On Building Inexpensive Network Capabilities

Craig A. Shue
Worcester Polytechnic Institute
cshue@cs.wpi.edu

Andrew J. Kalafut
Grand Valley State University
kalafuta@gvsu.edu

Mark Allman
International Computer Science Institute
mallman@icir.org

Curtis R. Taylor
Oak Ridge National Laboratory
taylorcr@ornl.gov

ABSTRACT

There are many deployed approaches for blocking unwanted traffic, either once it reaches the recipient's network, or closer to its point of origin. One of these schemes is based on the notion of traffic carrying *capabilities* that grant access to a network and/or end host. However, leveraging capabilities results in added complexity and additional steps in the communication process: Before communication starts a remote host must be vetted and given a capability to use in the subsequent communication. In this paper, we propose a lightweight mechanism that turns the answers provided by DNS name resolution—which Internet communication broadly depends on anyway—into capabilities. While not achieving an ideal capability system, we show the mechanism can be built from commodity technology and is therefore a pragmatic way to gain some of the key benefits of capabilities without requiring new infrastructure.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols—*Applications*; C.2.0 [**Computer Communication Networks**]: General—*Security and protection*

General Terms

Security, Design, Experimentation

Keywords

DNS, NAT, Capabilities

1. INTRODUCTION

The traditional principle of *being conservative in what you send and liberal in what you accept* has strengthened the Internet along many dimensions over the years. However, as the volume and type of security threats multiply over time, showing leniency towards unexpected traffic has diminishing appeal. This change in attitude manifests in the widespread use of firewalls, intrusion detection and prevention systems, anti-virus systems and spam filters. These mechanisms are put in place to implement conservativeness in terms of what traffic is acceptable upon arrival. However, the senders and the network itself remain blind to the decisions taken by these recipient-centered mechanisms.

Several alternate mechanisms have been proposed to keep unwanted traffic away from recipients, for instance to mitigate denial-of-service (DoS) attacks [11, 12, 23–25]. Additionally, the off-by-default concept includes notions in the routing tables for what services are allowed at the recipient, leaving the remainder unreachable [4]. An alternate proposal is based on *capabilities* [1]. Using this approach, a communication initiator must present a valid capability to the network and end system in order to engage in communication. Many methods of obtaining capabilities are possible, varying from asking a capability server which fulfills requests at a limited rate, to presenting some credentials to authenticate to the capability server before being granted access to some sensitive service (e.g., a company's file server). Without a valid capability, an initiator cannot even attempt to access an end host. This can isolate end hosts from random probing and vulnerability testing by attackers.

Obtaining a capability provides a new *decision point* whereby the originating peer, the type of traffic, the intended duration of communication, and other aspects can be vetted before communication starts. Introducing this decision point and enforcing its judgments naturally leads to new complexity. The initiator must first obtain a capability and then use it in all subsequent traffic. If the network wants to enforce the capability, it needs to validate the capabilities as they traverse the network. Finally, the service being protected has to furnish a server to distribute capabilities. One suggestion to deal with a number of the logistical difficulties is to use the DNS infrastructure to distribute capabilities [19, 20]. In this way, we do not need additional infrastructure to serve capabilities, but they are simply included as a component of a natural part of the process (e.g., a DNS response).

In this paper we ask the question: *Can we simply use the DNS request itself as the decision point in lieu of a distinct capability?* The obvious answer to this question is "no" since host names are simply abstractions for IP addresses which serve as the actual communication endpoints. In other words, while a DNS server can refuse to answer a query, that does not prevent the peer from accessing the given host using an IP address (e.g., as cached from a previous lookup). However, if we were to borrow a notion from the attackers' playbook and allow services to flux across IP addresses (a la fast fluxing servers within the underground economy [21]) then DNS responses would gain prominence as a required component of communicating with a server. This observation leads to our proposal of a lightweight capabilities mechanism whereby the name of a service is sta-

ble but the IP address of the service is not and therefore *the IP address becomes the capability* returned to a given DNS query. Subsequent traffic that knows the IP address (i.e., carries the capability) will be granted access to the end host, while traffic arriving at a currently unused IP address will be discarded.

Our approach allows (*i*) using DNS for the capability setup logistics without requiring additional machinery and (*ii*) the reuse of all the network's considerable machinery for acting upon traffic based on IP addresses. Our scheme does sacrifice some properties of the original capability vision. For instance, while traditional capabilities call for allowing any element in the network to validate traffic we do not enable such verification. Therefore, while traditional capabilities can protect both a service and its network by keeping unwanted traffic far away, our scheme can protect the service itself, but not the service's network (e.g., which could still come under DoS attack). While our mechanism is less powerful, we believe our approach of largely leveraging existing technologies provides a pragmatic tradeoff between gaining some of the benefits of capabilities while easing implementation. Further, with modest, optional additions to DNS, we enable a collaboration between edge networks to prevent unwanted traffic near the originator. While this does not make the recipient "liberal" in what it accepts, it empowers the originating edge network to be conservative in what it sends and it sets up a situation whereby the actions of the recipient are at least transparent (e.g., as suggested in [14]).

2. LIGHTWEIGHT CAPABILITIES

As sketched above, the basic idea of our proposal is to flux server IP addresses so that clients are required to periodically query the DNS for a server's address. Such a mechanism turns the DNS lookup into a decision point whereby the server's network can decide to allow or deny service to some client before that client can communicate in any way with the server. Further, this prevents arbitrary clients from probing to learn about a server (e.g., scanners).

We build the mechanism from existing technology, specifically DNS servers and Network Address Translators (NAT). We assign each server in the network a static private address. We then empower the DNS server to drive the translation table in the NAT. A granted capability manifests as both a DNS response to the requesting client and a mapping between some unpredictable public address and the private address of the server requested in the DNS query. The translation in the NAT is set to expire when the TTL given in the DNS response expires. As sketched, this machinery is confined to the server's network. This eases the deployment path because clients do not have to be modified. Further, it follows the natural incentives present in the network, since the server's network is ultimately the entity with a direct interest in the server's security. While such a scheme is straightforward, a number of design issues immediately arise which we tackle in the remainder of this section. Issues surrounding an actual implementation will be discussed in the next section.

2.1 Capability Decisions

A first question that arises is how a DNS server can make a decision whether to grant a capability (i.e., return an IP address) based on an incoming lookup. This is ultimately a policy decision and therefore we do not tackle the particulars of how an organization would go about vetting a lookup. However, we note that several approaches are possible.

- **Leveraging Previous Behavior:** The output of an institution's general security monitoring infrastructure (e.g., network- or host-based intrusion detection systems, server logs, scan detectors) can be distilled to a blacklist of known malicious actors at any given time.
- **Rate Limits:** The DNS server could enforce a rate limit for each protected server to limit the damage from DDoS attacks that target specific servers. Of course a rate limit in the DNS server merely moves this attack to being a denial-of-capability attack. However, the impact on legitimate clients is not increased with our approach since a traditional DoS attack will likely limit legitimate access to the given server. In fact, impact on legitimate clients accessing other servers during a denial-of-capability attack will be overall lower than during a full-blown DoS attack. This is due to the lightweight decision process at the DNS server, which may be more challenging to overwhelm computationally than some application servers.
- **Strong Authentication:** While not required, a scheme whereby DNS requests are cryptographically authenticated would allow the DNS server to verify the requester.

There are additional mechanisms for making policy decisions such as client puzzles [17] or assessing the client's capabilities in some fashion. These mechanisms can help address traditional weaknesses of capabilities, such as "denial of capability" attacks [3]. Such extensions for negotiating with and extracting additional information from a client will require additional DNS protocol machinery. While potentially worthwhile, we defer such considerations to future work.

Finally, we note that in conjunction with granting a capability via a DNS response, the network can also leverage the significant abilities of network elements to operate on traffic involving particular IP addresses to further drive policy. For instance, a synergistic policy could call for granting of a capability followed by traffic shaping the resulting activity to some low amount of the available capacity.

2.2 Coupling Capabilities with Activity

Often client hosts use institution- or ISP-provided DNS resolvers to conduct DNS queries on their behalf [13]. These DNS resolvers can serve a large number of clients, and often cache results. This makes correlating a particular DNS lookup with subsequent traffic difficult at best. Hence, when using DNS lookups as capabilities, the NAT will have no way to concretely verify a valid capability was actually retrieved for the given traffic. This could, for instance, allow for *capability sharing* whereby a host could retrieve an IP address and share it across a botnet to DDoS a server. This lack of strong coupling between capability granting and use in our system can be mitigated.

With enough public IP addresses and a short enough TTL, we could correlate capabilities and activity by mapping a particular DNS resolver R to a unique public IP address A for each queried server S. In this case, we can map activity back to a DNS resolver regardless of the actual clients used. This would then allow us to take broad action to thwart observed problems—e.g., by removing the translation between

A and S. Additionally, since we can with high likelihood couple traffic with an R, we can use this as input to future policy decisions involving lookups from R. In § 3.3 we address the issue of how much address space would be needed to provide such a mapping.

Additionally, we can attempt to tighten access even more using the AS numbers of the DNS resolvers. In the work by Mao *et al.* [13], the authors found that 64% of clients were in the same autonomous system as their local DNS server. This provides a reasonable heuristic for default client-resolver associations, but can be further refined. With the unique mapping sketched above, a network could use history and some additional heuristics to likely achieve better associations. If associations between resolvers and client networks can be determined, the destination can build its translation table in the NAT such that the translations only work for hosts within the identified networks. This serves two purposes. First, it prevents arbitrary hosts not covered by a granted capability from reaching the given server. Second, it allows us to leverage the public IP address space further. For example, some public address A could map to server S_1 for traffic arriving from network N_1 and to server S_2 for traffic arriving from some other network N_2.

An alternative approach to inferring the link between a DNS resolver and subsequent traffic is to design a mechanism to make that coupling explicit. This mechanism may optionally be used to achieve the benefits described above, at the expense of being a larger modification to DNS operations. Contavalli *et al.* proposed a DNS option to allow recursive resolvers to indicate the network that originated the DNS query [6]. This option would provide authoritative DNS servers with more detailed network information about a query's origin, allowing better responses for content distribution networks (CDNs). Google's public DNS service is already using the approach with particular CDN partners [15]. We formalize this option into a mechanism whereby DNS requests can carry an *On-Behalf-Of* (OBO) annotation. This would give the netmask of the networks served by the DNS resolver making a request.[1] This would allow a NAT translation table to be setup to only conduct translations for the given networks and not more generally and hence makes capability sharing attacks—whereby a NAT entry is instantiated by some request and then shared widely with attackers—more difficult to mount.

An immediate problem with an OBO request is that DNS requests are easy to craft and therefore it would be straightforward to send a DNS request that claimed to cover arbitrary address space to enable a capability sharing attack. Our notion thwarts this by requiring OBO annotations to include a verification IP address V in each advertised network block. A reverse lookup on V should then yield the same OBO information as advertised in the DNS request. Additionally, the information in V's record may be signed using DNSSEC [2]. In this fashion we force DNS resolvers to control the reverse mapping of at least one address in their claimed blocks. This requirement is problematic for alternative resolvers such as Google Public DNS [8] since the public nature of the service means (*i*) there are no natural network blocks they service and (*ii*) Google does not control the requisite reverse DNS records. We could mitigate the problems surrounding such systems by requiring a small number of whitelisted public DNS servers to provide the specific host making the request. Such an approach makes the reverse lookup less crucial because the capability sharing potential is quite low. Further, we can generally trust that big service providers such as Google are setting OBO properly based on the request they receive and hence the chances of request spoofing are low. Finally, in this fashion we can notice a large volume of capability requests being laundered through a public DNS server and apply appropriate policy to capability requests.

[1]Note that the netmask could be host-specific (i.e., a /32), allowing only the specific client for which the DNS request is being made.

2.3 Dealing With a Crowd

We must also consider that when a DNS resolver acts on behalf of a set of clients, the capabilities issued treat all clients in this set identically. This will result in ambiguous situations whereby a DNS resolver is serving both well-meaning and malicious hosts and yet the granularity of our capability mechanism is not fine enough to handle this situation. We can handle this by blocking or throttling hosts that are viewed as problematic (e.g., by an IDS). This is no different from the current situation. However, given that we have introduced a new decision point for granting access to servers, our position is that we can design a more transparent system.

In particular, we add another DNS annotation that allows the granting of a capability *with exceptions*. So, a local public IP address A might be returned in response to a DNS query, but with an additional note that says "except for IP address *a.b.c.d*". In other words, the resolver should return the mapping to any of its clients except *a.b.c.d*. To verify the authenticity of such exceptions, DNS responses carrying them may be signed [7]. This exception could be either within the announced OBO networks in the request or presumed to be covered by the requesting DNS resolver based on previously developed history (as sketched in § 2.2). The server's network would still be wise to block traffic from the given IP address given that the DNS resolver may not understand the annotation. However, by making the service's intentions clear in the returned capability we empower the client's network to do several things. First, the client's network could block the outgoing traffic, unburdening the network from carrying traffic that does not represent useful work and will ultimately be discarded. Second, it adds transparency to the process—as advocated in [14]—in that it indicates who is blocking traffic rather than the usual situation whereby all one can tell is the traffic is falling into some black hole. Finally, it indicates to the client's network that the server's network believes the given host is malicious. In many cases the client's network has a vested interest in cleaning up such problems.[2]

2.4 Additional Considerations

We now briefly address several additional issues with our lightweight capability mechanism.

Non-DNS-based Communication: Not all communication uses hostnames and therefore the DNS. For instance, many peer-to-peer systems have different ways of determining IP addresses for peers in the system. A general capability system would incorporate these applications, yet our

[2]Obviously in some cases, such as a user employing Google Public DNS, there is no strong relationship and this third benefit does not hold.

lightweight mechanism that relies on DNS and uses IP addresses as the capabilities does not work for these applications. While this is a drawback, we believe there is still much benefit to our system for standard infrastructure-level servers (e.g., an institution's web or mail servers).

Alternate Structure: We base our system on NATs, but could instead simply make the servers themselves change IP addresses. For instance, instead of making the DNS server drive the translation table of the NAT, it could drive the allocation policy of a DHCP server. One practical drawback of this is that long-lived connections would be problematic in the face of servers actually changing their network-level addresses, whereas we have found NATs to handle long-lived connections without trouble (see § 3). A second issue with DHCP is that as discussed in § 2.2 and § 3.3 it is sometimes useful to assign more than one public IP address to a host and making request multiple DHCP addresses at the appropriate times adds additional machinery to the overall process. Assuming one could mitigate that problem, the NAT-vs.-DHCP question is largely one of engineering tradeoffs and not one of fundamental differences.

Denied Capabilities: Given that our proposal calls for distinguishing wanted from unwanted traffic, there is a question of what to do with the unwanted traffic. Ultimately this is a policy decision. The default response to traffic that arrives without a capability is to drop it because there is no translation table entry to direct the traffic. Another option is to vector such traffic to a honeypot to learn about attacks. Likewise, when a DNS server is going to deny a capability it could instead give the requester a capability and translation table entry that sends the client to a honeypot.

Richer Capabilities: As discussed in § 2.3 we could add exceptions to the granted capabilities such that the origin network could potentially block traffic known to be unwanted. However, such exceptions are potentially just the beginning of additional information that could be included in the returned capabilities. For instance, records could include a notion that only the first N hosts or M connections to contact the target server will be granted access, after which a new capability will be required. This would provide a further way to scope incoming traffic to deal with capability sharing attacks. The parameterization of capabilities can be quite rich—for example, aggregate or per connection capacity limits, concurrent connection limits, or host fan-in limits could be used. Ultimately, such information in the capability may be ignored; given the loose nature of the system we have proposed, the recipient cannot count on the origin enforcing the specified policies. However, such annotations are useful in two respects: (*i*) they make the policies at the recipient more transparent, aiding the problem of opaque network failures confounding users and administrators, as sketched in [14], and (*ii*) this information allows the origin network to understand how their traffic will be treated at the recipient and hence plan for the best possible use of their own resources (e.g., outgoing capacity).

3. IMPLEMENTATION

To validate our proposal and identify issues, we implemented our approach in a small testbed. We used an Ubuntu Linux system as our NAT and BIND9 as our DNS server. We wrote a small tool to both update entries in our DNS zone file and the translation mappings in the NAT (via `iptables`). We used Windows XP, Windows 7, and Ubuntu Linux for our clients. Further, we used the following web browsers as clients: Mozilla Firefox 5, Google Chrome 10 and 12, and Internet Explorer 6 and 8.[3] These browsers together account for roughly 94% of web traffic [22].

In our unoptimized testbed, we find that updating a DNS record takes about 13 msec and updating a NAT rule takes about 4 msec. These delays suggest our capability mechanism would add only minimal delays to connection establishment. In addition, our small-scale testing has uncovered two issues that need further attention. First, we find that NAT translation tables do not always persist across updates leading to broken connections. Second, we find that DNS TTLs are not universally honored and therefore our naive capability system must cope with traffic to IP addresses that are no longer valid. We consider these issues and their implications further in the next two subsections. A final implementation question that lingers is the amount of address space required by our system. We tackle this in § 3.3.

3.1 NAT Issues

NATs must track two types of translations in their internal tables. First, a table N maps an external IP address to an internal IP address for new connections. A second table E tracks established connections via the ephemeral port number in the transport protocol header. In our scheme, table N is controlled by the DNS server. If table E is cleared when the corresponding entry in table N is updated, the NAT will sever ongoing communication every time a capability expires and needs to be updated. We find these mappings to be independent in the Linux and OpenBSD NAT implementations. Therefore, changing (or expiring) a capability has no impact on established connections. However, in the NAT implementation in a Linksys WRT54Gv2 router, we found that changes to N do in fact clear the corresponding connections in E and therefore interrupt connections.

The Linksys results are not particularly troubling because (*i*) this is a small client-side device that is not suitable for the sort of infrastructure-level task we propose in this paper, (*ii*) with our mechanism, the NAT device is controlled by the operators protecting the servers and so they can ensure the NAT treats the N and E tables independently, (*iii*) NATs must have two tables as we have sketched above and so any linkage between the two is artificial and can be detangled in the NAT implementation if necessary, and (*iv*) the Linux and OpenBSD implementations show that some NATs do in fact behave in the required fashion already. Therefore, we believe that while care must be taken to ensure a given NAT is working as intended, this problem is not burdensome.

3.2 TTL Issues

A larger issue with our proposal is the use of expired hostname-to-IP address mappings. Previous studies have found the use of expired DNS entries [16]. Likewise, our testbed measurements illustrate the problem. All client operating systems we tested honor the TTLs returned in DNS responses. However, in an effort to mitigate cross-site scripting attacks [9] web browsers "pin" hostname-to-IP address mappings without regard for the TTL in the DNS response. All the browsers we tested pin. Internet Explorer is the most

[3]This browser-centric test will be motivated below as we find that most of the problems with our approach involve web traffic.

Network	Conns.	Within DNS TTL	Outside DNS TTL
ICSI	443K	310K	133K (30.1%)
CCZ	817K	753K	64K (7.9%)
LBNL	6.1M	5.2M	838K (13.8%)

Table 1: DNS and Connection Behavior

egregious by pinning for 30 minutes after the DNS lookup.[4]

Given the results from our testbed that browsers often do not honor DNS-given TTLs, we turned to empirical measurements of traffic from several vantage points. At each site we analyzed all outgoing connections on June 30, 2011 in addition to DNS query logs. We gathered data from the Lawrence Berkeley National Laboratory (LBNL)[5] which encompasses activity from 3-4,000 users and the International Computer Science Institute (ICSI) which covers roughly 100 users. Additionally, we collected data from a fiber-to-the-home residential network run by Case Western Reserve University (the "Case Connection Zone" or CCZ). We further winnowed the data by ignoring connections for which we did not see a DNS response containing the remote peer's IP address.[6] Table 1 shows high-level results. DNS TTL violations are not rare in any of the networks we observed, ranging from 8–30% of the connections.

In Figure 1, we show the distribution of the time interval between a DNS response expiring and the use of the given remote IP address. We find a number of cases where the interval is short ($\leq$ 10 sec) and likely reflects a coarse-grained expiration procedure in DNS resolvers. However, there are also many cases whereby the use is an egregious violation of the TTL. For instance, we find that in the LBNL dataset 18% of the invalid uses come more than one hour after the name expiration. In all three networks, we find that web browsers are the predominant user of expired name-to-IP address mappings accounting for at least 90% of the invalid uses. While we do not delve into these results in great detail here, the relative frequency of the phenomenon is clearly an issue our capability mechanism will face.

Dealing with the issue of remote peers employing expired DNS results is ultimately a policy issue. We outline three strategies for addressing this issue, but since it is a policy issue, different networks can choose different courses of action. The possibilities include:

- One approach is to accept that the clients are broken and let them handle communication failures after the NAT entry (i.e., capability) expires. While this requires no additional complexity at the server and forces the system to work as specified, it may alienate fickle users (e.g., potential customers).

- Another approach is to grant capabilities for long periods of time such that the impact on mis-behaving clients is reduced. This prevents broken communication and errors for the user, but may be too coarse-grained to effectively deal with problems as they arise.

- While the above two methods are bounds on the solution space, the third approach aims to strike a balance: the NAT tracks the remote hosts that have used a mapping, and rather than completely expiring translation table entries, we can tighten the mappings to continue to allow only those hosts previously observed as using the translation for some grace period. This allows the NAT to restrict new use of the capability while at the same time taking steps to not break current and benign behavior. Further, this directly addresses "pinning" because the translation table tightening is done on a host level and the problems caused by pinning are caused by hosts that have previously used a particular lookup. The grace period may be defined such that it does not cover all cases of client mis-behavior and hence there is an incentive for the clients to move towards conformance.

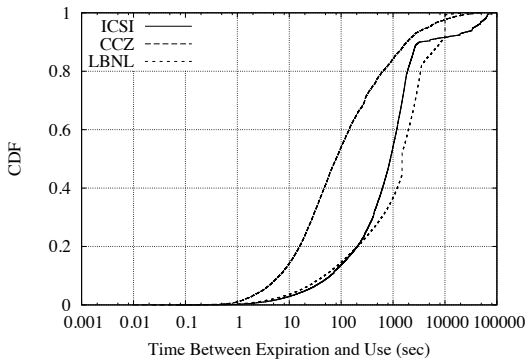

Figure 1: Distribution of time intervals between DNS expiration and IP address use.

A related issue is the impact of DNS pre-fetching, which is becoming popular in browsers to reduce the waiting time users experience after clicking on a link. As long as the TTL returned in a pre-fetched DNS response is respected, the practice has no impact on our capability mechanism. However, if a browser were to employ both DNS pre-fetching and IP address pinning the grace period described above would not ameliorate the potential problems. In this case, the browser would both pin an IP address to a name and not generate any traffic for some time period after the name resolution. So, simply allowing hosts to continue using a binding they have previously used for some grace period is not possible since there has been no use.

We ran quick experiments with Chrome and Firefox and have found that the pre-fetching process simply warms the operating system's DNS cache (as explained for Chrome in [18]). Indeed when we click a link with a pre-fetched hostname after the DNS TTL has expired the hostname is resolved a second time. This means there is no browser-level pinning and hence our capability mechanism will work as expected for these two browsers.

3.3 Address Space Requirements

Our capability system obviously requires some amount of public IP address space, but an outstanding question is how

[4]On the other hand, IE6 is the only browser we tested that will timeout and ultimately load the page when an IP address pin is wrong. However, this behavior changed in IE8.
[5]Note, due to a measurement glitch, we use only 15 hours of data from the laboratory due to missing DNS logs outside of this range.
[6]For example, peer-to-peer applications do not use the DNS and directly route traffic using IP addresses.

many addresses are required. In some sense, our scheme requires no new address space. For instance, if an edge network has N Internet-facing servers now, then all our scheme calls for is the random rotation of the corresponding N address to different servers. We may leverage a small number of additional addresses to cushion the transition of a machine from one IP address to another.

While this minimal set of addresses is in some ways sufficient, it is not ideal. As sketched in § 2.2, to be able to trace behavior back to particular DNS resolvers, allowing future capability requests can be handed out more judiciously, we would like to hand out a unique mapping from a public address A to a private address S for each DNS resolver R that seeks a capability. We can then assume with fairly high likelihood that traffic to A for the given TTL in the response is from hosts associated with R.

To assess the feasibility of the mapping described above, we analyzed roughly 30 hours of traffic logs from LBNL.[7] We found 8.4 million successful incoming connections to roughly 2,000 servers. We then simulated the evolution of a translation table required for assigning each client a unique address[8] for each internal server while assuming TTLs of 20 seconds. We find a required peak table size of nearly 2,200; however, the median is less than 1,200 and the 99^{th} percentile is roughly 1,400. In fact, across the 30 hours, we found we needed more than 2,000 addresses (which we know to be available as they are employed by the servers today) for only 19 seconds. Increasing the pool by 10% compared to the current allocation would leave us fully covered. We also note that while small DNS TTLs will result in more DNS traffic, previous studies have shown that TTLs in the range of 20 seconds do not adversely impact the DNS process [5]. Further, when using DNS replies as capabilities long TTLs do not provide for fine-grained control and hence are likely not desirable.

As a second data point we also ran the simulation using TTLs of 60 seconds and as expected this increased the necessary table size—with a required table size exceeding 2,000 over 80% of the time. When more table entries are required than them number of local public addresses available, we will have to map more than one R to a given A, which may degrade our ability to map activity to capabilities. Over long time periods, we still may be able to assess traffic from particular networks. For instance, if we observe unwanted traffic on some given A we can start to isolate the resolvers given A in an attempt to isolate the bad actors and hence use a different capability allocation process (or the exception mechanism outlined in § 2.3) in the future.

We stress that the above results are illustrative, not conclusive. The results show the behavior at one institution and should not be taken as "representative" or "typical" of other enterprise networks. While promising, future work will develop an understanding of the requisite server pool size with data from a range of edge networks.

A further refinement would be for the capability server to take into account open server ports during the allocation. Consider two servers: S_1 running only a web server and S_2 running only an email server. Now assume resolver R_1 looks up S_1 and resolver R_2 looks up S_2. If the capability server understands this configuration, the DNS server could map both of these servers to a single public IP address A and the resulting traffic could be traced back to R_1 and R_2 by leveraging the destination port number in the incoming traffic (since these two requests were for different hostnames and hence different protocols). Assuming a network has a number of servers with non-overlapping open ports, this offers an avenue for further public address savings.

Finally, we note that the question of a unique mapping becomes nearly moot if IPv6 is employed. Under such a scenario an edge network will have more than enough addresses to flux servers randomly and give each resolver R a unique A in the address space for each server S.

[7]Note, the missing DNS logs that winnowed this dataset to 15 hours above does not impact our analysis here as we disregard the normal DNS process to simulate our own system.
[8]Note, assuming no clients share a DNS resolver is the worst case. We ran an experiment considering clients within the same /24 as using the same DNS revolver and, as expected, this results in a smaller required translation table.

4. DISCUSSION

We have designed a scheme that *roughly approximates* the traditional capability notion. Traditional capabilities offer an architecture whereby clients are pre-approved to communicate with servers. This pre-approval is done in a way that any network element can discard traffic that does not carry an approval. Therefore, both the server and the server's network are protected because malicious traffic targeting either will be discarded well before it arrives at the destination edge network. While our approach does not offer the same degree of protection, we believe it does offer a useful point in the design space, as follows.

- The cost of our approach is lower than traditional capabilities since we are leveraging several pieces of the current system: DNS servers become capability servers, IP addresses become capabilities and NATs become capability enforcers. In the basic case only the server's network needs to change. The worst case from a deployment perspective includes also involving DNS resolvers. This contrasts with involving every network element in the case of traditional capabilities.

- As with traditional capabilities, when no clients have been given access to a particular server the server is completely unavailable. Therefore, even scanning will not find hosts that are not being legitimately used.

- Using our approach, a server does not have a fixed location in the IP address space. Therefore, before legitimate communication a client will naturally perform a DNS resolution. In the basic case, once a capability is granted to one client and a translation is setup in the NAT then any host on the Internet can use that translation. To do so the remote host must either know about the translation or stumble upon it via probing (during the limited window the server is available on the given IP address). We conclude that our approach mostly keeps traffic without capabilities away from protected end systems.

- There is more vulnerability to scanning in our scheme compared to a traditional capability scheme. However, our scheme also raises the bar for scanners to be successful. First, fluxing servers across IP addresses makes IP-based hit lists ineffective. Further, scan detectors work by observing that legitimate traffic is usu-

ally productive, whereas scanning is often unproductive because scanners are guessing at targets (e.g., this manifests as an incoming SYN to a non-existent host or closed port) [10]. By fluxing servers constantly we make it more likely scanning traffic will be unproductive and hence scanners can be readily identified. Finally, returning to a particular server is more difficult because of the lack of a stable IP address.[9]

- Through the use of On-Behalf-Of annotations (§ 2.2) and capabilities with exception (§ 2.3) we can winnow down the set of hosts that can take advantage of a particular NAT table translation. This further protects the server.
- As noted above, traditional capabilities protect the server's network and our approach does not. By issuing capabilities with exceptions our approach can empower the client's network to block unwanted traffic and hence modestly addresses the pushback notion of traditional schemes.

5. CONCLUSION

We have described an approach to implementing capabilities with only a restructuring of currently prevalent technologies (i.e., DNS servers and NATs). The system borrows the notion of IP address "fluxing" from botnets to prevent arbitrary hosts from connecting to particular servers. Rather, by making a DNS request for the server's name the client is given the current address of the server, which acts as a capability. This changes the DNS from a handy abstraction into a *decision point* in the system. By denying a DNS request, the server can thwart communication. We have built a small version of this system and explored the issues in realizing our approach. While our analysis is preliminary, the approach is shown to be promising.

Acknowledgments

Discussions with Tom Callahan, Eamon Johnson and Vern Paxson were helpful during the development of this paper. The anonymous reviewers provided helpful feedback on our initial submission. Josh Blanton helped us understand OpenBSD's NAT implementation. Scott Barron, Ethan Blanton and Tom Callahan helped us understand DNS prefetching. Vern Paxson provided the LBNL data used in § 3. This work was sponsored in part by NSF grant CNS-0433702. Our thanks to all!

The submitted manuscript has been authored by a contractor of the U.S. Government under contract DE-AC05-00OR22725. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allows others to do so, for U.S. Government purposes.

6. REFERENCES

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. *SIGCOMM Comput. Commun. Rev.*, Jan. 2004.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, Mar. 2005.

[9]Note, if an attacker successfully compromises a host then it can "phone home" and therefore returning to a particular server likely becomes straightforward.

[3] K. Argyraki and D. Cheriton. Network Capabilities: The Good, the Bad and the Ugly. In *Fourth Workshop on Hot Topics in Networks*, Nov. 2005.

[4] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off By Default. In *ACM Workshop on Hot Topics in Networks*, 2005.

[5] S. Bhatti and R. Atkinson. Reducing DNS Caching. In *Global Internet Symposium*, Apr. 2011.

[6] C. Contavalli, W. van der Gaast, S. Leach, and D. Rodden. Client subnet in DNS requests. IETF Draft, January 2011.

[7] D. Eastlake. DNS Request and Transaction Signatures (SIG(0)s). RFC 2931, Sept. 2000.

[8] Google. Google Public DNS, July 2011. `http://code.google.com/speed/public-dns/`.

[9] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting Browsers from DNS Rebinding Attacks. *ACM Trans. on the Web*, 3(1), 2009.

[10] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy*, 2004.

[11] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. *ACM SIGCOMM Computer Communication Review*, Jan. 2004.

[12] X. Liu, X. Yang, and Y. Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *ACM SIGCOMM*, 2008.

[13] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A precise and efficient evaluation of the proximity between web clients and their local DNS servers. In *USENIX*, 2002.

[14] C. Muthukrishnan, V. Paxson, M. Allman, and A. Akella. Using Strongly Typed Networking to Architect for Tussle. In *ACM HotNets*, 2010.

[15] OpenDNS. How it works - a faster Internet, Oct. 2011. `http://www.afasterinternet.com/howitworks.htm`.

[16] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan. On the Responsiveness of DNS-based Network Control. In *ACM Internet Measurement Conference*, 2004.

[17] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *SIGCOMM*, 2007.

[18] C. Project. Dns prefetching. `http://www.chromium.org/developers/design-documents/dns-prefetching`.

[19] M. Rabinovich and O. Spatscheck. Evasive Internet: Reducing Internet Vulnerability Through Transient Addressing. In *IEEE Global Internet Symposium*, 2010.

[20] B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *SIGCOMM*, 2004.

[21] The Honeynet Project. Know your enemy: Fast-flux service networks, July 2007. `http://www.honeynet.org/papers/ff`.

[22] w3schools.com. Browser statistics, July 2011. `http://www.w3schools.com/browsers/browsers_stats.asp`.

[23] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*, 2004.

[24] A. Yaar, A. Perrig, and D. Song. StackPi: New packet marking and filtering mechanisms for DDoS and IP spoofing defense. *IEEE Journal on Selected Areas in Communications*, 24(10):1853–1863, 2006.

[25] X. Yang, D. Wetherall, and T. Anderson. TVA: a DoS-limiting network architecture. *IEEE/ACM Transactions on Networking*, 16(6):1267–1280, 2008.