Exploiting email address parsing with AWS SES

Exploiting email address parsing with AWS SES - Author not stated, nathandavison.com.

- Published: date not stated
- Original: <https://nathandavison.com/blog/exploiting-email-address-parsing-with-aws-ses>
- Preserved from: https://nathandavison.com/blog/exploiting-email-address-parsing-with-aws-ses (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

In this post I'm going to cover a technique I discovered recently to bypass user account email validation/verification in a web app I was testing. This app used AWS SES to send verification emails, and the domain of a user's verified email address was used to make some access control decisions in the app logic.

In an app such as this one where a certain email domain can grant certain privileges for a signed up user, a good target for pen testing is whether you can trick the app into treating your address as belonging to one domain, but the verification email goes to another. The impact of finding a bug like that would vary on what sort of importance the app places in the email address (or its domain) after validating it - in this app's case, the domain of the user signing up determined what level of access they had once verified and authenticated so, basically, we're talking access control bypass and privilege escalation.

After butting up against a lot of failures, I eventually stumbled across the following payload to achieve access control bypass in the app:

```
<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>[[email protected]](https://nathandavison.com/
```

When instructed to send an email to this address, SES will send the message to `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` however, in this particular instance, the app was treating this signup attempt as belonging to the domain `ddd.com`. The concept of this vulnerability is somewhat similar to HTTP request smuggling, in that the "frontend" (the web app) is parsing a value (the email address) different to the "backend" (AWS SES), causing a desynchronisation between the two and their intepretation of the value.

This is a failure in the app's logic primarily, as the app was not applying enough validation to the address a user supplied, but it's also interesting that SES parses this address at all - from what I can determine, RFC 5322 is fairly strict in that the `name-addr` spec of `[display-name] angle-addr` is only in that order, and not `angle-addr [display-name]`. However, the RFC also states that some legacy systems will use `angle-addr` with the `display-name` following inside a comment (i.e. between parentheses), which this payload isn't exactly, but it is somewhat close. The RFC also states the use of `name-addr` as opposed to this legacy format as a SHOULD and not a MUST. Either way, when you combine the app's faulty logic in determining the domain of the user being verified, and the somewhat relaxed address parsing by SES, you end up with a vulnerability that allows a user to signup as a member of an arbitrary domain.

A quick way to confirm that SES will handle this format is the following command using the AWS CLI - you'll need valid AWS creds configured in your environment, a valid from address allowed by the creds, and of course an email to target with the message, which may need to be verified if your AWS account is in sandbox mode:

```
aws ses send-email --from '[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)' --to '<[[email pro
```

The "test" message should arrive at the inbox you put in place of the `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)`. Interestingly, the `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` display name doesn't appear anywhere in the raw email source, which suggests perhaps it is not being treated as a display name at all, but is simply being ignored by SES. With that said, the following payload will send an email to `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` and use a display name of `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)`:

```
<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>([[email protected]](https://nathandavison.com
```

This more closely follows the legacy format touched upon earlier when referencing the RFC, as `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` is a comment inside parentheses (although the `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` is still non conforming).

The situation with the vulnerable app I encountered is very similar to the writeup by "Elliot Alderson" which resulted in a vulnerability being found in the Python email address parsing function `parseaddr` (CVE-2019-16056). Interestingly, for both pre and post CVE-2019-16056 versions of Python, `parseaddr` identifies `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` as the address in the above payload, which is consistent with SES and would avoid the disconnect between app and email server, so it seems a valid mitigation for any app (like the one I bypassed with the above payload) that uses SES would be to use a function like `parseaddr` which also extracts `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` from the payload. Like SES though, whether or not `parseaddr` *should* extract `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` from the payload as the email address rather than hit an error condition is another question.

What about other languages and packages that may be used to parse an address? Here I have some tests I've run against various languages and their popular functions and libraries for parsing email addresses:

| Language | Return when parsing `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` payload | Vulnerable? | | | Python + email.utils.parseaddr() | `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` | Maybe | | | NodeJS + email-addressess | `null` | No | | | NodeJS + address-rfc2822 | `throw new Error('No results')` | No | | | PHP + mailparse_rfc822_parse_addresses() | Array of `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` and `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` | Yes | | | PHP + Mail_RFC822::parseAddressList() | `Validation failed for: <[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` | No | | | PHP + PHPMailer | `Invalid address: (to): <[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` | No | | | Ruby + Mail::AddressList() | `Mail::AddressList can not parse \|<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)\|: Only able to parse up to "<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>"` | No | | | C# + System.Net.Mail.MailAddress | `An invalid character was found in the mail header: '>'` | No | | | Go + mail.ParseAddress() | `mail: expected single address, got "[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)"` | No | | | Java + email-rfc2822-validator EmailAddressParser.getAddressParts() | `null` | No | | | Java + javax.mail InternetAddress() | `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` | Maybe | |

The "Vulnerable?" column is asking whether the output of the command could be vulnerable to allowing the payload to be parsed differently in the app code compared to where AWS SES will send the email or, in other words, was the execution successful/error free and does the output contain `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)`? because if it does, it may be further interpreted by code as the email address. Of course, email platforms other than AWS SES may do the exact opposite, and send a message to `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` instead of `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` when fed the payload - this is why, if the method above returns any valid data at all instead of raising an error, I list it as "Maybe" vulnerable, as it could create a vulnerability in an app when paired with the "wrong" mail backend.

With that said, I considered the PHP `mailparse_rfc822_parse_addresses()` to be vulnerable because it returns the `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` in the array (albeit in the last array element), which could lead to code parsing the payload as an email being sent to `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)`. For instance, the following (somewhat contrived) PHP code would consider `ddd.com` to be the domain the user signed up with when given the `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` payload:

```php
function get_signup_domain($email) {
    // strip out commas and semi colons
    $email = str_replace([",", ";"], "", $email);
    // get the address
    $addr_a = mailparse_rfc822_parse_addresses($email);
    $addr = end($addr_a)['address'];
    // return the domain
    return substr($addr, strpos($addr, "@") + 1);
}
```

The code is stripping out symbols that might be used to combine email addresses, signifying a developer's effort to prevent multiple addresses being provided.

Because the payload still works with SES with some modifications (such as the `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>([[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection))[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` one mentioned already), I also tried variations of the payload to see if the outcome changed across the languages - most of the time the result was the same as above, but there was one noteable exception with this comment payload:

| Language | Return when parsing `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>([[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection))[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` payload | | | C# + System.Net.Mail.MailAddress | `"<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>" <[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>` | |

When given `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>([[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection))[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)`, the .NET `System.Net.Mail.MailAddress` method will return `"<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>" <[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>` rather than the `An invalid character was found in the mail header: '>'` error from the original payload. This also works with a space character instead of `([[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection))`, i.e. a payload of:

```
<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)> [[email protected]](https://nathandavison.com
```

This non-error return is problematic, because it may be possible for code to take this return and determine that `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` is the address receiving the email. Luckily, if you feed this output directly to SES, `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` will get the email because it correctly parses the quotes and `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>` becomes the display name, so there is no disconnect between code and mail backend in that scenario. The issue would be if `System.Net.Mail.MailAddress` was used to validate and extract the domain out of the address, but the original payload was sent directly to SES as the 'to' address, such as in the following snippet:

```
// this is the value the user provided during signup
string UserEmail = "<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)> [[email protected]](ht

// parse and validate the user provided email address
try {
    System.Net.Mail.MailAddress ParsedEmailAddress = new System.Net.Mail.MailAddress(UserEmail);
} catch (Exception ex) {
    ...
}

// no exception caught - validation passed! get the domain of the user's email address, and store it for later use in business lo
string UserDomain = ParsedEmailAddress.Host;

// UserDomain is "ddd.com"

// send the email to the user via AWS SES, using the original validated payload
sendValidationEmailViaSES(UserEmail);

// email was sent to "[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)"
```

You can see the parsing in action with this dotnetfiddle.net snippet.

The `UserDomain` here will be `ddd.com` and, because it passed `System.Net.Mail.MailAddress`'s validation, the original `<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)> [[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)` payload is trusted and used to send the verification email via SES, which will send the email to `[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)`. Once verified, the user account is associated with the `ddd.com` domain in the app, but the email was never sent to an inbox under the `ddd.com` domain.

The catch here is the developer would have to implement `sendValidationEmailViaSES` in a specific way - Amazon's own snippet of code on how to send an email in C# using SES has the following line:

```
message.To.Add(new MailAddress(TO));
```

This should prevent the vulnerability, because as shown, `MailAddress` will convert the payload into the compliant `"<[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>" <[[email protected]](https://nathandavison.com/cdn-cgi/l/email-protection)>` string. However, if the `sendValidationEmailViaSES` implementation instead did this:

```
message.To.Add(TO);
```

Then the message will be sent to SES with the payload intact, and the vulnerability would be in play. As mentioned, a developer may be inclined to do this, because the address was parsed and validated by `System.Net.Mail.MailAddress`.

The results across the languages suggest that this payload won't be effective in mass exploiting many web applications, however keep in mind these are functions and libraries built for parsing email addresses - if Google results for searches like "parse email addresses LANGUAGE_HERE" are anything to go by, a lot of developers will be following bad advice such as "just split on the last '@' symbol to get the email domain", which is probably what caused the app I was testing to be vulnerable. On the other hand, even if you do use a parser, you should still be wary of edge cases like those I listed - sometimes their output can be wrong or misleading, or at the very least too liberal in accepting non conforming addresses, and prone to introducing vulnerabilities. At the very least, a developer should make sure to investigate whether there is a better way in their language of choice to validate an email address beyond running an email parsing function and looking for errors (for instance, in PHP, use `filter_var()` with `FILTER_VALIDATE_EMAIL`), and use the output of the parser when sending the email even if the original payload was parsed successfully (which avoids the issue in the `System.Net.Mail.MailAddress` example).

Reporting

As a result of this research, I reported the following:

- mailparse_rfc822_parse_addresses parsing bug. The maintainers don't believe this is a security issue in PHP as users shouldn't be using this function for email validation.
- AWS don't consider the error-free parsing of the payload to be an issue on their end, saying "The issue you reported is caused by improper client-side validation within web applications which use Amazon SES".
- I couldn't find a way to report System.Net.Mail.MailAddress's parsing of the modified payload to Microsoft, and failed to get past triage in their .NET bug bounty process.