Hardening UDP Services against Response Amplification Attacks

6.875 Project, Spring 2016

Robert Sloan rsloan@mit.edu
Hunter Gatewood hgatewood@mit.edu

Abstract

DNS amplification attacks are one of the most powerful and common denial of service (DoS) attacks currently in use; and their existence is utterly preventable, totally the result of the poor protocol design of DNS.

In this paper, we will explore both the finer details of amplification attacks, and present as a number of novel techniques for hardening web services against amplification attacks.

1 Amplification Attacks *in-the-large*

The general principle of distributed denial of service (DDoS) attacks is that an attacker will send more traffic to a server than its network interface can deal with: as more and more traffic floods the server and its routers, the attacker's packets will fill up available buffer space and cause the server to drop packets, many of which will belong to legitimate clients, who will, as a result, be *denied service*.

The nuance of an amplification attack is that while the amount of traffic an attacker can generate alone will be limited, we can *amplify* it to something that will cause trouble in a variety of ways.

For example, in UDP protocols, we can *spoof* (or forge) a request from our target to, for example, a DNS server; and that server will respond to our target. If the response is larger than the request we had to send, then this will allow us to send that much more traffic to our target.

In recent years, this attack – called *DNS amplification* – has become the most popular DDoS attack by an order of magnitude [14]. Attacks have reached rates of 300 Gigabits per second [9], wasting enormous amounts of web resources [8].

The fact that it's still possible to do this is patently ridiculous, a result of the decentralized, "just-good-enough" [5] nature of the DNS, the necessarily public status of DNS servers, and the high response-size asymmetry afforded by certain DNS queries [11].

This paper aims to analyze these attacks, as well as provide novel techniques for their mitigation that will require no meaningful change in the semantics of DNS.

2 The *Why* of DNS Amplification

DNS is fundamentally a query language; requests are queries for the records corresponding to a given domain and the response lists those records. It is in this way unsurprising that there is this request-response size asymmetry (up to a factor of 179 [11])

Now, we see some obvious holes in this scheme:

– *Why not return fewer records?*

DNS is an eminently *extensible* protocol: servers can return arbitrary records, only some of which are actually domain names – mail server (MX) records, DNSSEC signature records, and TXT records are good examples of where this can get very large [13].

Because DNS does not have a facility to specify exactly which records the server should return, we generally return all of them to maintain backwards-compatibility. It's unclear if this issue is solvable with the DNS protocol.

– *Why would the server respond to just anybody?*

It shoudn't, and it usually doesn't; most Service Providers limit access to their DNS servers to a specific IP range. Many, however, don't bother to implement this basic security measure, which we will revisit under the heading *Open Resolvers*. Misconfiguration is another issue whose solution is unclear, given the diversity of software distributions in use.

– *Can we paginate or otherwise restrict the response?*

Theoretically, we do: the DNS RFC dictates that UDP response sizes should be limited to 512 bytes[10], requiring an additional TCP session to transact the rest.

In practice, however, we have seen much larger amplification rates, due in part to some servers providing larger responses (which we have seen in Wireshark), but also possibly because some clients are accepting TCP sessions they did not request (which we could not replicate with `glibc`'s `libresolv`).

Either way, something is incorrect here, but determining exactly which vendors' software is misbehaving is outside the scope of this project.

Other major issues follow:

2.1 Open Resolvers

Open Resolvers are name servers which will, as above, recursively serve *any* user, leading (perhaps unwittingly) to the crux of the amplification problem.

In the iterative mode of DNS, when a Name Server does not own the requested resource (and assuming no caching), it replies to the query with the location of a name server which is likely have the resource.

In the recursive method, however, a name server (after acting as the client itself) returns the resource itself, rather than the location of a name server. This means any server can return any record, of any size, which leads to the large amplification described above[15].

According to recent estimates, there may be many tens of thousands of large open relays currently operating, many from Chinese ISPs [16].

2.2 Decentralization

One of the hallmarks of the DNS is its decentralization, which, while important for long-term stability, does lead to problems of coordination:

– We can't update name server software consistently across vendors, which are globally distributed, multilingual, and intensely bureaucratic [12].

– No centralized authority exists to affect a concerted effort toward mitigating DNS amplification.

– We can't penalize misbehaving servers: name servers utilized in an amplification attack are not generally penalized legally or otherwise, and their administrators have no incentive to update their software [12].

– Many embedded network devices make updating difficult or impossible.

3 EDNS and the Possiblity of Extension

As described above, we can ascribe a large portion of the blame to the DNS extensions (so-called *EDNS*), which are used, for example, to support DNSSEC [2].

These protocol extensions do provide enormous utility (e.g. signing the DNS records to prevent cache poisoning), and so removing EDNS makes little sense; but couldn't this capability be put to work for our own ends? We will discuss this in the *Weak Client Validation* section.

4 Generally Workable Mitigation Strategies

4.1 Load-balancing and Redundancy

Most of the large websites (Google, Facebook, etc.) prevent this form of DDoS by simply being sufficiently well-replicated that *there isn't a single server to attack*.

This method is most commonly supported by CloudFlare – a service that provides caching, HTTPS, and other services to a website – and is becoming one of the best defenses for DoS, especially for static websites via, for example, Amazon or Akamai's CDN [1].

It is, however, not feasible for all applications; and it does not seem sustainable in general to simply assume that all relevant websites are run through a large CDN. Certainly, individual corporate networks will still be vulnerable.

4.2 Server-level Filtering

As mentioned above, we really should be, on the server level, preventing this sort of misconfiguration, by OS distribution-provided configuration files or otherwise (for example, by providing default `iptables`, `pf`, or `ufw` rules).

This is, however, immensely difficult; and getting a distribution's committers to cooperate, we expect, will be well-nigh impossible and a political disaster.

4.3 Router-level Filtering

It is further possible to deal with this traffic on the router level: we can simply have the service provider's routers reject unestablished DNS responses, or stop large streams of DNS traffic (a *kill-switch* method). This, however is infeasible for many reasons:

– Because the client port is unspecified, the router will not know *a priori* which traffic is a DNS response [12].

– This would require dealing with hardware vendors individually

– The logic required would fall under the heading of Software-Defined Networking (SDN): we created OpenFlow rules which would perform the appropriate routing. However, SDN does not yet have enough adoption to make this approach viable.

[1] https://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack/

4.4 Server-level Logic

Where we may be able to actually change something is in the DNS server software itself, with customized logic which maintains backward compatibility: changes to BIND would – eventually – be pushed down to most Linux or BSD-based servers, which comprise a large proportion of Internet servers.

Our candidate solutions will all be of this form: changes in the internal logic of the BIND server which preserve correct behavior while making the execution of an amplification attack much more difficult:

5 Proposed Hardening Strategies

5.1 Semantics of Response Prioritization

It will be easiest to express our solutions in terms of a single set of semantics: we will make the assumption that we can model any logic-based solution in terms of a change in the *prioritization* of DNS requests, as to maintain correctness we must serve the request *eventually*.

Say that we maintain partitions over IPs via some known function

$$\mathrm{Part} \ : \ \mathrm{IP} \to \Pi$$

Where Π is the type of client *partitions*, chosen so that we expect the empirical distribution of requests over partitions to be approximately uniform, or

$$\{a \leftarrow \mathcal{U}(\mathrm{IP}); \mathrm{Part}(a)\} \approx \mathcal{U}(\Pi)$$

For each partition, we record a *priority* $\delta \in [0, 1]$. We can then serve requests according to a *min-heap* over serving-times:

$$\tau_{r \in \mathrm{Requests}} = t_{\mathrm{now}} + d_{\max}(1 - \delta_r)$$

Where we at any time only serve requests up to t_{now}. This maintains the commonsense properties

- Requests with lower priority will always be served later.
- We will never delay a request by more than $d_{\max}$

Thereby providing implicit correctness. For an ordinary implementation, we can rely on the pseudorandomness of some existing hash function H, and set

$$\mathrm{Part}(a) = H(a)_{1..k}$$

for some k such that 2^k integers will not consume much memory; in our simulations, we set $k = 16$.

5.2 Simple Client-Validation Strategies

Now, we will increase or decrease a client range's δ whenever we receive, respectively, a positive or negative indication that its requests are legitimate:

- If we successfully establish a TCP session with a client, we divide $\delta(\pi)$ by α.
- If we are on the same class-B network, we divide $\delta(\pi)$ by α.

– We record the number of requests for each π as a proportion of total requests. If this proportion is higher than $5/|\Pi|$, we multiply $\delta(\pi)$ by α.

– We record the interval between requests for all partitions: for as long as a given partition is more than two standard deviations above the mean, we multiply $\delta(\pi)$ by α.

Where, obviously, $\alpha \in [0, 1]$; and we use $3/4$ for simulations.

5.3 Weak EDNS Response Validation

Those highly-heuristic strategies will likely work; but can we do better? Can't we maintain some state between the server and client to validate each other?

We suggest using an additional EDNS record (call it type OPT) to do that: on each server response, it will additionally provide a token, nominally equal to

$$H(\text{serverIP}\|\text{clientIP}\|k)$$

for some hash function H and ephemeral key k. The client has simply to save this value for any number of requests, to be specified in the query as another OPT record (much like a CSRF token)! It will be hard to forge because of the assumed pseudorandomness of H.

If we multiply $\delta(\pi)$ by α when a client fails to provide a valid OPT token, then in the steady state, well-behaving clients will have high δ, and a spoofed client will have $\delta = 0$, as desired.

5.4 Weak Request Proof-of-Work

Alternatively, we can step into the Bitcoin Zeitgeist and require that our client provide a proof-of-work before being allowed to receive DNS responsesṪhe easiest thing to do is to directly use the Bitcoin method of producing a number x that satisfies

$$H(\text{serverIP}\|\text{clientIP}\|x) = 0^k\|...$$

For some small fixed k, likely on the order of 20. This is strictly weaker than the response-validation method, and would not strictly-speaking prevent amplification attacks; but it would make them much more difficult and less push-button.

6 Results of Local Simulation

Toward the beginning of this project, we began developing a simulation toolchain to support the investigation of DNS amplification attacks; however, as our methodology became clear, the behavior of the system under our controls is *actually somewhat obvious*, and not particularly helpful. We have, however, summarized the results here:

– The response prioritization worked in simulation, because we could arbitrarily trigger additional positive-biasing or negative-biasing events. It's actually unclear, though, how often actual servers receive, for example, TCP extension requests. We would need actual traffic logs to determine the sort of configuration that would work in practice, but we can indeed make the performance sensitive to these positive indicators.

– When we condition the time-delay of transmission on the propagation of an unpredictable token, we necessarily see that spoofed connections never replicate this behavior, and will always be maximally delayed.

7 Large-Scale Viability

7.1 General Server Patches

The ideal case for a final deliverable is a patch to core software, such as *libbind* or *libresolv*; however, this is, as we have discovered, an enormously difficult problem. While it's very possible to make simple modifications to the inner request-handling logic of, for example, `glibc`, it's very complicated and likely difficult to maintain. As a result, it makes much more sense to adopt a more scalable solution.

7.2 A Network-Control Library

The real solution we have decided on (and made very limited progress toward) is a library to automatically provide the data structures and controls to handle prioritized request queues, as described above. The mathematics itself is very simple, and our Go implementation has a large degree of implicit parallelism.

Implementing it in this way has a number of advantages:

- Easy integration into a wide variety of tools
- Generalized testing, i.e. we won't have to test the method per-implementation
- Much easier to integrate into a network simulation framework

Naturally, patches are much more easily adopted, but we believe that these advantages provide ample justification for a library-based approach.

8 Conclusion

While the conclusion to a paper on an enormously difficult and probably intractable problem will necessarily be less than totally positive, we believe that these ideas are a positive contribution, detailing a different way to approach the problem, which should provide some encouragement to people attacking the *big problems* of the Internet.

The Weak Response Validation and control techniques are the main contribution of this paper: they are, so far as we know, a novel approach to lessening the strength of amplification attacks, and they may actually be feasible with the support of a large organization like Mozilla. We hope in the future to investigate what, exactly, it would take to put this into practice.

References

[1] Anagnostopoulos et al. (2013). *DNS amplification attack revisited*. Computers & Security, 39, 475-485.

[2] Arends, R. et al. (2005, March). *Protocol modifications for the DNS security extensions*. RFC 4035.

[3] Dagon, D., Antonakakis, M., Day, K., Luo, X., Lee, C. P., & Lee, W. (2009, February). *Recursive DNS Architectures and Vulnerability Implications*. In NDSS.

[4] Fachkha, C., Bou-Harb, E., & Debbabi, M. (2014, March). *Fingerprinting internet DNS amplification DDoS activities*. In New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on (pp. 1-5). IEEE.

[5] Handley, M. (2006, July). *Why the Internet only just works*. BT Technology Journal Vol 24 No 3.

[6] Kambourakis, G., Moschos, T., Geneiatakis, D., & Gritzalis, S. (2007). *Detecting DNS amplification attacks*. In Critical information infrastructures security (pp. 185-196). Springer Berlin Heidelberg.

[7] Kambourakis et al. (2007). *A fair solution to DNS amplification* attacks. In Digital Forensics and Incident Analysis, 2007. WDFIA 2007. Second International Workshop on (pp. 38-47). IEEE.

[8] Kravets, David (2011). *U.N. report declares Internet access a human right*. Wired.

[9] Markoff, John, & Perlroth, Nicole (2013). *Firm is accused of sending spam, and fight jams Internet*. The New York Times.

[10] Mockapetris, P. (1987, November). *Domain names—implementation and specification*. RFC 1035.

[11] van Rijswijk-Deij, Roland (2014). *DNSSEC and its potential for DDoS attacks - a comprehensive measurement study*. ACM Press.

[12] Vaughn, Randy (2006, March). *DNS amplification attacks*. ISOTF.

[13] Vixie, P., Graff, M., & Damas, J. (2013, April). *Extension mechanisms for DNS*. RFC 6891.

[14] Wueest, Candid (2014). *The continued rise of DDoS attacks*. Symantec Security Response.

[15] (2013, March). *DNS Amplification Attacks*. US-CERT. Alert TA13-088A.

[16] *Open Resolver Project*. `http://openresolverproject.org`.