Network Working Group
Internet-Draft
Updates: 3261 (if approved)
Expires: December 27, 2006

R. Sparks, Ed.
Estacado Systems
S. Lawrence
Pingtel Corp.
A. Hawrylyshen
Ditech Networks Inc.

June 25, 2006

Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Forking Proxies

draft-ietf-sip-fork-loop-fix-02

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 27, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document normatively updates RFC 3261, the Session Initiation Protocol (SIP), to address a security vulnerability identified in SIP proxy behavior. This vulnerability enables an attack against SIP networks where a small number of legitimate, even authorized, SIP requests can stimulate massive amounts of proxy-to-proxy traffic.

This document strengthens loop-detection requirements on SIP proxies when they fork requests (that is, forward a request to more than one destination). It also corrects and clarifies the description of the loop-detection algorithm such proxies are required to implement.

Table of Contents

1. Conventions and Definitions 3
2. Introduction 3
3. Vulnerability: Leveraging Forking to Flood a Network 3
4. Normative changes to RFC 3261 5
  4.1. Strengthening the requirement to perform loop-detection . 5
  4.2. Correcting and clarifying the RFC 3261 loop-detection algorithm 6
    4.2.1. Update to section 16.6 6
    4.2.2. Update to section 16.3 7
    4.2.3. Note to Implementers 8
5. Impact on overall network performance 8
6. IANA Considerations 9
7. Security Considerations 9
8. Acknowledgments 10
9. Change Log 10
  9.1. -01 to -02 10
10. References 11
  10.1. Normative References 11
  10.2. Informative References 11
Authors' Addresses 12
Intellectual Property and Copyright Statements 13

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

2. Introduction

Interoperability testing uncovered a vulnerability in the behavior of forking SIP proxies as defined in [RFC3261]. This vulnerability can be leveraged to cause a small number of valid SIP requests to generate an extremely large number of proxy-to-proxy messages. A version of this attack demonstrates fewer than ten messages stimulating potentially 2^70 messages.

This document specifies normative changes to the SIP protocol to address this vulnerability. According to this update, when a SIP proxy forks a request to more than one destination, it is required to ensure it is not participating in a request loop.

3. Vulnerability: Leveraging Forking to Flood a Network

This section describes setting up an attack with a simplifying assumption, that two accounts on each of two different RFC 3261 compliant proxy/registrar servers that do not perform loop-detection are available to an attacker. This assumption is not necessary for the attack, but makes representing the scenario simpler. The same attack can be realized with a single account on a single server.

Consider two proxy/registrar services, P1 and P2, and four Addresses of Record, a@P1, b@P1, a@P2, and b@P2. Using normal REGISTER requests, establish bindings to these AoRs as follows (non-essential details elided):

```
REGISTER sip:P1 SIP/2.0
To: <sip:a@P1>
Contact: <sip:a@P2>, <sip:b@P2>

REGISTER sip:P1 SIP/2.0
To: <sip:b@P1>
Contact: <sip:a@P2>, <sip:b@P2>

REGISTER sip:P2 SIP/2.0
To: <sip:a@P2>
Contact: <sip:a@P1>, <sip:b@P1>

REGISTER sip:P2 SIP/2.0
To: <sip:b@P2>
Contact: <sip:a@P1>, <sip:b@P1>
```

With these bindings in place, introduce an INVITE to any of the four AoRs, say a@P1. This request will fork to two requests handled by P2, which will fork to four requests handled by P1, which will fork to eight messages handled by P2, and so on. This message flow is represented in Figure 2.

```
                                          |
                                        a@P1
                                      /       \
                                    /           \
                                  /               \
                                /                   \
                             a@P2                   b@P2
                             /  \                   /  \
                           /      \               /      \
                          /        \             /        \
                        a@P1       b@P1        a@P1       b@P1
                        /  \       /  \        /  \       /  \
                     a@P2  b@P2 a@P2  b@P2  a@P2  b@P2 a@P2  b@P2
                      /\    /\   /\    /\    /\    /\   /\    /\
                                          .
                                          .
                                          .
```

Figure 2: Attack request propagation

Requests will continue to propagate down this tree until Max-Forwards reaches zero. If the endpoint and two proxies involved follow RFC 3261 recommendations, the tree will be 70 rows deep, representing 2^70 requests. The actual number of messages may be much larger if the time to process the entire tree worth of requests is longer than Timer C at either proxy. In this case, a storm of 408s, and/or a

storm of CANCELs will also be propagating through the tree along with the INVITEs. Remember that there are only two proxies involved in this scenario - each having to hold the state for all the transactions it sees (at least 2^69 simultaneously active transactions near the end of the scenario).

The attack can be simplified to one account at one server if the service can be convinced that contacts with varying attributes (parameters, schemes, embedded headers) are sufficiently distinct, and these parameters are not used as part of AOR comparisons when forwarding a new request. Since RFC 3261 mandates that all URI parameters must be removed from a URI before looking it up in a location service and that the URIs from the Contact header are compared using URI equality, the following registration should be sufficient to set this attack up using a single REGISTER request to a single account:

```
REGISTER sip:P1 SIP/2.0
To: <sip:a@P1>
Contact: <sip:a@P1;unknown-param=whack>,<sip:a@P1;unknown-param=thud>
```

This attack was realized in practice during one of the SIP Interoperability Test (SIPit) sessions. The scenario was extended to include more than two proxies, and the participating proxies all limited Max-Forwards to be no larger than 20. After a handful of messages to construct the attack, the participating proxies began bombarding each other. Extrapolating from the several hours the experiment was allowed to run, the scenario would have completed in just under 10 days. Had the proxies used the RFC 3261 recommended Max-Forwards value of 70, and assuming they performed linearly as the state they held increases, it would have taken 3 trillion years to complete the processing of the single INVITE that initiated the attack. It is interesting to note that a few proxies rebooted during the scenario, and rejoined in the attack when they restarted (as long as they maintained registration state across reboots). This points out that if this attack were launched on the Internet at large, it might require coordination among all the affected elements to stop it.

4. Normative changes to RFC 3261

4.1. Strengthening the requirement to perform loop-detection

The following requirements mitigate the risk of a proxy falling victim to the attack described in this document.

When a SIP proxy forks a particular request to more than one destination, it MUST ensure that request is not looping through this proxy. It is RECOMMENDED that proxies meet this requirement by performing the Loop-Detection steps defined in this document.

The requirement to use this document's refinement of the loop-detection algorithm in RFC 3261 is set at should-strength to allow for future standards track mechanisms that will allow a proxy to determine it is not looping. For example, a proxy forking to destinations established using the sip-outbound mechanism [I-D.ietf-sip-outbound] would know those branches will not loop.

A SIP proxy forwarding a request to only one location MAY perform loop detection but is not required to. When forwarding to only one location, the amplification risk being exploited is not present, and the Max-Forwards mechanism is sufficient to protect the network. A proxy is not required to perform loop detection when forwarding a request to a single location even if it happened to have previously forked that request (and performed loop detection) in its progression through the network.

4.2. Correcting and clarifying the RFC 3261 loop-detection algorithm

4.2.1. Update to section 16.6

This section replaces all of item 8 in section 16.6 of RFC 3261 (item 8 begins on page 105 and ends on page 106 of RFC 3261).

8. Add a Via header field value

The proxy MUST insert a Via header field value into the copy before the existing Via header field values. The construction of this value follows the same guidelines of Section 8.1.1.7. This implies that the proxy will compute its own branch parameter, which will be globally unique for that branch, and will contain the requisite magic cookie. Note that following only the guidelines in Section 8.1.1.7 will result in a branch parameter that will be different for different instances of a spiraled or looped request through a proxy.

Proxies required to perform loop-detection by RFC XXXX (RFC-Editor: replace XXXX with the RFC number of this document) have an additional constraint on the value they place in the Via header field. Such proxies SHOULD crate a branch value separable into two parts in any implementation dependent way. The first part MUST satisfy the constraints of Section 8.1.1.7. The second part is used to perform loop detection and distinguish loops from spirals.

This second part MUST vary with any field used by the location

service logic in determining where to retarget or forward this request. This is necessary to distinguish looped requests from spirals by allowing the proxy to recognize if none of the values affecting the processing of the request have changed. Hence, The second part MUST depend at least on the received Request-URI and any Route header field values in the received request. Implementers need to take care to include all fields used by the location service logic in that particular implementation.

This second part MUST NOT include the request method. CANCEL and non-200 ACK requests MUST have the same branch parameter value as the corresponding request they cancel or acknowledge. This branch parameter value is used in correlating those requests at the server handling them (see Sections 17.2.3 and 9.2).

Open Issue 1 : Why do all the Route header field values need to be included? Only the top two in the received request will normally influence processing (in general, the topmost and any others this proxy removes from the message, plus the topmost in the message the proxy emits). Can we simplify this to only include those Route header field values that actually affect the processing of the message?

Open Issue 2 : RFC 3261 recommends Call-ID, To-tag, and From-tag and CSeq be included in this second part. Why? This are guaranteed to be invariant each time the request passes through this proxy. Further the created value will only be compared with Vias received from previous times the request went through this proxy, so there is no possibility of comparing this value with a hash created from some other request with a different Call-ID, To-tag, or From-tag or CSeq. I've removed these fields from the recommended set for this version of the draft.

Open Issue 3 : RFC 3261 recommends including any Proxy-Require and Proxy-Authorization header field values in the second part. Why it it valuable to call these out as special cases? What is the use case where they will be the only thing in a message returning to a proxy that varies, making the return a spiral instead of a loop? I haven't been able to remember the motivation - we need to re-identify that and capture it here or remove the text making these anything other than a case covered by the abstract "fields used by the location service logic". This draft is taking the second option.

4.2.2. Update to section 16.3

This section replaces all of item 4 in section 16.3 of RFC 3261 (item 4 appears on page 95 RFC 3261).

4. Loop Detection Check

Proxies required to perform loop-detection by RFC-XXXX (RFC-Editor: replace XXXX with the RFC number of this document) MUST perform the following loop-detection test before forwarding a request. Each Via header field value in the request whose sent-by value matches a value placed into previous requests by this proxy MUST be inspected for the "second part" defined in Section 4.2.1 of RFC-XXXX. This second part will not be present if the message was not forked when that Via header field value was added. If the second field is present, the proxy MUST perform the second part calculation described in Section 4.2.1 of RFC-XXXX on this request and compare the result to the value from the Via header field. If these values are equal, the request has looped and the proxy MUST reject the request with a 482 (Loop Detected) response. If the values differ, the request is spiraling and processing continues to the next step.

4.2.3. Note to Implementers

A common way to create the second part of the branch parameter value when forking a request is to compute a cryptographic hash over the concatenation of the Request-URI, all Route header field values, and any other values used by the location service logic while processing this request. An MD5 [RFC1321] hash expressed in hexadecimal (base64 is not permissible for a token) is a reasonable choice of a hash function.

A common point of failure to interoperate at SIPit events has been due to parsers objecting to the contents of other's Via header field values when inspecting the Via stack for loops. Implementers need to take care to avoid making assumptions about the format of another element's Via header field value beyond the basic constraints placed on that format by RFC 3261. In particular, parsing a header field value with unknown parameter names, parameters with no values, parameters values with and without quoted strings must not cause an implementation to fail.

5. Impact on overall network performance

These requirements and the recommendation to use the loop-detection mechanisms in this document make the favorable trade of exponential message growth for work that is at worst case order n^2 as a message crosses n proxies. Specifically, this work is order m*n where m is the number of proxies in the path that fork the request to more than one location. In practice, m is expected to be small.

The loop detection algorithm expressed in this document requires a proxy to inspect each Via element in a received request. In the worst case where a message crosses N proxies, each of which loop detect, proxy k does k inspections, and the overall number of inspections spread across the proxies handling this request is the sum of k from k=1 to k=N = N(N+1)/2.

6. IANA Considerations

None.

7. Security Considerations

This document is entirely about documenting and addressing a vulnerability in SIP proxies as defined by RFC 3261 that can lead to an exponentially growing message exchange attack.

Alternative solutions that were discussed included

Doing nothing - rely on suing the offender : While systems that have accounts have logs that can be mined to locate abusers, it isn't clear that this provides a credible deterrent or defense against the attack described in this document. Systems that don't recognize the situation and take corrective/preventative action are likely to experience failure of a magnitude that precludes retrieval of the records documenting the setup of the attack. (In one scenario, the registrations can occur in a radically different time period than the invite. The invite itself may have come from an innocent). It's even possible that the scenario may be set up unintentionally. Furthermore, for some existing deployments, the cost and audit ability of an account is simply an email address. Finding someone to punish may be impossible. Finally, there are individuals who will not respond to any threat of legal action, and the effect of even a single successful instance of this kind of attack would be devastating to a service-provider.

Putting a smaller cap on Max-Forwards: The effect of the attack is exponential with respect to the initial Max-Forwards value. Turning this value down limits the effect of the attack. This comes at the expense of severely limiting the reach of requests in the network, possibly to the point that existing architectures will begin to fail.

Controlling the number of concurrent requests : Bounding the total number branches to which the original request can be forwarded simultaneously limits the impact of the attack at any given point in time. Proposals for limiting mechanisms where considered, but no consensus to adopt them currently exists.

Disallowing registration bindings to arbitrary contacts : The way registration binding is currently defined is a key part of the success of the kind of attack documented here. The alternative of limiting registration bindings to allow only binding to the network element performing the registration, perhaps to the extreme of ignoring bits provided in the Contact in favor of transport artifacts observed in the registration request has been discussed (particularly in the context of the mechanisms being defined in [I-D.ietf-sip-outbound]. Mechanisms like this may be considered again in the future, but are currently insufficiently developed to address the present threat.

Deprecate forking : This attack does not exist in a system that relies entirely on redirection and initiation of new requests by the original endpoint. Removing such a large architectural component from the system at this time was deemed a too extreme solution.

8. Acknowledgments

Thanks go to the implementors that subjected their code to this scenario and helped analyze the results at SIPit 17.

9. Change Log

RFC Editor - Remove this section before publication

9.1. -01 to -02

Integrated several editorial fixes suggested by Jonathan Rosenberg

Noted that the reduction of the attack to a single registration against a single URI as documented in previous versions, is, in fact, going to be effective against implementations conforming to the standards before this repair.

Re-incorporated motivation from the original maxforwards-problem draft into the security considerations section based on feedback from Cullen Jennings

Introduced replacement text for the loop detection algorithm description in RFC 3261, fixing the bug 648 (the topmost Via value must not be included in the second part) and clarifying the algorithm. Removed several other fields suggested by 3261 and placed open issues around their presence.

Added a Notes to Implementors section capturing the "common way" text and pointing to the interoperability issues that have been observed with loop detection at previous SIPits

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

10.2. Informative References

[I-D.ietf-sip-outbound] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", draft-ietf-sip-outbound-03 (work in progress), March 2006.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

Authors' Addresses

Robert Sparks (editor)
Estacado Systems
17210 Campbell Road
Suite 250
Dallas, Texas 75254-4203
USA

Email: RjS@nostrum.com

Scott Lawrence
Pingtel Corp.
400 West Cummings Park
Suite 2200
Woburn, MA 01801
USA

Phone: +1 781 938 5306
Email: slawrence@pingtel.com

Alan Hawrylyshen
Ditech Networks Inc.
1167 Kensington Rd NW
Suite 200
Calgary, Alberta T2N 1X7
Canada

Phone: +1 403 806 3366
Email: ahawrylyshen@ditechnetworks.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.