SIP Protector: Defense Architecture Mitigating DDoS Flood Attacks Against SIP servers

Jan Stanek, Lukas Kencl
Research & Development Center for Mobile Applications (RDC)
Czech Technical University in Prague
Technicka 2, 166 27 Prague 6, Czech Republic
{jan.stanek, lukas.kencl}@rdc.cz,

***Abstract*—As Voice-over-IP becomes a commonly used technology, the need to keep it secure and reliable has grown. Session Initiation Protocol (SIP) is most often used to deploy VoIP and therefore SIP servers, the base components of SIP, are the most obvious targets of potential attacks. It has been demonstrated, that SIP servers are highly prone to DDoS flood attacks, yet no generally accepted defense solution mitigating these attacks is available. We propose a novel defense architecture against SIP DDoS floods, based upon a redirection mechanism and a combination of source and destination traffic filtering, exploiting the combined advantage of all the three techniques. We show that the proposed solution effectively mitigates various types of SIP DDoS flood attacks, discuss its strengths and weaknesses and propose its potential usability for other protocols. We also provide results of performance evaluation of the defense solution deployed in a SIP testbed.**

I. Introduction

With ever higher proliferation of Voice-over-IP communication, its infrastructure – the signaling protocol (Session Initiation Protocol (SIP)) and control nodes (SIP servers) – become highly attractive targets for various attacks. SIP servers are special in their relatively complex processes (such as the user search in the SIP REGISTER command), thus enabling attacks of not very high request load (hundreds or thousands per second) to overwhelm the server [19]. While a typical flood attack from a single machine may easily be filtered away, more sophisticated, distributed denial-of-service (DDoS) attacks against SIP, typically being low in terms of number of packets used, can easily bypass a general DDoS-flood-attack mitigation mechanism, tailored to stop high-bandwidth consuming attacks.

In this work, we combine multiple mechanisms – a SIP Redirect Server, a rate-limiting firewall, a Network Address Translation (NAT) element and a specific shared mapping function – to construct a novel architecture for SIP DDoS flood attack mitigation. We provide a detailed description of various DDoS flood attack types and their handling by the architecture, describe its prototype implementation and carry out early performance evaluation in a SIP testbed. Finally, we discuss the role and possible improvements of the mapping function and general applicability to other protocols.

II. Related work

Attacks against VoIP infrastructure have been studied widely in recent years. The Voice over IP Security Alliance (VOIPSA) published a detailed description of threats in VoIP [1] in 2005. Blake published a guide [6] denoting the differences between VoIP and PSTN, Chavran and Chhabria [9] provided multiple design guidelines for use in VoIP security.

The fact that SIP servers are very prone to flood attacks is well known too. Deng and Shore proposed an advanced flooding attack on a SIP server in [2]. Luo et al. tested a CPU-based DoS attack on a SIP server in [3]. Liu and Lo simulated a DDoS attack against a SIP server in [13]. Voznak and Rezac proposed a SIP vulnerability testing tool in [18], Stanek and Kencl described a SIP flood simulation tool in [19]. Both these tools can be used to simulate various attacks in the SIP infrastructure.

Quite a few proposals for defense mechanisms against SIP DoS and DDoS flood attacks have already been described. Zhang et al. [4] proposed a defense solution capable of mitigating DNS-flood DoS attacks. Even though this solution works, it only mitigates DNS floods and is not usable against general SIP floods, which we want to mitigate using SIP Protector. Sisalem et al. [5] surveyed the topic of SIP vulnerabilities and suggested ideas for server design and efficient implementation, leading to reduction of DoS-attack impact. Even though the provided analysis of vulnerabilities in SIP infrastructure is very good, the suggested improvements are usually not followed when deploying SIP infrastructure and therefore protection against DoS attacks is not ensured. Ha et al. [10] and Akbar et al. [11] proposed SIP DDoS detection mechanisms. These mechanisms are only capable of alerting when the attack is detected and not mitigating it. Zhou et al. [12] proposed a defense solution based on history snapshots and whitelisting. This works in principle, yet effectively blocks any new, previously unseen user from using the SIP server when the attack is detected. SIP Protector seeks to avoid such behaviour. Some highly complex defense solutions capable of mitigating various DoS and DDoS attack types were proposed in works of Fiedler et al. [8], Nassar et al. [7] and Asgharian et al. [20]. In comparison to these, the basic principles of SIP Protector are relatively simple, so we hope it can be easily understood and adopted. Hussain et al. [21] proposed a proxy-based strategy capable of mitigating distributed floods consisting of SIP INVITE and SIP OPTIONS messages. This solution is architecturally similar to SIP Protector, however, it is not usable against floods composed of other SIP messages,

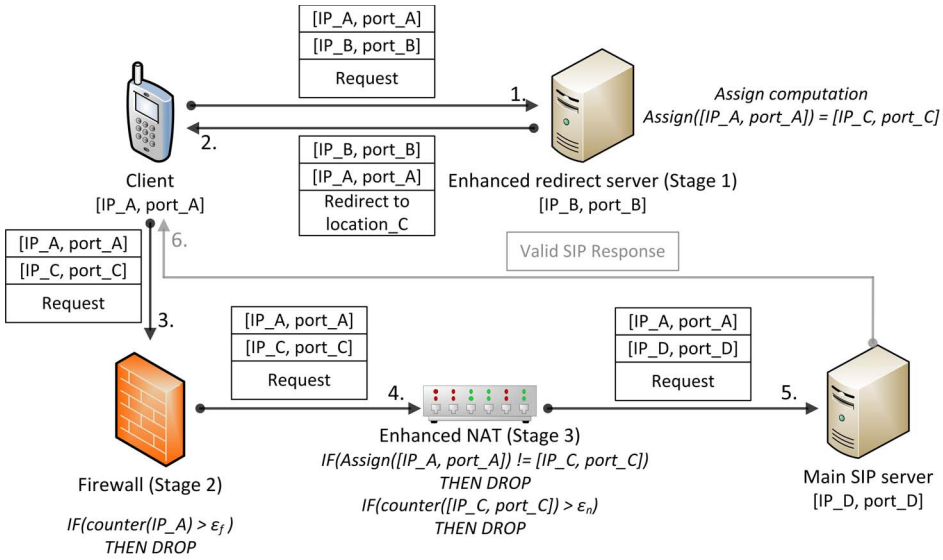


Fig. 1. SIP Protector architecture overview.

especially not the SIP REGISTER message. SIP Protector is designed to be independent of the message type used. Huici et al. [24] presented the "SIP Defender", a scheme specifically designed to counter very large flooding attacks against SIP devices. It is based on a network of securely connected SIP controllers, distributed among ISPs and exchanging information about potential sources of attack, whereas SIP Protector is designed as a standalone defense-device solution.

III. Defense Architecture

The proposed defense architecture consists of three independent, yet tightly cooperating stages (see Fig. 1). The basic idea is to use redirection and automated traffic limiting, from both the source and destination perspectives, over more stages preceding the actual SIP server. The three stages consist of an *enhanced Redirect Server*, a *firewall* and an *enhanced Destination Network Address Translator (NAT)*, all operating in harmony and sharing a hidden mapping function.

When a SIP request arrives at Stage 1 (the enhanced Redirect Server), a mapping function *Assign (src IP-P pair) → (dst IP-P pair)* is computed over the source IP address and port pair (IP-P pair) and the result is returned to the client in a redirection response as the IP-P pair of the destination SIP server. Once the client resends the request to the newly obtained IP-P pair, it arrives at Stage 2 (a rate-limiting firewall), where request rates per originator IP-P pair are stored over the recent time interval. If the request rate per originator exceeds a predefined firewall threshold ε_f, requests from this originator are dropped. If not, the request continues to Stage 3 (the enhanced Destination NAT), where the *Assign* mapping is verified and a threshold ε_n, associated to destination IP-P pairs, is checked. His request then either reaches the main SIP server or is dropped along the way. Concrete threshold values in our testbed are provided in Section V. Let us examine the three stages in more detail:

A. Stage 1 – Enhanced Redirect Server

A SIP redirect server typically has only one role - to respond to any valid request with a redirection response, containing the IP-P pair of the "main SIP server" i.e. a SIP server capable of processing requests. Redirect servers are typically used when the IP-P pair of the main server changes often, or when there is need for load balancing among multiple of them. We use the redirect server for a similar purpose - to enable virtually alternating our main server depending on the clients IP-P pair. This is done via enhancement of the redirect server with function *Assign*. For every request that arrives at the redirect server, *Assign* is computed over the source IP-P pair and outputs a new destination IP-P pair, returned to the client in a redirect response as the IP-P pair of the main server.

B. Stage 2 – Firewall with traffic limiting rules

The second stage is a typical firewall with limiting rules set per source IP-P pair. Each source IP-P pair is allowed only ε_f requests per time period. If traffic from one source reaches ε_f, all traffic from this source is filtered for a predefined time period. The threshold ε_f is set so that even the highest number of requests possibly appearing in legitimate traffic from one source in one time period can pass. For signaling protocols such as SIP this number is usually quite low (for example, we have chosen 50 requests per a 5-second period in our tests).

C. Stage 3 – Enhanced Destination NAT

The third stage is a typical Destination NAT, enhanced by computation of function *Assign* and a threshold ε_n per each NAT entry (e.g. destination IP-P pair). Upon request arrival, *Assign* is computed over the source IP-P pair and a check whether the request arrived at the appropriate destination IP-P pair is carried out. If it fails, the request is dropped, preventing potential attackers from flooding the NAT entry using requests from source IP-P pairs not assigned to it by *Assign*. If the check succeeds, a counter for the corresponding

entry is incremented. If the counter does not exceed threshold ε_n, the request is forwarded to the main server.

Because ε_n must be high enough not to limit legitimate traffic, its actual value is highly dependent on the ability of *Assign* to spread legitimate traffic among as many NAT entries as possible. If the spreading factor is high, ε_n can be set low and usability of stage 3 in attack mitigation is high. On the other hand, if *Assign* might concentrate all the legitimate traffic in a single NAT entry, then ε_n must be set to the maximum value of permitted requests per second (e.g. the processing capacity of the SIP server itself) making stage 3 usability in attack mitigation very low.

D. Function Assign

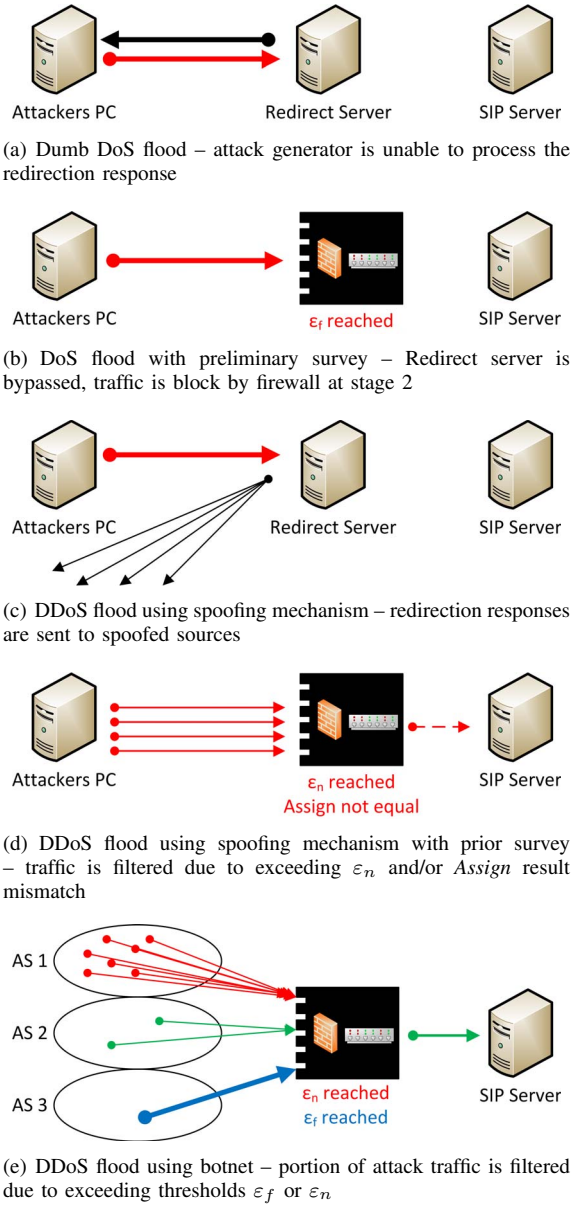
Function *Assign* is defined as follows

$$Assign(src \quad IP-P) = dst \quad IP-P$$

where *src IP-P* is the source IP-P pair of the request and *dst IP-P* is the SIP server destination IP-P pair generated by the function *Assign*. Function *Assign* has two somewhat contradictory goals: to spread legitimate traffic as evenly as possible across destination IP-P pairs (i.e. Defense NAT entries), while aggregating malicious requests together into fewer IP-P pairs. Theoretically, this goal should be feasible, as it is extremely hard for an attacker to assemble source IP-P pairs with such traffic patterns that would exhibit identical distribution properties as that of legitimate traffic. This is a key rationale of *Assign*: if it spreads legitimate traffic well, it is unlikely to spread attacker traffic well, leading to large fraction of it being blocked. Another important attribute of *Assign* is that it should be easily computable, consuming as little resources as possible.

For test purposes, we have chosen to exploit the location principle and define *Assign* as a mapping function of the request source IP address to its corresponding Autonomous System (AS) [16]. Since the distribution of IP subnets among ASes has no general pattern and ASes differ greatly in size too, it is not an easy task to find out which AS corresponds to a given IP address. Looking up the corresponding subnet in the database of all subnets of all ASes is very time-consuming. To achieve better effectiveness, we have taken the IP-AS mapping snapshot, converted all the included IP subnets into binary form and built a binary tree with AS numbers in leaf nodes. Computation of *Assign* thus reduces to a binary tree search.

The design of *Assign* is crucial for performance of the proposed architecture. A wisely chosen *Assign*, well adapted to the concrete typical traffic, the proposed solution can be very effective and requires neither cooperation with other defense mechanisms nor a complicated undergoing traffic analysis. As we demonstrate in Section V, the AS-IP mapping *Assign* function is far from ideal since it is static and does not guarantee a good distribution of legitimate traffic among NAT entries. It was designed purely for testing purposes and should be replaced with another function in the future. Ideas for better function candidates and further improvements are discussed in Section VI.



(a) Dumb DoS flood – attack generator is unable to process the redirection response

(b) DoS flood with preliminary survey – Redirect server is bypassed, traffic is block by firewall at stage 2

(c) DDoS flood using spoofing mechanism – redirection responses are sent to spoofed sources

(d) DDoS flood using spoofing mechanism with prior survey – traffic is filtered due to exceeding ε_n and/or *Assign* result mismatch

(e) DDoS flood using botnet – portion of attack traffic is filtered due to exceeding thresholds ε_f or ε_n

Fig. 2. Various types of DoS and DDoS flood attacks and their mitigation by SIP Protector

IV. Attacks

This section covers description of various types of flood attacks and shows how the proposed architecture mitigates them and why. We start by describing a simple DoS flood and proceed to more sophisticated attacks (see Fig. 2). We assume that the attacker initially knows only the publicly known IP address and port of the Redirect Server. The attacker may subsequently discover the redirection mechanism and even guess the threshold ε_f and use this information. We do not consider attacks (such as the shadow replay attack) that need other advantages of the attacker, such as a compromised legitimate client network, a compromised router or the like.

A. Simple flood

The simplest flood attack is deployed using only one PC generating high amount of packets and sending them con-

tinuously to the target server. The packets should be chosen wisely so that their processing consumes as much resources as possible. This type of attack is successfully mitigated by the proposed architecture. If the generator cannot process the redirection responses, the attack is mitigated by Stage 1 (see Fig. 2(a)), since the Redirect Server is not prone to flooding and attack traffic will never be forwarded to the actual SIP server. However, the attacker might find out that the publicly known IP-P pair leads to the Redirect Server that responds with another IP-P pair, and might send attack packets directly targeting Stage 2 (see Fig. 2(b)), or the attack generator may be able to process the returned redirect responses. If thus the attack traffic arrives at Stage 2, the attack will be mitigated there as the firewall limiting rule will block traffic from the client after it receives the first ε_f packets over a time interval.

An improved version of the previous attack would be to use multiple machines to generate malicious traffic. Either by using more real machines (e.g. a botnet) or by forging the source IP-P pair information in packets (spoofing).

B. Distributed flood using spoofing

If an attacker uses the spoofing mechanism without prior search, the attack is mitigated at Stage 1 (see Fig. 2(c)). SIP requests with the forged source IP-P pair information arrive at the Redirect Server and it sends the redirection responses, however, since the source IP-P pair information were spoofed, the responses never reach the real sources. If the attacker performs a survey first and finds out the matching IP-P pair, he may redirect malicious traffic there. However, as the destination IP-P pair is computed using *Assign* at the Redirect Server, the attacker obtains the corresponding target IP-P pair only for the machine (or machines) from which he does the survey. Therefore, either the attack traffic aggregates to a few IP-P pairs and easily crosses the threshold ε_n and is blocked there, or the majority of traffic arrives at wrong IP-P pairs and is dropped. If the attacker has many real computers under his control that may be used for the survey and later for the attack, he does not need to use IP spoofing and the situation transforms into a distributed flood using a botnet, described in the following paragraph.

C. Distributed flood using a botnet

If an attacker uses a botnet to generate a distributed flood, there are several alternatives. If the bots cannot process the redirect responses, the attack is mitigated at Stage 1. If the bots can process the redirect response or the attacker obtains the correct target IP-P pairs for the bots prior to the attack, then all malicious traffic reaches Stage 2 (see Fig. 2(d)). If traffic from any bot is higher then limit ε_f, the source IP-P pair of the bot is blocked and the total attack load is reduced. If the bot generates traffic such that it does not exceed ε_f, it continues to Stage 3. Since bots have previously obtained the correct destination IP-P pair, the check whether *Assign*(src IP-P) = dst IP-P is successful. Threshold ε_n is higher than ε_f, however, ε_n accounts for all the traffic arriving at the target IP-P pair. If the botnet traffic ends up in only a few target IP-P pairs, threshold ε_n will be exceeded in these NAT entries and the attack will be mitigated (Fig. 2(d) & 2(e)). In the bad case, attack traffic would spread well across the destination IP-P pairs, sneak under ε_n and flood the SIP server. This depends on the choice of *Assign* and is discussed in Section VI in more detail. If *Assign* succeeds in aggregating the attack into fewer NAT entries, even this sophisticated attack is mitigated.

In summary, the defense solution is capable of mitigating various flood attacks, from the simple ones, where all attack traffic is blocked, to the sophisticated ones, where the extent of attack mitigation depends on choice of the mapping function.

V. Performance evaluation

A. Prototype implementation

We have implemented a prototype of the architecture in a very straightforward manner, programmed in high-level programming languages and unoptimized. The Enhanced Redirect Server and the Enhanced Defense NAT applications were implemented in Python, using the *Twisted* library [23]. *Iptables*, the Linux software firewall implementation, with the *ipt_recent* module was used as a firewall. Direct AS-to-IP mapping function, described in Section III, was used as the *Assign* function. Therefore, the prototype source code (not counting *iptables*, where only proper configuration is needed) is short and its portability is ensured, however, both the Redirect Server and the NAT application are prone to flood attacks themselves. We have stress-tested both and the results have shown that the Redirect Server can handle at most 24000 rps and the NAT application only about 21000 rps. This limits the processing capacity of the entire architecture to about 20000 rps, which would not suffice for practical deployment. However, it is enough for a test demonstration of the architecture. Considering the relative simplicity of its respective parts, processing capacity should improve greatly once the implementation is optimized (programmed in low-level languages, *Assign* computation time minimized).

B. Attack simulation

As shown in Section IV, simple non-distributed floods and floods with spoofed sources are fully blocked using our architecture. Therefore we have focused on distributed floods, to test how well the proposed architecture handles these. Our SIP testbed consists of three machines (see Fig. 3).

The main server, a quadcore Xeon 2.5GHz with 8GB RAM, is running the SIP server. We use the Asterisk PBX [14] implementation of a SIP server. Asterisk is publicly available, open

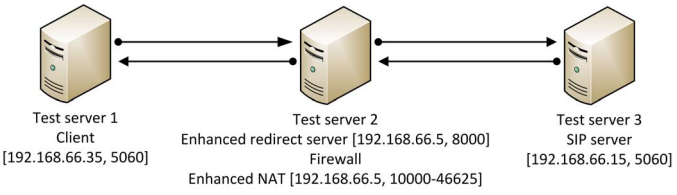


Fig. 3. SIP testbed

source and easy to install and configure and thus it is deployed widely in small and medium businesses. The most important parameter of a SIP server is its processing capacity (number of requests per second the SIP server can handle). We have set ε_n approximately to the processing capacity when using the most resource-consuming requests (for Asterisk these are SIP REGISTER requests). Our stress tests have shown that in our testbed this corresponds to approx. 250 rps. Using a different SIP server implementation, this value might be higher, even about ten times for some implementations, however, this would not affect the general results of the simulations. In fact, the results are easier to evaluate and discuss for lower processing capacities. ε_f is chosen accordingly to SIP nature – as it is a signaling protocol, there is no reason to send many messages over a short time period, and therefore ε_f was set to 50 requests per a 5-second period.

The middle machine hosts all the defense stages of the architecture. The tests were setup so that this machine always has enough capacity to process the requests and does not become a bottleneck.

The third machine is used to simulate the attackers botnet and legitimate traffic. Attacks with spoofed source addresses would be mitigated using the defense architecture, yet we ourselves are using the spoofing mechanism to simulate a botnet. This is possible as we possess the necessary knowledge of the destination IP-P pair assignment to source IP-P pair and therefore we do not need the response from the Redirect Server in practice (we let the Redirect Server generate it and drop it). This slight modification has no effect on the simulation results. Both the legitimate and attack traffic are generated by SIPp-DD [19], a SIP DDoS flood testing tool.

(i) General DDoS flood. We simulate the situation where the attacker runs an attack using a botnet without prior knowledge of the defense solutions deployed at the target. Legitimate traffic was simulated as a combination of SIP INVITE, OPTIONS and REGISTER requests so that CPU load on the SIP server fluctuates around 50% (for exact values refer to Tab. I). The test generator follows the patterns stated in [19], i.e. splits the attack traffic among sources that form a few groups with similar attack rates. Figure 4 shows CPU load before, during and after the attack. The attack is very strong initially, before blocking, but then rapidly mitigated. When examining where the attack load has been blocked, we find out that practically all of the attack traffic ends at Stage 2 – traffic from source machines is blocked due to reaching threshold ε_f. Note that a major part of the attack is blocked within 10s, however, the SIP server takes some time to recover (tries to repeatedly answer the attack requests, for which it did not get an acknowledgement). Legitimate traffic was only slightly affected – within the first 5 seconds of the attack, about

TABLE I
GENERAL DDOS FLOOD PROPERTIES

	rate (rps)	# sources	duration
legitimate traffic	150	50	200s
attack traffic	1000	100	100s

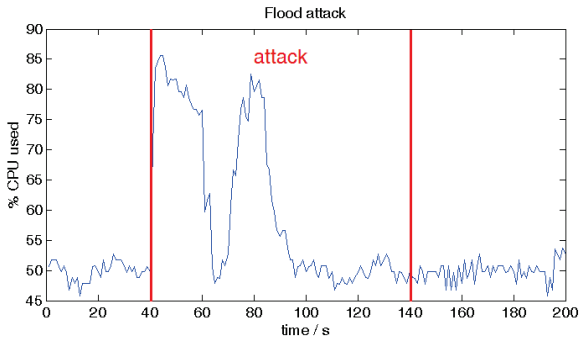


Fig. 4. CPU usage on the SIP server during a distributed flood attack. The attack starts at 40s and ends at 140s.

350 requests had to be resend. This demonstrates that using a firewall with traffic-limiting rules is effective against typical DDoS floods. The rest of the proposed architecture appears to be an overhead in this kind of attack. Let us then consider a smarter attack:

(ii) Tailored DDoS flood. This attack is designed to bypass Stage 2, not reaching ε_f for any source. This simulates a situation where the attacker found out about the firewall and estimated the threshold. This is doable, especially if the threshold does not change over time. The simulation is identical to the previous case, only the number of attack sources grows to 125. The attack is distributed among the sources uniformly – the rate per source is 8rps and does not exceed ε_f for any source. It is feasible for an attacker to get this many machines under control, however, it is almost impossible for them to be distributed among the ASes evenly. We have analyzed traffic from a real DDoS attack [17] and found out that the distribution of source addresses among ASes corresponds roughly to a power law. We have modified our generator accordingly and Fig. 5 shows the distribution of the sources of an attack among the ASes used in the simulation. Figure 6 depicts CPU load during the attack. The load rises up after the attack starts and floods the server until the attack ends. During the attack, legitimate traffic was badly affected and only occasionally a few requests were responded to properly. However, this does not mean that defense architecture would not work – after all, about 75% of the attack load was mitigated – yet the remaining portion was still enough to flood the server. The question is, can we do any better?

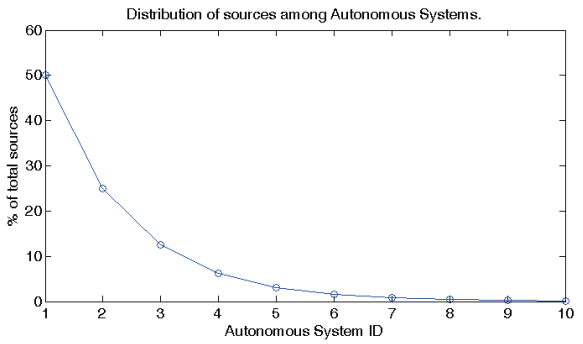


Fig. 5. Distribution of attack sources among the Autonomous Systems.

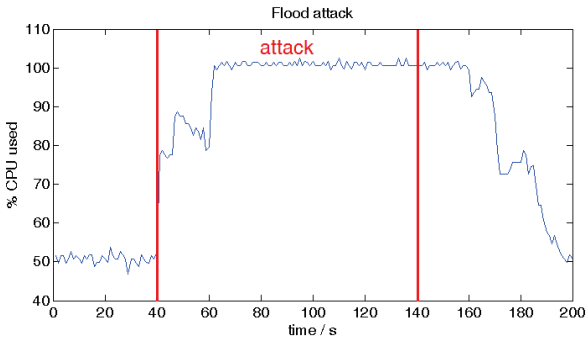


Fig. 6. CPU usage on the SIP server during a distributed flood attack.. The attack starts at 40s and ends at 140s.

VI. Discussion and Future work

In the second simulation, the architecture does not ensure attack protection. The reason rests in the big difference between thresholds ε_f and ε_n. The lower the thresholds, the better the protection. It is not feasible to decrease threshold ε_f, as it was derived from typical SIP properties and decreasing it further might result in blocking legitimate clients. Threshold ε_n is a better candidate for decrease. In our simulation, threshold ε_n equalled the processing capacity of the protected SIP server, based upon a theoretic assumption that all legitimate clients might belong to the same AS. When analyzing legitimate traffic at a real operator server, we found this assumption to be false. However, even if it were true, one could partition this AS further into smaller virtual areas. Splitting a major AS into two may let decrease ε_n by up to a half. The more evenly *Assign* distributes legitimate requests among destination IP-P pairs, the lower ε_n can be and thus the better protection the architecture could offer. The better the requests are spread among destinations, though, the higher the probability that in case of an attack, portions of legitimate traffic are blocked too (as they share the destination with attack sources and thus are blocked too, after ε_n is reached for the corresponding entry).

In our future work, we plan to improve *Assign* from a straightforward IP-AS mapping function to a more sophisticated version that would *(i)* dynamically change over time so that it cannot be guessed by the attacker. This is achievable easily – for example one may randomly rotate the one-dimensional array of the mapping of groups of sources to destination IP-P pairs each randomly chosen time interval (or for randomly chosen steps); *(ii)* maintain the location principle to group traffic from neighbors together (the neighborhood property might be redefined from the AS-IP connection to another relation); *(iii)* better spread traffic among the destination IP-P pairs. The latter two properties are harder to ensure. Adaptive hash functions may be good candidates to meet these requirements [22].

Another possibly useful improvement is employing a simple statefulness by defining states for a normal situation and for active attack and using different *Assign* functions in each situation. Switching from state ”normal” to state ”attack” can be triggered by exceeding one of the previously defined thresholds ε_n or ε_f. This should ensure that attack traffic is mapped mainly to other destination IP-P pairs than legitimate traffic that started before the attack.

VII. Conclusion

We have presented a prototype architecture capable of mitigating most DDoS flood attacks against SIP servers. The architecture is usable and provably working but the implemented prototype is not yet optimized and tested under real conditions. Specifically targeted attacks may still overcome the server, however, with randomization and further optimization of the mapping function *Assign*, it will become increasingly harder for an attacker to disguise attack traffic as legitimate.

The architecture possesses some clear advantages: it has potential for fast hardware implementation in a standalone network element, acting as a protecting device for a real SIP server. Furthermore, a similar mechanism may be used to protect infrastructure of other network protocols, most notably HTTP, as long as a redirection mechanism is put in place.

References

[1] J. Zar et al., *VOIPSA VoIP Security and Privacy Threat Taxonomy*. www.voipsa.org/Activities/VOIPSA_Threat_Taxonomy_0.1.pdf, 2005.

[2] X. Deng and M. Shore, *Advanced Flooding Attack on a SIP Server*. International Conference on Availability, Reliability and Security, 2009.

[3] Ming Luo, Tao Peng, C. Leckie, *CPU-based DoS Attacks Against SIP Servers*. NOMS, 2008.

[4] G. Zhang, S. Ehlert, T. Magedanz, D. Sisalem, *Denial of service attack and prevention on SIP VoIP Insfrastructures Using DNS Flooding*. IPTCOMM, 2007.

[5] D. Sisalem, J. Kuthan, S. Ehlert, *Denial of Service Attacks Targeting a SIP VoIP Infrastructure: Attack Scenarios and Prevention Mechanisms*. MNET, September/October 2006.

[6] E. A. Blake, *Network Security: VoIP Security on Data Network-A Guide*. In Information Security Curriculum Development Conference, 2007.

[7] M. Nassar, S. Nicollini, R. State, T. Ewald, *Holistic VoIP Intrusion Detection and Prevention System*. IPTCOMM, 2007.

[8] J. Fiedler, T. Kupka, S. Ehlert, T. Magedanz, D. Sisalem, *VoIP Defender: Highly Scalable SIP-based Security Architecture*. IPTCOMM, 2007.

[9] N. A. Chavhan and S. A. Chhabria, *Multiple Design Patterns for Voice over IP Security*. ICAC3, 2009.

[10] D. Y. Ha et al., *Design and Implementation of SIP-aware DDoS Attack Detection System*. ICIS, 2009.

[11] M. A. Akbar and M. Farooq, *Application of Evolutionary Algorithms in Detection of SIP based Flooding Attacks*. GECCO, July 2009.

[12] C. Zhou, C. Leckie, K. Ramamohanarao, *Protecting SIP Server from CPU-Based DoS Attacks using History-Based IP Filtering*. LCOMM, 2009.

[13] Chung-Hsin Liu and Chun-Lin Lo, *The Simulation for the SIP DDoS Attack*. NCM, 2009.

[14] Asterisk Private Branch eXchange, http://www.asterisk.org/.

[15] SIP: Session Initiation Protocol, http://www.ietf.org/rfc/rfc3261.txt.

[16] Autonomous System numbers FAQ, www.apnic.net/services/services-apnic-provides/helpdesk/faqs/asn-faqs.

[17] H. D. Moore et al. *Source list of attackers targetting metasploit.com*. http://digitaloffense.net/tools/ddos_sources_02.09.txt, 2009.

[18] M. Voznak, F. Rezac *SIP threats detection system*. DNCOCO, 2010.

[19] J. Stanek, L. Kencl *SIPp-DD: SIP DDoS Flood-Attack Simulation Tool*. ICCCN, 2011.

[20] Z. Asgharian et al. *A framework for SIP intrusion detection and response systems*. CNDS, 2011.

[21] I. Hussain, F. Nait-Abdesselam *Strategy based proxy to secure user agent from flooding attacks in SIP*. IWCMC, 2011.

[22] C. Martinez, W. Lin *Adaptive Hashing for IP Address Lookup in Computer Networks*. IEEE ICON, 2006.

[23] Twisted - an event-driven networking engine written in Python, http://twistedmatrix.com/trac/.

[24] F. Huici, S. Niccolini, N. d'Heureuse *Protecting SIP against Very Large Flooding DoS Attacks*. GLOBECOM, 2009.