Protecting SIP against Very Large Flooding DoS Attacks

Felipe Huici felipe.huici@nw.neclab.eu
Saverio Niccolini saverio.niccolini@nw.neclab.eu
Nico d'Heureuse nico.dheureuse@nw.neclab.eu
NEC Europe Ltd.

***Abstract*—The use of the Internet for VoIP communications has seen an important increase over the last few years, with the Session Initiation Protocol (SIP) as the most popular protocol used for signaling. Unfortunately, SIP devices are quite vulnerable to Denial-of-Service (DoS) attacks, many of them becoming unresponsive and even resetting with floods of only hundreds of packets per second.**

In this paper we introduce *SIP Defender*, a new distributed filtering architecture designed to protect SIP devices against large, flooding DoS attacks. In addition, we describe the implementation of the architecture's *SIP Controllers*, the network devices in charge of performing the actual filtering. We further present testbed performance figures for these, showing that a controller built on commodity hardware can forward an impressive 2.5 million packets per second for small SIP packets while applying one million filters as well as anti-spoofing mechanisms.

I. Introduction

The last few years have seen a sharp increase in the use of the Internet for voice communications. Indeed, in order to reduce operating costs and to provide additional services, many operators are providing voice-over-IP (VoIP) services and some have already started to replace their entire PSTN infrastructure [3]. The change, however, comes with drawbacks, since VoIP inherits the security problems of the IP network it runs over. Worse, telephony is critical infrastructure, and so it presents an attractive target for attack.

While several protocols exist, VoIP communications tend to rely on the Session Initiation Protocol (SIP) [18] for call signaling. Unfortunately, it is relatively easy to flood a device such as a SIP phone with messages so that it can no longer service legitimate requests, a Denial-of-Service (DoS) attack. These sort of attacks are likely to become more commonplace as more of these devices are connected to the public Internet.

Just how vulnerable are SIP devices to flooding DoS attacks? In the case of some of the most popular hardware phones, we conducted flooding attacks with INVITE messages, normally used to initiate calls. As shown in figure 1, even the best performing phone could not withstand a flood of only 180 messages per second. Further, some of the phones did not return to normal after the flood subsided, but ended up resetting instead. SIP proxies can handle larger numbers of transactions, up to thousands per second for Kamailio (formerly known as OpenSER) [10] and probably higher rates for hardware-based solutions. However, it would be trivial for a botnet to generate flood traffic to saturate a proxy's capacity, specially considering the fact that these can consist of as many as 1.5 million bots [4].

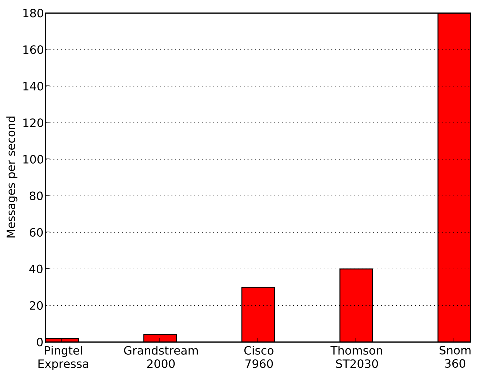

Fig. 1. SIP INVITE flood attacks on hardware phones. The values denote the rate at which the attack was successful.

It is clear from this discussion that SIP devices are quite vulnerable to DoS flooding attacks. What is also clear is that placing a device to filter the flood near the destination will not work, since by then the traffic from a botnet will have aggregated enough to DoS even the filtering device. Even if the device kept up, the attack could be large enough to saturate the link it is connected to, rendering the solution ineffective.

While many architectural (i.e., scaling to Internet-wide levels) solutions have been proposed over the years in the field of IP-level DoS attack prevention and mitigation, none have been even partially deployed. This is largely due to the fact that most of them present difficult deployment hurdles. The challenge is then to design an architecture that re-uses currently available mechanisms as much as possible in order to be easily deployed, while at the same time being able to cope with large attacks. In order to achieve this, we present *SIP Defender*, a distributed filtering architecture that enables victims to request that malicious traffic be filtered close to its sources.

The rest of the paper is organized as follows: section II describes the basic architecture in detail; section III presents anti-spoofing mechanisms for the actual traffic as well as for filtering requests; section IV describes the implementation of the filtering devices on commodity hardware, as well as a detailed performance evaluation; finally, section V discusses related work and section VI concludes.

II. SIP Defender Architecture

The aim is to design a distributed filtering architecture that can cope with large flooding attacks, reuses existing

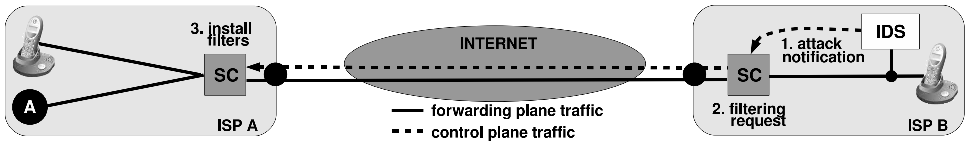

Fig. 2. Basic SIP Defender filtering architecture. SC stands for SIP Controller, A for attacker.

mechanisms as much as possible in order to be deployable in the current Internet, and cannot itself be used as a tool for Denial-of-Service attacks. In this section we present *SIP Defender*, an architecture that meets these requirements. We will first present the basic filtering process and then introduce anti-spoofing mechanisms to prevent attackers from using the architecture as an attack vector.

A. Basic Solution

We begin by assuming the presence of an Intrusion Detection System (IDS) at the victim's site capable of detecting the flooding attack. The architecture places special filtering boxes called *SIP Controllers* (SPs, or controllers for short) near the sources of traffic but also at the victim's ISP, as shown in figure 2.

The basic process begins with the IDS detecting the flooding attack. The IDS is also in charge of generating a list of the SIP users sending the malicious of traffic, along with those users being attacked; we will explain why the latter is needed later on in this section. In this step care must be taken that the user identities have not been spoofed, since ignoring this would allow an attacker to avoid detection. Even worse still, the attacker could cause the system to deny service to an unsuspecting victim by including this latter as the originator of the flood; we will tackle these issues in section III.

An alternative to the IDS generating the list of malicious sources of traffic would be for the ISP to allow its clients to use their own IDSes, creating their lists and sending them directly to the controller. Either way, the next step is for the local controller to figure out which remote controllers the malicious traffic is flowing through, a step needed in order to know where to send filtering requests to.

What does a filter in these requests look like? In the simplest case, it is made of a SIP `<to, from>` pair or a `<to>` singleton. While more advanced filters are certainly possible (we will touch upon this in the implementation section), in general we will assume that filters are of this simple form, since it is sufficient to mitigate SIP DoS flooding attacks.

The next step is for the local controller to distribute filtering requests containing the malicious sources reported by the IDS to the relevant remote SIP controllers (we explain how their IP addresses are obtained in the next section); defining the protocol used for this is outside the scope of this paper. A remote controller then receives the filtering request but has to perform a few basic checks before installing the filters in it. First, it needs to ensure that the request did in fact come from the controller it claims to have come from, otherwise attackers could easily install malicious filters.

A second check performed by the remote controller is making sure that the controller that sent the request is actually responsible for the user in the `to` field of the filter. Ignoring this step would allow malicious controllers to block traffic to unsuspecting victims, and so what is essentially needed to prevent this is a mapping of controllers to clients. Such a mapping should be cryptographically signed, lest it provide another avenue for attack. There are many ways to accomplish this, but our preferred solution is to use a robust peer-to-peer flooding protocol to distribute digitally signed mappings to all SIP controllers. This is basically the same robust flooding mechanism proposed in [6], and is extremely resilient to attack.

Once these checks are performed the remote controller finally installs the filters. The problem for the attack's victim is now to know when to remove the filters: if the attack traffic subsides, the victim cannot tell whether this is because the attack is being filtered or because it has actually stopped. One possibility would be to include a time out value as part of the filtering request, and let the victim's controller send another filtering request should the attack resume after the original filter expires. Alternatively, the controller could explicitly send a request to cancel the filter. Independent of this the remote controller would have to have a filter expiration policy to put a bound on how many filters it has at any one point.

B. Initial Deployment

Under full deployment every ISP would have SIP controllers deployed. Unfortunately this cannot initially be the case, and so we need to consider partial-deployment scenarios and their implications. As a visual aid, figure 3 shows a number of these. ISPs shown in grey have SIP Defender deployed, while those in white, called *legacy ISPs*, do not. In the normal scenario, calls from Alice to Chris go through ISPs A, E and G, as is also the case for calls from Bob to Chris. Please note that whether ISPs E and F have SIP Defender is irrelevant; this will become apparent in the next section when we describe the anti-spoofing mechanism.

The first attack scenario is the normal one, where both source and destination ISPs have SIP Defender deployed. Attacker A1 at ISP B sends a flood to user Chris. The IDS at ISP G detects this and sends a message with user A1 as the attacker to SC5, the local SIP controller. SC5 then looks up that SC2 is responsible for A1 and sends it a filtering request with the filter `<Chris, A1>`. Upon receipt, SC2 checks that the request did come from SC5 and that it is the controller responsible for user Chris. Finally, it installs the filter, blocking all SIP traffic from A1 to Chris.

In the second attack scenario A2 sends malicious traffic to user Chris. Unfortunately this time SC5 has no remote controller at ISP D to send a filtering request to. As a result, SC5 installs filter `<Chris, A2>` locally or should

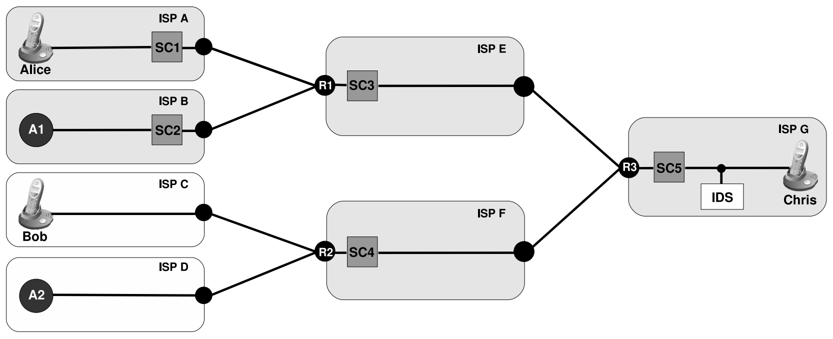

Fig. 3. Initial deployment scenarios. ISPs in white denote legacy ISPs without protection.

an upstream ISP have SIP Defender deployed, it can send the request to its controller instead, as is the case with ISP F and SC4 in the figure. This situation is naturally less than ideal, since the traffic is being filtered closer to the victim. In the next section we introduce a simple mechanism to improve this problem and also to provide some incentive for ISPs to deploy the architecture.

III. Anti-Spoofing

We need to ensure that attackers cannot use SIP Defender as a DoS attack tool in its own right. In the previous section we mentioned several parts of the filtering process that needed special attention. The first of these has to do with identity spoofing, basically making sure that a SIP user cannot send messages with somebody else's identity in the "from" header (or that of a non-existent user). If an attacker were able to do this, it could avoid detection or even create floods that would force SIP Defender to filter an unsuspecting victim's legitimate traffic.

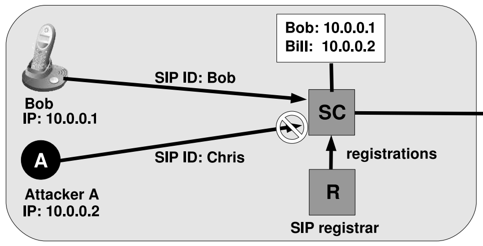

Fig. 4. SIP ingress filtering mechanism.

To prevent this situation, SIP controllers have added functionality to perform SIP ingress filtering. We assume that the ISP has a SIP registrar whose entries can be trusted and that keeps a mapping of IP addresses to user identities[1]. The controller periodically requests all new registrations and keeps a local copy of them (see figure 4). When attacker A sends a SIP messaging claiming to come from someone else (in this case Chris), the controller checks it against its local registrations to see if there is a match, and if there is not drops the packet. It is worth noting that this mechanism works even if the SIP identity used by A is Bob, assuming that the ISP has IP-level ingress filtering deployed in order to prevent A from claiming that packets are coming from Bob's IP address.

[1]Technically a SIP registrar does not keep actual IP addresses but URIs. The registrar could be modified to support this functionality or a separate service could be created for the same purpose.

As described so far, the SIP controllers prohibit spoofed SIP packets from reaching the Internet. However, this is not sufficient, since under partial deployment it is of course possible for attackers to send spoofed messages from legacy ISPs. The problem is that a victim cannot tell whether an incoming SIP packet came from a SIP Defender ISP or a legacy one. In the case of SIP it is current common practice for inter-site communication to take place over a TLS tunnel. As a result, assuming that both ends of the tunnels have SIP ingress filtering deployed, we now know that SIP To headers arriving over TLS tunnels are not spoofed. With this in place the rule is simple: filtering requests are issued only for packets arriving over a TLS tunnel and are accepted only if they arrive via the same means.

The TLS tunnel also solves the issue of figuring out which remote SIP controller to contact for a given SIP message. Upon the IDS detecting an attack, the local controller waits until another attack packet arrives and marks which tunnel it came from. It can then send filtering requests to the ISP at the other end of the tunnel; one way for these requests to reach the actual remote controller would be for SIP Defender ISPs to have a DNS entry of the form `controller@isp.com`. This approach would work well in the short-run, since currently most administrative domains have TLS connections to all entities they share SIP traffic with. In the longer run, as SIP adoption becomes even more widespread, it is likely that SIP communication will no longer happen only over TLS tunnels; in this case, cryptographically-based identity schemes such as Passport [13] or that found in RFC 4474 [16] can be used to replace the role played by TLS tunnels.

This leaves one last issue unresolved. While the attacker can no longer use the filtering architecture as a DoS tool, it can still, if located at a legacy ISP, flood the destination's SIP controller. In order to cope with this, the destination ISP installs a filtering rule at its edge router giving lower priority to SIP packets not arriving over a TLS tunnel (the router should be able to apply this rule at line rate without problems). This means that even under a flood packets arriving from a SIP Defender ISP will get through, and if these packets happen to be malicious they can of course be dropped using the normal filtering mechanism. Should the flood originate at legacy ISPs and be large enough to saturate the edge router's link, filters can be requested of intermediate SIP Defender ISPs such as ISP E and F in figure 3.

IV. Evaluation

One of the key elements to coping with large SIP DoS flooding attacks is performance. While ideally we would like the mechanisms described to be implemented in hardware, this is unlikely to be the case in the beginning. A compromise solution is to build the SIP controllers using commodity hardware, in order to give confidence that they can perform well enough to filter even large attacks. To this end, in this section we present performance results conducted on a network testbed.

A. Experimental Setup

We conducted all performance experiments on the Heterogeneous Experimental Network (HEN[20]). For our SIP controller tests we used a Dell 2950 with two Intel Xeon X5355 2.66GHz quad core processors, 8 GB of main memory and 3 quad port Intel 82571EB PCI express network cards. In addition, we used Dell 1950s to both generate and count traffic. Since the Dell 2950 acting as the controller has a maximum of 12 network interfaces, we connected three traffic generators and three traffic counters to it, as shown in figure 5.

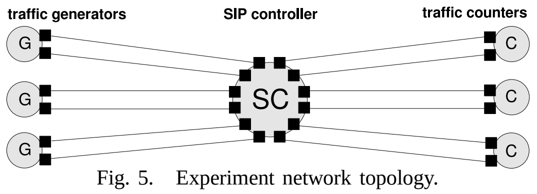

Fig. 5. Experiment network topology.

The reason for using two interfaces on each of the generators and counters is that we conducted tests that showed that these machines (all Dell 1950s) can perform these functions at line rate for all packet sizes. Using more network ports increases overall throughput but packets are no longer processed at line rate, which could potentially skew measurements regarding the SIP controller. In terms of software all computers ran a Linux 2.6 kernel and an e1000 polling driver. To generate, forward and count packets we used the Click modular router platform [11].

In order to prove the worth of an anti-DoS mechanism it is best to test the worst-case scenario. Since performance decreases with decreasing packet sizes, using small packet sizes during the tests gives greater confidence that the architecture would be able to cope with any large flooding attacks. In the case of SIP, there is no official minimum packet size as is the case with IP. However, the SIP RFC [18] shows that an ACK is one of the SIP messages with the fewest number of required headers. If we strip each of these down we end up with a payload size of about 240 bytes, for which the maximum theoretical rate is around 446,000 packet per second (pps) or about 929 Mb/s (this rate is less than 1Gb/s due to overheads such as the preamble and the inter-frame gap). Note that for our experiments we used UDP traffic since not only it is cheaper for an attacker to generate, but it is also easily spoofable and so more attractive from his point of view.

Of course this is not an absolute minimum, but it should give a good idea of the SIP controller's performance. Throughout the rest of this section we will always use these 240-byte packets unless otherwise stated. In addition, it is worth noting that an attacker can naturally send malformed SIP messages that are smaller than this. To deal with this, our implementation quickly discards any SIP messages that are not over a certain minimum size. An attacker can also attempt to slow down the parser with large SIP packets by placing crucial headers, such as `From` and `To`, at the end. However, most SIP implementations place these headers at the beginning, and so we can drop any SIP packets for which these headers are not found after a certain number of bytes.

B. Baseline Performance

Before testing the performance of the mechanisms discussed in the previous section it makes sense to see what the baseline performance of the system is. To do this, we had the SIP controller receive packets on one interface and forward them on another. The controller consists of Click *elements*, which are basic units of packet processing, connected to each other using a Click configuration file. In more detail, the controller receives packets from the polling driver and classifies them into IP and non-IP. IP packets are further processed, first by stripping the Ethernet header and then by going through a custom-made `MinLengthDropper` element, which drops packets that are considered too small to be valid SIP messages.

Next the collector performs some checks on the IP header before sending the packets to custom-made `SIPStrip/SIPUnstrip` elements. The first element strips the UDP or TCP header from the packet while the latter re-adds it (the reason that there is no processing between these two is that this provides a baseline measurement). Finally packets are sent to a queue before being sent out.

This Click configuration represents one *forwarding path* from a receiving interface to a sending one. The Dell 2950 we used as the SIP controller has twelve interfaces and eight CPU cores, so we wanted to see how the forwarding performance would scale as forwarding paths were added to the system, up to a maximum of six in this case. In greater detail, Click supports multi-threading, essentially allowing the user to assign parts of a forwarding paths to different CPU cores. As it turns out, the best performance results when an entire path is handled by a single core in order to prevent packets having to change cores; figure 6 shows performance figures for increasing numbers of concurrent forwarding paths, where each is assigned to a separate CPU core. The SIP controller can forward very small SIP packets at line rate for all six paths, or about 2.6 million packets per second (5,570 Mb/s).

Another basic function of the SIP controller is IP defragmentation. In order to derive performance figures for this we inserted the standard click element `IPReassembler` after the `CheckIPHeader` element. We then had the generators send minimum-sized (64-byte) fragments, which they did at a rate of about 1 million packets per second. Even in this extreme case the SIP controller was able to keep up, the forwarding rate matching the offered one. We also performed one more test this time mixing regular traffic with about 20%

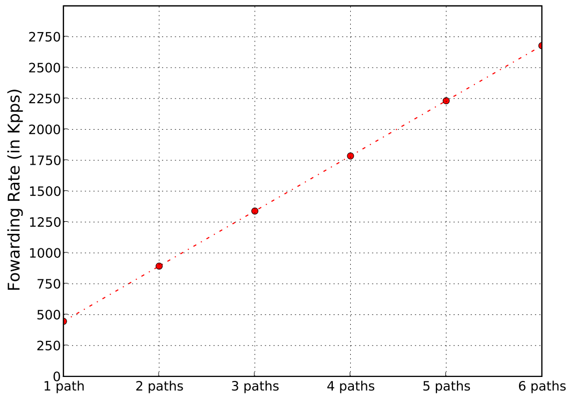

Fig. 6. Baseline forwarding performance for SIP controller and six forwarding paths for 240-byte SIP packets.

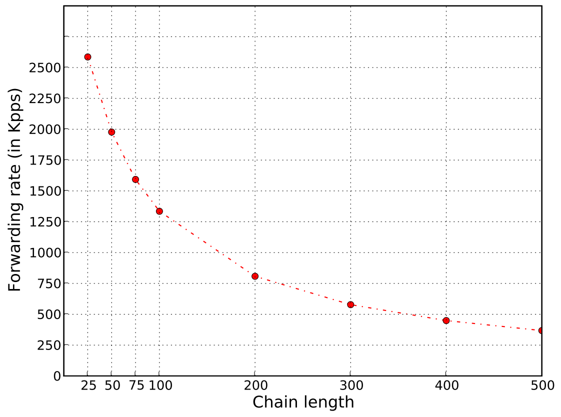

Fig. 7. Forwarding performance with long chains in the hash-based filtering element.

fragmented traffic (studies have shown that IP fragments in the Internet comprise about 1% of all traffic [9], [19], and so the 20% figure is a large upper bound). With this setup the SIP controller was again able to forward packets at line rate. This makes sense, since dealing with the a flow composed entirely of minimum-sized packets as was the case in the previous experiment puts a larger strain on the system.

C. SIP Controller Performance

The main function of the controller is to filter malicious packets. To achieve this we extended Click by implementing a fast SIP parser and a custom element called `SIPHashFilter`. We used the exact same Click configuration from the baseline evaluation with this new element inserted between the `SIPStrip` and `SIPUnstrip` elements. The element definition for the filter in the Click configuration file looks as follows:

```
filter :: SIPHashFilter("From" "URI", "To" "URI")
```

As can be seen, the element takes a list of string pairs. In this example the controller parses SIP packets, retrieves the URIs from the “To” and “From” headers and uses their concatenated values as inputs to the hash. Other filters are certainly possible, for example including only a “To” field in order to block all traffic from an ISP to a destination.

Since the filter is based on a hash, its performance depends largely on the length of its chains. To see the effect of long chains on forwarding performance we tweaked the hash so that all incoming packets would traverse a long chain of filters, essentially turning the hash into a long vector. The results in figure 7 show, as expected, a clear drop as the chain length increases. Even so, the forwarding rate achieved is quite high (e.g., 74% of the theoretically maximum rate for a chain length of 50). The result is even more reasonable if we consider that a 50-element chain is quite long: normally a hash would be balanced and have enough buckets so that this does not occur, and, in case it does, re-hashing would take place to correct the situation.

So far all experiments involved static packets, meaning that the values used as input to the hash do not change. This results in very good cache locality but is unfortunately an unrealistic scenario. To test the performance when the CPU core’s cache is being thrashed we instrumented the hash so that all its chains were of the same length and each packet was hashed to a different bucket. This setup results in a forwarding rate of about 2,100 Kpps (79% of the theoretical maximum) for a chain length of 10, dropping down to 317 Kpps (12% of the maximum) for a length of 100. What this shows is that even under an extreme scenario where each single packet hashes to a different bucket the system is capable of forwarding at high rates, but that special care should be taken so that the chains do not become too long.

Num Filters	Insertion Time (in msec)	Time/Filter (in msec)
500,000	3,559	0.0071
1,000,000	7,032	0.0070
5,000,000	35,376	0.0071
10,000,000	66,768	0.0067

Fig. 8. Filter insertion times while forwarding traffic.

Another important factor worth testing is filter insertion time. A controller should be able to insert new filters in a reasonable time while still forwarding traffic. Figure 8 shows that even inserting as many as 10 million filters takes only about one minute, certainly sufficient to quickly filter even the largest floods. In addition, since we used a large number of buckets for the hash (one million) the controller was able to forward packets at line rate throughout the whole process.

The controller’s last important function is SIP ingress filtering. For this purpose we implemented another Click element called `SIPIngressFilter` which retrieves the source IP address and SIP identity from the packet and makes sure that this pair matches one of the entries it has stored.

As a final performance test we created a Click configuration with all the elements of the controller in place. The only new element is `SIPMinChecker`, which splits packets depending on whether they are meant for SIP’s well-known port number, and drops those that are but are too small to be legitimate SIP packets. To conduct performance tests with this configuration we loaded the ingress filter in each forwarding path with 10,000 /24 prefixes (enough to support about 2.5 million addresses) and the hash filter with one million filters. Even with this large load the SIP controller was able to forward SIP packets at 95% of the theoretical maximum for all six paths, or a total of about 2.5 million packets per second.

These results clearly show that the combination of commodity hardware and Click constitute a viable platform for implementing the SIP Defender architecture. The rates demonstrate that SIP controllers can cope with large amounts of traffic and filters, certainly enough to cope with even large DoS flooding attacks. Indeed, the rates are good enough to provide a reasonable level of protection even in the case where the filtering has to be done at the victim's ISP because there are no remote controllers to send filtering requests to.

V. Related Work

The field of DoS at the IP level has seen lots of solutions proposed over the last years, yet most of them present difficult deployment issues and have not seen the light of day. Approaches tend to fall into two categories: capabilities-based systems and filtering ones. Capabilities systems [1], [22], [12] force sources to request permission to send from the receiver, thus attempting to prevent attacks. Filtering solutions [8], [2], [7], on the other hand, mitigate an attack once it has already started by inserting filters at different points in the network.

While schemes have been proposed to protect SIP against DoS [5], [14], [21], [17], none of them can cope with very large distributed attacks. VoIP Defender, in particular, claims to be highly scalable, but does not provide an architecture that can scale to large (i.e., Internet) networks, but rather a design based on using a load balancer to distribute traffic across a set of work servers. The work in [15] uses expensive hardware platforms in order to filter DoS attacks at only a single point, so does not scale to large networks.

VI. Conclusion

In this paper we introduced SIP Defender, a new distributed filtering architecture designed to protect against large, flooding SIP DoS attacks. SIP Defender allows victims to send filtering requests to filtering points called SIP controllers placed near the sources of the attack. Further, we covered how the architecture works in ideal conditions as well as under partial deployment scenarios with attackers spoofing their identities.

In addition, we implemented the SIP controllers using inexpensive off-the-shelf hardware and the Click modular router software package. We tested their performance on a network testbed, showing that a controller can forward small SIP packets at 95% of the theoretically maximum rate while having one million filters installed while performing anti-spoofing checks; since the platform used had twelve network interfaces, this boiled down to an impressive rate of about 2.5 million packets per second, or about 5.3 Gb/s. We believe this to be largely sufficient to cope with even very large SIP DoS flooding attacks.

References

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. In *Proc. ACM SIGCOMM 2nd Workshop on Hot Topics in Networks*, November 2003.

[2] K. Argyraki and D. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Usenix Annual Techical Conference*, April 2005.

[3] British Telecom. BT's 21st Century Network. http://www.btplc.com/21CN/.

[4] CNET. Bot herders may have controlled 1.5 million pcs. http://news.com.com/, October 2005.

[5] Jens Fiedler, Tomas Kupka, Sven Ehlert, Thomas Magedanz, and Dorgham Sisalem. Voip defender: highly scalable sip-based security architecture. In *IPTComm '07: Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications*, pages 11–17, New York, NY, USA, 2007. ACM.

[6] Mark Handley and Adam Greenhalgh. The case for pushing DNS. In *Proc. ACM HotNets IV*, November 2005.

[7] Felipe Huici and Mark Handley. An edge-to-edge filtering architecture against DoS. *SIGCOMM Comput. Commun. Rev.*, 37(2):39–50, 2007.

[8] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proc. Network and Distributed System Security Symposium, San Diego*. ISOC, Reston, VA., February 2002.

[9] Wolfgang John and Sven Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 111–116, New York, NY, USA, 2007. ACM.

[10] Kamailio. Kamilio SIP Server. http://www.kamailio.net/.

[11] E. Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, 18(3):263–297, August 2000.

[12] W. Lee and J. Xu. Sustaining availability of web services under distributed denial of service attacks. *IEEE Transactions on Computers*, 52(3):195–208, 2003.

[13] Xin Liu, Ang Li, Xiaowei Yang, and David Wetherall. Passport: secure and adoptable source authentication. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 365–378, Berkeley, CA, USA, 2008. USENIX Association.

[14] S. Niccolini, R.G Garroppo, S. Giordano, G. Risi, and S. Ventura. Sip intrusion detection and prevention: recommendations and prototype implementation. In *Proceedings of the 1st IEEE Workshop on VoIP Management and Security, 2006*, 2006.

[15] Gaston Ormazabal, Sarvesh Nagpal, Eilon Yardeni, and Henning Schulzrinne. Secure sip: A scalable prevention mechanism for dos attacks on sip based voip systems. In *IPTComm '08: Proceedings of the 2nd international conference on Principles, systems and applications of IP telecommunications*. ACM, 2008.

[16] J. Peterson and C. Jennings. Enhancements for authenticated identity management in the session initiation protocol (sip).

[17] Utz Roedig, Ralf Ackermann, and Ralf Steinmetz. Evaluating and Improving Firewalls for IP-Telephony Environments. In *Proceedings of the 1st IP-Telephony Workshop (IPTel2000), Berlin, Germany*, pages 161–166. GMD-Forschungszentrum Informationstechnik GmbH, April 2000.

[18] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[19] Colleen Shannon, David Moore, and K. C. Claffy. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Trans. Netw.*, 10(6):709–720, 2002.

[20] UCL Network Research Group. HEN: Heterogeneous Experimental Network. http://hen.cs.ucl.ac.uk.

[21] Yu-Sung Wu, Saurabh Bagchi, Sachin Garg, Navjot Singh, and Tim Tsai. Scidive: A stateful and cross protocol intrusion detection architecture for voice-over-ip environments. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 433, Washington, DC, USA, 2004. IEEE Computer Society.

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