Defending Against Bandwidth-Flooding Attacks

Dimitrios Melissovas
I&C, EPFL

Abstract—Bandwidth-flooding attacks are a type of Denial of Service attacks that target the victim’s tail-circuit. The stateless approach that the Internet takes with respect to routers makes launching such attacks, for a well-large enough group of attackers (or a botnet) a trivial task to undertake. Despite the fact that DDoS have garnered attention from both the research community and popular media outlets over the last decade, the threshold for launching a successful network-flooding attack remains frustratingly low. Through the selected papers, we explore three different approaches that we can use to mitigate bandwidth-flooding attacks: filtering malicious sources, enforcing admission control and regulating access to the common medium.

Index Terms—Internet, network security, filtering, capabilities

I. INTRODUCTION

I
n a Distributed Denial-of-Service attack (DDoS), a large number of hosts flood a victim with the goal of exhausting a bottleneck resource and cause significant performance degradation. A host may participate in a DDoS attack either voluntarily, or as a part of a botnet that consists of several compromised hosts. Bandwidth-flooding attacks are a type of DDoS attacks which target the victim’s tail circuit specifically with the intent of causing congestion and packet loss.

One of the first well-documented incidents of bandwidth-flooding attacks was the DDoS attack against the Gibson Research Corporation in 2001 [1]. Ever since, such attacks have grown both in terms of intensity and in terms of prominence in the news outlets. One of the most recent examples is an attack on Wikileaks, a popular whistle-blowing website [2]. The attack rendered the website inaccessible for more than a week.

There are two qualities that set bandwidth-flooding attacks apart from other categories of DDoS attacks. First, bandwidth-flooding attacks do not have to exploit any sort of victim-specific vulnerability, such as software bugs. Second, a victim cannot rely on local actions (e.g. installing a firewall rule) in order to mitigate a bandwidth-flooding attack. By the time the victim is able to take any action, the attacker packet has traversed the tail circuit and, thus, has already achieved its purpose. Therefore, a victim should, at the very least, request assistance from their Internet Service Provider (ISP) in order to mitigate a bandwidth-flooding attack.

The fact that the network has been designed to be flow-agnostic is the main reason that every host on the Internet is susceptible to bandwidth-flooding attacks. The rationale behind this is that it reduces both router complexity, and the amount of state that needs to be maintained. Routers, therefore, will only forward packets and propagate routing information. Instead, the responsibility for sharing the access medium fairly among competing flows has been delegated to the transport layer. Thus, when congestion occurs, a legitimate user will observe a packet-loss event and reduce their sending rate in order to ease congestion in the network. An attacking host, which has no incentive to comply, will continue flooding the victim at the maximum possible sending rate. An attacker may even take advantage of the exponential back-off that TCP uses in order to reduce the goodput of legitimate TCP flows to 0 [3].

In today’s Internet, an attack victim could provide their ISP with a list of IP addresses they consider to be malicious. The ISP, then, install filters and block unwanted traffic before it reaches the tail circuit and, thus, protect the victim. In the worst case, all clients of a particular ISP can simultaneously report that they are under attack, each from a different set of IP addresses.

The ISP must block all reported flows, since they are undesirable, but it should avoid blocking unreported flows1. The ISP must, thus, provision for a number of filters that is quadratic to the number of hosts on the Internet. If an ISP does not have sufficient resources to block each individual flow, they can use aggregate filters in order to block entire network prefixes. However, the negative effects of collateral damage due to filter aggregation may cause more damage to

I

1For instance, assume that A has made a request that it wishes C to be blocked, whereas B has not made any filtering request about C. The ISP must block flow (C, A) but it should not block flow (C, B). The reason being that A’s filtering request may be malicious, whereas C may be innocent.
the ISP and its clients than the attack itself.

It is unprofitable for a single ISP to overprovision for the worst-case scenario. A router must be able to determine at wire-speed whether a packet belongs to an unwanted flow. Thus, filters should be stored in the blocking router’s TCAM (Ternary Content Addressable Memory). Unfortunately, TCAM space is a scarce resource in routers, since TCAM is expensive, bulky and consumes considerably more energy than conventional DRAM. Given the relative rareness of massive DDoS attacks, the current pricing for TCAM memory and the per-flow state that a perfect defense would require, it may not be worth the return in investment for the ISP to protect all of its clients. At the same time, if an attack does happen, it may impossible for a victim and their ISP to recover from the damage done to their credibility.

Finally, source address spoofing further increases the difficulty of mitigating DDoS attacks in the Internet that we have. An attacker may spoof their source address in order to bypass any blacklist-based defenses, or turn them against the victim. For instance, a sophisticated attacker can impersonate legitimate users in order to implicate them in the attack and have them blacklisted. The net effects of such an attack on the victim are similar to filter aggregation; the victim shuts itself off from legitimate users.

Each of the forementioned issues is evidence to the fact that a victim cannot rely on their ISP in order to mitigate DDoS attacks effectively. Thus, a more comprehensive solution can only come through cooperation with other network elements.

II. PAPER REVIEW

Now let us examine three different solutions to the problem of mitigating DDoS attacks. The first paper describes AITF, a protocol which enables ISPs to cooperate with one another in order to avoid TCAM resource exhaustion. The second paper, SIFF, introduces capabilities. Capabilities are a cookie-like mechanism that minimises the amount of state in the network. SIFF requires that senders obtain explicit authorisation from the receiver before they are allowed to send traffic. The last paper, Portcullis, addresses one of the core problems of all capability-based solutions. It uses cryptographic proof-of-work puzzles in order to regulate access and enforce fairness in the connection setup channel.

An assumption that AITF and SIFF make is that a victim will eventually be able to detect most, if not all, malicious sources of traffic, in order to take action against them.

In order to make the text more concise and readable, we are going to use the following abbreviations for all three papers:

- R: Receiver
- S: Sender
- Rgw: Border router in R’s ISP, which connects the ISP to R’s tail circuit.
- Sgw: Border router in S’s ISP, which connects the ISP S’s tail circuit.
- Mi: Intermediate router at the i-th hop of the path from S to R. Sgw is M1 by default.

A. Scalable Network-layer Defense Against Internet Bandwidth-Flooding attacks

This paper describes Active Internet Traffic Filtering (AITF), a filter-based protocol which uses ISP collaboration in order to significantly reduce TCAM space requirements for filtering. In doing so, this paper demonstrates that filter-based approaches are both practical and scalable.

As mentioned previously, the victim’s gateway, Rgw, is unable to block all unwanted flows effectively, if it receives no help from the network. Once the botnet exhausts Rgw’s TCAM resources and, thus, its ability to block new sources of attack traffic, Rgw will have to start using filter aggregation. Although this strategy enables Rgw to block all sources of attack traffic, it introduces collateral damage, which is equivalent to punishing the victim of the attack.

AITF, instead, punishes malicious sources of traffic by leveraging cooperation among ISPs. AITF’s architecture enables wire-speed filters in expensive TCAM to be supplanted by memory log entries in conventional DRAM. Sgw, the gateway of a compromised host, S, will disconnect S from the network if S does not cooperate. Disconnecting S from the network requires only one single wire-speed filter at Sgw, regardless of how many hosts S is attacking at the same time. Compare this to the solution where each ISP must maintain a filter on behalf of its customers, if ISPs are not willing to cooperate.

A receiver, R, should be able to decide, at any point in time, that it does not wish to receive traffic from a sender, S. In AITF, R makes a filtering request to S and asks it to stop sending traffic for Wf seconds. In the meantime, both Rgw and Sgw intercept and process the filtering request. Sgw, in particular, records the filtering request in DRAM for future reference, and makes sure that S receives the request (e.g. by prioritising the filtering request over other packets). We refer to the memory entry as a Shadow Filter for brevity. If S does not cooperate and sends another packet to the victim within Wf seconds from the first request, R will send another filtering request. This time, however, Sgw will retrieve the previous Shadow Filter in DRAM and discover that S is a non-cooperating host. Sgw will, then, punish S by disconnecting it from the network for Wf seconds. Sgw is able to log Shadow Filters in DRAM because we expect the rate of filtering requests to be much lower than the line rate.

ISP collaboration and host punishment present AITF with its main two challenges. First, AITF must provide incentives for Sgw to deploy the protocol and punish S, if S does not cooperate. From Sgw’s perspective, it is not reasonable to sacrifice the user experience of one of its clients, simply because another host in the Internet made a filtering request. Second, AITF must ensure that the filtering mechanism does not, itself, get abused. S may i) accuse R of non-cooperation 3

3Wf is referred to as the filtering window. Wf should be large enough to make filtering requests worthwhile. Nevertheless, Wf should be small enough to allow a user to patch their machine and reconnect to the network. The paper proposes 10 minutes as a token value for Wf.

4The authors argue that each ISP can implement their own punishment policy, provided that they have sufficient resources to enforce it. Sgw could, for instance, use filter aggregation for S in order to make it make the state fit in the router’s TCAM. Nevertheless, Sgw must ensure that S is not able to send traffic to R for the following Wf seconds.

5Wf is referred to as the filtering window. Wf should be large enough to make filtering requests worthwhile. Nevertheless, Wf should be small enough to allow a user to patch their machine and reconnect to the network. The paper proposes 10 minutes as a token value for Wf.

6The authors argue that each ISP can implement their own punishment policy, provided that they have sufficient resources to enforce it. Sgw could, for instance, use filter aggregation for S in order to make it make the state fit in the router’s TCAM. Nevertheless, Sgw must ensure that S is not able to send traffic to R for the following Wf seconds.
in order to have them disconnected from the network. $S$ could also ii) fake filtering requests from other hosts. Moreover, $S$ could iii) spoof its IP address and attack a random host in the network, in order to implicate $R$ in the attack. Finally, $S$ may simply iv) flood the filtering mechanism with bogus filtering requests in order to disable it. Let’s see how AITF deals with each case separately.

Let us assume that $S_{gw}$ is not willing to cooperate (e.g. $S_{gw}$ does not use AITF) and does not filter $S$’s packets to $R$. $R_{gw}$ will use filter aggregation and block all traffic from $S_{gw}$ to $R$, much like non-collaborative filtering at the ISP level. AITF, in particular proposes that $R_{gw}$ uses escalation. That is to say, $R_{gw}$ should deal with $S_{gw}$ lack of cooperation in a recursive fashion. Assume that $S'_{gw}$ is $S_{gw}$’s upstream provider on the path from $S$ to $R$. In that case, $R_{gw}$ treats $S_{gw}$ as the attacker and $S'_{gw}$ as the attacker’s gateway and makes a filtering request to $S'_{gw}$ to block all attack traffic from $S_{gw}$ to $R$. The threat of losing all access to $R$ will motivate $S_{gw}$ to deploy AITF and punish its misbehaving clients.

$S_{gw}$ must verify independently that $S$ is indeed sending traffic to $R$, before it disconnects $S$. Otherwise, a host that creates bogus filtering requests may disconnect any and all hosts from the Internet. Upon intercepting the first filtering request from $R$, $S_{gw}$ starts monitoring $S$’s outgoing traffic. $S_{gw}$ will keep a record of each destination that $S$ has contacted in the past $W_m$ seconds. We refer to the structure that maintains these monitoring records as the Flow Cache. Once $S_{gw}$ intercepts $R$’s second filtering request, it examines whether the Flow Cache contains flow entry $(S, R)$. If this is the case, $S_{gw}$ will punish $S$. To put things into perspective, a Shadow Filter entry for $(S, R)$ proves that $S$ has already been warned once, whereas a Flow Cache entry for $(S, R)$ proves that $S$ has sent a packet to $R$ recently. $S_{gw}$ will punish $S$ iff both a Shadow Filter entry and a Flow Cache entry exist for $(S, R)$ when it receives the filtering request.

AITF uses a stateless three-way handshake to prevent hosts from faking filtering requests. Upon intercepting $R$’s filtering request, $F_{req}$, $S_{gw}$ will send a proof request packet to $R$ that contains $F_{req}$. This proof request packet contains a cookie, $F_{cookie}$, which should prevent attackers from guessing the contents of the packet. $F_{cookie}$ is a cryptographic hash of $R$’s address, with a key known only to $S_{gw}$. $R_{gw}$ intercepts the proof request packet before it reaches $R$, and sends a proof response packet to $S$ that contains $F_{req}$ and $F_{cookie}$. $S_{gw}$ intercepts the proof response packet, verifies that $F_{cookie}$ is present and processes $F_{req}$ only if the node on which the path between $R_{gw}$ and $S_{gw}$ can intercept the cookie and, thus, able to complete the handshake. It is worth noting that the three-way handshake does not traverse the tail circuit of the receiver in the direction of the flood from $R_{gw}$ to $R$. If this were not the case, $S_{gw}$’s proof request packet would most likely get lost due to congestion at the tail circuit, and the handshake would not get completed.

AITF uses a simple path identification mechanism, in order to prevent hosts from spoofing their source address. $S_{gw}$ generates a stamp and sends it to $R_{gw}$. $S_{gw}$, then, applies that

\[ W_m \text{ is referred to as the monitoring window.} \]
The paper does not elaborate how AITF addresses issues related to flow monitoring. Recall that $S_{gw}$ must track which hosts $S$ has sent packets to in the past $W_{in}$ seconds in a Flow Cache. This is per-flow state which must be maintained in TCAM, since consecutive packets may correspond to previously unseen destinations. Depending on how this is implemented, the Flow Cache mechanism can yield false negatives (e.g. if we use sampling), or false positives (e.g. if we use Bloom Filters). Thus, the Flow Cache could be a possible avenue of attack.

One of the main benefits of filter-based solutions, such as AITF, is that they are reactive and do not introduce overhead to the network if they are not invoked. Moreover, such mechanisms require only minimal modifications to hosts’ network stacks in order to work (e.g. firewall rules to ensure compliance to filtering requests). These modifications do not challenge any well-established concepts, such as connection-oriented or datagram traffic. This is not true for other defense mechanisms, such as SIFF, which is the subject of the following paper. Finally, although there exist some resource exhaustion attacks against AITF, which we have described, they require a certain level of sophistication on the attacker’s behalf. Furthermore, an attacker may have to dedicate a significant portion of their attack traffic in order to avoid punishment. This would diminish the intensity of the flooding attack on the victim, which is precisely the goal of DDoS attack mitigation mechanisms.

B. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks

Filtering requires per-flow state at routers (either in TCAM or in DRAM, as we saw in AITF), in order to block undesired traffic. The main reason behind this is that a host has no incentive to inform the network that its packets must be blocked (referred to as setting the evil bit in RFC 3514 [4]), since the network assumes that all flows are desirable by default. SIFF, instead, approaches this matter differently; a flow is undesirable unless proven otherwise. The main contribution of SIFF is that it carries is identical to the capability token that $M_i$ stamped during the handshake protocol. If this is true, it proves that the packet belongs to a privileged flow and $M_i$ will forward the packet. In order for capabilities to work, tokens should be tied down to a specific flow and must be made difficult to guess. Furthermore, capability tokens should expire, given that $R$ cannot revoke authorisation from $S$. Otherwise, $S$ would simply have to acquire authorisation from $R$ once, in order to bypass the defense mechanism altogether.

In SIFF, a router uses a keyed hash function with a slowly changing secret key to generate capability tokens for flows, which addresses all aforementioned challenges. $M_i$ computes capability $C_i$, which is equal to the last $Z$ bits of $\text{hash}_{\text{key}}(S||R)||R||M_{i-1}||R_{gw})$, where $\text{hash}_{\text{key}}$ is a keyed hash function, $Z$ is a configuration parameter and $\text{key}$ is a secret known only to $M_i$. The parameters of the hash function tie the capability down to a specific flow. $M_i$ can verify that a data packet is valid by re-computing the output of the hash function and comparing it to the $i$-th capability of the data packet.

$M_i$ generates a new key every $T$ seconds and keeps a window of $X \geq 2$ keys valid at any one time. Upon receiving an old but valid capability (i.e. the capability is valid for one of these $X$ secret keys, but not for the most recent one), $M_i$ will stamp an updated capability token on the data packet. If $R$ wishes to extend $S$’s authorisation period, $R$ must send the updated capability tokens to $S$. Otherwise, after sufficient time elapses, $S$ will be unable to use the privileged channel to send traffic to $R$. It should be noted that the capability refresh mechanism only uses privileged traffic and, thus, is more resilient to packet flooding attacks.

Much of the evaluation in the paper is an analysis of the effects of different values for $Z$ (number of bits in a capability token) and $X$ (the number of keys that are considered valid) on capability-guessing attacks. A larger value for $Z$ makes it more difficult for the attacker to guess a valid capability. At the same time it increases the marking space requirements for packets, since each packet carries $\#\text{hops}$ capabilities. A large value for $X$ makes it easier for an attacker to guess a valid capability (a capability is considered valid if any of the $X$ available keys produce the same hash function output). However a larger $X$ evens out the difference between the minimum and

5To be precise, $C_i$ is equal to the last $Z$ bits of $\text{hash}_{\text{key}}(S||R)||M_{i-1}||R_{gw})$, where $M_{i-1}$ is the previous hop and $R_{gw}$ is the last hop before the receiver.
the maximum number of seconds for which a capability is considered valid \((\text{max}_k = X \times T\) and \(\text{min}_k = (X - 1) \times T\). According to the authors, the probability that a randomly guessed capability will pass a particular router is:

\[
P(X, Z) = 1 - \left(1 - \frac{1}{2^X}\right)^X
\]

This is not entirely true, however. A capability token can carry an additional \(\log_2 X\) bits as a timestamp value, which explicitly describes which of the \(X\) available router keys was used. This scheme also reduces computational strain on routers; a router needs only evaluate the output of one hash function in order to verify that a capability is valid.

In fact, from a victim’s point of view, it makes sense to pick a large value for \(X\). A capability must be valid for at least \(RTT_{\text{max}}\) seconds, where \(RTT_{\text{max}}\) is the longest round trip time possible in the network. Therefore:

\[
\text{min}_k = (X - 1) \times T = RTT_{\text{max}}
\]

Moreover, \(R\) is unable to revoke capabilities from \(S\). Thus, if \(S\) is malicious it can flood the privileged channel for the duration that the capability is still valid. In the worst case, \(S\) can flood \(R\) for \(\text{max}_k\) seconds, where:

\[
\text{max}_k = X \times T = \text{min}_k + T = RTT_{\text{max}} + T
\]

Therefore, a large value for \(X\) will be offset by a small value for \(T\) and minimise the amount of time for which a receiver becomes vulnerable after authorising a capability request.

Arguably, the biggest drawback of SIFF, is that it is susceptible to flooding attacks against the capability request mechanism. A powerful enough botnet can generate a large number of capability requests and prevent legitimate hosts from ever connecting. This is the fundamental problem of all capability-based solution and the attack has been aptly named a Denial of Capability attack [5]. We defer a full discussion on DoC attacks to the summary of the last paper, Portcullis, which describes a potential solution. Now, let’s analyse some of the lesser issues in SIFF.

SIFF requires end-host modifications to work, which carry an influence to the design of several aspects of the network stack. If deployed, we assume that SIFF will most likely be implemented in a shim layer between the Transport Layer and the Network Layer at the network stack. This intermediate layer should handle sending capability request messages during connection setup and capability refresh messages during long periods of inactivity, in order to keep a connection alive. This may cause connectionless protocols to lose their utility and appeal, given that the sender must maintain some connection state in order to use the privileged channel.

Finally, a connection authorisation in SIFF is tied down to a particular path. If a flow gets forced to follow a different path (e.g. due to load balancing), \(S\) must use the handshake protocol again, in order to acquire authorisation for the new path. SIFF can avoid this hassle, if the only routers that validate capabilities are the ones that are guaranteed to be present in all paths from \(S\) to \(R\), such as \(S_{\text{gw}}\) and \(R_{\text{gw}}\).

As we discussed, it might be difficult for a legitimate user to complete the handshake protocol with the victim, while the victim is under attack. Nevertheless, once the user does manage to connect to the victim, the users session will no longer get disturbed by the attack. Another benefit of SIFF is that it is reasonably resilient to address spoofing attacks. Finally, a server is better able to do admission control by withholding capabilities from new users, if it experiences a benevolent flash crowd (sometimes referred to as the Slashdot Effect). SIFF yields these benefits while remaining stateless and requiring no inter-ISP collaboration.

C. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks

Despite their merits, capability-based solutions are inherently susceptible to Denial-of-Capability (DoC) attacks [5], which are DoS attacks against the capability request mechanism. What makes matters worse is that attackers will send request packets to the victims much more aggressively than legitimate users. Indeed, a legitimate user may send one request packet per round trip time at most, and could even back off if its requests are unsuccessful. Unlike data traffic, however, throughput (i.e. how many capability requests per second a sender is able to make) is only secondary when it comes to the request channel. Instead, the only performance metric of the request channel that we are interested in is connection setup delay.

Therefore, Access to the resource channel is a resource which we can regulate access to. Unfortunately, we cannot use a per-source fair sharing scheme, since that requires a significant amount of state stored in router TCAM. Instead, we could do fair sharing at a coarser granularity, for instance, per source domain. This, however, introduces collateral damage, since compromised hosts will still overpower legitimate users that reside in the same domain.

Portcullis takes a different approach and uses proof-of-work cryptographic puzzles in order to force hosts to share the request channel fairly. The rationale is that there is the disparity in access to computational power is much less than disparity in access to other kinds of resources among different hosts. Portcullis guarantees a maximum connection setup delay, which is a function of the aggregate computational power of the botnet. The authors argue that we can use the DNS infrastructure as a puzzle distribution platform, given that it is already well provisioned, distributed and, thus, resilient to DDoS attacks. The main challenges that Portcullis faces are securing the puzzle distribution protocol and the solution verification mechanism from abuse.

Portcullis assumes that the DNS root (or another centralised entity) generates and distributes puzzle seeds to the network. The DNS root generates a random value, \(h_0\), and hashes it repeatedly, using a secure hash function, in order to create the series of puzzle seeds, using the following method:

\[
h_i = \text{hash}(h_{i-1})
\]

Once the DNS root computes the \(n\)-th puzzle seed (referred to as the puzzle anchor), it starts releasing puzzle seeds, in the reverse order from which they were created, starting from the puzzle anchor first. Thus, the
DNS root will release seed $h_i$ before it releases seed $h_{i-1}$. The DNS root signs the puzzle anchor with its private key before announcing it to the network. By construction, a router, $M$, is always able to certify that a puzzle seed $h_i$ is valid. If $i = n$ (this is the puzzle anchor), $M$ can verify that the seed is authentic using the signature. Otherwise, the router can repeat the steps used during seed generation and verify that $h_i$ yields $h_n$ if hashed $n - i$ times.

A legitimate sender, $S$, must first obtain the most recent puzzle seed, $h_i$ from the DNS infrastructure, in order to connect to $R$. $S$ generates and solves a random puzzle, $P$ with a desired difficulty level, $l$, from $h_i$. Then, $S$ sends a capability request packet that contains both the puzzle, $P$, and the solution to $P$, $X$. Upon receiving the request packet, an on-path router, $M$, validates that $X$ is a solution to $P$ and forwards the request to the next hop. At the same time, $M$ will also record the puzzle that $S$ used in order to prevent collaborating hosts from sharing and reusing puzzle solutions.

Higher difficulty puzzles require more computation cycles to solve than lower difficulty puzzles. At the same time, however, packets that contain solutions to higher difficulty puzzles will receive higher priority from an intermediate router $M$, if $M$ becomes congested. The intuition behind this design choice is that a botnet has bounded computational power and can only become congested. The intuition behind this design choice is that a botnet has bounded computational power and can only become congested. The intuition behind this design choice is that a botnet has bounded computational power and can only become congested. The intuition behind this design choice is that a botnet has bounded computational power and can only become congested. The intuition behind this design choice is that a botnet has bounded computational power and can only become congested.

Thus, once $S$ backs off a sufficient number of times, it will solve a puzzle with difficulty level $h_{bot} + 1$ and it will manage to connect to $R$.

$S$, in particular, must find a solution $X$ for the puzzle, such that the last $l$ bits of $\text{hash}(X||r||h_i||l||R)$ are all zero, where $\text{hash}$ is a secure hash function known, $r$ is a random 64-bit nonce that $S$ generates, $h_i$ is the latest known puzzle seed and $R$ is the receiver address. $S$ can only use a brute force strategy to find a solution that satisfies the puzzle. Thus, by construction, a puzzle with difficulty level $l$ requires two times more computation cycles to solve than a puzzle with difficulty level $l - 1$. $S$ will send $P = r||h_i||l||R$ as the puzzle and $X$ as the solution. Since routers drop duplicate packets, senders are encouraged to pick $r$ at random.

However, a router, $M$, cannot record each individual puzzle that it has verified. This would, otherwise, amount to $r_{max} \times$ per-flow entries in the worst case, where $r_{max}$ is the maximum value attainable by $r$. Thus, the shear amount of state would defeat the purpose of using capabilities (i.e. offloading state from routers to senders and data packets). Portcullis, instead, uses Bloom Filters [6] to prevent puzzle reuse. Thus, $M$ will do a lookup on the Bloom Filter structure to check whether someone has already solved puzzle $P = r||h_i||l||R$. If not, $M$ will verify that the solution is valid and insert $P$ to the Bloom Filter.

By using Bloom Filters, however, Portcullis introduces false positives to puzzle verification. The rate of false positives in Bloom Filters is a function of the amount of memory allocated to the structure and the number of inserts. The authors argue that with a quota of 300 KB assigned to a Bloom Filter and 80000 requests/sec, the rate of false positives is approximately $\frac{F}{IPR}$ per request, over the period of one second. The authors use a circular buffer of $F$ Bloom Filters (wiping a Bloom Filter is the only way to delete entries properly) in order to cover a period of $F$ seconds. $M$ considers that $P$ has been solved before, if an entry for $P$ can be found in one of the $F$ available buffers. In this case, this amounts to $300 \times F$ KB worth of state for a false positive rate which is roughly equal to $\frac{F}{IPR}$ for $F$ seconds.

The authors provide analytical proof that the connection setup delay for each user, regardless of the strategy that the attacker uses, is $O(n_m)$, where $n_m$ is the aggregate computational power of the botnet. The assumption here is that the botnet cannot predict the exact timing of a user’s requests. Thus, high priority request bursts (i.e. requests that carry a solution to a high-difficulty puzzle) have a chance of missing the user’s request packet, which allows a user to connect. The authors also use simulation to demonstrate the effectiveness of various attacker strategies. The simulation results demonstrate that the best attacker strategy is to create just enough packets with the maximum possible priority level in order to saturate the victim’s tail circuit.

Although Portcullis guarantees that connection setup delay is bounded, it may still be significant. Users that become frustrated at long connection setup delays may use the services of the victim’s competitors.

Moreover, the puzzle distribution mechanism, itself, may become a target for DoS attacks. The DNS root must keep its resources to pace with attacker resource growth. This centralised puzzle generation mechanism is also a single point of failure from a security point of view. For instance, if an attacker manages to compromise the private key that the DNS root uses to sign puzzle anchors, they can propagate puzzle seeds which are incompatible to one another, similar to a DNS cache poisoning attacks [7].

Finally, the attacker may try to abuse the replay protection mechanism in order to block legitimate requests. The fact that the puzzle verification mechanism yields false positives violates the assumption that higher difficulty puzzles are always given priority over lower difficulty puzzles. Thus, the attacker can use this as a strategy for attacking underprovisioned routers; the attacker may generate a large volume of low-priority packets in order to ‘squat’ Bloom Filter memory space and make subsequent legitimate requests invalid. Note that memory budget in routers must increase proportionally to the number of connection requests, if we are to keep the rate of false positives reasonable.

Portcullis is a puzzle-based solution for countering Denial of Service attacks. Despite its complexity, Portcullis presents a solution which may be appealing for some types of services. This solution requires far less resources than per-host fair sharing, and avoids collateral damage (e.g. when a legitimate source is collocated with an attacker). Finally, from a high-level perspective, Portcullis is an interesting solution for addressing the problem of regulating access to resources, where throughput comes secondary, whereas maximum delay guarantees is the primary concern.
III. RESEARCH PROPOSAL

We have seen three papers that describe different approaches for dealing with DoS attacks; blocking, enforcing admission control and regulating access to a common resource. There are numerous other examples in the literature of papers that adhere to one or more of these principles at the same time. TVA [8] features a fine-grained capability scheme which allows a receiver to assign a rate-limit to flows that they authorise. TVA also tries to address Denial of Capability attacks by using per-ingress port fair sharing at each router. StopIt [9] is filtering protocol from the same authors, and proposes per AS-fair sharing as a fallback option, when filtering fails. StopIt also features a cookie-like mechanism that an ISP can use in order to offload filtering state to their clients. In Points of Control [10], neighbouring ISPs can collaborate to identify the entry points of malicious traffic to their network. The authors achieve this using IP encapsulation and a controlled BGP message announcement scheme that is feasible using commodity equipment.

My research plan involves answering the following two questions. First, can we combine the best of all worlds in order to create a comprehensive DoS mitigation solution. Second, how can we improve current best practices in ISPs in order to improve the resilience of hosts against DoS attacks?

One of the caveats of combinatorial approaches is that it may introduce unnecessary complexity. For instance, Argyraki and Cheriton have argued that if a filter-based mechanism is sufficient to protect connection setup in a capability-based solution, then the same mechanism can be used to protect all traffic in general [5]. Combining capabilities and filtering in that fashion only increases the complexity of the protocol without adding any utility. Nevertheless, we can leverage the stateless nature of capabilities in order to augment filter-based solutions.

Recall that flow monitoring is a crucial aspect of ATTF; an ISP must consult the Flow Cache so that it verify independently that one of its clients is misbehaving. As we have already discussed, Flow Cache could require a significant amount of state. Nevertheless, we can combine filtering with capabilities in order to reduce the amount of state that each ISP needs to maintain for its customers.

Assume that a receiver, \( R \), wishes to have all traffic from \( S \) blocked. The sender’s ISP, \( S_{gw} \), can use capabilities in order to offload flow monitoring to \( R \). In particular, \( S_{gw} \) marks capability tokens on outgoing traffic from \( S \). Upon receiving a packet from \( S, R \) can use the capability token as an irrefutable proof that it has received traffic from \( S \). From a high level perspective, \( S \) authorised \( R \) to make a filtering request by providing \( R \) with a capability token. In contrast to capability-based solutions, this mechanism is immune to Denial of Capability attacks against \( R \).

Currently, ISPs use dedicated pieces of equipment known as scrubbing boxes in order to mitigate DDoS attacks. Scrubbers are provisioned with a significantly larger amount of TCAM than conventional routers and are, therefore, more expensive. An example of a commercially available scrubbing product is Arbor Peakflow [11]. Once a receiver reports that it is under attack, the ISP reroutes all incoming traffic through a scrubbing box, which blocks the attack packets.

As a first step, we should investigate what resources it takes to block most bandwidth-flooding attacks today, and whether scrubbing boxes are sufficient to achieve this. It may be difficult to determine the limitations of scrubbing boxes, given that the design is proprietary. Nevertheless, we can tap onto insight from the research community and draw conclusions about how an optimal scrubber-based solution would look like. For instance, we can assume that when a scrubber runs out of TCAM resources to do filtering, it uses an optimal filter aggregation algorithm that minimises collateral damage, such as the one proposed by Soldo et al. [12].

As a second step, we should examine the monetary aspects of provisioning for DoS attacks today, and what is the return in investment for an ISP. Although the initial cost is high, scrubbing boxes can be a verifiable source of profit for ISPs, which can charge their customers a fee for mitigating bandwidth-flooding attacks. Nevertheless, the processing capability of a scrubbing box is limited by the throughput of its network interface. Thus, the ISP may have to purchase additional scrubbing boxes in order to protect all of its customers, if the ISP comes under a massive DDoS attack. Given that an ISP might fail to protect some of the customers, we should also measure the cost of downtime for the victims and the ISP. Moreover, if possible, we should try and capture the cost of collateral damage due to filter aggregation, when the ISP blocks legitimate flows.

Finally, we should explore what the optimal reaction from an ISP should be, when it receives a massive DDoS attack. For instance, multiple ISPs could enter an agreement to pool their resources if they receive a disproportionately large attack from the network. Just like other approaches that are based on collaboration, the goal would be to provide a simple and practical solution that would motivate adoption among the ISPs.

REFERENCES


