Received 17 January 2023, accepted 17 February 2023, date of publication 27 February 2023, date of current version 2 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3250106

RESEARCH ARTICLE

Detecting Network Scanning Through Monitoring and Manipulation of DNS Traffic

JAFAR HAADI JAFARIAN, (Member, IEEE), MASOUMEH ABOLFATHI, (Member, IEEE), AND MAHSA RAHIMIAN
Department of Computer Science and Engineering, University of Colorado (CU) Denver, Denver, CO 80204, USA

Corresponding author: Jafar Haadi Jafarian (haadi.jafarian@ucdenver.edu)

ABSTRACT In this paper, we propose an approach for detecting internal and external network scanning attacks on enterprise networks. In our approach, an inline scan detection system (SDS) monitors the ingress and egress flows of an enterprise network subnet and detects scanning probes based on the correlation of flows with preceding DNS query/responses and reducing TTL values of DNS Resource Records (RR). Through rigorous evaluation, we show that our method is effective against both external and internal port scanners and network worms, its effectiveness is independent of scanning rate or technique, and its deployment incurs negligible overhead on DNS and network response times. While the idea of detecting scans by correlating network flows with preceding DNS query/responses has been proposed in the literature, this work extends the state-of-the-art by offering four contributions: 1) we show that without decreasing TTL values of RRs in DNS responses, attackers can piggyback on cached DNS records to bypass our detection; thus we incorporate a TTL reduction mechanism to enhance the effectiveness of this approach, especially against stealthy and adaptive scanners; 2) while prior works work against internal scanners, we use the relatively new extension of DNS protocol, ENDS0 Client Subnet (ECS) option, to expand this approach toward detecting external scanners; 3) we present a novel adaptive scanning technique, called *DNS-cache-based scanning*, that exploits local DNS cache to bypass prior detection methods, and shows that, while prior approaches fail to defeat this threat model, our approach is effective against this evolved threat model as well; and 4) contrary to existing work that focuses on defeating fast network scanning worms, this approach is effective against any scanning, including stealthy scanning that uses conservative timing profiles to evade detection.

INDEX TERMS Network scanning, intrusion detection, domain name system (DNS), network worm.

I. INTRODUCTION

Network scanning refers to the process of identifying and enumerating active machines and their services in an address space by sending probes (*e.g.*, ICMP Ping requests) to IP addresses in this space and analyzing their responses. Network scanning is one of the indispensable steps in the reconnaissance stage of cyber intrusions, and it is one of the most important means for the discovery of potential targets or infected peers by malwares. While network worms such as Conficker [1] or Code Red [2] have not been very prevalent in the last decade due to their high detectability [3], self-inhibiting propagation [4], and lack of financial incentives, in recent years we have witnessed a resurrection of scanning attacks by malwares. One of the most recent examples is a novel class of Cryptomining malware like *WannaMine*, which scans target networks to discover potential machines that are susceptible to *EternalBlue* SMB vulnerability [5]. Another important example is the IoT-targeting malware called *Mirai* [6] that uses network scanning to discover unsecured IoT devices. In addition to target discovery, network scanning has been used by some malwares (*e.g.*, Stuxnet [7]) for discovering already-infected peers in the internal network for purposes like sharing the latest versions of the malware binary.

However, the most prominent adversarial use of network scanning is in the reconnaissance stage of cyber intrusions [8] to gather information about a target network, such as which

The associate editor coordinating the review of this manuscript and approving it for publication was Pedro R. M. Inácio.

IP addresses are actively used as well as more detailed information about machines assigned to these addresses, a process that is referred to as *network mapping*. Scans may range from simple pings (ICMP requests and responses) to more nuanced scans that may reveal the machine's software/versions via server banners or other network artifacts. [9]. Network scanning is usually done using off-the-shelf tools such as `Nmap` [10], `Masscan` [11], or `ZMap` [12], but sophisticated adversaries may devise their own novel (zero-day) scanning methods as well. Meanwhile, intrusion detection mechanisms (behavior-based and content-based) have not been able to mitigate the threats of scanning completely. By keeping the scanning rate low (*e.g.*, one scan/min), strategic attackers can evade behavior-based scanning detection mechanisms [13], while by using new or stealthy scanning methods (*e.g.*, stateless scans used by `Zmap` [14]), they can evade content-based (*aka* signature-based) detection techniques.

In this paper, we present a simple yet highly effective proactive technique that slows down scanning-based threat models in network enterprises. Our approach is primarily effective against internal scanners, but can also be extended to external scanners with some modifications. This approach takes advantage of a fundamental principle in TCP/IP networks that any regular communication from machine A to an IP address I must be preceded by a DNS query by A for attaining the IP address I and the DNS record must be still valid (unexpired) during connection setup as specified by its TTL. Any connection that does not follow this policy is marked as a scan. While the central idea of this internal scan detection mechanism is straightforward and proposed in several prior works [15], [22], in this paper, we show that the robust and evasion-free deployment of this detection mechanism requires manipulation (reduction) of TTL values of DNS responses, which are typically long (24 hours on average), to values in order of a few minutes. This is because when machine A has an unexpired mapping in its DNS cache for the IP address I, a scanner on machine A can scan I without being detected. By expediting the expiration of cached DNS records, we prevent a scanner on A from piggybacking on existing cached DNS records on the local machine to successfully scan IP addresses in the local DNS cache.

We also propose a variation of our internal scan detection approach for detecting external scanners targeting our enterprise network, based on the relatively new EDNS0 client subnet extension (ECS) [16] of DNS protocol. Using this option, a recursive resolver acting on behalf of a client, which could be in a different network than the client, can include part of the client's (*i.e.*, the query originator) IP address in the DNS query, thus allowing DNS servers to observe and adapt DNS responses to the geolocation of clients. Using this relatively new, but widely implemented feature, we propose a novel method for detecting external scans sent to a network with high accuracy and precision.

For both internal and external detection, the monitoring and TTL manipulation are enforced by a scanning detection system (SDS) which is located at the subnet gateways and, in addition to TTL manipulation, is responsible for the detection and filtering of internal and external scans. Deployment of our approach through SDS devices does not require any modification in existing network protocols, network devices, and machines. Through rigorous evaluation, we show that our approach is feasible against both internal and external scanners and incurs low overhead. We investigate its effectiveness against state-of-the-art scanners and network worms in small-scale and large-scale networks; we show that the combination of correlating connections with DNS responses and reducing TTL values can throttle their hit rate up to four orders of magnitude. We also show that unlike conventional signature-based or behavior-based detection techniques [15], [17], [18], our approach can defeat unknown scanning techniques and stealthy scanners using conservative scan rates. In addition to evaluating our approach against existing state-of-the-art scanning techniques, we discuss how the next generation of scanners could potentially adapt to evade our techniques, using a novel scanning technique called *DNS-cache-based scanning*. Using a simulation model based on a dataset of network traffic traces collected at Los Alamos National Laboratory corporate internal network [19], we show that, while such DNS-cache-based scanners can defeat all previous approaches based on DNS traffic and flow correlation, it has extremely low effectiveness against our approach, even in their best-case scenarios. We investigate the overhead of our internal and external scan detection approaches, especially using smaller TTL values, on the workload of authoritative DNS servers. We also evaluate the effect of deploying our approach on network response time and show that it is negligible. Finally, we discuss some of the practical considerations for the deployment of our approach, along with its limitations and directions for future work.

While the idea of detecting scans by correlating network flows with preceding DNS query/responses have been proposed in the literature [15], [22], this work offers several transformative contributions:

1) We demonstrate that without decreasing the TTL values of the DNS responses, attackers can piggyback on cached DNS responses to bypass this detection method; thus, we incorporate TTL manipulation to enhance the effectiveness of this approach, especially against stealthy scanners.
2) While prior works only work against internal scanners, we use the new extension of DNS protocol, ENDS0 client subnet option (ECS), to expand this approach toward detecting external scanners with high precision and accuracy.
3) We present a novel adaptive scanning technique, called *DNS-cache-based scanning*, that uses local DNS cache to bypass this detection method. We show that while prior approaches [15], [22] fail to defeat this evolved threat model, our approach is effective against it.
4) We present a complementary approach that provides a countermeasure against scanners that rely on reverse-DNS queries to bypass our defense.

5) Contrary to existing works, which typically focus on defeating fast network scanning worms, our approach is effective against scanners/worms with any scanning rate, including stealthy rates that use conservative timing profiles to evade detection.

The rest of the paper is organized as follows. Section II discusses the related work. Section III discusses the internal and external scanning threat models. In Sections IV and V, we present the algorithms and architecture of the proposed approach for defeating internal and external scanners, respectively. Section VI presents our evaluation results on the efficacy and the overhead of the proposed approaches and discusses practical challenges and limitations in implementing our approach. Finally, Section VII concludes the paper and identifies future works.

II. RELATED WORK

In this section, we briefly review previously proposed scan detection mechanisms in three categories: signature-based techniques, behavior-based techniques, and DNS-based techniques.

Signature-based techniques defeat scanning by identifying scanning probes through their signature. For example, `Snort` IDS [20] can identify various `Nmap` scans like TCP scans or even evasive fragmentation-based scans using their signatures [21]. The main well-known shortcoming of signature-based detection techniques is their susceptibility to evasion using scanning zero-day or new techniques [13].

Behavior-based techniques defeat scanning by identifying scanners through their probing behavior. Although all behavior-based scan detectors are initially designed for worm detection, they can be leveraged to detect other types of scanners. Threshold Random Walk (TRW) detector [3] identifies a scanning machine based on the notion that a scanner will have a higher connection failure rate than a machine engaged in legitimate operations. Another notable approach, Minimum Rim Width (MRW) detector [17] works based on the observation that scanning results in connections to many destinations; thus, if the number of first-contact connections from a machine to new destinations within a given window exceeds a certain threshold, the machine is identified as a scanner. Rate-based Sequential Hypothesis Testing (RBS) detector [18] measures the rate of first-contact connections to new destinations. It works based on the hypothesis that a worm-infected machine contacts new destinations at a higher rate than a legitimate machine. RBS measures this rate by fitting the inter-arrival time of new destinations to an exponential distribution. The TRWRBS detector [18] combines the TRW and RBS detectors and observes both the connection failure rate and the first contact rate. It performs sequential hypothesis testing on the combined likelihood ratio to detect worms. The problem with behavior-based detectors is their susceptibility to evasion against low-and-slow scanners that use conservative timing profiles. For example, Stafford et al. [13] shows that by adopting highly conservative scanning rates (*e.g.*, one scan per minute), a scanner can bypass all the behavior-based detectors. Moreover, these detectors are able to detect a scanner only after observing a relatively high number of scanning probes.

DNS-based techniques detect scanning by correlating connection requests with preceding DNS queries, similar to our approach. Whyte et al. [15] propose correlating connection requests with preceding DNS queries to detect network worms. Specifically, if no DNS activity is observed before a new connection is initiated, the connection is considered anomalous. In a similar work, Ahmad et al. [22] propose NEDAC (NEtwork and DAta link layer Countermeasure), which counters network worms by observing DNS activities and detecting the absence of DNS lookup in newly initiated outgoing connections. In this approach, when a datagram is transmitted to a destination address without prior DNS lookup, the source IP address is maintained in a cache, and its corresponding counter is incremented. A threshold value is set in order to assign a maximum number of distinct IP addresses a machine can attempt to contact without a prior DNS lookup per time duration. Upon reaching the threshold value within this duration, the detection system will mark the behavior as worm propagation and invoke the containment mechanism. These two approaches, which are closest to our work, only detect internal scanning (local machines scanning other local machines or remote machines) but do not work against external scanners (remote scanners scanning local machines). Moreover, TTL values of DNS responses are not reduced by either of the approaches and thus they both suffer from high false negative rates against scanners. Reducing TTL values makes the DNS-based detection technique robust and evasion-resistant. This is because the average DNS response TTL values are typically long (24 hours). With such a long TTL value, at any point, a machine would have a large DNS cache that includes DNS mappings for many internal and external machines. If a machine with a large DNS cache is compromised, and the attacker scans the network from this machine, scans to IP addresses in the DNS cache would not be detected, resulting in a large number of false negatives. More importantly, these approaches fail entirely against our evolved DNS-cache-based scanning technique, where a scanner only scans IP addresses already in the local DNS cache.

In a related work but pursuing a different objective, Shue et al. [24] combine the idea of IP randomization (fluxing server's IP address over time) and NATing so that clients are required to periodically query the DNS for a server's address, thus turning the DNS lookup into a decision point whereby the server's network can decide to allow or deny service to some client while preventing arbitrary clients from probing to learn about a server (*e.g.*, using port scanners). This work pursues a different objective as it uses the DNS lookup as a mechanism for enforcing capabilities (which clients are allowed to communicate with a server), rather than using DNS lookup to detect malicious scans.

In summary, while the notion of correlating DNS lookup with newly initiated connections to detect scans has been proposed in the literature [15], [22], existing works suffer from several shortcomings: 1) none of these works include TTL reduction as an indispensable mechanism to reduce false negatives against classic network scanning worms, 2) none of the works have considered how existing scanning techniques could be evolved to bypass their detection, and they only work against existing scanning techniques, 3) none of these works are capable of detecting external scanners that target enterprise networks and are only capable of detecting internally initiated scans, and 4) both of these works focus on fast scanning network worms, and the efficacy of these methods against conservative low-and-slow scanners have not been investigated. Our proposed approach addresses these shortcomings.

III. THREAT MODEL

Network scanning is among the most important tools in attackers' arsenal for discovering machines in an address space. Network worms such as Conficker or Mirai have used scanning to select the next potential targets for infection, while others like Stuxnet [7] have used scanning to discover other infected peers. However, the most insidious and prevalent use of IP/port scanning is in the reconnaissance stage of cyber intrusions [9]. In this initial stage, adversaries usually rely on off-the-shelf port scanning tools such as `Nmap` [10] or `Masscan` [11] to discover active machines in a target address space and also identify their open ports and running services.

Any scanning by network worms or port scanners, regardless of its rate or technique, requires sending probing datagrams to IP addresses in a target address space. Depending on the location of the scanner, this scanning could be internal or external.

Internal scanning refers to cases where the scanning agents are located within the enterprise network and scan either the internal address space (local-to-local scan) or public address space (local-to-remote scan).

External scanning refers to cases where the scanners are outside the perimeter of the enterprise network and scan the public address space of this network (remote-to-local scan) to discover its publicly accessible machines, which are usually located in the demilitarized zone (DMZ) of the enterprise networks.

Defeating scanners is a challenging problem because attackers use various evasion techniques to bypass the detection algorithms. For example, by decreasing the scanning rate to below a certain threshold (*e.g.*, one scan/min), attackers are able to bypass all behavior-based scanning detection algorithms [13].

We will first show how our technique can detect internal scanners and then discuss how it could be extended - with certain limitations - to detect external scanners.

IV. DETECTING INTERNAL SCANNERS

Our detection algorithm takes advantage of the fact that while benign users usually first attain an IP address using a DNS lookup before initiating a connection to it, scanners and network worms randomly select IP addresses from the targeted address space, and thus their probe or exploit datagrams sent to a destination IP address are not preceded by a DNS lookup.

Considering this differentiating factor, to detect internally initiated scans, we propose an algorithm based on the principle that any legitimate connection attempt by a machine A to a destination IP address I (either internal/local or public/remote) must be typically preceded by a DNS query by A to attain I. Meanwhile, IP addresses that must be rightfully excluded from this policy are identified and added to an allowlist prior to the deployment. Thus, if we observe a connection from machine A to a non-white-listed IP address I with no record of A querying the authoritative DNS to attain I, we mark this connection attempt as a scan. Our discussion here focuses on IPv4-based addressing and scans; in Section VI-D, we discuss how this approach could be extended to IPv6.

Since, in an enterprise network, the authoritative name servers and network machines are under the same authority, this policy can be enforced straightforwardly against internal scanners. To apply this policy, we must keep track of DNS queries and responses issued by every internal machine. However, if the interval between issuing a DNS query for attaining I and establishing the connection to I is too long, keeping track of these queries would incur non-trivial storage and lookup overhead. More insidiously, it allows attackers to take advantage of existing cached DNS records on the residing machine to bypass our detection, as discussed in Section IV.

In DNS responses, TTL (time-to-live) determines the duration in seconds for which the given `DNS RR` (resource record) is valid. The DNS RR should be considered *expired* after this interval. To shorten the duration between (1) the time when a DNS response provides an IP address I to an internal machine A is received and (2) the time when the corresponding communication attempt is made by A to I, we intercept DNS responses destined to machine A and modify the TTL of all DNS RRs within the response to relatively small values, *e.g.*, $60 - 300$ seconds, before forwarding the updated DNS response to A. This ensures that A only caches the most recent DNS records, and cached IP addresses in the local DNS of A are expired after a short interval.

While shorter TTL values are more desirable, they would result in receiving a higher number of DNS queries by the authoritative DNS server(s) of the network. The trade-off between the overhead and the effectiveness for different TTL values is investigated in Section VI.

Figure 1 shows the overall architecture for the deployment of our mechanism in a network. The scan detection gateways (SDS) are located at the boundaries of virtual or physical

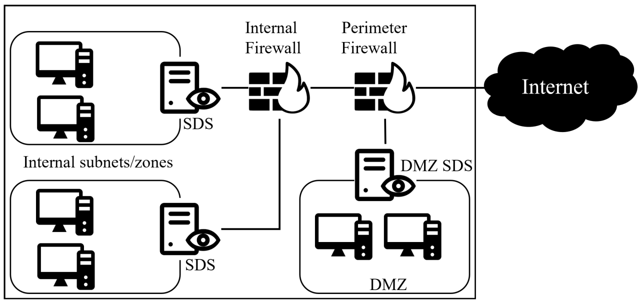

FIGURE 1. Location of SDS devices in enterprise network architecture.

LANs. SDS performs three major tasks: (1) they keep track of ingress DNS responses (and DNS RRs) received by each machine in their LAN; (2) they change the TTL values of the DNS responses to a given value (denoted by τ) for every ingress DNS response, and (3) they validate ingress connection attempts (*e.g.*, TCP SYN segment, first UDP datagram, first ICMP packet) according to the above policy, and mark connections with no or expired preceding DNS lookup as scans. Upon detecting a scan datagram, SDS could either drop it or just log it for further inspection, depending on the security policy. Note that each SDS is only responsible for its subnet (*e.g.*, LAN or VLAN). Therefore, the mechanism does not rely on any central entity, which makes the approach scalable.

Algorithm 1 shows the pseudo-code run by each SDS for the subnet under its control. When SDS observes a DNS response destined for a machine with IP_S, it modifies the TTL values of all type `A` resource records (in IPv4) in the DNS response and also records them in a hash table. Assume a DNS response to an IP_A includes a type `A` RR which provides the IP address IP_D. SDS changes the TTL value of the DNS RR to τ, which is a short TTL value given as input. It also records the new expiration time of each DNS RR in a hash table where the key is $IP_S||IP_D$ and the value is the current clock, denoted as *clock*, plus τ which is the new TTL. After performing this change for all RRs in the DNS response, SDS recalculates the UDP header checksum and then forwards the DNS response.

When SDS observes a communication attempt from IP_S to an IP address IP_D at a time t, it checks to see if the key $IP_S||IP_D$ has a matching and non-expired entry in the hash table. If so, it allows the connection; otherwise, it either drops the connection or raises an alert, depending on the predefined security policy. Also, the security operator also may decide to denylist or quarantine the scanning machine for a while until a proper investigation and infection removal are conducted. Since the SDS is within the same LAN/VLAN as the scanning machine, this filtering could be performed using the MAC address of the scanning machine. Since each SDS is only responsible for its own subnet, the clock could be local, and there is no need for global clock synchronization among SDS devices. This further simplifies the deployment and increases the scalability of the approach.

Note that using Alg. 1, the border SDS in Fig. 1 can also detect the egress internal scans that target external machines on the Internet (*i.e.*, local-to-remote scans). This is because the perimeter SDS in Figure 1 can monitor and manipulate ingress DNS responses from external DNS servers and also correlate ingress DNS responses with egress connections destined to external IP addresses to detect scans. While this egress filtering is more beneficial for others than the enterprise network itself, it serves the enterprise by making it a *good neighbor* and protecting its reputation.

Algorithm 1 SDS Algorithm for Detecting Internally-Initiated Scans

```
%marking origins of DNS queries
for every ingress DNS response destined to a machine with address $IP_S$ do
    for every type-A RR $\in$ DNS response providing address $IP_D$ do
        $ht[IP_S||IP_D] \leftarrow (clock + \tau)$
        update TTL of DNS RR to $\tau$
    update UDP checksum of DNS response
    forward response
%validating connections
for every new egress connection from $IP_S$ to $IP_D$ at time $t$ do
    if exists($ht[IP_S||IP_D]$) and $ht[IP_S||IP_D] \geq t$ then
        allow connection
    else
        drop/log connection
        block $IP_S$ (if policy allows)
```

An informed attacker may try to bypass our technique using reverse-DNS queries. For example, Nmap performs reverse-DNS resolution for every IP which responds to host discovery probes (*i.e.*, those that are online). Reverse DNS is represented by `DNS PTR` records in the zone files of authoritative DNS servers and stored in a special zone called *.in-addr.arpa* [25]. For example, the zone for the PTR record of private address range $10.*.*.*/8$ would be *10.in-addr.arpa*. An attacker would use the reverse-DNS in the following manner: first, they issue a reverse-DNS query for a given IP address. If they receive a response that maps that IP address to a domain name, they know that the IP address is online. Then, they would issue a DNS query for that domain name in order to precede the following scan with a DNS lookup and bypass our detection. Finally and after receiving the DNS response, they would scan the IP address without being detected.

To prevent this evasion scenario, we filter reverse-DNS queries, except for allowlisted users and domains. Reverse-DNS allowlisting provides benefits of the reverse-DNS while preventing its misuse by potential attackers. One of the main uses of reverse-DNS queries on the Internet is to verify that the sending mail server is not a malicious spammer. This is done by doing reverse-DNS lookups for the IP of the sending server to ensure that there is a reverse-DNS record associated with it in the zone files of the claimed authoritative DNS server. If not, the receiving mail server may consider our emails spam. Disabling reverse-DNS queries makes Internet mail servers consider our mail server as a spammer. Thus, rather than responding to all reverse-DNS queries, we only respond to the queries sent by allowlisted addresses and only those sent for machines with potentially legitimate purposes like mail servers. This practice enables us to take advantage of the benefits of reverse-DNS service for administrative purposes and spam detection while avoiding its abuse for malicious purposes.

V. DETECTING EXTERNAL SCANNERS

External scans are initiated from external machines and sent to public IP addresses of our enterprise network, in order to discover the publicly accessible machines within our network. While the perimeter firewalls and intrusion detection systems (IDS) can detect and filter many different types of scans from their signature or behavior of the scanning machine (*e.g.*, high rate of connections to new destinations), the scanners can still evade detection using new types of scans or stealthy low scan rates.

The main difference between using our approach for the detection of internal and external scans is that contrary to internal scans where the IP address of the DNS query originator is known to the authoritative DNS server, on the Internet, most queries come from recursive resolvers, and the source address of a DNS query is that of the recursive resolver rather than of the query originator [26]. While in legacy networks, recursive resolvers were usually located within the user's own autonomous system (AS), in modern networks, clients increasingly query a "cloud DNS" host or open resolver, such as the *Google Public DNS*, situated at a different AS [27]. The rise of open recursive DNS servers, which are typically situated in separated ASes from the users entails that recursive resolvers are not necessarily geographically close to the querying clients. This limitation does not allow us to adopt the internal scan detection approach against external scans because the recursive resolver querying on behalf of a client may be situated in a completely different AS than the client, and thus the query's source IP reveals no information about the IP address of the querying client.

To work around this limitation, we modify our approach by relying on a relatively new extension for DNS that allows DNS queries to carry information about the machine/network that originated that query to authoritative DNS servers. This extension, called EDNS0 Client Subnet (ECS) option, allows a recursive resolver to include a prefix of the IP address of the source query originator in the DNS query. Initially proposed as an experimental draft in 2011 and officially adopted in 2016 [27], ECS is specified in RFC 7871 [26] and enables a recursive DNS resolver, such as Google Public DNS, to specify the subnetwork for the host or client on whose behalf it is making a DNS query. This is generally intended to help speed up the delivery of data from content delivery networks (CDN) by allowing more efficient DNS-based load balancing to select a service address near the client machine.

ECS includes a truncated portion of the client's IP address, referred to as the client subnet, in all subsequent requests made by the recursive to an authority supporting ECS. ECS does not change the DNS resolution process but augments the information exchanged between recursive resolvers and authoritative DNS servers. When the recursive resolver and the authoritative DNS servers support ECS, the DNS queries and responses are extended with several new extra data fields, most importantly including *source prefix-length*, and *scope prefix-length*. *Source prefix-length* is an unsigned octet representing the leftmost number of significant bits of the client IP address to be used for the lookup (/24 is the default [27] and currently the most typical), and it is specified by the recursive resolver in the DNS query. In response, the source prefix-length mirrors the same value as in the corresponding queries. Accordingly, the DNS response includes a *scope prefix-length*, which denotes the leftmost number of significant bits of the given source address that the response covers. The authoritative DNS typically sets the scope prefix-length of a DNS response to the same value as the source prefix-length in the corresponding DNS query [26]. In queries, scope prefix-length must be set to 0 [26].

ECS protocol is widely adopted and constitutes a significant percentage of the DNS traffic. ECS is currently "on by default" for all traffic through many of the largest open DNS servers [27]. This extension has currently been adopted by many of the largest open DNS providers on the Internet, including Google Public DNS, OpenDNS, Quad9, and NextDNS [27]. A recent study in 2021 investigated 11.5 billion DNS queries on the Internet and observed that 69% were ECS enabled.

By enabling the ECS option, a recursive resolver could provide a prefix of the client IP address to the authoritative DNS server. A source prefix-length value of 32 means that the recursive resolver must add the complete client IP address information to its queries, while a value of 0 means that the resolver must not add any prefix of client address to its queries. The prefix length is recommended to be less than the full IP address for privacy purposes [26]. According to a recent longitudinal study exploring the deployment of ECS extension in 2021 [27], 54% of the announced prefixes are /24 and only 2.5% are less or equal to /16. This is consistent with the recommendation of RFC 7871, which determines that /24 can be leaked without compromising the client's privacy [26]. The study also reports that since 2015 the utilization of prefixes larger than /24 has increased considerably.

Figure 2 shows the architecture and communication schemes for a benign client (black lines) vs. a scanning attacker (orange lines). Suppose a client intends to communicate with our enterprise Web server on the domain

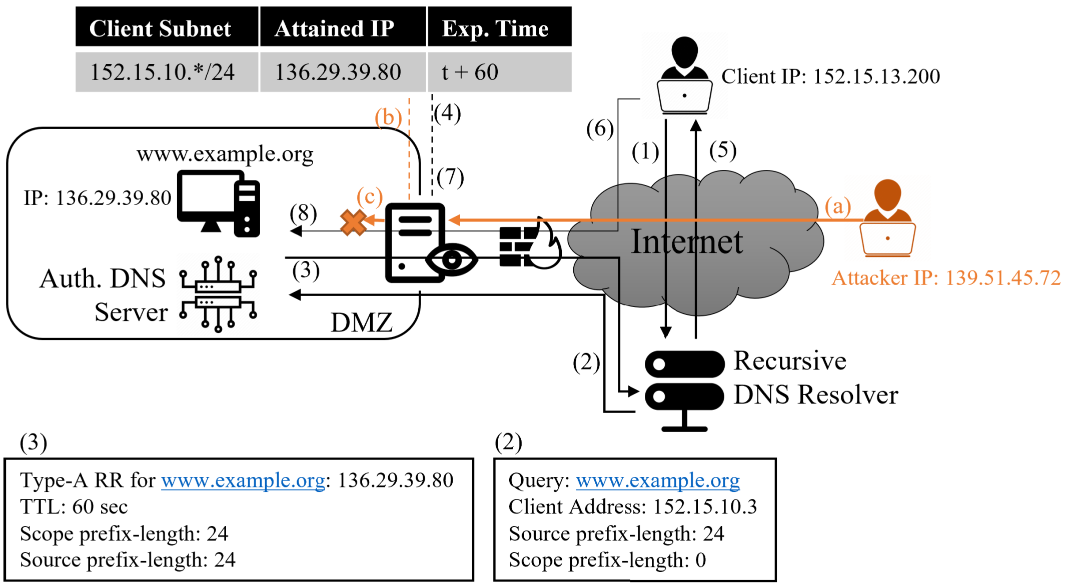

FIGURE 2. Communication protocol for external clients/scanners.

"www.example.org." 1) The client's machine first issues a DNS query to a recursive DNS resolver. In the usual case, where no ECS option was present in the client query, the recursive resolver initializes the option and sets the *source prefix-length* to a predetermined default value (24 is recommended by RFC). If the client query included an ECS option itself, the resolver sets *source prefix-length* to the shorter of the incoming query's source prefix-length and the server's default value. The resolver also includes the client's IP address but is truncated to the number of bits indicated by the *source prefix-length* field and padded with 0 bits to pad to the end of the last octet needed. 2) The resolver forwards the query with ECS data to the authoritative DNS server of the web server, which is located in our enterprise network. 3) The authoritative DNS server resolves the query and returns the IP address of the webserver to the resolver. The DNS server, by default, assigns small TTL values to DNS RRs, because, to achieve high effectiveness, we need to adopt the same strategy of announcing DNS resolutions with small TTL values. TTL values of 20 seconds or less are currently being widely used by content delivery networks (CDNs) [28] for load balancing. 4) The SDS device located at the perimeter of the DMZ intercepts the egress DNS response, and for every type-A RR, it records the client's subnet address, the given IP address, and the expiration time (based on the given TTL) in a *Connection Table*. It then forwards the response to the resolver. Algorithm 2 denotes the pseudo-code executed by the SDS devices to mark the origins of external DNS queries. 5) The resolver forwards the DNS response back to the querying client. 6) The client initiates a connection with the webserver's IP address. 7) The SDS device intercepts the incoming datagram (TCP SYN segment) and checks the Connection Table to see whether the client's IP address belongs to a subnet that has previously attained the IP address within the last TTL seconds. Algorithm 2 denotes the pseudo-code executed by the SDS devices to validate incoming connections. 8) Since the client's subnet has been included in the DNS query by the resolver, the client would be allowed to communicate with the web server with no alarm raised.

In contrast, a scanning attacker would initiate a connection or send a probe to the webserver's IP address before first querying the authoritative DNS server, as shown in Figure 2. Thus, unless a benign client from the same subnet as the attacker has queried the authoritative DNS server very recently (in the last TTL seconds), the attacker's connection request will be detected as a potential scan and logged for further inspection.

The main drawback of the proposed approach is that while ECS adoption has raised considerably and will continue to grow, and also utilization of prefixes equal to or larger than /24 has increased significantly [27], ECS deployment and use are not mandatory for servers, clients, and resolvers. A client, DNS resolver, or authoritative DNS server may not support ECS; alternatively, both the client and the DNS resolver acting on behalf of the client can optionally select a source prefix-length of 0 to fully anonymize the IP address of the client [26].

Since ECS deployment is not mandatory, we can not reject a non-compliant DNS query (*e.g.*, one with no ECS enabled or announcing a 0 source prefix-length) or drop a non-conforming connection. Without this mandatory enforcement, malicious scanners would prefer to either not use ECS or announce a *source prefix-length* of 0 to anonymize their identities and avoid detection. However, while opting out of using ECS with large prefixes is not necessarily

Algorithm 2 SDS Algorithm for Detecting Externally-Initiated Scans

```
%marking origins of DNS queries
for every egress DNS response to an external resolver do
    for every type-A RR ∈ DNS response, providing
    address $IP_D$ for a client with subnet
    $CS = \langle x.y.z.d/t \rangle$ do
        $\boldsymbol{T}[CS, IP_D] \leftarrow (clock + \tau)$
        update TTL of DNS RR to $\tau$
    update UDP checksum of DNS response
    forward DNS response
%validating connections
for every new ingress connection from an external
machine with address $IP_S$ to address $IP_D$ at time $t$ do
    if there exists an entry $[CS, IP_D]$ in $T$ where
    $IP_S \in CS$ and $T[CS, IP_D] \geq t$ then
        allow connection
    else
        log connection
```

malicious, opting in is definitely a sign of non-maliciousness. In Section VI-D, we discuss this limitation in further detail.

VI. EVALUATION

In this section, we evaluate the effectiveness of internal scan detection and external scan detection algorithms. We also present our results on evaluating the overhead of the approach, followed by a discussion on the limitations of the approach. Table 1 provides a summary of our primary evaluation methods, testbeds, and

A. DETECTING INTERNAL SCANNERS

To evaluate our approach, we first deploy our method on a consolidated testbed consisting of a few virtual machines to ensure the feasibility of our approach and evaluate its technical intricacies; we then deploy it on a large simulated network to evaluate its efficacy in large-scale scenarios.

1) SMALL-SCALED EVALUATION ON A REAL TESTBED

To prove the feasibility of our approach, we built a small network and deployed a proof-of-concept implementation of our approach on it. We used `VirtualBox 6.1` to host 3 virtual machines (VM) on a capable server. The three machines are part of a subnet with the network address 192.168.0.0./16. One VM represents a benign client machine. This machine runs a `python` script that uses Chrome Selenium to automatically browse the top 10, 000 websites in Alexa top domains [29] (discontinued starting May 2022). The second VM represents a scanner; on this machine, we executed `Nmap 7.91` to scan the internal network range.

The third VM acts as SDS (Scan Detection System) module. This machine has two interfaces, one connecting it to the virtual internal network and one bridge connecting it to the physical network and the Internet. This VM acts as the default gateway for two other VMS, and all traffic sent and received by them passes through this VM. A *python* script on this VM implements Algorithm 1. This script monitors and records DNS responses and updates their TTL values. It also monitors incoming connections from VMs and detects those not preceded by a non-expired DNS query/response as scans. The new small TTL value for DNS responses, τ, was set to 300 seconds.

After observing $N = 1,018,171$ packets from both benign and scanning VMs, the following results were achieved:

- Total (N) = 1,018,171
- True Negative (TN) = 116,765
- True Positive (TP) = 864,833
- False Negative (FN) = 2,152
- False Positive (FP) = 34,421 (18,096 correspond to existing but expired cached records)

Thus, the accuracy, precision, and recall attained in this experiment are as follows:

- Accuracy = (TP + TN)/N = 0.964
- Precision = TP/(TP + FP) = 0.961
- Recall = TP/(TP + FN) = 0.997

In this evaluation, we only experimented with one TTL value (300 seconds) and mainly focused on a feasibility study. The impact of TTL reduction is more tangible in large-scale networks, which will be discussed in the next section.

The approach suffers from false negatives; these false negatives occur when an IP address that exists in the local DNS cache is scanned by the scanner. These false negatives can not be eliminated completely, but by reducing the TTL value, we can reduce their likelihood, as demonstrated in the results of the large-scale evaluation.

More importantly, the approach has a relatively high number of false positives. These false positives occur when the machine attempts to communicate with a benign IP address that does not exist in the DNS cache. 52% of these false positives correspond to communicating with IP addresses with existing - but expired - DNS records. This could be due to - among other reasons - TTL pinning by the browsers (see Section VI-D2). 41% of the false positives are for communicating with IP addresses belonging to CDN proxies and cloud machines by reputable provides such as Amazon, Microsoft, Akamai, *etc.* Our investigation shows that these addresses are attained via a different means than DNS. However, these reputable network addresses could be identified and added to an allowlist over time, thus as discussed in Section VI-D1. The remaining 7% of false positives belonged to communications with private static addresses like the gateway IP address, and reserved or multicast addresses (224.0.0.0/24). These IP addresses could be straightforwardly identified and allowlisted.

2) LARGE-SCALE EVALUATION ON A SIMULATED TESTBED

To evaluate the approach for large-scale scenarios, we developed a simulated class *B* (/16) enterprise network based on the Los Alamos National Laboratory corporate internal

TABLE 1. Summary of evaluation methods.

Scan Type	Testbed	Configuration	Objective and Summary of Results
Internal	Small-scale realistic testbed consisting of 3 VMs	Setup::VM1:Scanner,VM2:Client,VM3:SDS TTL=300s	Evaluate against internal classic port scanners (Results: accu=0.96, prec=0.96, recall=0.99)
	Large-scale simulated testbed	Based on Los Alamos internal net. traces TTL = {60sec, 300sec, 600sec, 24hr}	1) Evaluate against classic port scanners (Fig. 4-5) 2) Evaluate against adaptive port scanners (Fig. 6) 3) Evaluate against classic network worms (Fig. 7) 4) Evaluate against adaptive network worms (Fig. 8)
External	Small-scale emulated testbed consisting of 4 VMs	Setup:: VM1:Client,VM2:Auth. DNS, VM3:Scanner,VM4:SDS TTL={60s, 300s, 600s, 24hr}	Evaluate against external port scanners (Fig. 12) (Results for TTL = 60s: accu= 0.99, prec=1, recall=0.99)

computer network flow datasets [19], which collectively represent the communication log in an internal network among 8495 machines over the course of 57 days. We used two datasets from this repository: 1) a *flow dataset* including network flow events collected from central routers within the network. Each event in this dataset presents a network flow event from a source machine to a destination machine at the given time and the given duration in seconds; and 2) a DNS dataset presenting DNS lookup events collected from the central DNS servers within the network. Each event in this dataset presents a DNS lookup at the given time by the source computer for the resolved computer.

To evaluate the effect of benign communications on the success rate of scanners, connections among these machines were simulated according to these datasets. For evaluations regarding port scanners, the port scanner was assumed to be residing on a machine that is identified in the dataset as `C1065`. This machine was chosen because it has the highest communication rate with other machines in the dataset. Therefore, our analysis here shows the best-case scenario for attackers. Similarly, for worms, this machine was assumed to be the initial infection point in order to evaluate our approach against the worst-case worm propagation scenario.

We investigate the effectiveness of the model against two classes of threat models: port-scanners and network worms. The main distinction is that port scanners solely probe machines for information gathering, while network worms infect and propagate. In worm propagation, each infected machine would turn into yet another scanner, and the number of scanners increases exponentially over time.

Effectiveness against Internal Port Scanners: Any scanning, regardless of its type or rate, will result in sending probes to randomly chosen addresses in a target address space. With SDS proxies deployed, such probes would be easily detected because scans to IP addresses will not be preceded by DNS queries. The only exceptions are the addresses in the local DNS cache of the scanner's machine, which have been obtained through some legitimate means, like browsing activities of a user on that machine. The number of these cached DNS entries depends on the TTL values of the DNS RRs; the smaller the TTL value, the lower the number of cached DNS records.

We used this simulated network to evaluate the effectiveness of the model. We used the following assumptions in our evaluation:

- Internal machines communicate according to the *Flow* and *DNS* logs in the Los Alamos National Laboratory corporate internal computer network flow datasets [19].
- A scanner on one of the network machines - with the highest communication rate among all machines - scans the network.
- We conducted our experiments on two types of network, an *Unprotected* network not using our defense model and a *Protected* network using our defense model. We conducted these experiments for different TTL values and scanners with different rates (aggressive vs. stealthy). We also conducted experiments against an evolved scanning technique.
- In our experiments, we assumed that in a protected network, once a scanning packet is detected, it is dropped, but the scanner is not denylisted.

Figure 3 compares the success rate of scanning in an unprotected network with its success rate in the same network protected by our technique. The success rate denotes the ratio of internal machines successfully probed by the attacker without being detected. In the figure, note that in an unprotected network and when the scanner is scanning with an aggressive rate (300 addr/sec), the attacker will be able to probe all machines in less than 20 minutes. In comparison, note that in the protected network, even with a 5-min TTL expiration interval, after 2, 000 minutes, the same aggressive scanning only succeeds in probing at most 6% of the machines. In contrast, the same aggressive scanner in an unprotected network takes only 20 seconds to scan 6% of network machines.

Aggressive scanning is very noisy and detectable by the detection techniques like TRW [3] or RBS [18]. With aggressive scans being detected and filtered out, attackers have to resort to stealthy scanning rates like `T0` or `T1` timing profiles in `Nmap`, which take an extraordinarily long time to scan thousands of machines or ports. With a conservative scanning rate (4 addr/sec), in the unprotected network, it only takes 200 seconds to probe 50% of the network machines,

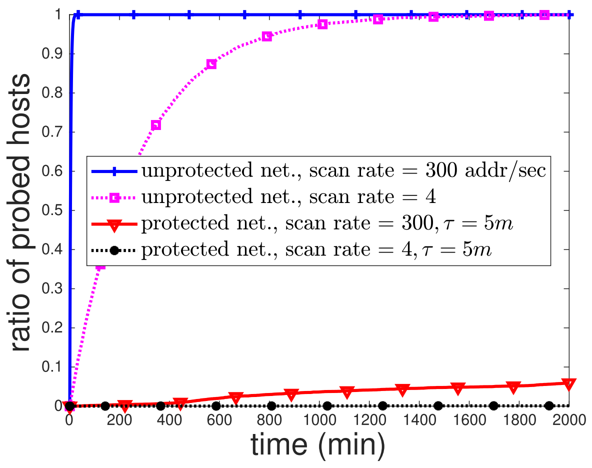

FIGURE 3. Comparing success ratio of scanners in protected vs. unprotected networks.

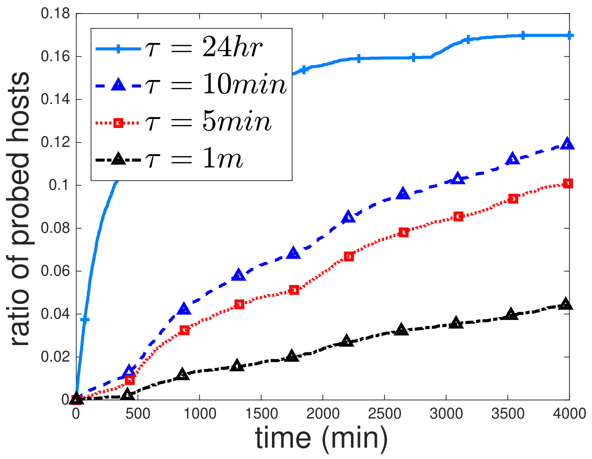

FIGURE 4. Success ratio of scanners in a protected network with various TTL values (scan rate = 300 addr/sec).

while in the protected network, the scan success rate does not surpass 0.01% even after 2,000 minutes.

These results are obtained by assuming that the scanning machine is not denylisted after detection. However, once we detect a few scans from a source, we can denylist it. When this is enforced as a security policy, then the scanning success rate will be close to 0.

Figure 4 compares the success rates of an aggressive scanner in a protected network with various enforced TTL values. Firstly, note that our approach is still effective even when no TTL value manipulation is enforced ($\tau = 24$ hours). Also, note that as the TTL value gets smaller, the effectiveness increases. For example, after 500 minutes and with $\tau = 24$ hours, the scanner succeeds in probing 11% of network machines. In contrast, with $\tau = 1$ minute, this ratio remains below 0.4%. This clearly emphasizes the significance of manipulating TTL values in achieving reasonable effectiveness against scanners.

But what if the attacker knows about our technique? Instead of using regular scanning, the attacker would passively monitor the local DNS cache on the residing machine to identify machines for which an unexpired DNS query has been issued and will only scan those. We refer to this technique as *DNS-cache-based adaptive scanning*. We consider this technique in order to evaluate our mechanism not only against existing scanning mechanisms but also against potential evasion techniques.

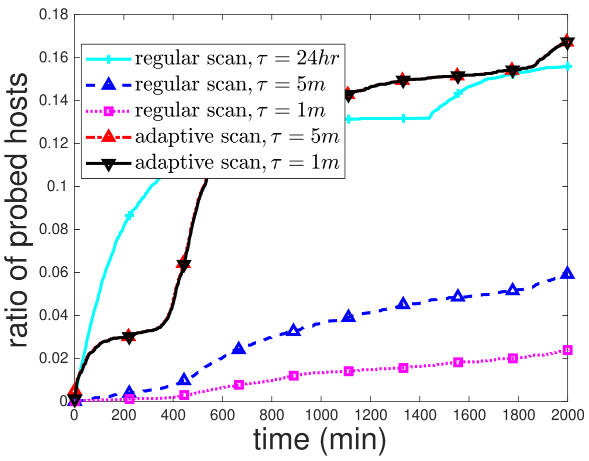

FIGURE 5. Success ratios of adaptive DNS-cache-based scan vs. regular scan in a protected network.

Algorithm 3 Adaptive DNS-Cache-Based Scanning Algorithm

```
while true do
    L = exec ipconfig /displaydns
    for every type-A record with IP $i \in L$ do
        scan $i$
```

Algorithm 3 shows the pseudo-code of an adaptive DNS-cache-based scanner. The scanner regularly monitors network connections using a command like `ipconfig /displaydns` on Microsoft Windows to discover unexpired (cached) DNS records, and will only scan those. Therefore, scanning will be only limited to addresses for which a valid mapping exists in the local DNS cache.

In the adaptive DNS-cache-based scanning of a protected network, the success rate depends on the rate at which a machine establishes benign connections with other machines in the network, and it is independent of the TTL values of DNS RRs. Figure 5 shows the success rate of adaptive scanners on the machine `C1065` in the Los Alamos network flow dataset and compares it with those of regular scanners in a protected network with all other parameters kept the same. First, note that in a protected network adaptive DNS-cache-based scanners outperform regular scanners that use random scanning, even when the scan rate is aggressive (300 addr/sec), and the TTL value is not manipulated ($\tau = 24$ hours). Secondly, note that although outperforming regular scanners, a DNS-cache-based scanner is still ineffective, and even after 2000 minutes, it has only probed less than 18% of machines. In comparison, a regular aggressive scanner would take less than 20 minutes to probe all machines in an unprotected network. Thirdly, note that for regular scanners, the TTL value has a noticeable effect on the effectiveness, while for DNS-cache-based scanners, the success rate is independent of the TTL value. This is because a DNS-cache-based scanner probe all the IP addresses that appear in the local DNS cache, and this is independent of the expiration rate of cached DNS entries.

Effectiveness against Network Worms: A worm would propagate infection across the internal network by automated exploitation of vulnerabilities. Network worms rely on scanning to discover vulnerable machines. Once a new machine is infected, it turns into another scanning machine. Thus, the number of scanners increases exponentially over time.

To evaluate the effectiveness of our approach against network worms that propagate internally, we rely on the same simulated testbed, including the *Unprotected* and *Protected* networks, but instead of a scanner, we assume a network worm initiates its propagation from one of the internal machines with the highest connection rate. This is the best-case scenario for worm propagation because the high communication rate means a highly-populated local DNS cache, which enhances the likelihood of evasion. We assume the worst-case scenario where all network machines are vulnerable to infection. Once a machine is successfully contacted without being detected, it becomes infected and starts scanning the address space. We conduct our experiments on these two networks using different TTL values and against an aggressive network worm with a scanning rate of 300 addr/sec, a stealthy network worm with a scanning rate of 4 addr/sec, and an evolved network worm that uses an evolved scanning technique based on local DNS cache to evade detection.

Figure 6 compares the propagation of a worm with various scanning rates in an unprotected vs. a protected network. Note that in an unprotected network, propagation is relatively instantaneous (less than 10 seconds). Even for a stealthy scanning rate of 4 addr/sec, the propagation takes less than 2 minutes. In comparison, a scanning worm in a protected network would have a much slower propagation and success rate. An aggressive worm would only infect 73% of network machines after 2000 minutes even with an aggressive scanning rate of 300 addr/sec and enforced TTL value of 5 minutes. In comparison, the same worm would infect 73% of machines in an unprotected network in merely 6 seconds. This shows a slowdown of $(2000 * 60)/6 = 20000$ times in terms of worm propagation. For a stealthy scanning rate of 4 addr/sec and $\tau = 5$ min, the percentage of infected machines in 2000 minutes falls to 60%. The same worm would infect 60% of machines in an unprotected network in 253 seconds. This shows a propagation slowdown of $(2000 * 60)/253 = 474$ times for infecting the same number of machines.

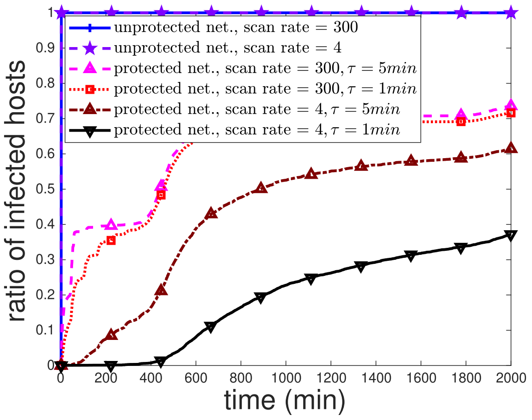

FIGURE 6. Worm propagation in an unprotected vs. protected network.

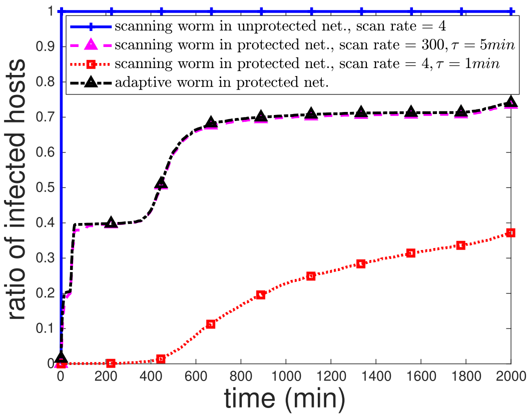

FIGURE 7. Comparing propagation of adaptive vs. random scanning worms in an unprotected vs. a protected network.

Also, in Figure 6 note the effect of enforced TTL values (τ) on the propagation speed. As the figure shows, worm propagation speed decreases with smaller TTL values. In the figure, note that for a low scanning rate of 4 addr/sec and a short expiration interval of $\tau = 1$ min, the worm would only infect 37% of network machines after 2000 minutes. This again emphasizes the significance of reducing TTL values in enhancing the efficacy of the approach.

Similar to adaptive port-scanners, attackers could develop an adaptive DNS-cache-based worm that, once it infects a machine, only scans IP addresses within its local DNS cache, as described in Alg. 3. Figure 7 compares the propagation of a random vs. an adaptive worm in an unprotected vs. a protected network. Note that an adaptive worm would infect 74% of network machines in 2000 minutes. Note that the propagation of an adaptive worm is comparable to that of a random scanning worm with an aggressive scanning rate (300 addr/sec) in a protected network with an enforced TTL value of 5 minutes. This is expected since a fast-scanning worm would have a high probability of successfully hitting all the machines within the local DNS cache during a 5-minute interval.

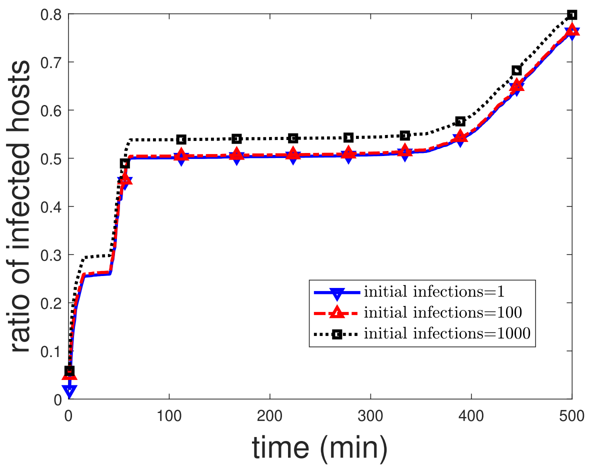

FIGURE 8. Propagation of adaptive worms in a protected network with various initial population sizes.

In previous figures, we assumed the propagation starts from one initially-infected machine. Figure 8 compares the propagation of an adaptive worm with various population sizes of initially infected populations in a protected network. Note that in a protected network, even with an initial infected population of 1000 machines, only 76% of network machines are infected after 2000 minutes, which is slightly higher than the original scenario. This again shows the effectiveness of our approach in throttling scanning-based attack vectors in internal networks.

B. DETECTING EXTERNAL SCANNERS

From the perspective of our network, the behavior of an external port scanner and an external network worm are the same because they both attempt to send packets to an IP address of our network without first querying the authoritative DNS. If a network worm infects an enterprise machine, that machine then becomes an internal network worm and will be countered by the internal detection method. Thus, in evaluating external scanners, we focus on a generic threat model where an external machine sends packets (*e.g.*, either scanning probes or exploits) to an IP address of our network.

The client subnet extension allows authoritative DNS servers to provide responses that are specific to a subnet address. After excluding the reserved addresses, the number of public IPv4 addresses on the Internet is close to 4.2 billion. If all the recursive resolvers implement EDNS0 Client Subnet option and announce client subnets with the recommended /24 length (*i.e.*, each subnet includes 2^8 addresses), and a public server has clients from $1,000$ different subnets at any time, then the probability that a random scanner on the Internet succeeds to scan (false negative) a public DMZ machine of our enterprise at that time is $(1000*(2^8-2))/(4.2*10^9) \simeq 5.9*10^{-5}$, which is very low.

Figure 9 generalizes this evaluation for different ECS subnet prefix lengths and the different numbers of clients from distinct subnets. Note that even when prefix length is 20 and clients are from $10,000$ distinct subnets of size

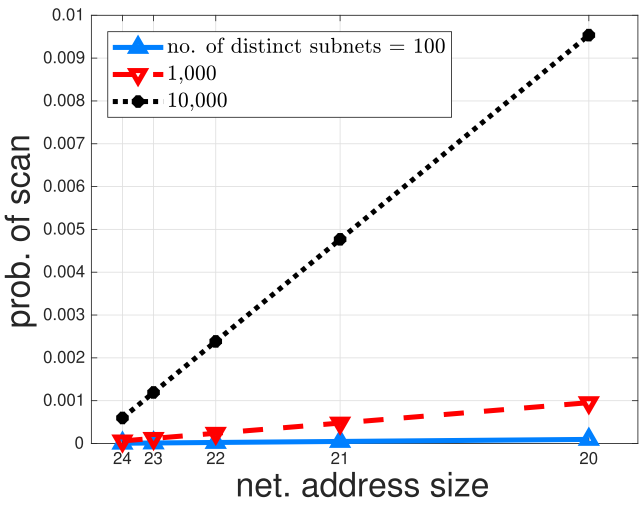

FIGURE 9. Hit rate of external attackers for various ECS subnet prefix lengths.

$2^{32-20} = 4096$, the success probability of a random scanner is still below 0.01.

In order to evaluate our external detection mechanism in a realistic but isolated testbed, we developed four different modules simultaneously running on a test server, collectively emulating the communication architecture and protocols of Fig. 2.

The first module, called the *User* module, emulates the behavior of regular clients. In this module, in an infinite loop, we first select a randomly-generated source IP address representing an external client; we then issue a DNS query - with ECS option enabled - from this client for the target domain name, `www.example.org`, to the *Authoritative DNS server* module; the source prefix-length is chosen using a weighted sampling based on the distribution of prefix lengths on the Internet infrastructure (see Fig. 10) as observed in a recent survey in 2021 [27], and the client subnet is also included in the DNS query based on the chosen prefix-length. After receiving the DNS response, the script sends a TCP SYN from this source IP address to port 80 of the web server. The web server is a Virtual Machine on the testbed server, which hosts a simple web app on port 80.

The *authoritative DNS server* module receives DNS queries and provides a simulated DNS response containing the same type-A RR to all of them; *i.e.*, all queries receive the same Web server IP and same TTL, where TTL value is modified across different experiments. Based on RFC 7871, the scope and subnet prefix-lengths are set to the subnet prefix-length in the DNS query. The TTL value of the DNS RR is set to the given TTL value for that experiment.

The *Scanner* module emulates the behavior of port scanners. In this script, in an infinite loop, we randomly select a source IP address as the address of an external scanning machine, and send a TCP SYN segment from this source IP to port 80 of the Web server, without querying the DNS.

Finally, the *SDS* module implements the proposed defense Algorithm 2; in an infinite loop, the script monitors DNS

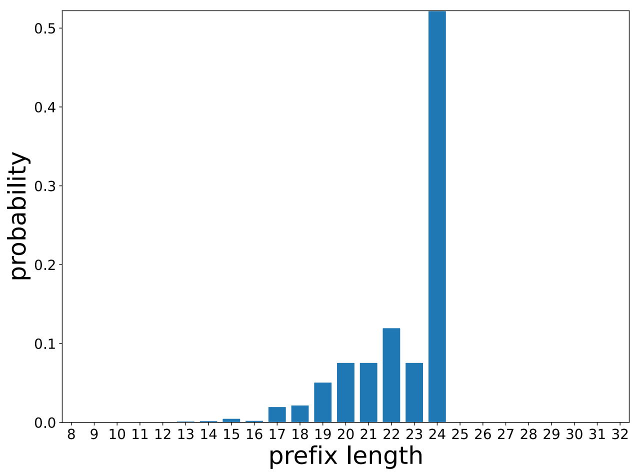

FIGURE 10. **The distribution of prefix lengths announced on the Internet as reported by [27].**

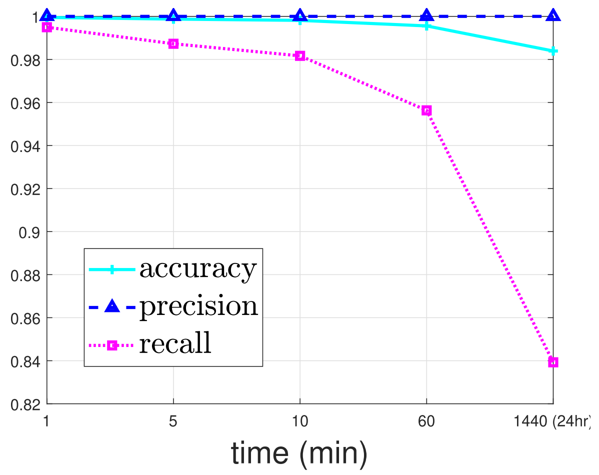

FIGURE 11. **Experimental results against external scanners.**

responses and records the client subnet address and length, and the expiration time (current time + TTL value) of the response in a table, called *Connection Table*. It also monitors new connections and checks them against the entries in the Connection Table: if the source IP belongs to a subnet in the table, it marks the source IP as benign; otherwise, it marks the source IP as malicious (a scan).

Figure 11 shows the results achieved on this experimental testbed for different TTL values. Each experiment is conducted with a different TTL value, and a total of 200,000 connections were observed for each experiment. On average, in each experiment, 90% of connections were benign, and the rest were scans. In Figure 11, note that the approach is still effective, even with TTL values of 24 hours; the precision is the same for all TTL values, which is because the approach does not have any false positives: no benign connection is identified as a scan. This is expected, because, in our simulation, any benign connection is always preceded by a pair of DNS query and response messages. However, we acknowledge that this result needs further investigation on production systems with real users. Especially, TTL pinning (see Section VI-D2) could result in false positives.

However, the approach expectedly incurs false negatives because if a scanning attacker is inside a client subnet from which a benign user has already received a non-expired DNS response, then their scans would be missed by SDS. The larger the client subnet lengths, the higher the likelihood of evasion by a scanning attacker. Obviously, larger subnet prefix-lengths are more desirable, as they increase the accuracy and recall of the approach. Also, note that both accuracy and - especially - recall drop as the TTL value increases because the number of false negatives increases as the TTL value increases. This is because, with a larger TTL, a scanning attacker has more time and thus a higher likelihood of piggybacking on a benign client's previous query, which still resides in the Connection Table, to evade detection. The recall significantly drops for larger TTL values, which proves the gravity of using small TTL values in lowering the number of false negatives, thus enhancing the efficacy of the approach. The average TTL value in current Internet infrastructure is 24 hours, which yields a recall value of 84%. With an enforced 1-minute TTL value, the recall increases to 99.4%.

C. OVERHEAD

In this section, we evaluate the impact of our detection method on authoritative DNS servers and communication latency.

1) OVERHEAD ON AUTHORITATIVE DNS SERVERS

Smaller TTL values for DNS RRs increase the number of DNS queries that are received by our authoritative DNS servers. Expectedly, the shorter the expiration interval, the higher the number of DNS queries. Figure 12 shows the number of DNS queries in a network with 8495 machines and various enforced TTL values. These results are achieved by simulating machine communications using the Los Alamos dataset [19]. In the figure, note that smaller TTL values increase the load on the authoritative DNS server. This increased load is a determining factor in identifying the optimal TTL value. In other words, the TTL value must be enforced such that the authoritative DNS server can handle the resulting expected extra load.

2) EFFECT OF TRAFFIC MONITORING AND MANIPULATION ON LATENCY

SDS devices act the same as legacy IDS/IPS devices. They receive traffic on an ingress interface, inspect and filter malicious traffic, and send the remaining packets on an egress interface. In practice, the latency of SDS processing on traffic depends on the implementation efficiency and the type and specifications of the machining hardware. In a proof-of-concept implementation for evaluating latency, we recorded that a socket read and write for MTU of 1500 bytes incurs a latency of on average 60 μs on each frame. Other operations like extracting A records from DNS responses, checking the hash table for expiration times, etc. require $< 10\mu s$

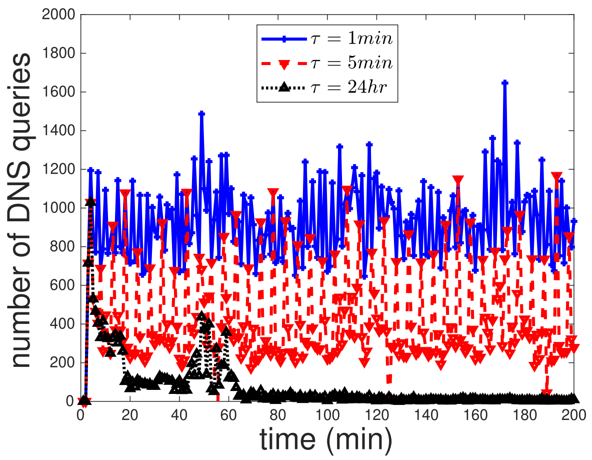

FIGURE 12. Number of DNS queries for various TTL values.

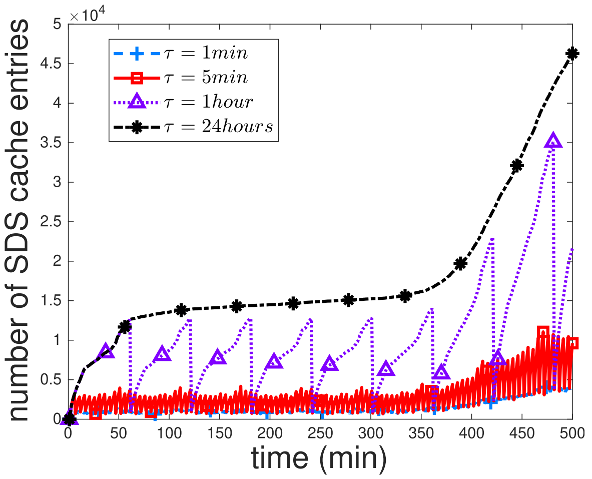

FIGURE 13. Number of SDS entries for various expiration intervals.

in total. We calculated these latencies under normal CPU and memory load. The average latency is less than 70 μs. This is comparable with evaluations of commercial Cisco IPS devices that have reported latencies in the order of 50 to 100 microseconds for different models [30]. Since packet round-trip delay times are usually in the order of milliseconds, the SDS processing time incurs negligible delay in response times.

3) COMPLEXITY AND SCALABILITY OF METHODS

In this section, we investigate the time and space complexity of the proposed methods. Alg. 1 presents our method for detecting internally-initiated scans including *local-to-local* scans as well as *local-to-remote* scans, while Alg. 2 presents our method for detecting externally-initiated or, *remote-to-local* scans. We investigate the time and space complexity for inspecting each of the three classes of local-to-local, local-to-remote, and remote-to-local scans, next.

Local-to-local connections: Suppose a network has n machines, and each machine only has one network interface, thus, one internal IP address is assigned to it. For a subnet that includes m machines, its SDS needs to store $m * (n - 1)$ DNS records in the worst case. Thus, the space complexity of the Connection Table is $\mathcal{O}(m \cdot n)$; thus, the local-to-local scan detection approach is scalable with regard to both the network size and subnet size.

The TTL value impacts the Connection Table size and, thus, the space complexity. The lower the TTL value, the smaller the Connection Table because SDS devices constantly delete expired DNS records from the table. Figure 13 compares the size of the local-to-local Connection Table of an SDS for various TTL values based on the communication traces of machines in the Los Alamos dataset [19]. Note that as the TTL value increases, the number of entries increases accordingly. This is because when DNS records are expiring at a lower rate, the SDS needs to keep track of a higher number of these records. Thus, while using smaller TTL values increases the load on authoritative DNS servers, it decreases the overhead on SDS devices while enhancing the efficacy of the approach. Since we use hash tables to store the Connection Table, the time complexity for insertion and search is $\mathcal{O}(1)$; Thus, the time required for inspecting a local-to-local connection is independent of the network or subnet size.

Local-to-remote connections: Suppose a subnet includes m machines (each with one network interface), and t denotes the maximum DNS cache size of machines in the subnet at any time; then, the space complexity for detecting local-to-remote connections on the SDS device of this subnet would be $\mathcal{O}(m \cdot t)$. The lower the TTL value, the lower the t and the smaller the Connection Table. The time complexity is again $\mathcal{O}(1)$ because this Connection Table is also implemented using hash tables. Thus, the method is scalable with regard to both the subnet size and the DNS cache size.

Remote-to-local connections: Suppose k denotes the number of publicly-reachable machines located in the DMZ of the enterprise network. Assume each public server has active clients (*i.e.*, their DNS responses have not expired yet) from at most p different ECS subnets at any time. The space complexity of the Connection Table of the DMZ SDS device for handling remote-to-local connections would be $\mathcal{O}(k \cdot p)$. Again, the lower the TTL value, the lower the p, and the smaller the Connection Table.

This Connection Table is implemented using a two-dimensional array. The time complexity for inserting a new entry into the table and for inspecting a remote-to-local connection to a public DMZ machine is $\mathcal{O}(p)$ because, as denoted in Alg. 2, for every incoming connection to a public machine, we need to inspect at most p entries in the Connection Table (*i.e.*, only active clients of the public server). Thus, the method is scalable with regard to both the number of public machines and the number of active clients.

D. PRACTICAL ISSUES FOR DEPLOYMENT

1) IDENTIFICATION OF ALLOWLISTED ADDRESSES

An important step in the deployment of our approach is identifying and excluding machines and IP addresses that are not obtained through DNS but through other methods. Network machines that are not named and are typically accessed through IP addresses, such as gateway machines or DNS servers, must be identified and allowlisted. Also, applications whose legitimate operations depend on the IP address, like a printer or a database server with a static IP address, or their IP addresses are attained from mechanisms other than DNS, such as peer-to-peer applications must also be identified and allowlisted. Some busy websites include URLs consisting of numeric IP addresses within the payload of an HTTP packet, in order to outsource their content, especially images, to other servers for faster page retrieval [15]. Such mechanisms are considered malpractices and strongly prohibited in HTTP RFC documentation (*e.g.*, 2616 and 1900). For internal Web servers, these malpractices could be prohibited as an enforced policy. For external cases where an internal machine is communicating with an external Web server that uses IP address embedding, these embedded addresses could result in mistakenly marking an internal machine as malicious. In our experiments on internal scanners in Section VI-A, we observed that almost 50% of false positives were observed in the Web browsing of popular websites belonging to CDN proxies and cloud machines, which are not obtained through DNS and potentially obtained through HTTP payloads. These IP addresses could be identified in a training phase and added to an allowlist. In this stage, rather than dropping detected scans, the SDS can merely generate alerts for further inspection. The network admins would then review the alert logs, inspect IP addresses that cause false positives and add them to the allowlist if needed.

2) DEALING WITH TTL PINNING

Another problematic scenario stems from the TTL pinning technique that is used by many Web browsers to defeat DNS rebinding attacks [31]. DNS rebinding attacks rely on DNS responses with small TTL values to bypass the same-origin policy of Web browsers and access the internal servers of enterprise networks [31]. To defeat this, modern browsers use a technique called TTL pinning: once the browser resolves a machine name to an IP address for the first time, it caches the result for a fixed duration, regardless of its TTL value [31]. TTL pinning is problematic for our approach because by ignoring the TTL value, the browser could initiate a connection with an IP address even after its TTL has expired. This new connection will be falsely identified as a malicious scan. However, the TTL pinning duration is usually short (*e.g.*, [60, 120) seconds in Firefox [31]) and comparable with our enforced TTL, τ. Moreover, to prevent potential false positives, we will add an exception for Web flows as follows: if a client re-establishes an HTTP/HTTPS connection to the same IP address within a fixed interval (*e.g.*, 120 seconds) since the TTL expiration, we will not mark it as a scan. While attackers could use this as an evasion technique, this has a negligible effect on the robustness of the approach, as it only permits probes to a very small set of ports (*e.g.*, 80, 443) of a target machine. Moreover, these scans could only come from machines that have recently queried the authoritative DNS of a target and initiated a legitimate HTTP connection with it, and only within a short interval after the previous TTL has expired.

3) PRACTICAL LIMITATIONS IN DETECTING EXTERNAL SCANS

Defeating external scanning depends on the cooperation of recursive resolvers. However, implementing or complying with the EDNS0 client subnet option by querying machines or recursive resolvers is not mandatory [26]. While originally designed as a mechanism for geographical load balancing and decreasing service provision latency, our work shows the potential of using this option as a robust security feature against scanning attacks. Moreover, ECS protocol is widely being adopted, and comparing the adoption rate from 2015 to 2019 shows a notable growth rate [27]. As of now, ECS-enabled DNS messages constitute a significant percentage of the DNS traffic. ECS is currently "on by default" for all traffic through many of the largest open DNS servers, including Google Public DNS, OpenDNS, Quad9, and NextDNS [27]. A recent study in 2021 [27] investigated several billion DNS queries on the Internet and reported that 69% were ECS enabled. Thus, over time we expect more clients and resolvers to adopt ECS. While attackers can continue avoiding the use of ECS, as the clients increasingly use ECS and provide their subnet addresses for better address resolution, the lack of using ECS could become a sign of suspicious behavior. Also, while attackers are fully motivated to opt out, we can provide incentives for benign clients to opt in; *i.e.*, enabling the ECS option and announcing large subnet prefixes of clients' IP addresses. For example, by giving a higher quality of service (QoS) to conforming clients and penalizing (*e.g.*, rate-limiting) non-compliant connections, we could provide incentives for collaboration by clients. The QoS could even become dependent on the precision (length) of the provided client's address, where the highest priority could be given to clients that include 24 bits or more of their IP address in the DNS query. Over time, as more clients opt in, the remaining non-compliant connections will dominantly belong to attackers, thus enabling us to adopt even more stringent throttling policies against them. We leave further investigations in this regard to future work.

4) EXTENSION TO IPv6 ADDRESS FAMILY

This paper only focuses on the IPv4 address family. Extending our approach to the IPv6 family is straightforward with minor changes. For example, instead of inspecting `DNS A` records in internal and external detection algorithms (Alg. 1 and 2 respectively, for IPv6, we need to record `DNS AAAA` records. The ECS option supports IPv6 addressing as well; the recommended subnet size (source prefix-length) for IPv6 addresses is 56 bits, based on RFC7871 [26].

However, due to the abundance of addresses in the IPv6 address domain, random IP scanning of IPv6 address ranges is not effective, and thus random scanning is not a viable threat model in IPv6, and intelligent heuristics need to be developed to reduce the search space [32], [33].

5) FUSION WITH DNSSEC AND OTHER CRYPTOGRAPHIC EXTENSIONS OF DNS

DNS-over-HTTPS and DNS-over-TLS: DNS does not have any built-in cryptographic capabilities to protect its traffic from on-path eavesdropping and man-in-the-middle attacks. To address this increasing concern that could impact the privacy of users, several standards, such as DNS-over-HTTPS (DoH) and DNS-over-TLS (DoT) have been proposed and widely adopted by major public DNS resolvers, including Google and Cloudflare, and implemented in the latest versions of popular operating systems like `Windows 11`. Both methods encrypt the DNS traffic between the client and its DNS resolver by transmitting it over a TLS channel.

With DoH or DoT in place, since the DNS traffic is encrypted, SDS devices located at the subnet edges can not read and change (the TTL values of) DNS responses since the TLS channel is established between the client and the resolver, and SDS does not have access to TLS keys. Thus, to fuse our approach with these emerging technologies, one viable option is to move and perform DNS traffic monitoring and TTL manipulation on the local recursive DNS resolver of the network; for every DNS query and corresponding response received from any machine, the local recursive DNS resolver updates the corresponding Connection Table. Simultaneously, for every new ingress connection to a machine, SDS devices will communicate with the DNS resolver through a secure channel and query whether a corresponding unexpired entry exists in the corresponding Connection Table for the source IP address. However, the extra time required for sending this query and receiving the response may incur a non-negligible latency on the communications; in future extensions of this work, we will conduct further inquiries in this regard.

DNSSEC: With no cryptographic protection, DNS is susceptible to cache poisoning attacks which could be conducted by off-path attackers through forging and spoofing DNS responses. DNSSEC adds a suite of extensions to the DNS protocol to ensure cryptographic authenticity and integrity of the DNS messages exchanged among recursive resolvers and DNS servers through the digital signing of zone data on authoritative DNS servers. DNSSEC uses public key cryptography to sign and authenticate DNS resource record sets (RRsets). A zone signs its authoritative RRsets by using a private key and stores the corresponding public key in a DNSKEY RR. A DNS resolver can then use the public key to validate signatures covering the RRsets in the zone and thus authenticate them.

Client operating systems do not support DNSSEC and cannot validate DNSSEC messages themselves. Rather, they delegate this job to a trusted recursive resolver that supports DNSSEC. Therefore, even when an enterprise uses DNSSEC, SDS devices can still read and manipulate DNS messages because DNSSEC protects the traffic between recursive DNS resolvers and DNS servers, but it does not impact DNS messages exchanged between client machines and recursive resolvers that pass through SDS devices. Thus, the original method would still work with DNSSEC. Further investigation of this issue is left to future work.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present a defense mechanism for protecting enterprise networks against *internal* and *external* network scanning attacks. These attacks are typically either conducted by cyber attackers in the reconnaissance stage of cyber intrusions to identify active machines in an address space or by network worms in the propagation stage for target discovery. Our approach acts based on the idea that a DNS resolution must typically precede any legitimate communication between two machines; Thus, any communication that is not preceded by a DNS query is marked as a scan, unless it is allowlisted based on the network security policy.

We propose the necessary architectures and algorithms for the deployment of our approach in detecting internally-initiated scans. Through experimentation, we show that this approach achieves over 96% accuracy even without excluding false positives related to allowlisted IP addresses. Using a public dataset of network flows, we simulate our approach in a large-scale network against port scanners and network worms and demonstrate that reducing TTL values (reducing expiration time of cached DNS records) is necessary to reduce the likelihood of evasion by scanners. Moreover, we show that by reducing TTL values, this approach slows down scanning success up to 20, 000 times, even in their best cases. Then, relying on a relatively new extension of DNS protocol, ENDS0 Client Subnet Option (ECS), we develop a variation of this approach for detecting externally-initiated scans conducted against our enterprise network. Through a realistic emulation built based on the latest statistics on ECS adoption by DNS servers, we confirm that our approach can achieve over 98% accuracy in detecting external scans. We discuss the overhead of this approach, in terms of increased load on authoritative DNS servers, and communication latencies, and confirm that both are reasonable. Finally, we highlight the practical challenges and limitations of the approach.

In the future, we will conduct a full-fledged implementation of evaluation of our approach against externally-initiated threats. We also plan to deploy our approach in a production network to conduct a more fine-grained investigation of cases where our approach does not apply or could disrupt the operations of the network.

REFERENCES

[1] S. Shin and G. Gu, "Conficker and beyond: A large-scale empirical study," in *Proc. 26th Annu. Comput. Secur. Appl. Conf.*, Dec. 2010, pp. 151–160.

[2] D. Moore, C. Shannon, and K. Claffy, "Code-red: A case study on the spread and victims of an internet worm," in *Proc. 2nd ACM SIGCOMM Workshop Internet Measurment*, 2002, pp. 273–284.

[3] S. E. Schechter, J. Jung, and A. W. Berger, "Fast detection of scanning worm infections," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2004, pp. 59–81.

[4] C. C. Zou, D. Towsley, and W. Gong, "On the performance of internet worm scanning strategies," *Perform. Eval.*, vol. 63, no. 7, pp. 700–723, 2003.

[5] P. Security. (2017). *WannaMine—New Cryptocurrency Malware Exposes Failings Traditional Anti-Virus Tools*. [Online]. Available: https://www.pandasecurity.com/mediacenter/mobile-news/wannamine-cryptomining-malware/

[6] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[7] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet dossier," *Symantic Secur. Response*, vol. 5, no. 6, p. 29, 2010.

[8] (2023). *Reconnaissance Techniques; Active Scanning: Scanning IP Blocks*. [Online]. Available: https://attack.mitre.org/techniques/T1595/001/

[9] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues Inf. Warfare Secur. Res.*, vol. 1, p. 80, Apr. 2011.

[10] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009,

[11] Z. Durumeric, M. Bailey, and J. A. Halderman, "An internet-wide view of internet-wide scanning," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 65–78.

[12] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 605–620.

[13] S. Stafford and J. Li, "Behavior-based worm detectors compared," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2010, pp. 38–57.

[14] R. Hiesgen, M. Nawrocki, A. King, A. Dainotti, T. C. Schmidt, and M. Wählisch, "Spoki: Unveiling a new wave of scanners through a reactive network telescope," 2021, *arXiv:2110.05160*.

[15] D. Whyte, E. Kranakis, and P. C. Van Oorschot, "DNS-based detection of scanning worms in an enterprise network," in *Proc. NDSS*, 2005, pp. 1–17.

[16] IS Consortium. (2019). *EDNS Compliance Report: 2019-07-22T00:31:18Z*. [Online]. Available: https://ednscomp.isc.org/compliance/summary.html

[17] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "A multi-resolution approach forWorm detection and containment," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2006, pp. 189–198.

[18] J. Jung, R. A. Milito, and V. Paxson, "On the adaptive real-time detection of fast-propagating network worms," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2007, pp. 175–192.

[19] A. D. Kent, "Cyber security data sources for dynamic network research," in *Dynamic Networks and Cyber-Security*. Singapore: World Scientific, 2016, pp. 37–65.

[20] M. Roesch, "Snort: Lightweight intrusion detection for networks," *Lisa*, vol. 99, no. 1, pp. 229–238, Nov. 1999.

[21] Z. Jammes and M. Papadaki, "Snort IDS ability to detect Nmap and Metasploit framework evasion techniques," *Adv. Commun. Comput. Netw. Secur.*, vol. 10, p. 104, Nov. 2013.

[22] M. A. Ahmad and S. Woodhead, "Containment of fast scanning computer network worms," in *Proc. Int. Conf. Internet Distrib. Comput. Syst.* Cham, Switzerland: Springer, 2015, pp. 235–247.

[23] M. A. Ahmad, S. Woodhead, and D. Gan, "A countermeasure mechanism for fast scanning malware," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Servicesq*, Jun. 2016, pp. 1–8.

[24] C. A. Shue, A. J. Kalafut, M. Allman, and C. R. Taylor, "On building inexpensive network capabilities," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 2, pp. 72–79, Apr. 2012.

[25] A. Gupta and L. S. Sharma, "Detecting attacks in high-speed networks: Issues and solutions," *Inf. Secur. J., A Global Perspective*, vol. 29, no. 2, pp. 51–61, Mar. 2020.

[26] C. Contavalli, W. Van Der Gaast, D. Lawrence, and W. Kumari. *Client Subnet in DNS Queries*, document RFC 7871, 2018. [Online]. Available: https://tools.ietf.org/html/rfc7871

[27] A. Kountouras, P. Kintis, A. Avgetidis, T. Papastergiou, C. Lever, M. Polychronakis, and M. Antonakakis, "Understanding the growth and security considerations of ECS," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–16.

[28] O. Hohlfeld, J. Ruth, K. Wolsing, and T. Zimmermann, "Characterizing a meta-CDN," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2018, pp. 114–128.

[29] (May 2022). *Alexa Top Websites*. [Online]. Available: https://www.alexa.com/topsites

[30] Cisco. (2015). *Performance of Cisco IPS 4500 and 4300 Series Sensors*. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/security/ips-4500-series-sensors/white_paper_c11-716084.html

[31] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh, "Protecting browsers from DNS rebinding attacks," *ACM Trans. Web*, vol. 3, no. 1, pp. 1–26, Jan. 2009.

[32] T. Chown. *IPv6 Implications for Network Scanning*, document RFC 5157, 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5157

[33] A. Murdock, F. Li, P. Bramsen, Z. Durumeric, and V. Paxson, "Target generation for internet-wide IPv6 scanning," in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 242–253.

JAFAR HAADI JAFARIAN (Member, IEEE) received the B.Sc. degree from the University of Tehran, Tehran, Iran, in 2005, the M.Sc. degree from the Sharif University of Technology, Tehran, in 2008, and the Ph.D. degree from the University of North Carolina Charlotte, Charlotte, NC, USA, in 2017. He is currently an Assistant Professor with the Department of Computer Science and Engineering, University of Colorado (CU) Denver. He is also the Founder and the Director of the Active Cyber and Infrastructure Defense (ACID) Laboratory, CU Denver. His current research interests include active cyber defense for emerging threats on infrastructures, big data analytics for cyber threat intelligence, and security of cyber-physical systems and critical infrastructures. In his prior research, he has also worked on web security, access control, and privacy. He has over 40 peer-reviewed publications in the security and privacy domain, and his work especially on active cyber defense, has received noteworthy attention from academia and industry.

MASOUMEH ABOLFATHI (Member, IEEE) received the B.Sc. degree in computer science from the University of Tehran, Tehran, Iran, and the M.Sc. degree in computer science from Tehran Polytechnic University, Tehran, Iran. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of Colorado (CU) Denver, Denver, CO, USA. Her research interests include data-driven cybersecurity, privacy-enhancing technologies, security and privacy, network traffic analysis, machine learning, and deep learning.

MAHSA RAHIMIAN received the M.Sc. degree from Colorado Technical University. She is currently pursuing the Ph.D. degree with the University of Colorado (CU) Denver. She is also working for the state of Colorado as an interoperability analyst, providing leadership and technical support to several cybersecurity projects. Her research interests include cybersecurity, physical security, and network traffic analysis.