The Sender Rewriting Scheme

Shevek
spf@anarres.org

June 5, 2004

Contents

1 Introduction

This version of the document is contemporary with version 0.30 of Mail::SRS and version 1.0.11 of libsrs2.

1.1 Overview

Sender Permitted From (SPF, `http://spf.pobox.com/`) is a mechanism for preventing sender forgery in SMTP transactions, thus allowing domain owners control over who may send mail from their domain. The Sender Rewriting Scheme (SRS, `http://www.anarres.org/projects/srs/`, `http://spf.pobox.com/srs.html`) is a mechanism for rewriting sender addresses when a mail is forwarded in such a way that mail forwarding continues to work in an SPF compliant world.

This document outlines the design of the SRS convention and explains some of the gaming behind it. It also provides notes to implementors and links to some useful resources.

1.2 Who does this affect?

SRS must be implemented by all mail servers which forward mail from a domain which does not designate the sender of that mail in its SPF record. This includes:

- Many servers which support .forward files.
- Third party mail forwarders.

This does **not** include:

- All mail user agents.
- Most mailing list providers.
- All simple send/receive-only mail servers which do no forwarding.
- Company internal smarthosts which forward only outgoing mail from the company and are listed in the company's SPF record.

1.3 How is it implemented

A modification to the forwarding mechanism within the mail server will perform address rewriting whenever a mail is forwarded. Patches are being made available with the Mail::SRS distribution for all major mail servers.

2 The SRS Mechanism

2.1 The Problem

SPF requires the client address to match the return path in a mail. This prevents clients from submitting mails with forged return paths. However, consider what happens when a mail goes over a forwarder.

Hop	**Client id**	**Recipient**	**Return path**	**SPF**
1	`source.com`	`alias@forward.com`	`user@source.com`	Accept
2	`forward.com`	`target@destination.com`	`user@source.com`	Reject

Now `destination.com` will look up the rules for `source.com` and quite rightfully reject the mail.

2.2 The Reversable Scheme

`destination.com` will only accept mails with a return path matching the client-id. The client-id of the forwarded mail is `forward.com`, therefore the return path must also be via `forward.com`. Therefore, `forward.com` must rewrite the return path in such a way that it is an address at `forward.com`, yet mails to that address will still reach `user@source.com`. The simplest possible scheme is as follows:

Hop	**Client id**	**Recipient**	**Return path**
1	`source.com`	`alias@forward.com`	`user@source.com`
2	`forward.com`	`target@destination.com`	`user=source.com@forward.com`

The SPF response may be omitted from the table since it will always accept. Now when a bounce or reply mail is received by `user=source.com@forward.com`, `forward.com` will rewrite the destination address to `user@source.com` and forward it.

2.3 Our First Spammer

Our first spammer thinks as follows:

> "*I want to spam* `user@source.com`*. I will send my mails to* `user=source.com@forward.com` *and use* `forward.com` *as an open relay.*"

The SRS scheme must ensure that it is not possible to construct an SRS address which forwards mail to an arbitrary user. This is achieved by introducing a cryptographic element into addresses such that it is possible to spot tampering with the destination user or host fields. The actual SRS address format looks like this:

SRS0=*HHH*=*TT*=hostname=local-part@local-part

The fields are as follows:

- **SRS0:** This identifies the address as an SRS address. This allows hosts to distinguish easily between ordinary email addresses and SRS return addresses directed to the same mail server. The `0` is in some sense a version number for the protocol; it is used later.
- ***HHH*:** This is a cryptographic hash of the fields *timestamp*, *hostname* and *local-part*. This hash may only be generated or validated by someone in possession of the secret key used in the cipher, that is to say, `forward.com`. If a spammer tries to forge an SRS address, he will not be able to generate a valid hash, and `forward.com` will simply reject the mail.
- ***TT*:** This limits the validity of an SRS address. It is expected that bounces will return within a few days, at most a month, of an email being sent. If a spammer gets hold of an SRS address for a user, then they can use the SRS system to bounce mails to that user. The limited validity of the timestamp reduces considerably the value to the spammer of the SRS system, since none of his target addresses is valid for more than a few days. The spammer cannot falsify the timestamp in an SRS address because this would cause a failure of the cryptographic check on the forwarder.
- ***hostname*:** The hostname of the original sender and final recipient for bounces.
- ***local-part*:** The local-part of the original sender and final recipient for bounces. The order of the hostname and local-part was chosen to allow the use of the = character as a separator.

The SRS address format has been designed for simplicity of both human and machine readability. Encoding and escaping formats have been avoided This enables humans to parse SRS addresses at a glance and allows simple regular-expression or substring-match engines to continue to operate on these addresses.

The = character was chosen as the separator because it is not used in hostnames (although this is by no means guaranteed, it looks set to stay) or base64 encoding. This means that the hostname is a single, unescaped field, and therefore an SRS address parser may treat the first 4 =-separated fields as tag, hash, timestamp and hostname respectively. The rest of the SRS address is the original local-part, and may contain = signs, or indeed any other character. Thus this encoding format for addresses avoids the necessity for escaping.

2.4 Our Second Forwarder

Why have just one forwarder when you can have lots? We cannot limit ourselves to just one hop. Therefore we must consider what SRS does over multiple hops.

Hop	Client id	Recipient	Return path
1	source.com	alias@forward.com	user@source.com
2	forward.com	target@bouncer.com	SRS0=*HHH*=*TT*=source.com=user@forward.com
3	bouncer.com	target@destination.com	SRS0=*HHH*=*TT*=forward.com=SRS0=*HHH*=*TT*=source.com=user@bouncer.com

2.5 The Shortcut Scheme

Clearly, this multiple forwarder business is going too far. The address above barely fits on the page, and exceeds the limit suggested by RFCs 2821 and 2822. However, the purpose of SRS is not to cause a returning mail to traverse the entire outgoing path, but to get it back to the original sender. So how about if we rewrite

SRS0=*HHH*=*TT*=source.com=user@forward.com

at bouncer.com

SRS0=*HHH*=*TT*=source.com=user@bouncer.com

using bouncer.com's cryptographic secrets to generate a new hash. Now when bouncer.com receives any bounce message, it will forward it to user@source.com. Now the system looks like this;

Hop	Client id	Recipient	Return path
1	source.com	alias@forward.com	user@source.com
2	forward.com	target@bouncer.com	SRS0=*HHH*=*TT*=source.com=user@forward.com
3	bouncer.com	target@relay.com	SRS0=*HHH*=*TT*=source.com=user@bouncer.com
4	relay.com	target@destination.com	SRS0=*HHH*=*TT*=source.com=user@relay.com

Now the return path will not grow without bound as we forward the mail repeatedly.

This is the shortcut strategy for SRS.

2.6 Our Second Spammer

Our second spammer is very clever. He thinks:

> "*I can get a forwarder to forward mails back to a host* A *if I can get it to generate an SRS address for* A*. Let's pretend* A *send a mail to* B*, which forwards it to* C *which forwards it to* D*, which is me. I can create the alias* C *on any public forwarding server.*[1] *But how to get* A *to mail it (either directly, or via* B*)?*
>
> "*Aha! I will pretend to be* B*, and send* C *a mail as if I was forwarding it from* A*.* C *will then generate an SRS address with his secret, and mail it to me!* C *has no way of knowing* A *didn't originally send me the mail, and will forward replies back to* A*!*"

[1] The term "*public forwarder*" is used to mean a host on which a spammer can set up a forwarding address pointing to himself.

This is clearly convoluted, but then so are the spammers. This exhibits the vulnerability in the simple shortcut strategy above. The problem is that someone who did not directly receive a mail from `A` is willing to forward back to `A`.

In the forwarding chain `A` $\rightarrow$ `B` $\rightarrow$ `C` $\rightarrow$ `D`, we cannot remove `B`, since `B` is the only person who can guarantee that a mail really came from `A` and record this cryptographically in the SRS system.

So we keep `B` and `C` will forward to `B` instead of `A`. `D` may skip `C` and forward directly to `B`. `E` may also forward directly to `B`. Anything goes, as long as we don't drop `B`. But now `C` needs to record in the SRS local-part the SRS address at `B` and wrap this up in all the cryptographic overhead. This means that our SRS address is suddenly twice as long as it was in the simple shortcut scheme.

2.7 The Guarded Scheme

What do we actually *need* in order to forward a mail back to `B` yet make sure `A` never receives a spam?

- We don't need the timestamp. Our timestamp would be the same as, or very close to `B`'s timestamp anyway.
- We don't need the extra `SRS0` prefix. We will know that we are a second hop, and that we must prepend `SRS0` before returning the bounce.

We are left with an address format for secondary hops which looks like:

`SRS1=`*`HHH`*`=forward.com==`*`HHH`*`=`*`TT`*`=source.com=user@bouncer.com`

which contains enough information for any hop > 2 in the forwarding chain to forward the mail back to `B` as an SRS0 mail. The only extra overhead is one = and the hostname for host `B`.

There is a new hash in this SRS1 address. Any SRS1 host will hash an address using its own secret to prove its veracity, and must check this hash when reverse-forwarding a bounce back to the original SRS0 address.

The purpose of this new hash is to prevent a spammer from using SRS as an extremely restricted open relay to addresses beginning with "SRS0". This hash may be seen as unnecessary from the perspective of protecting users from spam since if `B` is an SRS host, then the SRS0 address will contain a hash, which will be checked before the mail reaches a user. If `B` is not an SRS host, then the address "SRS0" is unlikely to exist. Therefore anyone tries to fake an SRS return path to `C` via `B`, `B` will drop it. However, some parts of the community were concerned about this possible loophole, so a hash was included in SRS1.

No generally applicable weaknesses are known in this scheme and it is recommended that it be deployed. Hosts which implement public forwarding should see section 2.10.

2.8 An Example Transaction

Hop	**Address**	**Return path**
1	`user@source.com`	`user@source.com`
2	`alias@forward.com`	`SRS0=`*`HHH`*`=`*`TT`*`=source.com=user@forward.com`
3	`target@bouncer.com`	`SRS1=`*`HHH`*`=forward.com==`*`HHH`*`=`*`TT`*`=source.com=user@bouncer.com`
4	`relay.com`	`SRS1=`*`HHH`*`=forward.com==`*`HHH`*`=`*`TT`*`=source.com=user@relay.com`
⋮	⋮	⋮
n	**Message bounces**	
$n+1$	`SRS1=`*`HHH`*`=forward.com==`*`HHH`*`=`*`TT`*`=source.com=user@somewhere.com`	
$n+2$	`SRS0=`*`HHH`*`=`*`TT`*`=source.com=user@forward.com`	
$n+3$	`user@source.com`	

2.9 Our Third Spammer

A third spammer? There is certainly no shortage of spammers. Anyway, our third spammer plays what is known as the "Five Party Game".

Imagine a forwarding chain `A` → `B` → `C` → `D` → `E`. If `E` bounces the mail, it will go back to `D`, which was the last SRS forwarder, thence to `B` and then `A`.

Our spammer sets up a forwarding address on `D` pointing to himself. He then mails himself via `D`. Now he has an SRS1 return path which routes via `D` to `B`, and he can send mail to a username starting `SRS0` at `B`.

This attack is *not generally feasible* since a spammer cannot generally set up a forwarding address on `D`. In the case where it is possible for a spammer to do this, for example, **on a public forwarding host, a database scheme should be used**. In any case, the impact of this attack is low since the spammer must perform 3 SMTP transactions in order to send mail to a username which must start with `SRS0`.

2.10 Database Schemes

SRS provides sufficient convention to implement a reversable return path rewriting scheme. Hosts must be aware of the rewriting conventions in order to shortcut hops in the return path. If more information is required than is provided by the standard SRS format, and later shortcutting around the present hop is not a priority, then a host may rewrite the return path as it sees fit as long as:

- The rewriting complies with SPF.
- The rewritten address complies with the SRS format.

The SRS system as described requires no form of database or backing store whatsoever. However, in the presence of store, it is possible to generate a *fresh* address to use as the rewritten sender address. When a bounce comes back, this address may be looked up in the database to discover the original sender (and presumably to check that the hash, timestamp, etc are valid). This keeps resender addresses short and sweet, since they need no longer contain timestamps, original addresses, or any other semantic information. It also allows for the provision of a certain amount of anonymity in the resending process, if this is desired.

An example database driven implementation is provided in the Perl reference implementation `Mail::SRS`, available from CPAN.

2.11 Other attacks: Database storage

2.12 A study on the 64 character limit

3 The SRS Format: A Formal Description

3.1 SRS separators

The SRS separator is a = sign. The first separator, which immediately follows the string `SRS0` or `SRS1` may be any of =, + or −. Throughout these examples, the = is used consistently as an initial separator.

3.2 SRS0 addresses

SRS0 addresses have the form:

`SRS0=`*opaque-part*`@`*domain-part*

where *opaque-part* may be defined by the SRS0 forwarder, and may only be interpreted by this same host. By default, the Guarded mechanism of the Mail::SRS distribution implements this as:

`SRS0=`*HHH*`=`*TT*`=orig-domain=orig-local-part@domain-part` where *HHH* is a cryptographic hash and *TT* is a timestamp. The Database mechanism of the Mail::SRS distribution implements SRS0 as:

`SRS0=`*key*`@`*domain-part*

where *key* is a database primary key used for retrieving SRS-related information. Other implementations are possible.

3.3 SRS1 addresses

SRS1 addresses have the form:

`SRS1=`*HHH*`=orig-local-part==`*HHH*`=`*TT*`=orig-domain-part=orig-local-part@domain-part`

where *HHH* is a cryptographic hash, which may be locally defined, since no other host may interpret it. The double == separator is introduced since the first = is the SRS separator, and the second = is the custom separator introduced by the SRS0 host and might alternatively be + or −. This double separator might therefore appear as =+ or =−.

The SRS1 format is rigidly defined by comparison to the SRS0 format and must be adhered to, since SRS1 addresses must be interpreted by remote hosts under separate administrative control.

4 Notes on the Mail::SRS Implementation

An example implementation is available as `Mail::SRS` from CPAN. The latest version should always be available from `http://search.cpan.org/~shevek/`. The home page for this distribution is at `http://www.anarres.org/projects/srs/`. Some computational features of the implementation are described here.

4.1 Case Sensitivity

RFC2821 *requires* that case be preserved in the *local-part* of an email address, but requests that implementations do not rely on this behaviour in remote MTAs. Consequently the Mail::SRS implementation is not case sensitive. The cryptographic hashes are encoded in base 64, but are compared case insensitively. The implementation will issue a warning if it detects that a remote MTA has smashed case, but this will not affect the functionality of the code. Timestamps are encoded in base 32.

4.2 The Hash

The hash is an SHA1 based HMAC, as provided by the Perl `Digest::HMAC_SHA1` or C OpenSSL libraries. The hash is base64 encoded. Only the first few characters of the hash are used: Since the hash is uniform, this only reduces the number of guesses required to fake an SRS address proprtionately (expoentially) with the length of the hash.

Since the hash is opaque to everyone except the host that created it, it is possible for an SRS compliant host to use any hash algorithm at all. The SHA1 HMAC is a recommendation, and anyone changing this should know what they are doing and have a *very* good reason why.

The length of this hash controls the amount of security it provides. The Mail::SRS distribution generates a hash of length 4 in base64, which gives 24 bits of security. This is probably a good minimum. If space is at an absolute premium, it may be possible to reduce the length of the SRS1 hash, but *not* the SRS0 hash. If you don't understand why, don't do it.[2]

[2]Consider the relative value to the spammer of being able to generate a valid fake address at each level, compare with the cost of cracking n bits of SHA1, and see section 2.7.

4.3 The Timestamp

The resolution is one day, and it is a 10 bit number stored as 2 base32 characters. The timestamp is computed as

$$\frac{\texttt{unix\_time}}{(60 \times 60 \times 24)} \bmod 2^{10}$$

This gives 1024 possible timestamps, with approximately a 3.5 year cycle before a timestamp becomes valid again.

4.4 The 64 Character Limit

RFCs 2821 and 2822 state that there is a possible upper limit of 64 characters on local-parts in SMTP addresses. SRS introduces by default 21 characters plus two hostnames as overhead into the local-part. Since the length of a hostname is bounded at a few hundred characters, there is potential for exceeding the 64 character limit on the local-part.

MTAs or SMTP systems which are known to enforce this limit are:

- MailGuard on the Cisco PIX

5 Notes on the libsrs2 Implementation

All SRS implementations must be interchangable, and thus they must all comply with the reference Mail::SRS implementation. libsrs2 includes a set of Perl bindings and a testsuite to demonstrate this conformance.

6 Other Proposals

6.1 SES - Signed Envelope Sender

Signed Envelope Sender, or SES, was originally proposed around `http://archives.listbox.com/spf-discuss@v2.listbox.com/200402/0901.html`. It doesn't work.

The proposal is that if email addresses are signed, then no spammer can forge an email address. However, email addresses can be used for replay if they ever become visible to spammers. There have been various proposed extensions to SES, some of which I list here, and none of which make it '*work*' without at least reintroducing something logically equivalent to SRS. If I have missed any extensions or definitions of SES, please let me know so that I can add them to this document.

- **Time limit addresses.** Now a spammer can only replay each address for a limited time. Finding 100 fresh email addresses per day on the internet isn't hard, especially when every single email sent has a fresh address on it.

- **Hash the headers or body into the signature.** This breaks (at least) MTAs, which like to add headers, and mailing lists, which like to modify the body.

6.2 RFROM/DAVE/SUBMITTER

There is a suggestion to extend the mail protocol to have both SUBMITTER, the "responsible address" or PRA, and MAIL FROM, the "origial source" of the message.

The purpose of SUBMITTER is not entirely clear. Some parts of the community seem to think it will stop spam, some think it will stop joe jobs, and some think it will stop phishing. I believe none of these is true, but my main concern is with the prevention of joe jobs.

See `http://archives.listbox.com/spf-discuss@v2.listbox.com/200406/0145.html` for my original thesis on SUBMITTER, as posted to the spf-discuss list. I reproduce and expand upon that thesis here.

SUBMITTER is the "responsible address" or PRA for a mail. The PRA is validated according to SPF rules. The MAIL FROM recieves bounces.

But suddenly we are verifying one address and sending bounces to another!

The possible responses I expect from the proponents of SUBMITTER are:

- "But we send bounces to SUBMITTER" - In this case, MAIL FROM is redundant, and we are proposing rewriting SMTP for no purpose. The current system just puts the PRA in MAIL FROM. This should continue.
- "We can verify MAIL FROM as well." - Differently to the verification of SUBMITTER? How?

The effect of introduing SUBMITTER is to entirely defeat the original purpose of SPF, that is, to permit spammers to send their bounces to arbitrary locations.

I appreciate the political constraints introduced by attempting to cooperate with uncooperative parties with differing agendas to that of SPF, but what is being achieved here is a total destruction of the utility of the protocol.

SUBMITTER is therefore bad and should be excluded from any future protocol.

7 More Information

- The author of this document may be contacted at `spf@anarres.org`.
- Implementations of SRS are available from `http://www.libsrs2.org/`.
- I was originally introduced to the concept of SRS by Meng Wong (`mengwong@dumbo.pobox.com`), who originally documented SRS at `http://spf.pobox.com/srs.html`.
- SPF is documented at `http://spf.pobox.com/`.
- An enjoyable book on classical game theory is "The Games Afoot! Game Theory in Myth and Paradox" by Alexander Mehlmann. It is a highly enjoyable read and an easy introduction to the process of gaming for even non-mathematicians.
- Apologies to those others whom I may have missed in the early drafts of this document. Please let me know as soon as possible.