DNS Water Torture Detection in the Data Plane

Alexander Kaplan, Shir Landau Feibish
The Open University of Israel

CCS CONCEPTS

• **Networks** → **Network security**.

KEYWORDS

DNS Security, Programmable Networks, Data Plane, Network Measurement

ACM Reference Format:
Alexander Kaplan, Shir Landau Feibish. 2021. DNS Water Torture Detection in the Data Plane. In *SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21 Demos and Posters), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3472716.3472854

1 INTRODUCTION

DNS Water Torture (also known as Random Subdomain attack) has been gaining popularity since the severe impact of the 2016 Mirai attack on Dyn DNS servers, which caused a large number of sites to become unavailable [2]. In these attacks, the attacker generates an enormous amount of DNS requests to a particular domain with a random distinct subdomain in each request. The traffic will usually come from many sources (i.e. a botnet) and will be seemingly legitimate, causing DNS resolvers to waste much of their resources looking for non-existent domains.

One existing solution is rate limiting, which is not effective in cases where the attack is highly distributed [2]. A more robust solution is provided by DNSSEC, which enables a range of subdomains to be declared as non-existent following a single NXDOMAIN response. However, the deployment of DNSSEC has been limited [3] and the resolver needs to explicitly support this feature [12]. Our suggested solution can be applied to traffic on the way to the DNS resolver, meaning it does not require any resolver compatibility and can potentially react to the attack at an earlier stage and avoid much of the malicious traffic generated by the attack.

Our Contribution We present WORD, a system for statistical detection of DNS Water Torture that is implemented directly in the data plane using the P4 language. WORD efficiently collects data about DNS requests and responses on a per-domain basis, and alerts the control plane if malicious traffic is detected. The solution we present succeeds in detecting the attack within the notably confined resources of the data plane, while reducing false positives by separately addressing domains which naturally have large amounts of subdomains (e.g. wordpress). In addition, our solution is easily expandable to further DNS related data plane processing, such as other types of DNS attacks, or collection of other DNS statistics in the data plane.

SIGCOMM '21 Demos and Posters, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8629-6/21/08.
https://doi.org/10.1145/3472716.3472854

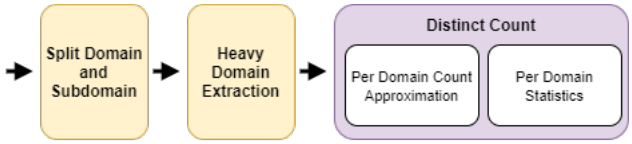

Figure 1: WORD Overview

2 THE WORD SYSTEM

2.1 WORD Overview

The WORD (**W**ater t**OR**ture **D**etection) system is composed of three main parts highlighted in figure 1. Following is a brief overview of each.

Splitting Domain and Subdomain. We define the *domain* to be the part of the DNS query identifying a site and the *subdomain* to be the labels of the query preceding the domain. In order to be able to detect changes on a per-domain basis, for each DNS packet that is received, WORD starts by splitting the subdomain and the domain from the DNS query.

Heavy Domain Extraction. We define a *heavy domain* to be a domain that has a large amount of legitimate subdomains. Examples for such domains include cloud services and blog hosting websites (e.g. wordpress) [9]. Since heavy domains are expected to produce plenty of requests for distinct subdomains we adjust the way WORD processes these domains. Specifically, for heavy domains, we only collect statistics on NXDOMAIN responses, which are expected for non-existent subdomains, such as randomly generated subdomains.

Distinct Count. Finally we maintain an approximate distinct count of the subdomains we encounter for each domain. Additionally we collect more statistics for potentially attacked domains, such as percentage of NXDOMAIN responses and percentage of requests answered.

2.2 Processing

Splitting Domain and Subdomain In order to detect the attack we need to be able to separate the domain and the subdomain from the DNS query. We use the hash of the top 3 labels in the query string to determine the top-level domain (TLD) that this domain belongs to. Using a publicly available list of TLDs [8] we generate a static mapping from these labels to the total number of labels in the domain. For example, with "maps.google.com" we would get 2 labels while "maps.google.co.uk" would yield 3. With this number we are able to split the labels into subdomain and domain, and use this information to calculate the hash of both. These hashes will be used in the next stages to approximate the count of distinct subdomains for each domain. The list of TLD mappings could be adjusted to fit a more specific environment, e.g. a router serving a DNS resolver of a predefined DNS zone. It should be noted that relying on a publicly available list allows us to automate the update process which ensures the TLDs used are always up to date.

Heavy Domain Extraction Heavy domains (which have many legitimate subdomains) are processed separately in WORD; instead of looking at the DNS requests, we will process only DNS responses that are of type NXDOMAIN. This has the benefit of avoiding legitimate subdomains while sacrificing some of our reaction ability to a potential attack, as WORD only begins counting once a response has been sent. We conjecture that DNS resolvers that are in charge of resolving these heavy domains are more tolerant to large amounts of requests, and will therefore still be able to provide responses for some duration when an attack starts, thus allowing us to detect and react accordingly.

Distinct Count While ideally we would like to count each distinct subdomain we see for every domain, the memory constraints in the data plane wouldn't be able to support it. As a possible solution we've considered the use of probabilistic data structures - such as Count-Min Sketch [5], Bloom Filters [11] or HyperLogLog [6]. While these structures are often used in the data plane and could be used in this case as well, we preferred to use a broader approach that could be used in conjunction for multiple purposes. We thus turn to BeauCoup [4] - a coupon collection [7] based system. While potentially less accurate, it provides the ability to approximate a distinct count that is within a predetermined margin of error from the actual result - which could be adjusted depending on the usage. In addition to providing a statistical approximation of the amount of distinct subdomains seen for each domain we could potentially collect more data for each DNS request without significantly affecting performance or the error rate, allowing future expansion.

Optimization of Coupon Collection We attempt to avoid cases where a small number of coupons are collected. Since we expect an attack to manifest with an extremely high number of requests [9] we are not interested in the estimation of domains with a small number of subdomains, and want to avoid the memory overhead of collecting coupons for them. We try to lower the probability of collecting the first coupon by introducing a coin flip following the first coupon collection, choosing to ignore the coupon with some probability. Further work is required here to asses how this affects the expected margin of error, and the approximate count we are trying to achieve.

Domain Statistics Additional data we can collect once we've decided to collect a coupon for a domain is total, *non-distinct*, count of requests, responses and NXDOMAIN responses. Since most domains do not have large amounts of subdomains, coupons are only expected to be collected for a small amount of domains. We allocate registers and store this data only after the first coupon has been collected for some domain. Once we have these counters we can generate alerts for domains presenting a large percentage of dropped requests (request with no response) or a large percentage of NXDOMAIN responses (out of the total responses received for the domain). We note that due to limitation of the data plane, percentages are calculated approximately. Additionally, while BeauCoup [4] may perform a single memory access per packet, WORD would need to perform additional memory accesses in order to maintain these statistics.

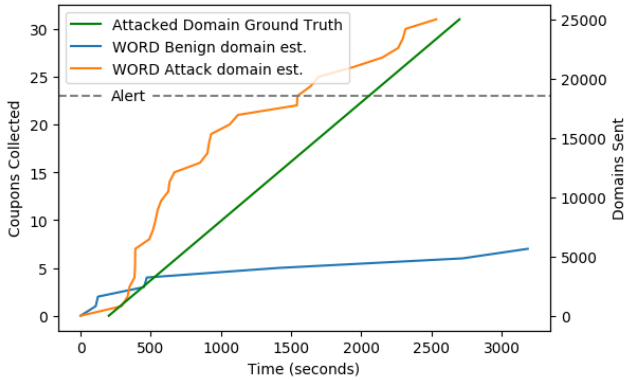

Figure 2: Coupons collected over time

3 EVALUATION

We implemented the DNS parsing and processing using P4, and ran the tests on a BMv2 switch. We set the parameters to process data in 1 hour windows while creating an alert after 23 coupons have been collected, approximating 10,000 distinct subdomains. The evaluation was done using a trace of DNS traffic captured in a university campus [10]. The trace contains a total of around 1.6 million DNS requests and responses generated by about 4000 users over the span of an hour. Together with the trace of benign traffic we used a script to simulate a DNS Water Torture attack featuring 25,000 randomly generated subomdains for a single domain.

Figure 2 shows the coupons collected for the targeted domain versus the maximum number of coupons collected for *all other* domains. Our alert threshold is reached when around 15,000 distinct queries are sent. While this is a fairly high error rate we do see the approximation trend is quite similar to the real trend. We note that the error rate is likely caused by the coin flip we introduced, a feature which we plan to compensate for in the future. On the other side we see that no more than 7 coupons are collected for the rest of the domains, for a trace containing approximately *10,000* distinct domains, out of which only *121* distinct domains collected at least one coupon.

4 FUTURE WORK

Using our simulated attack data we show an ability to concisely detect attack traffic in the data plane. In the future we plan to implement our system on a hardware switch and perform further analysis using real attack traces. In addition we intend to explore the possibility of mitigating this attack directly in the data plane. We plan to expand this research to other attacks, such as NXNSAttack [1], as well as detecting attempts to perform subdomain enumeration.

REFERENCES

[1] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. NXNSAttack: Recursive DNS inefficiencies and vulnerabilities. In *USENIX Security*, pages 631–648, 2020.
[2] Akamai. Whitepaper: Dns reflection, amplification, dns water-torture.
[3] APNIC. Dnssec validation rate by country.
[4] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. In *ACM SIGCOMM*, pages 226–239, 2020.

[5] Graham Cormode and Shan Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *Journal of Algorithms*, 2005.
[6] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156, 2007.
[7] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.
[8] Mozilla Foundation. Public suffix list.
[9] Xi Luo, Liming Wang, Zhen Xu, Kai Chen, Jing Yang, and Tian Tian. A large scale analysis of dns water torture attack. In *International Conference on Computer Science and Artificial Intelligence*, pages 168–173, 2018.
[10] Manmeet Singh. 10 days dns network traffic from april-may, 2016, 2019.
[11] Peng Xiao, Zhiyang Li, Heng Qi, Wenyu Qu, and Haisheng Yu. An efficient ddos detection with bloom filter in sdn. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1–6. IEEE, 2016.
[12] Petr Špaček. NXNSAttack: Upgrade resolvers to stop new kind of random subdomain attack. 2020.