Network Working Group
Internet-Draft
Updates: 3261 (if approved)
Intended status: Standards Track
Expires: January 4, 2009

R. Sparks, Ed.
Tekelec
S. Lawrence
Bluesocket Inc.
A. Hawrylyshen
Ditech Networks Inc.
B. Campen
Tekelec

July 3, 2008

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

draft-ietf-sip-fork-loop-fix-07

Status of this Memo

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. Additionally, this document defines a Max-Breadth mechanism for limiting the number of concurrent branches pursued for any given request.

Table of Contents

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 2^71 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.

This normative update alone is insufficient to protect against crafted variations of the attack described here involving multiple AORs. To further address the vulnerability, this document defines the Max-Breadth mechanism to limit the total number of concurrent branches caused by a forked SIP request. The mechanism only limits concurrency. It does not limit the total number of branches a request can traverse over its lifetime.

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 1.

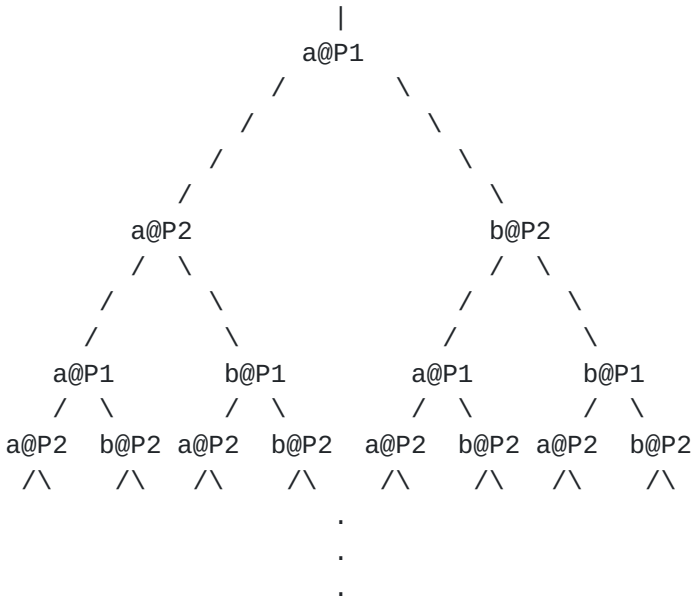


Figure 1: Attack request propagation

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 2^71-1 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 2^70 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.

Loop-detection, as specified in this document, at any of the proxies in the scenarios described so far would have stopped the attack immediately. (If all the proxies involved implemented this loop-detection, the total number of stimulated messages in the first scenario described is reduced to 14, and in the variation involving one server, the number of stimulated messages is reduced to 10.) However, there is a variant of the attack that uses multiple AORs where loop-detection alone is insufficient protection. In this

variation, each participating AOR forks to all the other participating AORs. For small numbers of participating AORs (10 example), paths through the resulting tree will not loop until very large numbers of messages have been generated. Acquiring a sufficient number of AORs to launch such an attack on networks currently available is quite feasible.

In this scenario, requests will often take many hops to complete a loop, and there are a very large number of different loops that will occur during the attack. In fact, if N is the number of participating AORs, and provided N is less than or equal to Max-Forwards, the amount of traffic generated by the attack is greater than N!, even if all proxies involved are performing loop-detection.

Suppose we have a set of N AORs, all of which are set up to fork to the entire set. For clarity, assume AOR 1 is where the attack begins. Every permutation of the remaining N-1 AORs will play out, defining (N-1)! distinct paths, without repeating any AOR. Then, each of these paths will fork N ways one last time, and a loop will be detected on each of these branches. These final branches alone total N! requests ((N-1)! paths, with N forks at the end of each path).

Forwarded Requests vs. Number of Participating AORs

N	Requests
1	1
2	4
3	15
4	64
5	325
6	1956
7	13699
8	109600
9	986409
10	9864100

Forwarded Requests vs. Number of Participating AORs

In a network where all proxies are performing loop-detection, an attacker is still afforded rapidly increasing returns on the number of AORs they are able to leverage. The Max-Breadth mechanism defined in this document is designed to limit the effectiveness of this variation of the attack.

In all of the scenarios, it is important to notice that at each forking proxy, an additional branch could be added pointing to a

single victim (that might not even be a SIP-aware element), resulting in a massive amount of traffic being directed towards the victim from potentially as many sources as there are AORs participating in the attack.

4. Updates 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 location, 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 will protect the network to the extent it was designed to do (always keep the constant multiplier due to exhausting Max-Forwards while not forking in mind.) 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 create 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 used when processing 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 vary with 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).

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. Impact of Loop-detection 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 which is N(N+1)/2.

4.2.4. Note to Implementors

A common way to create the second part of the branch parameter value when forking a request is to compute a hash over the concatenation of the Request-URI, any Route header field values used during processing the request and any other values used by the location service logic while processing this request. The hash should be chosen so that there is a low probability that two distinct sets of these parameters will collide. Because the maximum number of inputs which need to be compared is 70 the chance of a collision is low even with a relatively small hash value, such as 32 bits. CRC-32c as specified in [RFC4960] is a specific acceptable function, as is MD5 [RFC1321]. Note that MD5 is being chosen purely for non-cryptographic properties. An attacker who can control the inputs in order to produce a hash collision can attack the connection in a variety of other ways. When forming the second part using a hash, implementations SHOULD include at least one field in the input to the hash that varies between different transactions attempting to reach the same destination to avoid repeated failure should the hash collide. The Call-ID and CSeq fields would be good inputs for this purpose.

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.

Removing, obfuscating, or in any other way modifying the branch parameter values in Via header fields in a received request before forwarding it removes the ability for the node that placed that branch parameter into the message to perform loop-detection. If two elements in a loop modify branch parameters this way, a loop can never be detected.

5. Max-Breadth

5.1. Overview

The Max-Breadth mechanism defined here limits the total number of concurrent branches caused by a forked SIP request. With this mechanism, all proxyable requests are assigned a positive integral Max-Breadth value, which denotes the maximum number of concurrent branches this request may spawn through parallel forking as it is forwarded from its current point. When a proxy forwards a request, its Max-Breadth value is divided among the outgoing requests. In turn, each of the forwarded requests has a limit on how many concurrent branches they may spawn. As branches complete, their portion of the Max-Breadth value becomes available for subsequent branches, if needed. If there is insufficient Max-Breadth to carry out a desired parallel fork, a proxy can return the 440 (Max-Breadth Exceeded) response defined in this document.

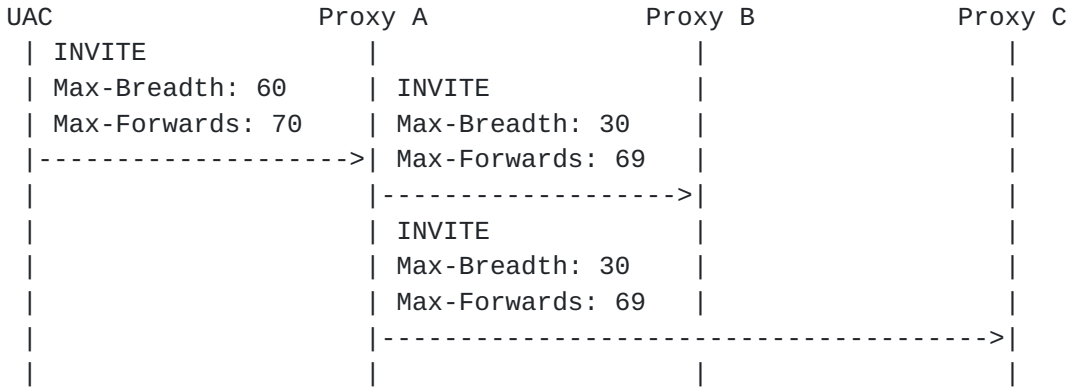
This mechanism operates independently from Max-Forwards. Max-Forwards limits the depth of the tree a request may traverse as it is forwarded from its origination point to each destination it may be forked to. As Section 3 shows, the number of branches in a tree of even limited depth can be made large (exponential with depth) by leveraging forking. Each such branch has a pair of SIP transaction state machines associated with it. The Max-Breadth mechanism limits the number of branches that are active (those that have running transaction state machines) at any given point in time.

Max-Breadth does not prevent forking. It only limits the number of concurrent parallel forked branches. In particular, a Max-Breadth of 1 restricts a request to pure serial forking rather than restricting it from being forked at all.

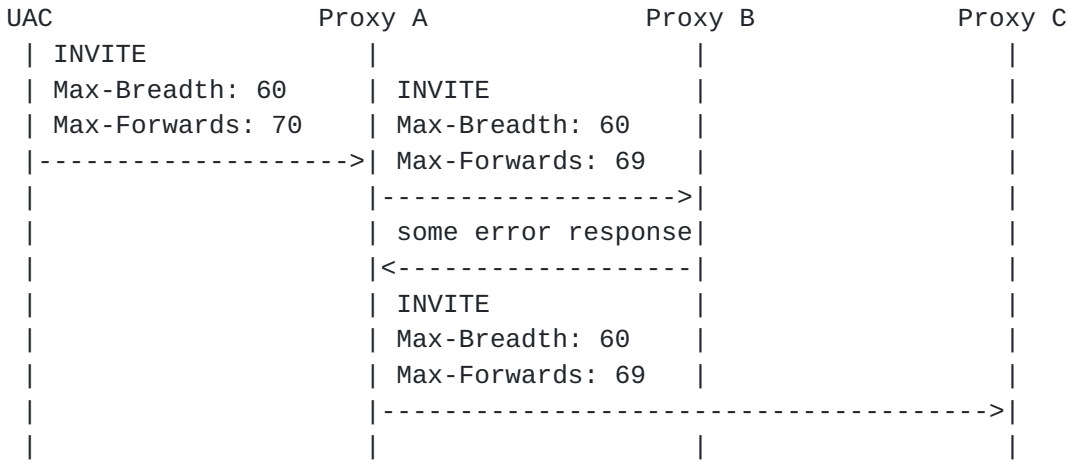
A client receiving a 440 (Max-Breadth Exceeded) response can infer that it its request did not reach all possible destinations. Recovery options are similar to those when receiving a 483 (Too Many

Hops) response, and include affecting the routing decisions through whatever mechanisms are appropriate to result in a less broad search, or refining the request itself before submission to make the search space smaller.

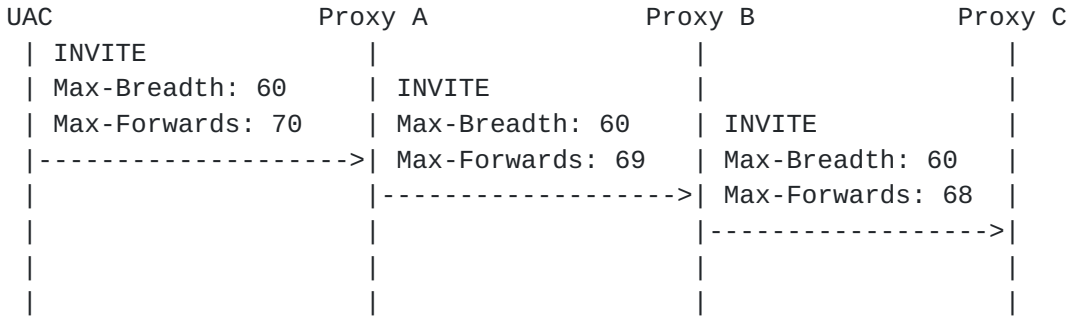
5.2. Examples

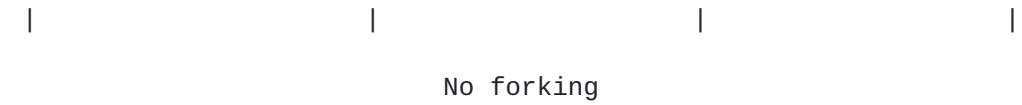


Parallel forking

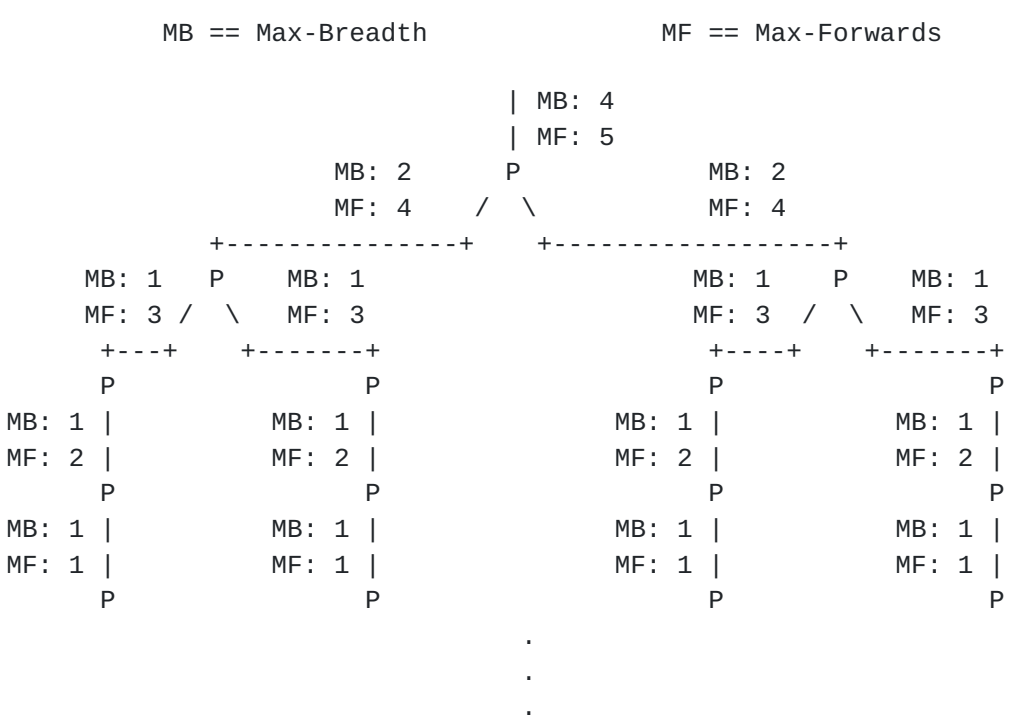


Sequential forking





No forking



Max-Breadth and Max-Forwards working together

5.3. Formal Mechanism

5.3.1. "Max-Breadth" Header

The Max-Breadth header takes a single positive integer as its value. The Max-Breadth header takes no parameters.

5.3.2. Terminology

For each "response context" (see [RFC3261] Sec 16) in a proxy, this mechanism defines two positive integral values; Incoming Max-Breadth and Outgoing Max-Breadth. Incoming Max-Breadth is the value of the Max-Breadth header field value in the request that formed the response context. Outgoing Max-Breadth is the sum of the Max-Breadth of all forwarded requests in the response context, that have not received a final response.

5.3.3. Proxy Behavior

If a SIP proxy receives a request with no Max-Breadth header field value, it MUST add one, with a value that is RECOMMENDED to be 60. Proxies MUST have a maximum allowable Incoming Max-Breadth value, which is RECOMMENDED to be 60. If this maximum is exceeded in a received request, the proxy MUST overwrite it with a value that SHOULD be no greater than its allowable maximum.

All proxied requests MUST contain a single Max-Breadth header field value.

SIP proxies MUST NOT allow the Outgoing Max-Breadth to exceed the Incoming Max-Breadth in a given response context.

If a SIP proxy determines a response context has insufficient Incoming Max-Breadth to carry out a desired parallel fork, and the proxy is unwilling/unable to compensate by forking serially or sending a redirect, that proxy MUST return a 440 (Max-Breadth Exceeded) response.

Notice that these requirements means a proxy receiving a request with a Max-Breadth of 1 can only fork serially, but it is not required to fork at all - it can return a 440 instead. Thus, this mechanism is not a tool a user-agent can use to force all proxies in the path of a request to fork serially.

A SIP proxy MAY distribute Max-Breadth in an arbitrary fashion between active branches. A proxy SHOULD NOT use a smaller amount of Max-Breadth than was present in the original request, unless the Incoming Max-Breadth exceeded the proxy's maximum acceptable value. A proxy MUST NOT decrement Max-Breadth for each hop or otherwise use it to restrict the "depth" of a request's propagation.

5.3.3.1. Reusing Max-Breadth

Because forwarded requests that have received a final response do not count towards the Outgoing Max-Breadth, whenever a final response arrives, the Max-Breadth that was used on that branch becomes available for reuse. Proxies SHOULD be prepared to reuse this Max-Breadth in cases where there may be elements left in the target-set.

5.3.4. UAC Behavior

A UAC MAY place a Max-Breadth header field value in outgoing requests. If so, this value is RECOMMENDED to be 60.

5.3.5. UAS behavior

This mechanism does not affect UAS behavior. A UAS receiving a request with a Max-Breadth header field will ignore that field while processing the request.

5.4. Implementor Notes

5.4.1. Treatment of CANCEL

Since CANCEL requests are never proxied, a Max-Breadth header-field-value is meaningless in a CANCEL request. Sending a CANCEL in no way effects the Outgoing Max-Breadth in the associated INVITE response context. Receiving a CANCEL in no way effects the Incoming Max-Breadth of the associated INVITE response context.

5.4.2. Reclamation of Max-Breadth on 2xx Responses

Whether 2xx responses free up Max-Breadth is mostly a moot issue, since proxies are forbidden to start new branches in this case. But, there is one caveat. For INVITE, we may receive multiple 2xx for a single branch. Also, 2543 implementations may send back a 6xx followed by a 2xx on the same branch. Implementations that subtract from the Outgoing Max-Breadth when they receive an INVITE/2xx must be careful to avoid bugs caused by subtracting multiple times for a single branch.

5.4.3. Max-Breadth and Automaton UAs

Designers of automaton UAs (including B2BUAs, gateways, exploders, and any other element that programmatically sends requests as a result of incoming SIP traffic) should consider whether Max-Breadth limitations should be placed on outgoing requests. For example, it is reasonable to design B2BUAs to carry the Max-Breadth value from incoming requests over into requests that are sent as a result. Also, it is reasonable to place Max-Breadth constraints on sets of requests sent by exploders, when they may be leveraged in an amplification attack.

5.5. Parallel and Sequential Forking

Inherent in the definition of this mechanism is the ability of a proxy to reclaim apportioned Max-Breadth while forking sequentially. The limitation on outgoing Max-Breadth is applied to concurrent branches only.

For example, if a proxy receives a request with a Max-Breadth of 4, and has 8 targets to forward it to, that proxy may parallel fork to 4

of these targets initially (each with a Max-Breadth of 1, totaling an Outgoing Max-Breadth of 4). If one of these transactions completes with a failure response, the outgoing Max-Breadth drops to 3, allowing the proxy to forward to one of the 4 remaining targets (again, with a Max-Breadth of 1).

5.6. Max-Breadth Split Weight Selection

There are a variety of mechanisms for controlling the weight of each fork branch. Fork branches that are given more Max-Breadth are more likely to complete quickly (because it is less likely that a proxy down the line will be forced to fork sequentially). By the same token, if it is known that a given branch will not fork later on, a Max-Breadth of 1 may be assigned with no ill effect. This would be appropriate, for example, if a proxy knows the branch is using the SIP outbound extension [I-D.ietf-sip-outbound].

5.7. Max-Breadth's Effect on Forking-based Amplification Attacks

Max-Breadth limits the total number of active branches spawned by a given request at any one time, while placing no constraint on the distance (measured in hops) that the request can propagate. (ie, receiving a request with a Max-Breadth of 1 means that any forking must be sequential, not that forking is forbidden)

This limits the effectiveness of any amplification attack that leverages forking, because the amount of state/bandwidth needed to process the traffic at any given point in time is capped.

5.8. Max-Breadh Header Field ABNF Definition

This specification extends the grammar for the Session Initation Protocol by adding the following extension-header:

```
Max-Breadth = "Max-Breadth" HCOLON 1*DIGIT
```

6. IANA Considerations

This specification registers a new SIP header field and a new SIP response according to the processes defined in [RFC3261].

6.1. Max-Forwards Header Field

This information should appear in the header sub-registry under http://www.iana.org/assignments/sip-parameters.

RFC XXXX (this specification)

Header Field Name: Max-Breadth

Compact Form: none

6.2. 440 Max-Breadth Exceeded response

This information should appear in the response-code sub-registry under http://www.iana.org/assignments/sip-parameters.

Response code: 440

Default Reason Phrase: Max-Breadth Exceeded

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.

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.

The Max-Breadth mechanism defined here does not decrease the aggregate traffic caused by the forking-loop attack. It only serves to spread the traffic caused by the attack over a longer period, by limiting the number of concurrent branches that are being processed at the same time. An attacker could pump multple requests into a network that uses the Max-Breadth mechanism and gradually build traffic to unreasonable levels. Deployments should monitor carefully and react to gradual increases in the number of concurrent outstanding transactions related to a given resource to protect against this possibility. An alternative design of the Max-Breadth mechanism that was considered and rejected was to not allow the breadth from completed branches to be reused Section 5.3.3.1. Under this alternative, an introduced request would cause at most the initial value of Max-Breadth transactions to be generated in the network. While that approach limits any variant of the amplification vulnerability described here to a constant multipler, it would dramatically change the potential reach of requests and there is beleif that it would break existing deployments.

8. Acknowledgments

Thanks go to the implementors that subjected their code to this scenario and helped analyze the results at SIPit 17. Eric Rescorla provided guidance and text for the hash recommendation note.

9. Change Log

RFC Editor - Remove this section before publication

9.1. -06 to -07

Cleaning up some things based on WGLC and review for publication request (like refreshing references)

Added a sentence to the overview discussing what a client might do if it got a 440

Reinforced that a UAC will ignore a Max-Breadth header

Updated the reference to CRC32C - from 3309 to 4960

Integrated fixes from Jan Kolomaznik's review

9.2. -05 to -06

Integrated Max-Breadth based on working group discussion of the secdir review

Added a paragraph pointing out that removing or modifying other node's branch parameters defeats their ability to loop detect

Moved the total number of messages from O(2^70) to O(2^71) based on an observation by Jan Kolomaznik. To see this, note that the total number of requests is the sum from i=0 to Max-Forwards of 2^i which is 2^(Max-Forwards+1) - 1. The point of the text doesn't change - (the point being that the number is _big_).

Made the new 4xx concrete (choosing 440)

Added a sentence reinforcing that if you forward to only one branch, you still potentially have a constant multiplier of messages in the network as Max-Forwards runs out (based on feedback from Thomas Cross.)

9.3. -04 to -05

Boilerplate update, editorial nits fixed

9.4. -03 to -04

Addressed WGLC comments

Changed the hash recommendation per list consensus

Reintroduced Call-ID and CSeq (list discussion rediscovered one use for them in avoiding repeated hash collisions)

9.5. -02 to -03

Closed Open Issue 1 "Why are we including all of the Route headers values?". The text has been modified to include only those values used in processing the request.

Closed Open Issues 2 and 3 "Why did 3261 include Call-ID To-tag, and From-tag and CSeq?" and "Why did 3261 include Proxy-Require and Proxy-Authorization?". The group has not been able to identify why these fields would be included in the hash generally, and successful interoperability tests have not included them. Since they were not included in the text for -02, the text for this version was not affected.

Removed the word "cryptographic" from the hash description in the non-normative note to implementers (per list discussion) and added characterization of the properties the hash chosen should have.

9.6. -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-15 (work in progress), June 2008.

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

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

Authors' Addresses

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

Email: RjS@nostrum.com

Scott Lawrence
Bluesocket Inc.
10 North Ave.
Burlington, MA 01803
USA

Phone: +1 781 229 0533
Email: slawrence@bluesocket.com

Alan Hawrylyshen
Ditech Networks Inc.
823 E. Middlefield Rd
Mountain View, CA 94043
Canada

Phone: +1 650 623 1300
Email: alan.ietf@polyphase.ca

Byron Campen
Tekelec
17210 Campbell Road
Suite 250
Dallas, Texas 75254-4203
USA

Email: bcampen@estacado.net

Full Copyright Statement

Intellectual Property