstein, "Syn cookies." <http://cr.yp.to/syncookies.html>
- [17] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing," RFC 2827, 2000.
- [18] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang, "SAVE: Source Address Validity Enforcement Protocol," in *Proc. of INFOCOM 2002*, 2002.
- [19] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against ddos attacks," in *Proc. of the Symposium on Network and Distributed Systems Security (NDSS 2002)*, 2002.
- [20] A. Yaar, A. Perrig, and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in *Proc. of IEEE Symposium on Security and Privacy, 2004*, 2004.
- [21] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," in *SIGCOMM '05*. New York, NY, USA: ACM Press, 2005, pp. 241–252.
- [22] N. Brownlee, kc Claffy, and E. Nemeth, "DNS Measurements at a Root Server," Cooperative Association for Internet Data Analysis - CAIDA, 2001.