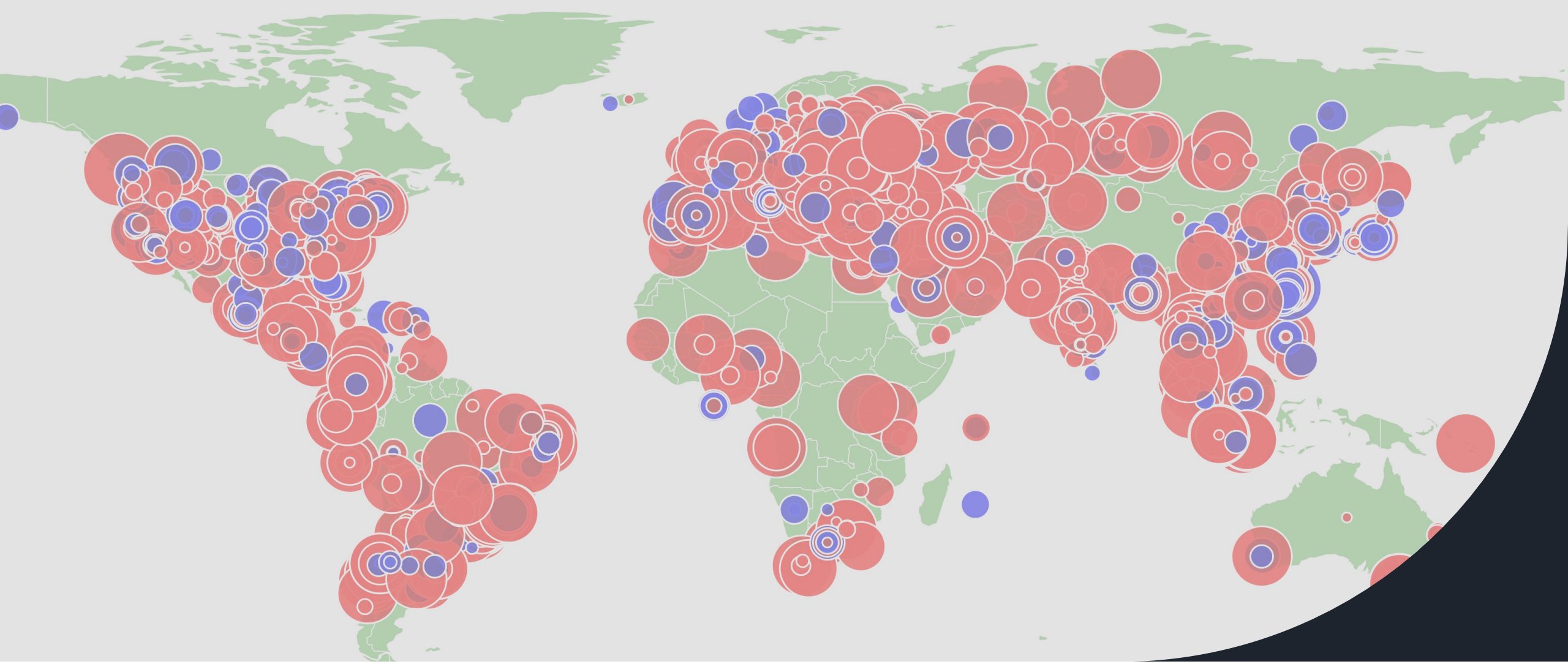
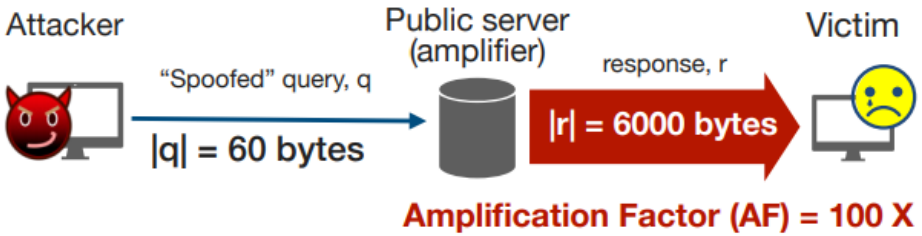


Accurately Measuring Global Risk of Amplification Attacks

Nischith Shadagopan M N

What is an amplification attack?

- In an amplification attack, the attacker spoofs a victim's IP address and sends queries to a server.
- These queries are chosen is such a way so as to induce a large response from the server to the victim.
- This leads to Distributed Denial of Service (DDoS) attacks.
- All stateless protocols are susceptible to such attacks



Problem Statement

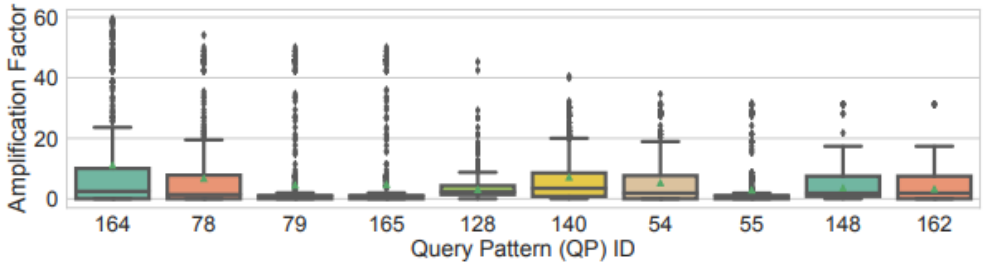
- This paper aims to solve the problem of finding a way to proactively map the risk of amplification attacks across servers and protocols with a small network footprint.
- Solving this problem helps network operators and Internet security experts assess the risk of amplification attacks. This helps in pointing out the high risk attacks and work on fixing those issues
- This paper also helps in identifying amplification inducing query patterns and also their risk. With this information network operators can proactively take defensive measures. It can also guide protocol designers in designing future protocols and also assess the effectiveness of defenses.

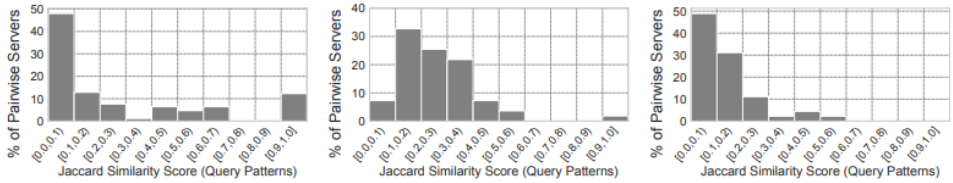
Contemporary solutions

- Solution 1 : We can scan for the number of open servers in the network and multiply an empirically chosen Amplification Factor (AF henceforth) with the number of servers to quantify the risk. This assumes a constant AF for all servers and protocols.
- Solution 2 : Solution 1 doesn't account for different patterns having different AF. We can probe the servers with known patterns to assess amplification risk and construct signatures.
- Solution 3 : Solution 2 doesn't take into consideration the change in amplification of a pattern across servers. We can save some probes by sending the query pattern to only one server out of a group having the same software setup.

Problems with existing solutions

- A small scale measurement study was conducted using DNS and it was found that these solutions misestimate amplification risk and also miss many amplification inducing query patterns.
- Amplification factor varies across servers for the same protocol. This breaks the assumption of S1.
- Query patterns which induce high amplification varies from server to server. This breaks the assumptions of S2 and S3.



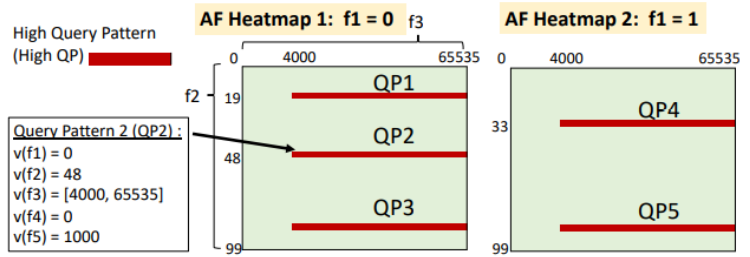


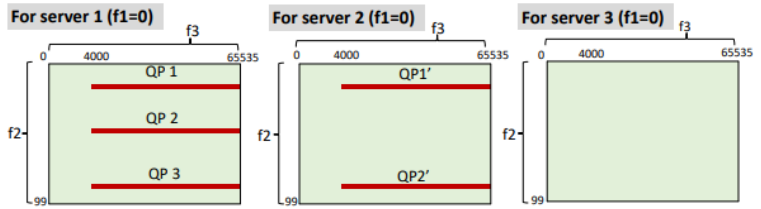
(a) Microsoft 6.1 (b) Dnsmasq 2.52 (c) Dnsmasq 2.40

Challenges

- Most protocols have a very large header space
- Amplification inducing query patterns have complex relations between the various fields
- As seen before the amplification induced by a pattern depends on the server and also the set of amplification inducing patterns varies from server to server

Fields: $F = \{f_1, f_2, f_3, f_4, f_5\}$
Accepted values for each field: $AV(f_i)$
1. f_1 takes a value from 0 to 1; $AV(f_1) = [0, 1]$
2. f_2 takes a value from 0 to 99; $AV(f_2) = [0, 99]$
3. f_3 takes a value from 0 to 65535; $AV(f_3) = [0, 65535]$
4. f_4 takes a value from 0 to 7; $AV(f_4) = [0, 7]$
5. f_5 takes a value from 0 to 1; $AV(f_5) = [0, 1]$





Some insights to keep in mind

- Amplification inducing query patterns exhibit locality. That is if we find one particular query pattern that induces amplification, we can find other such patterns by changing a few fields.
- If the percentage of amplification inducing queries is e, then we can find one such pattern in roughly 1/e trials.
- Fields with a very large range of values usually do not affect amplification. Even if they do it is usually in a range.
- Though servers don't share many query patterns, there is some commonalities which can be exploited.

Single server algorithm

- The algorithm runs in two stages
- Random sampling stage : In this stage we sample random queries to discover a high amplification query pattern.
- Per field search : In this stage we try to find similar pattern to a give query by varying the fields. It uses a log sampling for large fields and exhaustive search for smaller ones.

Algorithm 1: Per-Field Search

```
1  Function PerFieldSearch(QtoAF, Q^start, AF^thresh):
2      Q^explore = {Q^start}; PatternsFound = {}
3      while Q^explore is not empty do
4          q ← Extract from Q^explore
5          if IsNewPattern(q.pattern , PatternsFound) then
               /* Search neighbors for a new pattern   */
6              PatternsFound.insert(q.pattern)
7              tmpQtoAF = SearchNeighbor(q, AF^thresh)
8              QtoAF.insert(QtoAF^neighbor)
9              Q^explore = Q^explore ∪ tmpQtoAF.keys()
10         else
               /* if not new, skip exploration        */
11             MergeQueries(q.pattern, PattersFound)
12     return QtoAF
```

Multi server algorithm

- The main addition from before is the Probing stage. In this stage some of the queries found for a server in the random sampling stage are also tried with other servers. This increases the chances of finding a good starting set of queries

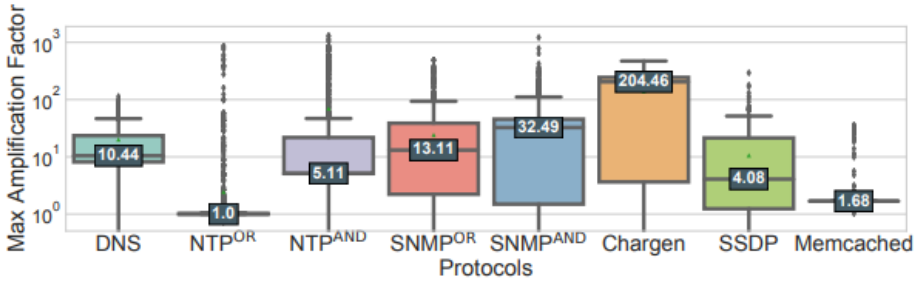
Algorithm 2: AmpMap algorithm for multiple servers

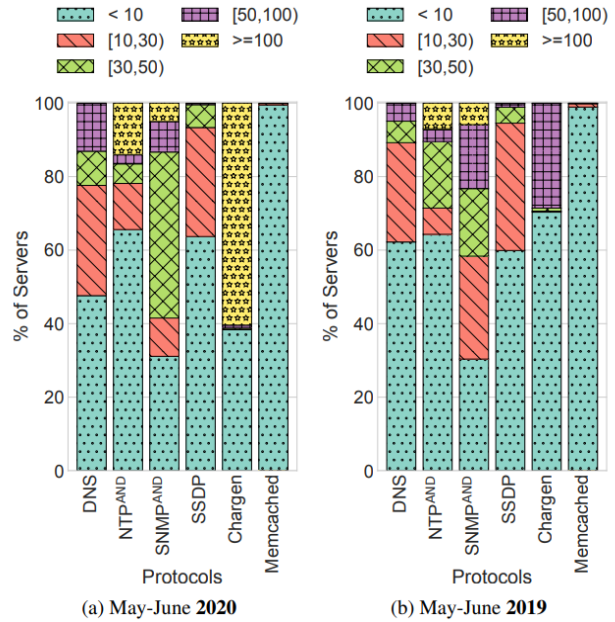
Input: B_{total}: query budget
$AV(f_i)$ for $i = 1,..,n$: accepted value for each packet field
S: a set of servers
Output: PerServerQToAF : maps each query to corresponding AF

1 $PerServerQToAF = \{\}$ /* Step 1: Random Search */
2 **for** $s \in ServerSet$ **do**
3 RunRandomUpdateMap(B_{rand},$PerServerQToAF[s]$)
/* Step 2: Pick probes based on current obs. */
4 Q^{probe} = PickProbes($PerServerQtoAF$, B_{probe})
/* Run additional probes per server */
5 **for** $s \in S$ **do**
6 $ProbeQToAF_s$ = SendQuery(Q^{probe})
7 $PerServerQToAF[s]$.insert($ProbeQToAF_s$)
/* Step 3: Per-field search for each server */
8 **for** $s \in S$ **do**
9 Q_s^{start} = FindTopKQueries($PerServerQToAF[s]$,K)
10 AF^{thresh} = ComputeThresh($PerServerQToAF[s]$)
11 PerFieldSearch($PerServerQToAF[s]$, Q_s^{start}, AF^{thresh})
12 **return** $PerServerQToAF$

Results

- Maximum amplification factor varies significantly from server to server
- Maximum amplification factor distribution varies from server to server
- Maximum amplification factor distribution also varies with time

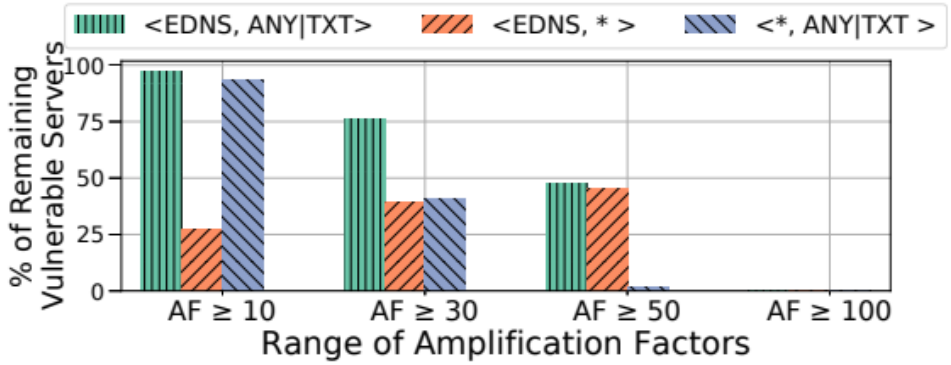




Assessing amplification risk

- Even for known patterns, prior work misestimates the amplification risk
- Prior work misses out on many query patterns that induce amplification and hence even if the known patterns are mitigated, there is substantial residual risk

	Known Pattern	Risk Quantification		Results
		Prior Work	AmpMap	
DNS	EDNS:0,ANY [1,57]	287K	149K	1.9× ↑
	EDNS:0,ANY,TXT [57,62]	Unknown	183K	N/A
DNS (domains w/o DNSSEC)	ANY,TXT [57,62]	Unknown	126K	N/A
NTP OR	monlist [2,57]	5,569K	13K	427× ↑
NTP AND	monlist [2,57]	5,569K	635K 12	8.8× ↑
SNMP OR	*GetBulk* [3,57]	64K	223K	3.5× ↓
SNMP AND	*GetBulk* [3,57]	64K	317K	5× ↓
Chargen	Request	3588K	1399K	2.9× ↑
SSDP	Search [3,57]	308K	126K	2.7× ↑
Memcached	Stats [3,17]	100M [3]	18K	5.6K × ↑



Limitations and critque

- Ampmap currently only supports some UDP based protocols. It needs to be extended to work on many other protocols including TCP based protocols.
- I feel this approach of finding amplification query patterns is a more practical way of dealing with the problem than finding bugs in protocols.
- I feel AI can be used to help in finding amplification inducing query patterns. AI can also be used for parameter tuning.
- Ampmap has currently been tested on a few servers but it needs to be scaled to be used on the whole internet.

Limitations and critique

- Ampmap currently assumes the knowledge of the protocol format but sometimes this knowledge can be private and hence Ampmap can benefit from protocol inferencing methods.
- This work is radically different from prior approaches. First to automatically identify amplification inducing query patterns and also to assess their risk. What I liked the most is that this is a very proactive approach and it is continous.

References

- Soo-Jin Moon, Yucheng Yin, Rahul Anand Sharma, Yifei Yuan, Jonathan M. Spring, and Vyas Sekar. Accurately measuring global risk of amplification attacks using ampmap. In 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, August 2021
- link