Protecting SIP Proxy Servers from `Ringing`-based Denial-of-Service Attacks

William Conner and Klara Nahrstedt
Department of Computer Science, University of Illinois at Urbana-Champaign
{wconner,klara}@uiuc.edu

Abstract

As Internet telephony systems continue to replace existing Public Switched Telephone Network systems, proxy servers running the Session Initiation Protocol (SIP) will continue to grow in importance for Voice-over-IP deployments that use SIP for call signaling. Since the protection of the global telecommunications infrastructure is critical to people's everyday lives, ensuring the availability of SIP proxy servers under attack should be a high priority. This paper first describes a disruptive denial-of-service attack that exploits the semantics of the SIP protocol to exhaust resources at a stateful SIP proxy server. Unlike previous approaches that focus on flooding-based denial-of-service attacks, we consider attacks that do not result in high incoming call traffic rates at the SIP proxy server. After describing this semantic-based attack, we then propose a new algorithm to reduce the effects of such an attack. Our algorithm has been implemented in a SIP proxy server and evaluated extensively through experiments on a local testbed.

1. Introduction

Voice-over-IP (VoIP) technology is continuing to emerge as a feasible option to replace traditional telephone systems that use the Public Switched Telephone Network (PSTN). In comparison to PSTN-based systems, VoIP allows packet-switched data networks to be utilized for telephony and is more easily integrated with Internet-based services, such as e-mail or Web browsing.

The Session Initiation Protocol (SIP) is an application-layer signaling protocol for session creation and management [1]. These sessions are typically associated with applications requiring real-time communication, such as Internet telephony or instant messaging. In addition to its use by many Internet telephony service providers, SIP is also incorporated into the IP Multimedia Subsystem (IMS) architecture defined by 3GPP [2]. The IMS architecture facilitates the delivery of Internet-based multimedia communication services to mobile users by telecommunications service providers (e.g., cellular phone service companies).

As SIP continues to play a larger role in our global telecommunications infrastructure, it will be critically important to secure SIP-based infrastructures from disruptive attacks. For example, a group of attackers might be hired by a competitor of company *X* to disable company *X*'s VoIP-based telephone system through an attack on their SIP proxy servers. A similar attack on Web servers owned by different companies from a malicious competitor has previously been reported [3]. Attacks on governmental Web servers have also been reported [4].

As VoIP systems continue to be deployed within enterprises, it is not unreasonable to assume that attackers will target SIP proxy servers in the same way that Web servers are currently targeted. Therefore, the protection of SIP-based infrastructures will be required to ensure the availability of critical telecommunication services for the enterprise.

This paper makes the following contributions:

1. We describe and measure the effects of a denial-of-service (DoS) attack on SIP proxy servers. This attack exploits the semantics of the SIP protocol without requiring a high traffic rate from the attackers. Unlike previous work [5,6], we do not focus on flooding-based DoS attacks.
2. We designed and implemented an algorithm that releases resources associated with potentially malicious transactions at the SIP proxy server when it experiences heavy load. Our algorithm makes more resources available for legitimate transactions.

In the next section, we will introduce some background information about the basic SIP call scenario that we consider throughout the paper. Section 3 describes our attack model. Section 4 presents our algorithm to provide some protection of the SIP proxy server from attacks. Experimental results appear in Section 5 and related work appears in Section 6. Finally, we conclude and briefly discuss future work in the last section.

2. Call Scenario

As mentioned in our introduction, SIP is a signaling protocol for session management. Since many different scenarios for call establishment are possible, we have decided to focus on one basic call scenario throughout this paper for the sake of clarity. Although our techniques are also applicable to more complicated scenarios with more entities and different message sequences, this basic scenario is sufficient to clearly illustrate the attack that we consider and the corresponding protection mechanism that we propose.

Our basic call scenario includes the following three logical SIP entities: user agent client (UAC), user agent server (UAS), and stateful SIP proxy server. These SIP entities and others are defined in [1]. Figure 1 depicts our basic call scenario.

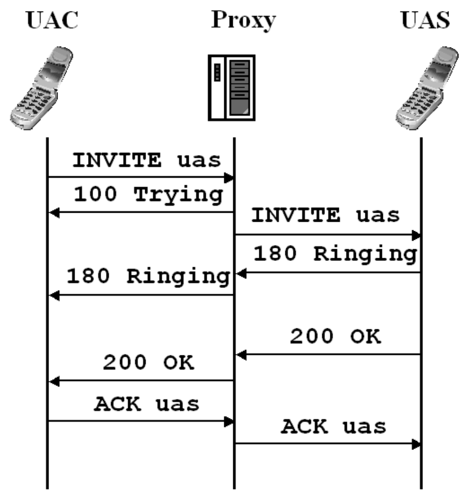

Figure 1. Basic SIP Call Scenario

In Figure 1, the UAC (i.e., caller) requests the initiation of a communication session with the UAS (i.e., callee) by sending an `INVITE` request via the proxy[1]. Upon receiving the `INVITE` at the proxy, the proxy will first send a `100 Trying` provisional response to the UAC to prevent unnecessary retransmissions of the `INVITE` request. Next, the proxy will forward the `INVITE` to the UAS. Upon receiving the `INVITE` at the UAS, the UAS will immediately send a `180 Ringing` provisional response to the UAC via the proxy. Similar to a telephone, the UAS is now in the ringing state. Once the UAS picks up the call, it sends a `200 OK` final response to the UAC via the proxy. The UAC will acknowledge reception of the `200 OK` final response with an `ACK` request. An interactive communication session has now been established between the UAC and UAS, which allows them to communicate directly.

Although it is not shown in Figure 1, an established session can be terminated when either the UAC or UAS sends a `BYE` request to the other party. The other party will then acknowledge reception of the `BYE` request by responding with a `200 OK` final response.

In Figure 1, the proxy can be thought of as a server that handles all incoming calls for the UAS. For example, a company using VoIP might issue logical SIP URIs to its employees (e.g., sip:alice@company.com). However, since employees might be at different locations[2] throughout the day, it will be necessary to map the logical SIP URIs to actual contact IP addresses (e.g., sip:alice@128.174.254.29) when an incoming call is received. In conjunction with a registrar and location service, the company can use SIP proxies to route incoming `INVITE` requests to the UAS at its current location. More details on using registrars and location services with SIP can be found in [1].

Before describing our attack model, it is important to distinguish between stateless and stateful proxies, which are both defined in RFC 3261 [1]. A stateless proxy simply forwards messages based only on the message header parameters without maintaining any transaction context. A stateful proxy will maintain transaction state to associate responses with previously handled requests. In Figure 1, the stateful proxy will create a transaction context when it receives the `INVITE` request. The proxy will release the resources associated with that transaction when it forwards the `200 OK` final response from the UAS to the UAC.

One reason for using a stateful proxy rather than a stateless proxy is to enable the deployment of richer functionality within the proxy. For example, a user agent *U* might register multiple contact IP addresses

[1] *Proxy* and *proxy server* will be used interchangeably in this paper.

[2] Example locations could possibly include a desktop computer at the office or a laptop computer at home.

for its single logical SIP URI. When a stateful proxy receives an `INVITE` request for *U*, it can fork that request by forwarding the `INVITE` request to all of the registered contact addresses for *U* in parallel. When *U* picks up the call at one contact address, the stateful proxy can then send a `CANCEL` request to the other contact addresses where *U* did not pick up the call. Stateless proxies cannot support this forking behavior since they do not maintain any transaction context. Furthermore, RFC 3261 specifically states that requests forwarded to more than one location must be handled statefully [1].

3. Attack Model

Our attack model considers DoS attacks on stateful proxies originating from attackers that do not rely on traffic flooding or deviations from the SIP protocol defined in [1]. In our attack model, malicious or compromised user agents exploit the semantics of the SIP protocol by intentionally increasing the amount of transaction state that must be maintained by a stateful proxy.

As mentioned in Section 2, a stateful proxy must maintain transaction context from the time it receives an `INVITE` request from the UAC (i.e., incoming call from caller) until it forwards the `200 OK` final response from the UAS to the UAC (i.e., call pickup at the callee). A stateful proxy with limited resources can only maintain a finite amount of transaction state before service disruptions start to occur.

Notice that the length of time for which the proxy must maintain transaction state for an `INVITE` request can be artificially increased by the UAS purposely staying in the ringing state for an unusually long amount of time before picking up the incoming call. Such UASs will be referred to as *excessively ringing* UASs. If one or more UACs repeatedly send `INVITE` requests (even at a relatively low traffic rate) to one or more excessively ringing UASs located behind a particular SIP proxy, then that stateful proxy can potentially become the target of what we call a `Ringing`-based DoS attack, which we named after the `180 Ringing` provisional response that indicates the UAS is in the ringing state.

In a `Ringing`-based DoS attack, excessively ringing UASs consume resources at the stateful proxy for a longer period of time, which makes these resources unavailable for other incoming calls. As we will show in our experimental results in Section 5, even a relatively low rate of incoming calls to excessively ringing UASs can significantly disrupt overall service.

Many possibilities exist for a user agent to become a willing or unwilling participant in a `Ringing`-based DoS attack. First, attacking UACs might collude with disgruntled insiders within an enterprise that willingly act as excessively ringing UASs. Second, attacking UACs might automatically scan a list of SIP URIs to detect unresponsive callees. These unresponsive callees might unwillingly become excessively ringing UASs during an attack since they will not pick up the incoming calls from the attacking UACs. Lastly, attacking UACs might utilize a botnet located behind a target SIP proxy to act as excessively ringing UASs. In the Web application domain, such botnets have previously been rented to disrupt online businesses [7]. Evidence of using VoIP networks to build botnets has already been reported [14].

4. Random Early Termination (RET)

Since `Ringing`-based DoS attacks depend on the delay between receiving an `INVITE` request at the proxy and forwarding the corresponding `200 OK` response, simply terminating all local transactions at the proxy associated with currently ringing calls that exceed a certain ringing time threshold might seem like a straightforward solution at first glance. However, using this simple approach by itself is not enough to meet our requirements described in the following subsection.

4.1. Requirements

The first requirement is that any solution should consider that even well-behaved UASs (i.e., callees) have different response times to incoming calls. For example, answering a cell phone from one's pocket might take less time than answering a telephone in a different room. Consequently, determining an appropriate ringing time threshold to apply to all incoming calls is not as obvious as it seems and should not be the only parameter considered.

Secondly, given the difficulty of determining absolute ringing time thresholds, it would not be desirable to terminate transactions unnecessarily when the proxy can easily handle the current amount of transaction state. Therefore, under light loads, the proxy should allow potentially legitimate incoming calls with long ringing times to have an opportunity to be picked up whenever possible.

Lastly, rather than using some rigid threshold, incoming calls should be dropped according to the duration of their ringing times. Specifically, an incoming call that has been ringing for a slightly long time should be less likely to be dropped at the proxy than a call that has been ringing for a very long time.

4.2. Algorithm

Given our requirements, we developed the Random Early Termination (RET) algorithm for dropping transactions suspected of involving excessively ringing UASs when the proxy is under heavy load. The goal of our algorithm is to give stateful proxies some protection from `Ringing`-based DoS attacks. As the name implies, RET is partly inspired by the Random Early Detection (RED) algorithm used by routers for congestion avoidance [8]. However, there are several substantial differences between the two algorithms that will become more evident in this section. For example, one major difference is that probabilistic dropping in RET considers transaction state used at the proxy by a particular incoming call. In contrast, RED considers bandwidth used at the router.

The basic idea behind RET is that stateful proxies should drop incoming calls most likely to involve excessively ringing UASs when the total amount of transaction state at the proxy becomes too large. Dropping calls involves terminating the associated local transactions at the proxy and sending the appropriate cancellation and/or error messages to the user agents involved. By dropping these incoming calls, more resources will be available for newer incoming calls that might otherwise be dropped without protection from RET.

One critical aspect in the design of RET is selecting which calls should be dropped. Upon receiving a new `INVITE` request, the proxy will initialize the corresponding transaction context. A current timestamp will then be added to this transaction context to later determine the transaction's age. To implement RET, the stateful SIP proxy maintains a sorted list of all currently active transaction contexts ordered from the oldest to the newest as shown in Figure 2. Transactions involving excessively ringing UASs are dropped from the sorted list periodically by running RET with a specified frequency.

In order to prevent a newly generated call from being dropped too soon, we added the condition that a transaction can only be eligible for termination if it's age exceeds a certain minimum ringing time threshold *MRTT*, which is a constant value that can be set as a parameter. *MRTT* is also used for probabilistic dropping as described in the following paragraphs.

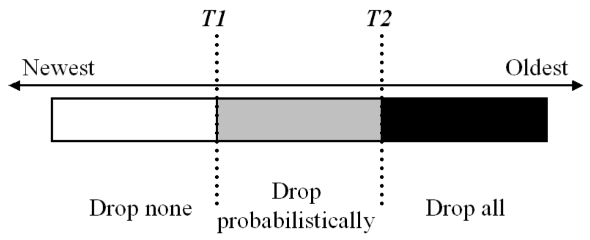

Figure 2. Sorted List of Active Transaction Contexts

Two threshold parameters *T1* and *T2* are associated with the number of active transaction contexts currently in the sorted list. All transactions from the newest transaction context up to transaction context *T1* (indicated by the white region in Figure 2) will not be considered for termination. Therefore, under a light load with no more than *T1* currently active transactions, the proxy will not drop any incoming calls regardless of how long they have been ringing. This prevents unnecessary call drops. If the number of concurrent transactions is greater than *T1*, then all transactions between *T1* up to *T2* (indicated by the gray region in Figure 2), which are older than *MRTT*, will be dropped with a certain probability *Prob* shown in Equation 1. The value *age* in Equation 1 is determined by taking the difference between the current time and the transaction context's timestamp.

$$Prob = 1 - e^{-\frac{age - MRTT}{MRTT}} \quad (1)$$

In the RET algorithm (shown later in pseudo-code), remember that only transactions older than *MRTT* are eligible for dropping. Equation 1 was chosen in order to create a substantially higher probability of dropping calls with ringing times significantly larger than *MRTT* compared to the probability of dropping calls with ringing times slightly over the *MRTT*.

Finally, if the number of concurrent transactions is greater than *T2*, then all transaction contexts between *T2* up to the oldest transaction (indicated by the black region in Figure 2), which are older than *MRTT*, will definitely be dropped.

Notice that transaction contexts are only removed from the sorted list of active transactions when one of the following events occurs: (1) stateful proxy receives a `200 OK` final response from the UAS indicating that the call was picked up, or (2) incoming call is dropped according to RET. Incoming calls with shorter ringing times are thus less likely to be dropped by RET.

Pseudo-code for the RET algorithm follows.

```
RET Algorithm.

MRTT = minimum ringing time threshold parameter
numactive = current number of transaction contexts in sorted list L

// Number of mandatory drops X
X = MAX( 0, numactive – T2 )
// Number of probabilistic drops Y
Y = MAX( 0, numactive – X – T1 )

// Currently active transaction contexts in L ordered
// from oldest to youngest (based on timestamp) starting
// from index 0 with the 0th transaction being the oldest

currenttime = current local time at SIP proxy

// Drop all transactions above T2 older than MRTT
for i = 0 to X–1
  trans = L[i]
  if ( currenttime – trans.timestamp ) > MRTT
    DROP trans

// Probabilistically drop transactions between T1 and T2
// older than MRTT
for i = X to X+Y–1
  trans = L[i]
  if ( currenttime – trans.timestamp ) > MRTT
    age = currenttime – trans.timestamp
    ratio = ( age – MRTT ) / MRTT
    DROP trans with probability 1 – exp(–ratio)
```

As shown by the pseudo-code, if the sorted list of transaction contexts has *N* transaction contexts corresponding to the state for *N* currently active transactions, then the RET algorithm will process at most *N*–*T1* transaction contexts as potential candidates for call drops. Therefore, the RET algorithm is *O*(*N*). However, in order to reduce overhead, the frequency of the RET algorithm can be adjusted. For example, our experiments in Section 5 ran the RET algorithm at the SIP proxy once every two seconds.

One consequence of using the safeguard parameter *MRTT* to prevent the dropping of recently initiated transactions is that an attacker could potentially flood the stateful proxy with many new `INVITE` requests whose resulting transactions would not be dropped due to their young age. However, such flooding-based DoS attacks have been previously addressed and are outside the scope of our work [5,6]. The RET algorithm is designed to prevent `Ringing`-based DoS attacks that exploit the semantics of the SIP protocol. Flooding-based prevention mechanisms can be combined with RET to collectively prevent both types of DoS attacks.

Another concern is that stealthy UASs might reduce their ringing times during an attack for a given traffic rate to avoid being dropped according to RET. However, by reducing their ringing times to avoid detection, the attackers would also reduce the amount of transaction state in the SIP proxy, which is our desired goal.

One final concern is that legitimate calls that happen to be ringing for a long period of time might be mistakenly dropped by RET. However, remember that the RET algorithm only drops calls when the SIP proxy is under heavy load. When a SIP proxy experiences a strain on its local resources, one side effect of RET is that calls consuming more resources (i.e., calls ringing for a longer time) would be more likely to be dropped. An argument could be made that this behavior of RET is desirable with respect to fairness. We experienced relatively few mistaken call drops during our experiments described in Section 5.

5. Performance Evaluation

In order to evaluate RET, we added code for the RET algorithm to an actual SIP proxy by modifying our JAIN SIP-based implementation of a stateful SIP proxy server [9]. User agent traffic for our experiments was generated by the open source SIPp testing tool [10]. Our local testbed had three machines connected over a local area network with each machine running one of the following: (1) emulated SIPp UACs, (2) emulated SIPp UASs, and (3) stateful JAIN SIP proxy.

We were able to control the incoming call rate from the UACs and ringing times at the UASs by adjusting the appropriate parameters in SIPp. For each experiment, we used SIPp to emulate benign UACs calling benign UASs and malicious UACs calling malicious UASs. Since both benign and malicious calls went through the same proxy in the experiments, malicious calls were able to cause disruptions in benign calls through `Ringing`-based DoS attacks on the proxy. Benign calls had ringing times uniformly distributed between 0.5 and 5 seconds while malicious calls had ringing times uniformly distributed between 30 and 120 seconds. According to RFC 3261, a UAS has up to three minutes between responses (e.g., between sending a `180 Ringing` provisional response and a `200 OK` final response), which means that the malicious calls in our experiments do not violate the SIP specification, which is consistent with our attack model that assumes no deviation by malicious user agents from the SIP protocol [1].

In all of our experiments, the aggregate incoming traffic rate at the SIP proxy was 80 calls per second (cps) for a duration of three minutes. After the three minutes of new incoming calls, we continued to observe the system until all calls had ended either successfully or unsuccessfully. The only variable in the experiments was the fraction of the fixed aggregate incoming call traffic rate originating from malicious UACs. The incoming call rate *MR* from the malicious UACs varied between 0 and 40 cps. The incoming call rate *BR* from the benign UACs was always *BR* = 80–*MR* in each experiment to maintain a fixed aggregate call rate of 80 cps.

We deliberately chose a low incoming call traffic rate to clearly demonstrate that the call failures experienced at the benign UACs were caused by the `Ringing`-based DoS attack on the proxy. If a high traffic rate had been used, then it would not be clear whether the calls failed due to bandwidth exhaustion at the proxy, which has previously been considered in flooding-based DoS attacks, or an excess amount of transaction state maintained at the proxy, which is the focus of our work.

As described in Section 4, the RET algorithm utilizes three parameters. For our experiments, the maximum ringing time threshold *MRTT* was set to ten seconds. Thus, an incoming call is only eligible to be dropped after ten seconds. The first transaction context threshold parameter *T1* was set to 250 concurrent transactions. The second transaction context threshold parameter *T2* was set to 300 concurrent transactions. The parameters *T1* and *T2* were chosen based on the observed transaction state needed at our SIP proxy to handle the specified incoming call rate of 80 cps when all calls are benign. A real deployment of a stateful SIP proxy using RET could similarly use performance testing to determine appropriate *T1* and *T2* values. The RET algorithm was run once every two seconds at the proxy during our experiments. Our measurements indicate that the average RET execution time did not exceed 20 milliseconds in any of our experiments.

We varied the incoming call rate *MR* from the malicious UACs to observe the effects when different fractions of the fixed aggregate incoming call rate belong to malicious calls. The effect of different malicious call rates on a stateful proxy that does not use RET is shown in Table 1. Table 2 shows the effect of different malicious call rates on a stateful proxy that runs the RET algorithm once every two seconds.

In Tables 1 and 2, the column *BR* corresponds to the incoming call rate from benign UACs and the column *MR* corresponds to the incoming call rate from malicious UACs. The column *MF* corresponds to *MR* divided by *BR*+*MR* (i.e., the fraction of malicious calls in the aggregate incoming call rate of 80 cps). The columns *AVG* and *PEAK* correspond to the average and peak number of concurrent transactions (NCT) in the stateful proxy, respectively. The number of concurrent transactions corresponds to the number of simultaneously active transactions requiring some transaction state in the proxy.

Table 1. Number of Concurrent Transactions (NCT) in an Unprotected Stateful SIP Proxy

BR (cps)	*MR (cps)*	*MF (MR/80)*	*AVG (NCT)*	*PEAK (NCT)*
80	0	0.00	146.9	323
76	4	0.05	319.1	609
72	8	0.10	480.4	864
64	16	0.20	757.2	1288
40	40	0.50	1441.5	2512

Table 2. Number of Concurrent Transactions (NCT) in a RET-Protected Stateful SIP Proxy

BR (cps)	*MR (cps)*	*MF (MR/80)*	*AVG (NCT)*	*PEAK (NCT)*
80	0	0.00	146.0	327
76	4	0.05	184.3	380
72	8	0.10	205.0	396
64	16	0.20	248.0	431
40	40	0.50	382.6	613

As expected, there is no substantial difference between an unprotected proxy and a RET-protected proxy when the percentage of malicious calls is 0%. However, the benefits of RET become apparent even when the percentage of malicious calls is as low as 5% (i.e., malicious UACs are only calling at a rate of 4 cps). In that case, RET reduces the average number of concurrent transactions at the proxy by 42% and reduces the peak number of concurrent transactions at the proxy by 38%. As the fraction of malicious calls *MF* increases, so does the benefit of using RET. For example, when half the calls are malicious, RET reduces the average number of concurrent transactions at the proxy by 73% and also reduces the peak number of concurrent transactions at the proxy by 76%. Based on our experiments, RET releases a significant amount of resources to reduce the amount of transaction state created by `Ringing`-based DoS attacks.

Figure 3 shows how the amount of transaction state varies over time at both unprotected and RET-protected proxies experiencing a malicious incoming call rate *MR* = 4 cps. Similarly, Figure 4 shows the transaction state over time when *MR* = 40 cps. In both figures, the gray line indicates the number of concurrent transactions at an unprotected proxy while the black line indicates the number of concurrent transactions at a RET-protected proxy.

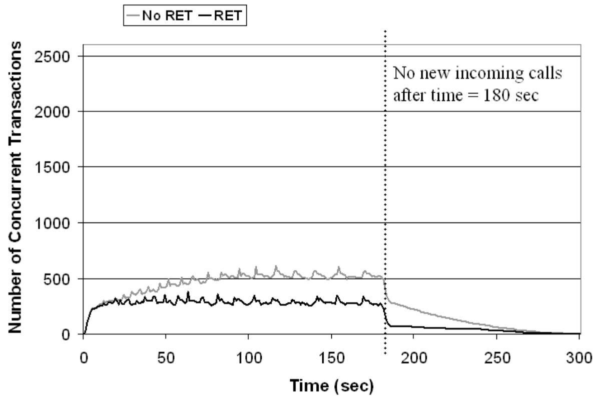

Figure 3. Transaction State over Time (*MR*=4)

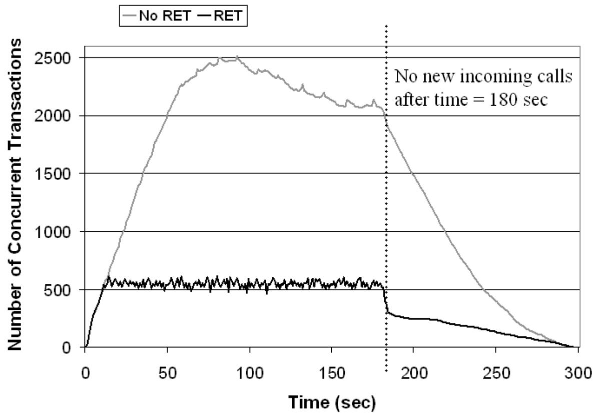

Figure 4. Transaction State over Time (*MR*=40)

In addition to measuring the amount of transaction state within the proxy, it is also important to measure the effect of `Ringing`-based DoS attacks and RET on disruptions experienced by benign user agents. Tables 3 and 4 show the number of call failures *Fails* experienced by benign UACs due to resource exhaustion at the proxy caused by the ongoing `Ringing`-based DoS attack. Tables 3 and 4 also show the number of mistakenly dropped benign calls *Drops* caused by RET. *Fails* and *Drops* are indistinguishable from the UAC's perspective. The column *Total* indicates the total number of outgoing calls from benign UACs during the experiments. Although the ringing times at benign UASs are set to be no more than five seconds, it is possible for a benign transaction to be occasionally delayed beyond ten seconds due to contention at the machines emulating the benign UASs and the proxy.

Table 3. Overall Experience at Benign UACs with an Unprotected Stateful SIP Proxy

BR (cps)	*MR (cps)*	*MF (MR/80)*	*Fails (# of calls)*	*Drops (# of calls)*	*Total (# of calls)*
80	0	0.00	0	0	14398
76	4	0.05	113	0	13679
72	8	0.10	750	0	12958
64	16	0.20	1877	0	11518
40	40	0.50	2825	0	7199

Table 4. Overall Experience at Benign UACs with a RET-Protected Stateful SIP Proxy

BR (cps)	*MR (cps)*	*MF (MR/80)*	*Fails (# of calls)*	*Drops (# of calls)*	*Total (# of calls)*
80	0	0.00	0	0	14398
76	4	0.05	0	4	13679
72	8	0.10	0	11	12958
64	16	0.20	0	35	11518
40	40	0.50	0	40	7199

Table 3 indicates that even a relatively low incoming malicious call rate *MR* of 4 cps can cause 113 call failures (i.e., *Fails*) due to transaction state overload at the proxy. Increasing the malicious call rate *MR* substantially increases the number of call failures. For example, a malicious call rate of 16 cps, which is one-fifth of the aggregate incoming call rate at the unprotected proxy, causes 1877 call failures (i.e., 16.3% call failure rate) for benign UACs. This result indicates that `Ringing`-based DoS attacks can be quite disruptive to well-behaved UACs even with relatively low malicious call rates.

The results in Table 4 indicate that RET can eliminate all of the call failures due to transaction state exhaustion at the proxy, but the possibility of mistakenly dropping benign calls (i.e., *Drops*) is introduced. For example, when the malicious call rate is 8 cps, there are 11 call drops at the RET-protected proxy in Table 4. However, it is important to note that the number of calls mistakenly dropped for a given malicious call rate in Table 4 (i.e., *Drops* column) is much less than the corresponding number of call failures for a given malicious call rate in Table 3 (i.e., *Fails* column). For example, when the malicious call rate is 40 cps in Tables 3 and 4, the RET-protected proxy has 98% fewer call drops compared to call failures at the unprotected proxy.

6. Related Work

To the best of our knowledge, previous work on protecting SIP proxies from DoS attacks has only considered flooding-based attacks [5,6]. One type of previously considered flooding-based attack involves sending `INVITE` requests at a rate high enough to overwhelm the targeted SIP proxy attempting to process all of the requests. Unlike a flooding-based attack, the `Ringing`-based DoS attack that we consider only involves exploiting the semantics of the SIP protocol to exhaust resources at the stateful proxy. The `Ringing`-based DoS attack does not require malicious UACs to induce a high incoming call rate at the proxy through flooding and would not be detected using previous solutions designed for flooding-based attacks. Therefore, RET is a complementary approach that could be used in coordination with existing mechanisms that focus on flooding-based attacks.

Some previous work has also considered protecting emergency call services (e.g., dialing '911' in the United States) in the PSTN from DoS attacks originating from VoIP networks [11]. Fuchs et al. apply intrusion detection techniques at the emergency call center to address this problem. In contrast, RET focuses on protecting proxies within the VoIP infrastructure, rather than PSTN.

Other work related to SIP security has considered many topics ranging from billing attacks that lead to overcharging of VoIP subscribers [12] to intrusion detection for VoIP systems [13]. Although interesting, these topics are beyond the scope of our work.

7. Conclusion and Future Work

Our experimental results indicate that `Ringing`-based DoS attacks can be quite disruptive by exhausting the limited resources of a stateful SIP proxy. Unlike flooding-based DoS attacks, these `Ringing`-based DoS attacks do not require attackers to induce a high incoming traffic rate at the proxy. To mitigate the negative effects of such an attack, we designed and implemented the RET algorithm that selectively drops calls that have been ringing for a long time to release the associated transaction resources at the proxy. Our performance evaluation demonstrates that RET is very effective at protecting stateful proxies from `Ringing`-based DoS attacks.

Currently, we only consider attacks on a single proxy. In the future, we plan to expand on this work by considering the protection of entire distributed infrastructures consisting of several SIP proxy servers.

References

[1] 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.

[2] K. Munasinghe and A. Jamalipour, "A 3GPP-IMS based approach for converging next generation mobile data networks", *IEEE International Conference on Communications (ICC)*, 2007.

[3] K. Poulsen, "FBI busts alleged DDoS mafia", *http://www.securityfocus.com/news/9411*, August 2004.

[4] M. Lesk, "The new front line", *IEEE Security and Privacy*, vol. 5, no. 4, July/August 2007.

[5] B. Reynolds and D. Ghosal, "Secure IP telephony using multi-layered protection", *Network and Distributed System Symposium (NDSS)*, 2003.

[6] H. Sengar, H. Wang, D. Wijesekera, and S. Jajodia, "Fast detection of denial-of-service attacks on IP telephony", *IEEE International Workshop on Quality of Service (IWQoS)*, 2006.

[7] S. Kandula, D. Katabi, M. Jacob, and A. Berger, "Botz-4-sale: surviving organized DDoS attacks that mimic flash crowds", *Networked Systems Design and Implementation (NDSI)*, 2005.

[8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, August 1993.

[9] Java API for SIP Signaling, *https://jain-sip.dev.java.net/*.

[10] SIPp, *http://sipp.sourceforge.net/index.html*.

[11] C. Fuchs, N. Aschenbruck, F. Leder, and P. Martini, "Detecting VoIP based DoS attacks at the public safety answering point", *ACM Symposium on Information, Computer, and Communications Security (ASIACCS)*, 2008.

[12] R. Zhang, X. Wang, X. Yang, and X. Jiang, "Billing attacks on SIP-based VoIP systems", *USENIX Workshop on Offensive Technologies (WOOT)*, 2007.

[13] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia, "VoIP intrusion detection through interacting protocol state machines", *International Conference on Dependable Systems and Networks (DSN)*, 2006.

[14] M. Hines, "Attackers get chatty on VoIP", *http://www.pcworld.com/businesscenter/article/132389/attackers_get_chatty_on_voip.html*, May 2007.