Fine penetration tests for fine websites

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

Audit-Report RNP & Thunderbird Integration 08.2020

Cure53, Dr.-Ing. M. Heiderich, M. Wege, BSc. J. Hector, MSc. D. Weißer,
MSc. R. Peraglie, Dr. N. Kobeissi

Index

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

Introduction

*“RNP is a set of OpenPGP (RFC4880) tools that works on Linux, *BSD and macOS as a replacement of GnuPG. It is maintained by Ribose after being forked from NetPGP, itself originally written for NetBSD.”*

From https://www.rnpgp.com/software/rnp/

This report documents the results of a large-scale assessment of the RNP C++ OpenPGP implementation and its integration with the Thunderbird email client. Carried out by Cure53 in 2020, the project entailed a broad-scope and multi-week examination. It featured both a penetration test and an audit of the source code underpinning the RNP C++ Open PGP complex. The focus was also solidly placed on how the community-maintenance of Thunderbird plays into the general security of the integration. It is also crucial to note that the project was requested by Mozilla and funded through the Mozilla Open Source Support (MOSS) scheme.

As the project combines approaches, links entities and aspects, it was vital to craft an appropriate work structure. The work was ultimately split into three work packages (WPs) as a means to ascertain that various key objectives of stakeholders were being met. In WP1, the testing team examined the RNP C++ OpenPGP library’s handling of PGP messages and its broader logic. Next, WP2 zoomed in on the Thunderbird integration of the RNP C++ library, with a strong focus on the cryptographic realm. Finally, WP3 examined the state of web security within the Thunderbird integration of the RNP C++ library.

Since all components in scope are available publicly as Open Source Software, a white-box methodology was used. It needs to be noted that the scope was chosen very carefully. For example, both the cryptography audits against the included Botan library, as well as general Binary Fuzzing work were considered OOS. A key problem that needed to be tackled from Mozilla’s point of view was whether Thunderbird integrates the RNP library well in terms of security. In other words, Cure53 was to find out if problems caused by the integration or the RNP-parts used by Thunderbird could signify security problems, primarily in the context of PGP messaging and key management.

The project started on time and was specifically executed in mid-August 2020, in CW32 and CW33. It progressed efficiently and leveraged the skills and expertise of six senior testers. The team members invested a total of forty days into the penetration testing and auditing work. The communications for this project were done in a dedicated Matrix chat channel, while the Element platform was used for the Cure53 and the developer teams to gather data, plan and then execute this assignment. The teams were able to invite all relevant personnel into this channel and the communications can be evaluated as

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

consistently fluent and productive, with the in-house team specifically being kept up-to-date about the progress and emerging findings.

The examination yielded a total of fourteen findings, seven of which were considered to be security vulnerabilities of varying severity levels, while the remainder are general weaknesses with lower exploitation potential. It needs to be noted that none of the findings reached *Critical* or even *High* severity scores, which is quite commendable. While the highest severity ascribed to two individual findings stood at *Medium*, most issues actually had limited impact.

In the following sections, the report will first shed light on the scope and key test parameters, as well as briefly details on the WPs. Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Finally, the report will close with broader conclusions about this summer 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team’s observations and collected evidence. Tailored hardening recommendations pertinent to the inspected parts of RNP, Thunderbird and the integration of RNP into Thunderbird - especially as it relates to PGP messaging and some peripheral tasks - are also incorporated into the final section.

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

Scope

- **Penetration tests & audits against RNP & its Thunderbird 78.1 integration**
  - **WP1**: RNP C++ OpenPGP library's handling of PGP messages & logic bugs
    - https://github.com/rnpgp/rnp
    - commit 650b3bce5bad75b64c7d3768d2df74a55740df15
  - **WP2**: Thunderbird integration of RNP C++ library, focus on crypto
    - http://ftp.mozilla.org/pub/thunderbird/candidates/78.1.0-candidates/build1/
  - **WP3**: Thunderbird integration of RNP C++ library
    - *See above*

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 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. *RNP-01-001*) for the purpose of facilitating any future follow-up correspondence.

RNP-01-001 WP1: Integer overflow due to expiration time of PGP v3 keys *(Low)*

While auditing the RNP source code, it was discovered that the function *pgp_key_get_expiration()* is potentially vulnerable against an Integer overflow for PGP key versions below v4. The referred function obtains the validity time in days and multiplies it by the value of 86400 in order to get the expiration time in seconds. The multiplication of *key->pkt.v3_days * 86400* can overflow and, thus, potentially return an incorrect value.

Setting a validity time in days to, e.g., 49711, will result in an integer larger than *MAX_INT* (4294967295), translating to an overflow. This is demonstrated below:

*49711 * 86400 = 4295030400*
4295030400 - 4294967295 (MAX_INT) = 63105

key->pkt.v3_days is defined as *uint16_t* which means it can store an Integer value of up to 65,535. The native function *rnp_key_get_expiration()*, which in turn invokes *pgp_key_get_expiration(),* gets invoked through the JavaScript layer via several function calls, for example *isKeyExpired()* or *addKeyAttributes()*.

Affected Files:
rnp/src/lib/pgp-key.cpp
comm/mail/extensions/openpgp/content/modules/RNP.jsm

Affected Code:

pgp-key.cpp:

```
uint32_t
pgp_key_get_expiration(const pgp_key_t *key)
{
    return (key->pkt.version >= 4) ? key->expiration : key->pkt.v3_days * 86400;
}
```

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

RNP.jsm:

```
[...]
if (RNPLib.rnp_key_get_expiration(handle, key_expiration.address())) {
      throw new Error("rnp_key_get_expiration failed");
    }
    if (key_expiration.value > 0) {
      keyObj.expiryTime = keyObj.keyCreated + key_expiration.value;
    } else {
      keyObj.expiryTime = 0;
    }
    keyObj.expiry = EnigmailTime.getDateTime(keyObj.expiryTime, true, false);
[...]
```

As a result of this potential Integer overflow, it will set an incorrect value of *keyObj.expiryTime*. Cure53 wants to point out that fixing this vulnerability should ensure that the multiplication operation cannot overflow when the expiration time is calculated.

RNP-01-004 WP1: Potential Integer underflow in *partial_dst_write()* *(Low)*

During the audit of the RNP source code, it was discovered that the function *partial_dst_write()* is potentially prone to an Integer overflow due to the declaration of *wrlen* as signed integer. When *wrlen* is becoming negative, the code calling *dst_write()*, wrlen will be cast to *size_t*, a big unsigned integer.

It has to be noted though that the vulnerable code path cannot be reached due to the condition check, namely if *len* is greater than *param->partlen - param->len*. Nevertheless, this issue is reported for the sake of completeness.

Affected File:
rnp/src/librepgp/stream-write.cpp

Affected Code:

```
static rnp_result_t
partial_dst_write(pgp_dest_t *dst, const void *buf, size_t len)
{
    pgp_dest_partial_param_t *param = (pgp_dest_partial_param_t *) dst->param;
    int                      wrlen;

    if (!param) {
        RNP_LOG("wrong param");
        return RNP_ERROR_BAD_PARAMETERS;
    }

    if (len > param->partlen - param->len) {
        /* we have full part - in block and in buf */
        wrlen = param->partlen - param->len;
```

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

```
        dst_write(param->writedst, &param->parthdr, 1);
        dst_write(param->writedst, param->part, param->len);
        dst_write(param->writedst, buf, wrlen);
[...]
```

Cure53 recommends to change the data-type of *wrlen* from "*int*" to *"size_t"* in order to avoid the potential risk of an Integer underflow.

RNP-01-005 WP1: Literal packet parsing allows for Integer underflow *(Low)*

While auditing the provided sources for potential vulnerabilities, it was discovered that a malicious user can set the packet length of a literal packet to an arbitrary length. This results in an Integer underflow upon calculation of the size of the received data. The calculated size is in turn used to ensure that *read* operations are not advancing beyond the provided data. Due to the underflow, this can result in out-of-bound *reads.*

It was impossible to leverage this in a meaningful way during the assignment, which explains the *Low* severity rating. Below is the code excerpt showing the vulnerability with the relevant parts highlighted.

Affected File:
rnp/src/librepgp/stream-parse.cpp

Affected Code:

```
static bool get_pkt_len(uint8_t *hdr, size_t *pktlen)
{
[...]
    switch (hdr[0] & PGP_PTAG_OF_LENGTH_TYPE_MASK) {
    case PGP_PTAG_OLD_LEN_1:
        *pktlen = hdr[1];
        return true;
    case PGP_PTAG_OLD_LEN_2:
        *pktlen = read_uint16(&hdr[1]);
        return true;
    case PGP_PTAG_OLD_LEN_4:
        *pktlen = read_uint32(&hdr[1]);
        return true;
    default:
        return false;
    }
}
[...]

rnp_result_t init_literal_src(pgp_source_t *src, pgp_source_t *readsrc)
{
[...]
```

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

```
    /* Reading packet length/checking whether it is partial */
    if ((ret = init_packet_params(&param->pkt))) {
        goto finish;
    }
[...]
    if (!param->pkt.indeterminate && !param->pkt.partial) {
        src->size = param->pkt.len - (1 + 1 + bt + 4);
        src->knownsize = 1;
    }
[...]
```

As can be seen, an arbitrary 32-bit value can be supplied by a malicious user. In the *init_literal_src()* function, this value is used to calculate the size of a source. The resulting *size* parameter is 64-bit in cardinality. When actually looking at the disassembly, however, it shows that 64-bit arithmetic is performed with signed extension of values.

Disassembly of arithmetic operations:

```
[...]
   0x000000000005209b <+1223>:    mov    rax,QWORD PTR [rbp-0x18]
   0x000000000005209f <+1227>:    mov    rdx,QWORD PTR [rax+0x20]
   0x00000000000520a3 <+1231>:    movzx  eax,BYTE PTR [rbp-0x1d]
   0x00000000000520a7 <+1235>:    movzx  eax,al
   0x00000000000520aa <+1238>:    add    eax,0x6
   0x00000000000520ad <+1241>:    cdqe
   0x00000000000520af <+1243>:    sub    rdx,rax
[...]
```

In order to demonstrate the issue, the following Proof-of-Concept (PoC) PGP message can be sent via mail.

PoC PGP Message:

```
-----BEGIN PGP MESSAGE-----

rARiCWhlbGxvLnR4dFx1TW9IZWxsbywgd29ybGQhCg==
=SXA7
-----END PGP MESSAGE-----
```

Upon receiving and opening the mail, the RNP library is called to parse the message. Using GDB to observe the calculated size makes it possible to inspect the flaw.

GDB breakpoint inspection:

```
(gdb) p /x src->size
$47 = 0xfffffffffffffff5
```

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

Although it was not possible to trigger a memory corruption using this issue within the time given for this assessment, it is nonetheless recommended to ensure that the parsed size is reasonable and that the subtraction cannot result in a value below zero.

RNP-01-006 WP2: Evaluation of password strength insufficient *(Low)*

It was observed that the password strength evaluation function included in the Thunderbird OpenPGP module is highly insufficient.

Affected File:
comm/mail/extensions/openpgp/content/modules/passwordCheck.jsm

Affected Code:
The entire file constitutes the affected code. The password strength algorithm suffers from the following issues:

- **Small "banlist":** A list of roughly 500 banned password items is provided in the *COMPLEXIFY_BANLIST* variable, which largely has negligible impact on the security of the password evaluation process.
- **Insufficient logic**: A password evaluation logic based on minimum characters and the "banlist" above is employed. However, this logic meant that the passwords "*password123"* and "*harrypotter*" (among others) were deemed secure, despite being some of the most used passwords in the world[1].

This functionality does not appear to currently be called from anywhere within the extension, therefore there is no practical security impact. Nevertheless, it is still recommended to either address the password strength algorithm's weaknesses or to remove the code entirely in order to prevent potential future misuse.

RNP-01-007 WP1: *encrypt_secret_key()* does not wipe *keybuf* from memory *(Low)*

While auditing the RNP source code, it was discovered that the function *encrypt_secret_key()* used for encrypting data does not properly clear the derived key stored in *keybuf* before the function returns. *keybuf* is reserved on the stack, and in the case of encryption successful, it does not get wiped from memory using *pgp_forget().* It has to be noted that the buffer gets properly zeroed in memory in the error case, as depicted in the code snippet below.

Affected File:
rnp/src/librepgp/stream-key.cpp

[1] https://en.wikipedia.org/wiki/Wikipedia:10,000_most_common_passwords

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

Affected Code:

```
Rnp_result_t
encrypt_secret_key(pgp_key_pkt_t *key, const char *password, rng_t *rng)
{
       uint8_t           keybuf[PGP_MAX_KEY_SIZE];
       [...]
       /* derive key */
       if (!pgp_s2k_derive_key(&key->sec_protection.s2k, password, keybuf,
       keysize)) {
           RNP_LOG("failed to derive key");
           ret = RNP_ERROR_BAD_PARAMETERS;
           goto error;
       }
       [...]
       if (!pgp_cipher_cfb_start(
             &crypt, key->sec_protection.symm_alg, keybuf, key-
       >sec_protection.iv)) {
           RNP_LOG("failed to start cfb encryption");
           ret = RNP_ERROR_DECRYPT_FAILED;
           goto error;
       }
       pgp_cipher_cfb_encrypt(&crypt, body.data, body.data, body.len);
       pgp_cipher_cfb_finish(&crypt);
       key->sec_data = body.data;
       key->sec_len = body.len;
       [...]
       return RNP_SUCCESS;
rrror:
       pgp_forget(keybuf, sizeof(keybuf));
       pgp_forget(body.data, body.len);
       free_packet_body(&body);
       return ret;
}
```

Sensitive data, such as keys, passwords, etc., should be wiped from memory using the *pgp_forget()* method in order to reduce the potential risk of disclosing sensitive information to local attackers who are in position to read the process memory.

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

RNP-01-012 WP1: Logic issue potentially leaves key material unlocked *(Medium)*

During the audit of the RNP source code, it was discovered that there is a potential logic flaw related to locking/unlocking and protecting/unprotecting keys. The function - named *rnp_key_unlock()* - unlocks a key and has secret key material for use without password protection. This puts all values into memory so the key becomes usable for corresponding operations (e.g. changing attributes or exporting keys).

As an example, the flow of operations is as follows:

1. Unlock the key by invoking *rnp_key_unlock()*
2. Perform some operations on the key, e.g. export it by using *rnp_key_export()*
3. Lock the key again by invoking *rnp_key_lock()*

It was identified that the *rnp_key_lock()* might not be invoked in case the desired operation, in the example above *rnp_key_export()*, is running into an error. This situation potentially allows an attacker to perform operations on already unlocked keys, without needing to unlock them beforehand.

A similar pattern has been observed for *rnp_key_protect/rnp_key_unprotect()*, which gets invoked when keys are imported through *importKeyBlockImpl()*. A protected key is encrypted and can be safely held in memory. Invoking *rnp_key_unprotect()* on a given key removes the encryption from the key.

Affected File:
comm/mail/extensions/openpgp/content/modules/RNP.jsm

Affected Code:

```
async backupSecretKeys(fprs, backupPassword) {
       [...]
       if (RNPLib.rnp_key_unlock(expKey, internalPassword)) {
              throw new Error("rnp_key_unlock failed");
       }
       if (RNPLib.rnp_key_export(expKey, output_to_memory, exportFlags)) {
              throw new Error("rnp_key_export failed");
       }
       if (RNPLib.rnp_key_lock(expKey)) {
              throw new Error("rnp_key_unlock failed");
       }
       [...]

async importKeyBlockImpl(
    win,
    passCB,
    keyBlockStr,
```

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

```
  pubkey,
  seckey,
  permissive = false,
  limitedFPRs = []
) {
    [...]
    if (k.secretAvailable) {
          [...]
          let rv = RNPLib.rnp_key_unprotect(impKey, recentPass);
          [...]
          if (RNPLib.rnp_key_get_subkey_count(impKey, sub_count.address()))
          {
                throw new Error("rnp_key_get_subkey_count failed");
          }
          [...]
    }
    [...]
    if (k.secretAvailable) {

          if (RNPLib.rnp_key_protect(impKey2, newPass, null, null, null, 0))
          {
                throw new Error("rnp_key_protect failed");
          }
    [...]
```

Locking/protecting the keys in case an operation performed on an unlocked/unprotected key fails is important because it reduces the potential risk of having unlocked/unprotected keys in memory. Therefore, it is recommended to properly catch any occurring exceptions and re-lock/protect keys again.

RNP-01-014 WP1: Key manipulation via uncertified Auto-Import *(Medium)*

It was found that Thunderbird allows importing primary keys and subkeys that are not bound to a valid cryptographically secure signature. Additionally, Thunderbird automatically imports detected, attached PGP primary keys with an already trusted fingerprint, as it has an extended expiry time. This introduces the risk of attackers obtaining a primary key with an extended or unset expiry time of a trusted person, while it has not yet been imported into the PGP key ring of the victim.

The attackers can abuse this by adding a malicious subkey, which has been used by Thunderbird for email encryption, signature verification and key certification. Although the subkey is flagged as invalid by RNP, it is silently accepted and imported by Thunderbird as soon as the user opens the attacker's mail.

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

Affected File:
comm/mail/extensions/openpgp/content/ui/enigmailMessengerOverlay.js

Affected Code:

```
commonProcessAttachedKey(keyData, isBinaryAutocrypt) {
 let errorMsgObj = {};
 let preview = EnigmailKey.getKeyListFromKeyBlock(
   keyData,
   [...]
 );
 [...]
 for (let newKey of preview) {
   let oldKey = EnigmailKeyRing.getKeyById(newKey.fpr);
   if (!oldKey) {
     [...]
     continue;
   }
   [...]
   let newHasNewValidity =
     oldKey.expiryTime < newKey.expiryTime ||
     (oldKey.keyTrust != "r" && newKey.keyTrust == "r");

   if (!newHasNewValidity)
     continue;
   [...]
     !EnigmailKeyRing.importKeyDataSilent(
       window,
       keyData,
       isBinaryAutocrypt,
       "0x" + newKey.fpr
     )
[...]
```

It is advised that the validity of the PGP keys returned from the RNP library is respected by Thunderbird and actions ensue accordingly. Additionally, Cure53 recommends for trusted primary keys not getting automatically imported. Instead, the user should be informed and asked about importing the new key. It is crucial to alert the user about every new subkey and user-identity which is about to be newly trusted.

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

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.

RNP-01-002 WP3: Automatic handling of *autocrypt-gossip* header *(Info)*

While auditing the code which integrates the RNP library into Thunderbird, it was discovered that Thunderbird handles the *autocrypt-gossip* header within an encrypted PGP message. This usually allows for automatically updating keys and the current implementation does so without notifying the user. Due to a missing database table, however, the code in question does not function properly. After some discussion with the developers, it turns out that the handling of the *autocrypt-gossip* header is not supposed to be enabled at all. Below is a code excerpt that is responsible for handling the header.

Affected File:
comm/mail/extensions/openpgp/content/modules/mimeDecrypt.jsm

Affected Code:
```
handleResult(exitCode) {
[...]
  try {
    this.extractEncryptedHeaders();
    this.extractAutocryptGossip();
  } catch (ex) {
    console.debug(ex);
  }
[...]
```

Additionally, when the *processAutocryptHeader()* function is called, the supplied *fromAddr* value is extracted from the *autocrypt* header itself. This negates the check whether or not the specified address matches the *From* header, especially since both values become identical.

Affected File:
comm/mail/extensions/openpgp/content/modules/mimeDecrypt.jsm

Affected Code:
```
[...]
for (let i in gossip) {
  let addr = EnigmailMime.getParameter(gossip[i], "addr");
  try {
```

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

```
    let r = await EnigmailAutocrypt.processAutocryptHeader(
      addr,
      [gossip[i].replace(/ /g, "")],
      msgDate,
      true,
      true
    );
[...]
```

This issue was ultimately not exploitable due to the missing database table, however it nevertheless poses a risk. It is therefore recommended to avoid handling this header altogether. Should support for this header be required for some reason, it could be considered to require a confirmation from the user before updating any key material.

RNP-01-003 WP3: Possible race condition when reading from disk *(Info)*

When processing a PGP-signed email, the signature data is extracted and written to the filesystem before calling the RNP library, which in turn reads the signature data back from the filesystem. This introduces a race condition due to the timing and predictable file-path wherein a malicious local user can swap out the dumped file for a malicious signature file. During this assessment, however, it was not possible to exploit this in any meaningful way. The following shows a code excerpt that is responsible for writing the file to disk before calling the RNP library.

Affected File:
comm/mail/extensions/openpgp/content/modules/mimeVerify.jsm

Affected Code:

```
onStopRequest() {
  [...]
  if (this.protocol === PGPMIME_PROTO) {
    [...]
    this.sigFile = EnigmailFiles.getTempDirObj();
    this.sigFile.append("data.sig");
    this.sigFile.createUnique(this.sigFile.NORMAL_FILE_TYPE, 0x180);
    EnigmailFiles.writeFileContents(this.sigFile, this.sigData, 0x180);

    if (!EnigmailDecryption.isReady(win)) {
      return;
    }

    let sigFileName = EnigmailFiles.getEscapedFilename(
      EnigmailFiles.getFilePath(this.sigFile)
    );
    let keyserver = EnigmailPrefs.getPref("autoKeyRetrieve");
    let options = {
```

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

```
      keyserver,
      keyserverProxy: EnigmailHttpProxy.getHttpProxy(keyserver),
      fromAddr: EnigmailDecryption.getFromAddr(win),
      mimeSignatureFile: sigFileName,
    };
    const cApi = EnigmailCryptoAPI();
[...]
    this.returnStatus = cApi.sync(cApi.verifyMime(this.signedData, options));
[...]
```

It is recommended to avoid using the filesystem in order to pass the signature information to the library. Instead, the information should be passed via memory, as done with the signed data itself.

RNP-01-008 WP3: Partially unencrypted email insufficiently detected *(Low)*

It was found that a partially unencrypted email has been insufficiently detected as such by Thunderbird. This introduces the risk of users erroneously assuming the entire email was encrypted and therefore ascribing it with an incorrect security status.

Proof-of-Concept **.eml* file:

```
To: user2@domain.com
From: user <user@domain.com>
[...]
Content-Type: multipart/mixed;
 boundary="------------32989E6E4C7AEB7775BAD494"
[...]
--------------32989E6E4C7AEB7775BAD494
Content-Type: text/html; charset=utf-8

aaa

--------------32989E6E4C7AEB7775BAD494
Content-Type: text/html; charset=utf-8

-----BEGIN PGP MESSAGE-----

hQGMAybeLH[...]PfY=
=7kwb
-----END PGP MESSAGE-----
[...]
--------------32989E6E4C7AEB7775BAD494
Content-Type: text/html; charset=utf-8

zzz

--------------32989E6E4C7AEB7775BAD494--
```

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

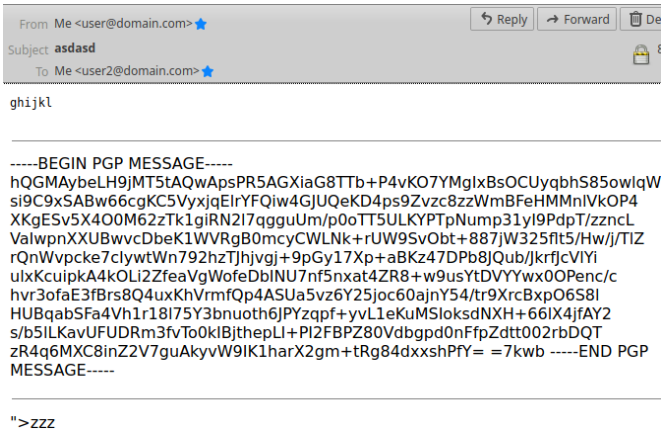

Fig.: A partially encrypted email displayed as fully encrypted.

It is recommended that the logic for detecting encrypted content receives improvements by design. If parts of the message were decrypted, the whole message should be checked for unencrypted parts. If any parts of the parts were not decrypted, the application should alert the user about partially unprotected information.

RNP-01-009 WP1: *mem_dest_own_memory()* callers do not check for *NULL* (*Info*)

During an audit of the RNP source code, it was discovered that the function *mem_dest_own_memory()* allocates memory using *malloc()* and potentially returns *NULL* to the caller in case *malloc()* fails. The caller of this function, *key_to_bytes()*, does not properly check for *NULL* being returned and might return *NULL* to *rnp_get_public_key_data()* and *rnp_get_secret_key_data()*, potentially leading to a Denial-of-Service (DoS) situation when dereferencing a *NULL* pointer.

It is important to emphasize that neither of the two functions - i.e. *rnp_get_public_key_data()* and *rnp_get_secret_key_data()* - are used by the JavaScript layer of Thunderbird at the time of this security audit. As a consequence, this issue is only reported in order to point out the importance of reviewing the RNP source code diligently.

Affected File:
rnp/src/librepgp/stream-common.cpp

Affected Code:

```
void *
mem_dest_own_memory(pgp_dest_t *dst)
{
       [...]
```

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

```
    /* in this case we should copy the memory */
    void *res = malloc(dst->writeb);
    if (res) {
        memcpy(res, param->memory, dst->writeb);
    }
    return res;
}

static rnp_result_t
key_to_bytes(pgp_key_t *key, uint8_t **buf, size_t *buf_len)
{
    [...]
    *buf = (uint8_t *) mem_dest_own_memory(&memdst);
    [...]
}

rnp_result_t
rnp_get_public_key_data(rnp_key_handle_t handle, uint8_t **buf, size_t *buf_len)
try {
    [...]
    return key_to_bytes(key, buf, buf_len);
}

rnp_result_t
rnp_get_secret_key_data(rnp_key_handle_t handle, uint8_t **buf, size_t *buf_len)
try {
    [...]
    return key_to_bytes(key, buf, buf_len);
}
```

Since *malloc()* can potentially return *NULL*, any callers of a function that return memory allocated through *malloc()* should carefully check for *NULL* and return an error in that case.

RNP-01-010 WP1: Outdated and vulnerable Botan library version *(Info)*

According to the official RNP installation documentation[2] (dated to the time of the assessment - 2020.08.14), it is recommended to use Botan 2.12.1. Cross-checking Botan 2.112.1 with the official advisory page[3] of Botan shows that one security issue affects this problem. As quoted from Botan:

2020-03-24: Side channel during CBC padding

[2] https://github.com/rnpgp/rnp/blob/master/docs/installation.adoc
[3] https://botan.randombit.net/handbook/security.html

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

The CBC padding operations were not constant time and as a result would leak the length of the plaintext values which were being padded to an attacker running a side channel attack via shared resources such as cache or branch predictor. No information about the contents was leaked, but the length alone might be used to make inferences about the contents. This issue affects TLS CBC ciphersuites as well as CBC encryption using PKCS7 or other similar padding mechanisms. In all cases, the unpadding operations were already constant time and are not affected. Reported by Maximilian Blochberger of Universität Hamburg.

Fixed in 2.14.0, all prior versions affected.

The recommended version of Botan should be checked and it must be ensured that a recent version is used to prevent potential exploitation of existing vulnerabilities.

RNP-01-011 WP1: Potential overflow in *librepgp* due to invalid size check *(Low)*

While investigating the handling of streams in the RNP library, it was discovered that a function used to initialize output streams checks the length of the file-path incorrectly. This could enable attackers with control over the path parameter to write a *NULL*l character past an allocated buffer. Shown below is the affected source code in the RNP library.

Affected File:
rnp/src/librepgp/stream-common.cpp

Affected Code:

```
init_file_dest(pgp_dest_t *dst, const char *path, bool overwrite)
{
    [...]
    pgp_dest_file_param_t *param;

    if (strlen(path) > sizeof(param->path)) {
        RNP_LOG("path too long");
        return RNP_ERROR_BAD_PARAMETERS;
    }
    [...]
    param = (pgp_dest_file_param_t *) dst->param;
    param->fd = fd;
    strcpy(param->path, path);
[...]
```

In order to reject overly long paths, it is checked if the length of the parameter exceeds the size of the buffer the string is copied to. However, the check does not account for the additional *NULL* character that is appended by *strcpy()*. If the path's length matched the

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

size of *param->path*, *NULL* character would be written past the buffer. It was not further investigated if the issue is relevant in a remote context. It is recommended to reject paths with a size equal to the size of *param->path* in order to ensure that the additional *NULL* character written by *strcpy()* fits in any case.

RNP-01-013 WP2: Forbidden cipher-suites/algorithms recommendations *(Info)*

Code was observed in the RNP/Thunderbird integration layer within the OpenPGP module. This seems to indicate a placeholder for the future blocklisting of algorithms within the usage of OpenPGP in Thunderbird. Recommendations are provided here with regards to how to best follow up with future blocklisting procedure.

Affected File:
comm/mail/extensions/openpgp/content/modules/RNP.jsm

Affected Code:

```
policyForbidsAlg(alg) {
    // TODO: implement policy
    // Currently, all algorithms are allowed
    return false;
},
```

Since PGP benefits from a truly federated ecosystem, any forbidding of algorithms could cause breakage with other PGP users that do not rely on Thunderbird. Such breakage can only occur on decryption however, and not in the process of key generation, import or encryption. It is considered possible for Thunderbird to impose limits which rule out insecure algorithms without a serious impact on usability or availability.

RFC4880[4] specifies the following algorithms for support and usage in OpenPGP:

```
9.1.  Public-Key Algorithms

      ID           Algorithm
      --           ---------
      1          - RSA (Encrypt or Sign) [HAC]
      2          - RSA Encrypt-Only [HAC]
      3          - RSA Sign-Only [HAC]
      16         - Elgamal (Encrypt-Only) [ELGAMAL] [HAC]
      17         - DSA (Digital Signature Algorithm) [FIPS186] [HAC]
      18         - Reserved for Elliptic Curve
      19         - Reserved for ECDSA
      20         - Reserved (formerly Elgamal Encrypt or Sign)
      21         - Reserved for Diffie-Hellman (X9.42,
```

[4] https://tools.ietf.org/html/rfc4880

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

```
                  as defined for IETF-S/MIME)
```

9.2. Symmetric-Key Algorithms

```
 ID           Algorithm
 --           ---------
 0          - Plaintext or unencrypted data
 1          - IDEA [IDEA]
 2          - TripleDES (DES-EDE, [SCHNEIER] [HAC] -
              168 bit key derived from 192)
 3          - CAST5 (128 bit key, as per [RFC2144])
 4          - Blowfish (128 bit key, 16 rounds) [BLOWFISH]
 5          - Reserved
 6          - Reserved
 7          - AES with 128-bit key [AES]
 8          - AES with 192-bit key
 9          - AES with 256-bit key
 10         - Twofish with 256-bit key [TWOFISH]
```

9.4. Hash Algorithms

```
ID           Algorithm                          Text Name
--           ---------                          ---------
1          - MD5 [HAC]                          "MD5"
2          - SHA-1 [FIPS180]                    "SHA1"
3          - RIPE-MD/160 [HAC]                  "RIPEMD160"
4          - Reserved
5          - Reserved
6          - Reserved
7          - Reserved
8          - SHA256 [FIPS180]                   "SHA256"
9          - SHA384 [FIPS180]                   "SHA384"
10         - SHA512 [FIPS180]                   "SHA512"
11         - SHA224 [FIPS180]                   "SHA224"
```

Similarly, GnuPG, the most popular implementation of OpenPGP, supports the following algorithms (as of version 2.2.19):

```
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
```

Based on the above information, the following recommendations can be made in terms of restricting support for algorithms:

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

Public Key Algorithms:

- **RSA:** RSA keys which are smaller than 2048-bit should be disallowed, as per the recommendations made in NIST Special Publication 800-57 Part 3 Revision 1[5]. RSA public key cryptography usage should be supported otherwise.
- **ElGamal:** The default GnuPG setting of 2048-bit keys for ElGamal should be enforced, and smaller keys should be rejected.
- **DSA:** DSA usage should be discouraged and perhaps eliminated entirely in favor of EdDSA, which is supported in the most recent versions of most PGP implementations. If DSA is to be used, a key length of 2048-bit is recommended.

Symmetric Ciphers:

- **CAST5, IDEA, Blowfish and 3DES:** While not generally considered to be practically "broken", usage of CAST5, IDEA, Blowfish and 3DES should be forbidden due to recent advancement in the practical exploitation of ciphers with a block size of 64-bit[6]. These four block ciphers suffer from a 64-bit block size and therefore are best avoided in favor of AES.

Hashing Algorithms:

- **SHA1:** Usage of SHA1 should be forbidden entirely due to recent major advancements, which allow the practical obtention of chosen-prefix collisions on SHA-1, specifically with an impact on the PGP ecosystem.[7]

The current optimal cipher configuration for OpenPGP in general appears to be Ed25519 for signing, RSA-2048 for public key cryptography, AES for symmetric encryption and SHA2 for hashing.

[5] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf
[6] https://sweet32.info/
[7] https://eprint.iacr.org/2020/014.pdf

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

Conclusions

It should be reiterated that this Cure53 summer 2020 examination represented a highly complex audit connected with an unusually intricate integration of an important application. After spending forty days deploying penetration tests and audit methods on the RNP C++ OpenPGP implementation and its integration with the Thunderbird email client, six members of the test team identified both strengths and weaknesses within the given scope. On the one hand, finding only fourteen flaws in the complex can be seen as a positive indicator. On the other hand, the testing team needed a very long time to become familiar with the targets due to their complexity, in particular in terms of how Thunderbird uses the RNP library. From this perspective, the findings must be seen in the broader context.

To clarify, the auditors had to come up with their own tooling to trace calls and create execution graphs, so as to make sense of the integration logic. While the testers generally encountered well-written and logically structured code, especially the originally Enigmail-based OpenPGP code within Thunderbird necessitates dead code elimination and some general clean-up activities. As a side-effect of legacy and remnants, vulnerabilities have been unintentionally and coincidentally mitigated on multiple occurrences. The bottom line is, however, that even minimal modifications to seemingly security-irrelevant aspects could make this fragile code insecure quite quickly.

Moving on to specific findings, it should be underlined that some of the integration code seems to have been merged pretty much straightforwardly from the original Enigmail codebase, consequently leading to issues such as RNP-01-002. The RNP integration passes user-data directly to the C/C++ library where most of the parsing and processing takes place. But, the C/C++ aspect and the JavaScript code still contains TODO items, some of which are security-relevant.

The C/C++ part seems well written but not completely free from inconsistencies, for example when it comes to the use of *malloc()/calloc()*. There are also some calls to inherently dangerous functions like *strcpy()*. In most cases the buffers are checked manually, yet RNP-01-011 demonstrates how the outcome may be flawed. Therefore, it is generally advisable to simplify the code by removing its dead parts, minimizing redundancies and increasing code reuse. It has to be noted that PGP and mail encryption is a complex process, as proven by many implementation- and protocol-flaws from the past. Accounting for this fact, Thunderbird makes a good overall impression in identifying, understanding and handling risks associated with the PGP mail encryption.

Despite this, it is quite likely that the limited time-frame of this audit may have been insufficient to protect Thunderbirds users against state-funded advanced persistent

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

threat, especially those with enough resources to identify high-severity bugs either in the Key distribution/WKS auto-look up, the RNP library or the integration of RNP into Thunderbird. Again, this corresponds to the overwhelming complexity and attack-surface associated with PGP and e-mail related cryptography more broadly.

Generally there are a few instances where things were spotted that could have a pretty severe impact, such as the already mentioned RNP-01-002 and RNP-01-014 serving as another example. Both showcase issues which are prevented by code that does not give the impression of having the purpose to prevent said issues. For instance, the reason why RNP-01-002 could not be exploited, was the missing database table. If the table had existed, the vulnerability would have had a pretty severe impact.

On the Thunderbird integration side, Cure53 checked for various issues revolving around signature verification. It was tested if providing a message where only a part is signed was possible, with the goal of tricking the UI to indicate that the entire message is signed. Given that Enigmail has undergone a previous audit, the integration into Thunderbird had a good foundation to begin with and, ultimately, no severe issues were uncovered. Stemming from the fact that the integration is implemented using JavaScript, memory corruption issues are not really as prevalent as is typically the case with C/C++ implementations. Because of this, extra focus was given to logical issues, which are described in more detail in RNP-01-002 or RNP-01-012.

On the RNP library side, the code is fairly complex and hard to track given the design of nested objects, with various items pointing to various *read* functions for reading the actual input. The parsing of PGP packets and general handling of input data was inspected in finer detail. It can be stated that parsing and handling is fairly correct overall, with some minor exceptions such as RNP-01-005. The code was additionally checked for typical use-after-free issues, yet none were discovered in the time allocated for this assessment. Although the issues mentioned in this report do not exceed *Medium* severity, some improvements can still be made.

The usage of the cryptographic implementation in the underlying RNP cryptographic library was audited, along with the integration of PGP functionality into Thunderbird via the JavaScript bridge that binds RNP and other functionality to the Thunderbird OpenPGP extension. While no issues were found with the cryptographic implementations present in the C/C++ RNP library, the large size of the library and the fact that it handles PGP data structures, often of arbitrary length, makes it impossible to completely rule out the presence of memory safety or overflow issues. The Thunderbird OpenPGP module had some incomplete elements, such as the Autocrypt implementation and the password strength checker. The former resulted in RNP-01-002

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

and the latter can be seen in RNP-01-006. The Thunderbird integration layer withstood the Cure53 attempts at a compromise.

The JavaScript code has some signs of being inconsistent. The function *encryptAndOrSign()* in R*NP.jsm,* for example, invokes functions and usually checks for proper return values. Within the same function though, inside the *else*-path of an *if*-statement, the code invokes a similar kind of operation, i.e. *rnp_op_sign_set_hash(),* without the return value being checked. Even though this has no security impact *per se* and calculating a hash should not fail, this is an indicator that the developers do not follow a consistent approach.

Binary fuzzing was out-of-scope, nevertheless it is considered crucial to add further testing harnesses to RNP, since only *rnp_dump_packets_to_output()* and *rnp_load_keys()* are fuzzing targets at the time of the audit. Botan, which is the library used by RNP to perform crypto operations, should be monitored more regularly for any security advisories that come up. At the time of the security audit, RNP still recommended to use an outdated and potentially vulnerable Botan version as part of their official installation guideline.

While performing the audit against Thunderbird, researchers of the Ruhr University Bochum published a paper[8] which introduces three key attacks against email encryption that are related to Thunderbird more broadly. Although not always related to RNP, several risks are emitted by those key attacks that will be briefly inspected here. The first attack vector, *Key Replacement (A1),* allows attackers to silently import S/MIME keys to the victim's keystore. Although this is not related to RNP, a similar PGP-related issue was found in this audit and described in RNP-01-014. Both issues silently import key data without user-interaction while RNP-01-014 additionally does not properly verify the validity of the keys imported by RNP. It is, therefore, advisable to prompt for user-interaction when modifying trusted keys of the user or changing the import behavior.

The second attack concerns the *Decryption/Sign Oracle (A2)* and requires an attacker-controlled IMAP server in which Thunderbird stores unencrypted draft messages. It is not advisable, as argued by the authors of this paper, to solely mitigate this attack by limiting decryption to the receiving context. Applying this technique would grant attackers an equivalent decryption oracle by moving the encrypted email into the receiving context - that is the inbox folder - of the IMAP server. If the victim clicks on *Reply-To* to that message - similar to what has been described in *TB-01-005*, the contents are decrypted and added to a new composed message and eventually stored unencrypted into the *Drafts* folder before the victim removes the contents prior to sending.

[8] "*Mailto: Me Your Secrets.On Bugs and Features in Email End-to-End Encryption*", J. Mueller et al., 2020, https://www.nds.ruhr-uni-bochum.de/media/nds/veroeffentlichungen/2020/08/15/mailto-paper.pdf

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

To mitigate this issue, it is therefore generally recommended that draft messages (and all similar folders) should be stored encrypted *but especially unsigned* with the *sender's* PGP key. This will only turn the attacker's decryption oracle into an encryption oracle when the attacker has control over the IMAP server. The message parts extracted from the *mailto:* pseudo protocol should only be inserted into the editor as a plain-text message *that will not be decrypted*. If technical limitations deny encrypting an individual draft email, the user should be prompted for interaction. While implementing the mitigations, it should be explicitly checked that *TB-01-005* cannot be revived by attackers.

Finally, the *Key Extraction attack (A3)* allows attaching arbitrary files accessible from Thunderbird to a newly composed email. If the user is not focused and sends the email, sensitive PGP keys and other files could be exposed. To raise the alertness of the user, it is advisable to prompt for interaction if the suggested files should be attached. Alternatively, this feature could be removed as a whole.

To sum up, the Cure53 auditors believe that they achieved good coverage during this summer 2020 project, though judging by the complexity and the limited budget granted to the assessment, there is still plenty of need for additional auditing at some future point in time. While the testers believe that an exhaustive analysis of the entire OpenPGP/RNP complex is impossible to achieve within such a relatively short time frame, the clear absence of the *High* and *Critical* issues stands.

Cure53 would like to thank Jochai Ben-Avie & Tom Ritter of Mozilla for their excellent project coordination, support and assistance, both before and during this assignment. Cure53 would further like to express gratitude to Kai Engert, Ronald Tse and Nickolay Olshevsky as well as the rest of the maintainer team who aided the assignment with valuable advice and input.