Pentest-Report Enigmail by Cure53

The following is an extract of a Pentest Report created by Cure53 (www.cure53.de). The report was funded by Posteo (www.posteo.de) and the Mozilla's Secure Open Source fund (MOSS; https://www.mozilla.org/en-US/moss/secure-open-source/) and covers both Thunderbird and Enigmail. As not all vulnerabilities have been fixed at the time of writing (2017-12-10), this extract only mentions the vulnerabilities that needed fixing in Enigmail. The full report will be published once all issues are fixed.

The Enigmail Project likes to thank Posteo and MOSS for sponsoring this highly valuable report.

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *TBE-01-001*) for the purpose of facilitating any future follow-up correspondence.

TBE-01-002 Enigmail: Weak Parsing Causes Confidentiality Compromise *(Critical)*

The tests revealed a weakness in email parsing. Specifically, this flaw might lead to a vulnerability in which Enigmail can be coerced to use a malicious PGP public key with a corresponding secret key controlled by an attacker. An example scenario for this attack is outlined next.

1. Bob sends an email to Alice. The email appears to be from Bob and is signed and encrypted under Mallory's PGP identity.

2. Mallory, a network attacker that can only modify Bob's "*Full Name*" field in SMTP communications, changes Bob's "*Full name*" field in a specific way that, because of one aspect of this vulnerability, is covert. In other words, Alice cannot detect the manipulation.

3. Alice replies to Mallory's email. Due to Mallory's surreptitious modification of Bob's "*Full Name*" field, however, Alice's message response ends up using a completely different PGP key than the initial one of Bob. This PGP key could be controlled by Mallory, or could actually be any other PGP key at all.

As exemplified above, this leads to a complete and silent Man-in-the-Middle (MitM) compromise of the email thread. Evidently, the associated level of risk signifies a vulnerability with a "Critical" severity and impact.

Two regular expressions in Enigmail lie at the core of this issue. They can be used for spoofing an arbitrary email address. The description below explains how this leads to a critical issue, but it should be noted that the flaw has additional further impact, as can be seen in TBE-01-004.

Enigmail's *funcs.jsm* defines a *stripEmail* function which is supposed to extract the email address contained in <*evil@example.com*> from a comma-separated list of emails. As a first sanity check, a

regular expression is used to make sure that no two <> follow each other. This is done by making sure that they are separated by commas:

```
EnigmailFuncsRegexTwoAddr = new RegExp("<[^>,]*>[^,<]*<[^>,]*>");
```

The problem is, however, that this regular expression can be fooled if the attacker injects an additional pair of <> and includes a comma in the spoofed email address:

```
<good@example.com,><evil@example.com>
```

Then the second regular expression tries to match the email in between the <> pair to extract it:

```
EnigmailFuncsRegexExtractPureEmail = new RegExp("(^|,)[^,]*<([^>]+)>[^,]*",
"g");
```

This causes the first email with the comma to be matched, instead of the correct one in *evil@example.com*:

```
<good@example.com,><evil@example.com>
```

In the efforts to act correctly, Enigmail actually makes things worse by stripping away the comma:

```
mailAddrs.replace(/[,;]+/g, ",").replace(/^,/, "").replace(/,$/, "");
```

Therefore, it can be supposed that Bob's *"Full Name"* field has been changed from `Bob Bobbington` to `Bob Bobbington <mallory@gmail.com,>`.

This change could be made not only by Mallory but also by Bob himself should he wish to deceive Alice. If Bob's *"Full Name"* was specified as shown above, then Enigmail will look up the PGP key under mallory@gmail.com when Alice attempts to reply to Bob. The latter will be used for encryption instead. Now, this example is far from "covert" as mentioned in the introduction of this issue. However, it is also important to consider that Mallory can equally alter Bob's "*Full Name*" field to Bob Bobbington <bob@gmail.com,>

The above appears to completely match Bob's genuine email address, namely bob@gmail.com. Yet in fact it does not, because the "a" one sees in "*gmail*" is actually the UTF-8 Cyrillic character "*а*". As a result, the string above does not match the original string in "bob@gmail.com" which represents Bob's actually true email address. To clarify, Mallory could upload a new identity posing as Bob to PGP key servers. If Mallory maliciously used a Cyrillic character for differentiation, Enigmail would be tricked into automatically fetching the fake identity. In effect, Enigmail would encrypt that information, thereby extending the implications of this vulnerability with an aspect of a covert component.

This vulnerability could be remedied by double-checking the regular expressions used for parsing. The verification should be performed in order to disallow malicious injections of email identifiers. It should be emphasized that Enigmail uses these identifiers as the basis for the SQLite database lookup, internally employed for retrieving the corresponding PGP identities. For that reason, flaws in this level of parsing can be fatal, as demonstrated by this multi-layered finding.

TBE-01-005 Enigmail: Replay of encrypted Contents leads to Plaintext Leak *(High)*

It was found that an attacker can retrieve plaintext of encrypted mails, provided that they were previously sent to the victim. This can be achieved by including the encrypted data block into the email's body. If the victim responds to the email in question without discarding the original message, the decrypted content is leaked to the attacker. Enigmail supports partially encrypted emails wherein only a selection of the

message's body is encrypted. This is what makes the attack realistic, since encrypted message blocks can be hidden in longer conversations.

Steps to reproduce:

- Mallory intercepts an encrypted message sent from Alice to Bob.
- Mallory starts a conversation with Bob. In order to make this attack work, Bob must not discard the original message when replying to an email.
- At some point when the conversation is long enough, Mallory slips the intercepted PGP block into the conversation and leaves the rest of the email unencrypted.
- When Bob receives the message, the PGP block will be decrypted automatically.
- As Bob will likely not read the earlier conversation again, he will have no way of noticing the additional text. However, if he expectedly responds to the message, the decrypted content will be leaked to Mallory.

An alternative way to exploit this issue requires social engineering and makes use of Thunderbird's *forwarding* feature. Actions that need to be completed for this alternative route are enumerated next.

Steps to reproduce:

1. Mallory intercepts an encrypted message which is sent from Alice to Bob.
2. Mallory sends Bob a very long text message which includes the encrypted PGP block and a short text which convinces Bob to forward this email to Trudy without reading the actual message.
3. If Bob follows Mallory's instructions and forwards the email, Enigmail will automatically decrypt the included PGP block and the plaintext is leaked to Trudy.

It should be noted that this issue is rather a design flaw. Specifically, it predominantly relies on the unawareness or lazy behavior of users.

Thunderbird already displays an info box when an email contains partly encrypted data. However, this message can be easily overseen or ignored. It is recommended to leave messages about partial encryption when they are being forwarded or responded to. It is alternatively recommended to display a clear warning when a response to a partly encrypted message is composed or when such a message is forwarded.

TBE-01-021 Enigmail: Flawed parsing allows faked Signature Display *(Critical)*

Enigmail will incorrectly find and verify signatures of attached email files. The error lies in parsing the email and failing to separate the contents of the email from the contents of the attachment. If an attached email is signed, Enigmail will verify that signature against the text of the attached email, but it will appear to the user as if the entire message was signed. This allows an attacker to create a forged email, e.g., from *bob@cure53.de*, that has an email as an attachment signed by *bob@cure53.de*. To the recipient it seems as if the message in its entirety - rather than just the attachment - was signed by Bob.

Steps to reproduce:

1. Save an email with a correct signature from the victim.
2. Create a new email and add the *saved* email as an attachment.
3. Send the email to the targeted recipient.

4. The target will now see the email as having a valid signature from the victim.

Sample email body:

```
Delivered-To: jonas@cure53.de
Return-Path: <mario@cure53.de>
To: Mario Heiderich <mario@cure53.de>, Jonas Magazinius <jonas@cure53.de>
From: "Dr.-Ing. Mario Heiderich" <mario@cure53.de>
Subject: This is totally signed by mario@cure53.de!
Date: Fri, 29 Sep 2017 12:35:19 +0200
Content-Type: multipart/mixed;
  boundary="------------AEA294334A39599F740CD34A"

This is a multi-part message in MIME format.

--------------AEA294334A39599F740CD34A
Content-Type: text/plain; charset=windows-1252
Content-Transfer-Encoding: quoted-printable


Hey!

Just writing this totally legit email and it's totally signed by me
(mario@cure53.de).

/Mario

--------------AEA294334A39599F740CD34A
Content-Type: message/rfc822;
  name="poc.eml"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment;
  filename="poc.eml"
Subject: [REDACTED]
To: jonas@cure53.de
From: "Dr.-Ing. Mario Heiderich" <mario@cure53.de>
Content-Type: multipart/signed; micalg=pgp-sha256;
  protocol="application/pgp-signature";
  boundary="PWpC1qlx6dsQoTPWMjFMqgqCjLq1TuoEA"

This is an OpenPGP/MIME signed message (RFC 4880 and 3156)

--PWpC1qlx6dsQoTPWMjFMqgqCjLq1TuoEA
Content-Type: multipart/mixed;
  boundary="MkhracRKbd653uoMlB5pR9frBfLDD2DJK";
  protected-headers="v1"
From: "Dr.-Ing. Mario Heiderich" <mario@cure53.de>
To: jonas@cure53.de
Subject: [REDACTED]

--MkhracRKbd653uoMlB5pR9frBfLDD2DJK
Content-Type: text/plain; charset=utf-8
Content-Language: en-US
Content-Transfer-Encoding: quoted-printable

Hi,
[REDACTED]
```

```
Cheers,

=2Emario


--MkhracRKbd653uoMlB5pR9frBfLDD2DJK--
--PWpC1qlx6dsQoTPWMjFMqgqCjLq1TuoEA
Content-Type: application/pgp-signature; name="signature.asc"
Content-Description: OpenPGP digital signature
Content-Disposition: attachment; filename="signature.asc"

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2

iQIcBAEBCAAGBQJZdwtDAAoJEMJshYCQ9wra/7kP/20hr3PCSO4Lm0eZ6OCpuhGj
p04h38Mx6Jxrn+i85yMA/Bk7aU48spraWNm9cVBv8sFnVLdSTs9IiNcNsEznUCM3
KMxkva+E8u3+uuOZEGlo70L/c8EFIkXT2TrW241ZMJFLzhvcAaQLKD4V+cnsJ6CS
bV9v0WYfFH3sS4ImTj1VPVGKfLgYQnxZK/OTnxVM7oHwb4ibshqGBic2L4C4afDI
K8MRc4Fek+llKPBqH/1Am72tTyyweGFyRAfJJ5BfxJTrSSJ08KPMya6NHQq4QG0A
63Sy1Ji1l5j9BoK+Y7VolwmONDnBYLnyTkN/UoPl/6C7rA8SVQzuQtG/qihXete6
6vrlwEADuS904BZv3BJuhwIw9irmqFSjMFcx4gRldZzvyII7MD7IvtSouSsbwSTZ
3swiifz5fNRUrKq4yNarLCqOKbXn+W0mSjS6Ft23wnMosadGNyT49t6f9ZPILpuB
kL2Cro1Sihsrryzg/Y5NG52Dy2BFH7VfBHjIIl++1dTU6nnfGCZ3XWdnXB5sX2BH
i+cZ2GFiu05ICgi7tdIAjL7Zwh0P1Pf4uAwZ4o5F7Ilxo1ez5LFMTPMoVa1R1E8t
bS/DwqhzTad5EXhhJknpNDt8VZJpx+XjHbD+QW4z8OTlLSVQ2UYnLZXqQsgzK8yE
hGGHg2U2a9dCF7psD2Cf
=VrRa

-----END PGP SIGNATURE-----
--PWpC1qlx6dsQoTPWMjFMqgqCjLq1TuoEA--
--------------AEA294334A39599F740CD34A--
```

As it is already clear from the opening description, it is recommended fix the parsing of the email in the signature verification flow. It should be ascertained that the whole email - instead of the attached emails only - is signed.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

TBE-01-001 Enigmail: Insecure Random Secret Generation *(Low)*

Enigmail's implementation of *pretty Easy privacy* (pEp)[1] generates security tokens with calls to JavaScript's *Math.Random()* feature. This does not signify a cryptographically secure pseudo-random umber generator[2] approach.

[1] https://en.wikipedia.org/wiki/Pretty_Easy_privacy

[2] https://stackoverflow.com/questions/5651789/is-math-random-cryptographically-secure

Affected File:

/enigmail-source/package/rng.jsm

Called in:

/enigmail-source/packagepEpAdapter.jsm:

Affected Code:

```
gSecurityToken = EnigmailRNG.generateRandomString(40);
 [...]

/**
 * Create a string of random characters with numChars length
 */
function generateRandomString(numChars) {
  let b = "";
  let r = 0;

  for (let i = 0; i < numChars; i++) { r = Math.floor(Math.random() * 58);
   b += String.fromCharCode((r < 10 ? 48 : (r < 34 ? 55 : 63)) + r);
  } return b;
}
```

The *generateRandomString()* function employs *Math.random()*, which is not an advised route in this realm. Instead, it is recommended to make use of the already present and considerably more secure random number generators that are referenced in the same file (*rng.jsm*). Generally, the most widely available secure source of pseudo-randomness in JavaScript is the *window.crypto.getRandomValues()* function, and it should be used exclusively for sensitive random value generation contexts.

TBE-01-003 Enigmail: Regular Expressions Exploitable for Denial of Service *(Low)*

Regular expressions used to parse user-input or *gnupg* output are specified too broadly. As such, they give way to Denial of Service (DoS) attacks. In the instances that were discovered, arbitrary-length inputs were accepted as valid for attachment headers, URL protocol headers, and email address links. This can allow an attacker to pass an extremely large string into internal Enigmail functions, causing Denial of Service on the client-side and ultimately crashing the client.

No further negative effect has been observed as part of this issue, so it is not viewed as actually compromising any user-security. Neither it is able to accomplish anything other than inconveniencing or disrupting the user's workflow.

Affected File:

/enigmail-source/package/decryption.jsm

Affected Code:

```
if (attachmentHead.match(/-----BEGIN PGP \w+ KEY BLOCK-----/)) {
 // attachment appears to be a PGP key file
```

Affected File:

/enigmail-source/package/decryptPermanently.jsm

Affected Code:

```
if (attachmentHead.match(/-----BEGIN PGP \w+ KEY BLOCK-----/)) {
  // attachment appears to be a PGP key file, we just go-a-head
  resolve(o); return;
}
```

Affected File:

/enigmail-source/ui/content/enigmailMessengerOverlay.js

Affected Code:

```
// Hyperlink URLs
var urls = text.match(/\b(http|https|ftp):\S+\s/g);
```

Affected File:

/enigmail-source/ui/content/enigmailMessengerOverlay.js

Affected Code:

```
// Hyperlink email addresses
var addrs = text.match(/\b[A-Za-z0-9_+.-]+@[A-Za-z0-9.-]+\b/g);
```

Across all of the detected examples, it was possible to replace instances of regular expression matching for arbitrary-length inputs with inputs of fixed length. Similarly, a predetermined but long range of, for example, 1 to 1024 characters, could be accomplished. Conversely, for the PGP header, a predetermined length of 1 to 10 characters is sufficient.