
Security Report

of Audit of "pEp Engine" on behalf of p≡p

For: p≡p
Switzerland/Luxembourg

By: SektionEins GmbH
- from here on "SektionEins" -
Breite Straße 159
D-50667 Cologne
Germany

Author: Ben Fuhrmannek

Contents

Document Version History	5
1. v0.1 - 2016-02-15 - Internal Draft	5
2. v0.2 - 2016-07-08	5
3. v0.2a - 2016-07-20	5
4. v0.2b - 2016-07-27	5
5. v0.2c - 2016-08-18	5
6. v0.2d - 2016-09-07	6
7. v0.2e - 2016-10-18	6
8. Notes	6
Executive Summary	7
General	8
1. Persons	9
2. Procedure	9
2.1. Source Code Audit	9
2.2. Issue reporting	10
2.3. Report generation	10
2.4. Re-audit and report extension	11
2.5. Risik Assessment	11
Issues	12
1. Buffer Overflows	12
1.1. Local buffer overflows	12
1.2. Buffer overflow: platform_unix.c: gpg_conf_path	13
2. Missing validity checks	14
2.1. Missing NULL check: only assert() in identity_list_add . . .	14
2.2. assert() without if in update_identity()	14
2.3. assert(keylist) without if in keymanagement.c: myself() . . .	15
2.4. assert without if in keymanagement.c: do_keymanagement() .	15
2.5. Unchecked limit: ensure_config_values max input values .	15
2.6. Unchecked input: ensure_config_values key value length .	16
2.7. assert without if in message_api.c: encrypt_PGP_MIME() . .	16
2.8. assert without if in message_api.c: copy_fields()	17
2.9. assert without if in stringpair_list_append()	17

3.	Memory leaks	18
3.1.	Memleak: etpan_mime.c: part_new_empty()	18
3.2.	Memleak in identity_list_dup()	19
3.3.	Memleak in myself(): _identity is never freed	19
3.4.	Possible memleaks in myself(): keylist after return	20
3.5.	Memleak *pair in message_api.c in error case	20
3.6.	Possible memleak: message_api.c: encrypt_PGP_MIME() . .	21
3.7.	Possible memleak: message_api.c: encrypt_PGP_in_pieces ciphertext not freed before reuse	21
3.8.	Memleak ptext not freed before reuse in message_api.c: _decrypt_message()	21
4.	Other memory management issues	23
4.1.	Double free(ptext) in message_api.c:encrypt_PGP_MIME in case of enomem: or pep_error	23
4.2.	Double free - Incorrect memory allocation in stringpair_list_append()	23
5.	Recursion in C	24
5.1.	Unnecessary recursion: stringlist_dup()	24
5.2.	Unnecessary recursion in bloblist.c: free_bloblist()	24
5.3.	Unnecessary recursion: bloblist_dup()	24
5.4.	Tail-recursion: bloblist_add()	25
5.5.	Unnecessary recursion: identity_list_dup()	25
5.6.	Unnecessary recursion: free_identity_list()	25
5.7.	Unnecessary recursion: identity_list_add()	25
5.8.	Unnecessary recursion: stringpair_list_dup()	26
5.9.	Unnecessary recursion: stringpair_list_add()	26
5.10.	Unnecessary recursion: stringlist_add()	26
5.11.	Unnecessary recursion: free_stringlist()	26
6.	Other issues	27
6.1.	Sqlite: old version 3.8.6 instead of 3.8.11.1	27
6.2.	Missing NULL checks in free_bloblist	27
6.3.	Sqlite: embedded instead of just linked statically	27
6.4.	Improvement: use strl*() instead of str*() style functions . .	28
6.5.	Possibly unintended sideeffect: seperate_short_and_long() .	29
6.6.	Unchecked return value: update_identity() in message_api.c	30
6.7.	Missing out-of-memory check after malloc()	30

6.8.	31 bit random number generator with static seed (Android)	31
7.	Comments	32
7.1.	Improvement: keymanagement.c: update_identity(): strdup instead of strdup	32
7.2.	bloblist_add() does not allow empty blobs	32
7.3.	Possible caveat: is_fileending() returns false if strings are equal	32
7.4.	decrypt_message() auto-imports attached keys	33
7.5.	Typo in function name seperate_short_and_long()	33
7.6.	Potentially incorrect logic in fread() wrapper	34
7.7.	Stringpair struct naming prone to errors	34
7.8.	Confusing behaviour: stringpair_list_add(. . . , NULL)	35
Notes	36

Document Version History

1. v0.1 - 2016-02-15 - Internal Draft

- after five days of code review in August/September 2015
- HG revision: 342:4c626ba43fa7

2. v0.2 - 2016-07-08

- after four days of code review in July 2016
- HG revision: 798:2dce08e4fc24

In order to speed up the auditing process for this first release for the pEpEngine the following unused files have been omitted from the audit:

- /sync
- /asn1
- /src/sync*
- /src/*asn1*
- /src/pgp_netpgp* (netpgp is used with iOS)

3. v0.2a - 2016-07-20

- re-checked all issues after bugfixes on July 20th 2016
- HG revision: 909:e39e9fec2da0

4. v0.2b - 2016-07-27

- added section “Executive Summary”
- added status *fixed* for fixed issues as requested by pEp
- added clarification, that “*reviewed*” really refers to an actual code review by the auditor

5. v0.2c - 2016-08-18

- re-checked all open issues
- audit of changed code since HG revision 909
- HG revision: 1028:a1ba15c1ac04

6. v0.2d - 2016-09-07

- It should be noted that HG revision 1028:a1ba15c1ac04 was tagged with the release version 0.8.0¹.

7. v0.2e - 2016-10-18

- Clarified pEp legal entities on the front page and appended a notes section with all addresses.

8. Notes

- Code revision numbers directly relate to the mercurial (hg) development repository publicly available here: <https://cacert.pep.foundation/trac/>. If no branch name is explicitly given, the *default* branch was audited.
- The versioning scheme for this document is 0.x where x increments for each audit. If a majority of issues is resolved, the version may change to 1.0. Up to 26 minor changes to the document are indicated with an incrementing letter postfix, e.g. 0.2a.

¹see <https://cacert.pep.foundation/dev/repos/pEpEngine/rev/d80a4c0d77e4>

Executive Summary

A source code audit of the “pEp engine” was performed by SektionEins on behalf of p≡p.

The following table outlines all issues with a risk rating of *critical*, *high* or *medium*. *Trac ID* corresponds to the issue ID in the publicly accessible bug tracker <https://cacert.pep.foundation/trac/>. All issues should have a status of *fixed* or *closed*.

Trac ID	Severity	Status	Description
#40	high	fixed	Double free(ptext) in message_api.c:encrypt_PGP_MIME in case of enomem: or pep_error
#95	high	fixed	Double free - Incorrect memory allocation in stringpair_list_append()
#15	high	closed	Buffer overflow: platform_unix.c: gpg_conf_path
#14	high	fixed	Local buffer overflows
#10	medium	fixed	Missing out-of-memory check after malloc()
#13	medium	fixed	Sqlite: old version 3.8.6 instead of 3.8.11.1
#21	medium	fixed	Memleak: etpan_mime.c: part_new_empty()
#24	medium	fixed	Memleak in identity_list_dup()
#30	medium	fixed	Memleak in myself(): _identity is never freed
#35	medium	fixed	Memleak *pair in message_api.c in error case
#36	medium	fixed	Improvement: use strl*() instead of str*() style functions

General

This document describes the source code audit of the “pEp engine” that was performed by SektionEins on behalf of p≡p. Additional reviews are shown in the version history of this document. All testing was done remotely from the SektionEins office in Cologne, Germany. SektionEins was provided with the application’s source code and access to a Trac system for documentation access and issue reporting.

The main goal of the audit was an evaluation of the pEp engine in order to harden the code and protect it from attackers with knowledge of the source code, as the pEp engine is or will eventually be open source software. Security relevant issues were identified in the code and first documented in the Trac system.

The pEp engine consists of roughly 9819 lines of C code (without comments/empty lines and libraries).

The embedded sqlite library was only checked implicitly for known vulnerabilities by comparing versions.

What is the pEp engine? "The p≡p engine is a Free Software library encapsulating implementations of:

- Key Management
- Trust Rating
- Abstract Crypto API
- Message Transports

p≡p engine is written in C99. It is not meant to be used in application code directly. Instead, p≡p engine is coming together with a list of software adapters for a variety of programming languages and development environments."¹

¹shortened excerpt from README.txt

1. Persons

The audit and its documentation was performed by Ben Fuhrmannek.

2. Procedure

The audit consisted of the following parts:

1. Source Code Audit
2. Issue Reporting
3. Report Generation
4. Re-audit
5. Report Extension

2.1. Source Code Audit

The source code audit is the most important part of the auditing process. This audit mainly consisted of manually reading source code.

The pEp engine is written in C99. As such, a particular focus was placed on checking memory management and string handling.

From a high-level perspective the source code audit was done systematically by checking

- Input and Data Flow
- Trust Relationships
- Assumptions and Misplaced Trust
- Interfaces
- Environmental Attacks
- Exceptions/Signals/Interrupts

Specifically the code was checked in-depth for the following security related problems:

- Memory Corruption
- Off-by-One Errors
- Stack Overflows
- Heap Overflows
- Data/String Encoding Problems
- Byte Order
- Arithmetic Boundaries

- Type Conversion Problems: Signed/Unsigned, Truncation, Comparison
- Pointer Arithmetic
- Typos
- Algorithmic Boundaries, e.g. non-deterministic behaviour, infinite loops, recursion
- Algorithmic Flaws
- String Handling: bound vs. unbound functions
- Meta characters and Filters
- File Handling, e.g. permissions
- Race Conditions
- Process related problems: e.g. IPC, Signal handling
- High Level Flaws: Logical errors, Invalid/Incomplete Protocol implementation
- Network related problems with low level (IP/TCP/Firewall), high level protocols (e.g. HTTP, SMTP)

2.2. Issue reporting

All issues listed in this report have been added to the Trac ticket system publicly available here: <https://cacert.pep.foundation/trac/>

Note: The *priority* field in Trac may not be related to the *severity* of the issue.

2.3. Report generation

Each audit is finalized with a result report detailing all findings and comments. Issues are reported with unbiased technical facts. Apart from answering technical questions the p≡p project was not involved in the actual auditing process or the report generation.

Each issue is outlined with a title and the following information:

- Status: open/fixed/closed
- Severity: critical/high/medium/low/comment
- Tracker ID: #YY
- Added with document version: XX

Some issues are described further with code samples and recommendations where necessary.

Issues can have one of the following statuses:

- *open*: The issue is not resolved.
- *fixed*: The issue and its corresponding source code was reviewed by SektionEins and is now resolved.
- *closed*: The issue is resolved. This state can occur in any of the following cases:
 - The issue was merged with another issue.

- The issue was resolved without code changes, e.g. by clarifying an update process.
- The issue was no security bug, but a comment on suspicious behaviour, and it was explained by pEp that it works as intended.
- the issue was invalid.

2.4. Re-audit and report extension

After each re-audit of the pEp engine, this report is updated. Updates to this report are done by adding a document version and outlining any changes either in the document version history or with the issue changed.

Issues that turn out to be invalid will be documented as such or may be removed entirely.

The cycle of re-audit and report extension should be repeated until all issues will have been resolved.

Note: Issues are only closed after a review of the corresponding source code by SektionEins. A short comment such as *“closed after review”* or *“closed after bugfix”* means the auditor closed the issue after a code review done by SektionEins.

2.5. Risik Assessment

For each issue uncovered during the audit, a simple risk assessment was performed. Based on damage potential and exploitability as evaluated by the auditor, the following risk levels were assigned:

- *Critical:* Critical issues are a serious security risk and must be fixed immediately. Example: remote code execution
- *High:* High risk vulnerabilities can compromise the entire application under certain conditions. Example: hard to exploit buffer overflows
- *Medium:* Medium level problems may not be sufficient on its own to build an exploit, but can lead to unexpected application behaviour or crashes. Example: memory allocation without return value check
- *Low:* Low risk vulnerabilities cannot easily be exploited on its own, but may become a problem in combination with other issues. Example: minor memory leaks
- *Comment:* Recommendations and issues not related to security vulnerabilities are marked as comments.

Issues

1. Buffer Overflows

1.1. Local buffer overflows

- Status: fixed
- Severity: high
- Tracker ID: #14
- Added with document version: 0.1

In case the length of the value of the environment variable TRUSTWORDS exceeds $\text{MAX_PATH} - 2$ (here MAX_PATH is 1024), then len is set to a negative value, which in case of the unsigned type size_t is a rather big number ($2^8 - 1$ or $2^8 - 2$). As a result, the check in line 70 passes and the strncpy leads to a buffer overflow.

Listing 9.1: platform_unix.c

```
66 if (tw_env = getenv("TRUSTWORDS")) {
67     char *p = stpncpy(buffer, tw_env, MAX_PATH);
68     size_t len = MAX_PATH - (p - buffer) - 2;
69
70     if (len < strlen(SYSTEM_DB_FILENAME)) {
71         assert(0);
72         return NULL;
73     }
74
75     *p++ = '/';
76     strncpy(p, SYSTEM_DB_FILENAME, len);
77     done = true;
78 } else {
```

The same problem can be found in unix_local_db

Listing 9.2: platform_unix.c

```
96 if ((home_env = getenv("HOME"))){
97     char *p = stpncpy(buffer, home_env, MAX_PATH);
98     size_t len = MAX_PATH - (p - buffer) - 2;
```

and twice more in ensure_gpg_home()

Listing 9.3: platform_unix.c

```
133 len = MAX_PATH - (p - path) - 2;
```

Listing 9.4: platform_unix.c

```
144 len = MAX_PATH - (p - path) - 3;
```

and once more in `ensure_gpg_agent_conf()`

Listing 9.5: `platform_unix.c`

```
203 size_t len = MAX_PATH - (p - agent_path) - 2;
```

Updates in version 0.2 of this document:

- Reopened this issue in Trac
- Added a more elaborate description

Updates in version 0.2a of this document:

- Issue reviewed and closed

1.2. Buffer overflow: `platform_unix.c: gpg_conf_path`

- Status: closed
- Severity: high
- Tracker ID: #15
- Added with document version: 0.1

The following check may fail:

```
if (len < strlen(gpg_conf_path) + strlen(gpg_conf_name))
```

Updates in version 0.2 of this document:

- This issue was added to issue #14 and closed.

2. Missing validity checks

In several instances variables are being checked for valid values using `assert()` only. However, `assert()` in C can and should be removed for production with the `-DNDEBUG` compile flag. In addition to `assert()` the program flow should be protected by proper conditional statements.

2.1. Missing NULL check: only `assert()` in `identity_list_add`

- Status: fixed
- Severity: low
- Tracker ID: #26
- Added with document version: 0.1

Listing 9.6: `identity_list.c`

```
53 DYNAMIC_API identity_list *identity_list_add(identity_list *id_list, pEp_identity ↵  
    *ident)  
54 {  
55     assert(ident);
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

2.2. `assert()` without if in `update_identity()`

- Status: fixed
- Severity: low
- Tracker ID: #29
- Added with document version: 0.1

Listing 9.7: `keymanagement.c`

```
179 status = set_identity(session, identity);  
180 assert(status == PEP_STATUS_OK);
```

Updates in version 0.2 of this document:

- Closed issue after bugfix.

2.3. assert(keylist) without if in keymanagement.c: myself()

- Status: fixed
- Severity: low
- Tracker ID: #32
- Added with document version: 0.1

Listing 9.8: keymanagement.c

```
233 status = find_keys(session, identity->address, &keylist);
234 assert(status != PEP_OUT_OF_MEMORY);
235 if (status == PEP_OUT_OF_MEMORY)
236     return PEP_OUT_OF_MEMORY;
237
238 assert(keylist);
```

Updates in version 0.2 of this document:

- Closed issue after code changed significantly and issue is gone.

2.4. assert without if in keymanagement.c: do_keymanagement()

- Status: fixed
- Severity: low
- Tracker ID: #33
- Added with document version: 0.1

Listing 9.9: keymanagement.c

```
304 if (identity->me) {
305     status = myself(session, identity);
306     assert(status != PEP_OUT_OF_MEMORY);
307 } else {
308     status = recv_key(session, identity->address);
309     assert(status != PEP_OUT_OF_MEMORY);
310 }
```

Updates in version 0.2 of this document:

- Closed issue after bugfix.

2.5. Unchecked limit: ensure_config_values max input values

- Status: fixed
- Severity: comment
- Tracker ID: #91
- Added with document version: 0.2

The function `ensure_config_values()` can only handle 32 key/value pairs as input. This limit is checked by `assert()`, which is insufficient.

Listing 9.10: `pgp_gpg.c`

```
32 assert(length <= sizeof(unsigned int) * CHAR_BIT);
```

`ensure_config_values()` tries to match keys from a given key/value list to the keys of a `gpg.conf`-style config file and appends keys and values where the keys were not found. This is also limited to 32 key/value pairs.

At the moment there are no calls to this function with a list bigger than 32 elements. However, in order to avoid future problems, a hard check is recommended.

Updates in version 0.2a of this document:

- Bugfix addresses the issue correctly, but leaks the file pointer `f`, now.
- Issue reopened in Trac.

Updates in version 0.2c of this document:

- Issue reviewed and closed

2.6. Unchecked input: `ensure_config_values` key value length

- Status: fixed
- Severity: low
- Tracker ID: #92
- Added with document version: 0.2

The function `ensure_config_values()` expects a key/value list as two lists for key and values separately. Equal length of these lists is checked only by `assert()`, which is insufficient.

The `pEp` engine provides a suitable implementation of a key/value list named *stringpair list*. An easy way to fix this issue would be to use this *stringpair list* instead of two separate lists.

Updates in version 0.2a of this document:

- Issue reviewed and closed

2.7. `assert` without `if` in `message_api.c`: `encrypt_PGP_MIME()`

- Status: fixed
- Severity: low
- Tracker ID: #39
- Added with document version: 0.1

Listing 9.11: message_api.c

```
341 status = mime_encode_message(_src, true, &mimetext);  
342 assert(status == PEP_STATUS_OK);
```

Updates in version 0.2 of this document:

- Updated line numbers

Updates in version 0.2a of this document:

- Issue reviewed and closed

2.8. assert without if in message_api.c: copy_fields()

- Status: fixed
- Severity: low
- Tracker ID: #93
- Added with document version: 0.2

Listing 9.12: message_api.c

```
164 static PEP_STATUS copy_fields(message *dst, const message *src)  
165 {  
166     assert(dst);  
167     assert(src);
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

2.9. assert without if in stringpair_list_append()

- Status: fixed
- Severity: low
- Tracker ID: #94
- Added with document version: 0.2

Listing 9.13: stringpair.c

```
118 assert(stringpair_list);
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

3. Memory leaks

This category describes problems with memory management where memory is allocated but not freed. If this broken memory handling repeats it may eventually exhaust the entire memory and lead to errors.

Note: The term *memory leak* is abbreviated as *memleak*.

3.1. Memleak: etpan_mime.c: part_new_empty()

- Status: fixed
- Severity: medium
- Tracker ID: #21
- Added with document version: 0.1

The variable `param` is never freed in case of `enomem.mailmime_parameter_new()` allocates memory, which will be properly freed after `clist_append(parameters, param)` (line 115). If anything happens before line 115, e.g. `goto enomem` (line 109), the memory is never freed.

Listing 9.14: etpan_mime.c

```
98 param = mailmime_parameter_new(attr_name, attr_value);
99 assert(param);
100 if (param == NULL)
101     goto enomem;
102 attr_name = NULL;
103 attr_value = NULL;
104
105 if (content->ct_parameters == NULL) {
106     parameters = clist_new();
107     assert(parameters);
108     if (parameters == NULL)
109         goto enomem;
110 }
111 else {
112     parameters = content->ct_parameters;
113 }
114
115 r = clist_append(parameters, param);
```

Updates in version 0.2 of this document:

- Reopened issue
- Added detailed description

Updates in version 0.2a of this document:

- Issue reviewed and closed

3.2. Memleak in identity_list_dup()

- Status: fixed
- Severity: medium
- Tracker ID: #24
- Added with document version: 0.1

The variable `_ident` is never freed in case of an error later in the function.

Listing 9.15: identity_list.c

```
21 DYNAMIC_API identity_list *identity_list_dup(const identity_list *src)
22 {
23     assert(src);
24
25     pEp_identity *_ident = identity_dup(src->ident);
26     if (_ident == NULL)
27         return NULL;
28
29     identity_list *id_list = new_identity_list(_ident);
30     if (id_list == NULL)
31         return NULL;
32
33     if (src->next) {
34         id_list->next = identity_list_dup(src->next);
35         if (id_list->next == NULL) {
36             free_identity_list(id_list);
37             return NULL;
38         }
39     }
40
41     return id_list;
42 }
```

Updates in version 0.2a of this document:

- `_ident` is still not freed in case of an error in `new_identity_list()`.
- Issue reopened in Trac

Updates in version 0.2c of this document:

- Issue reviewed and closed

3.3. Memleak in myself(): `_ident` is never freed

- Status: fixed
- Severity: medium
- Tracker ID: #30
- Added with document version: 0.1

Listing 9.16: keymanagement.c

```
pEp_identity *stored_identity;
...
status = get_identity(session,
                      identity->address,
                      identity->user_id,
                      &stored_identity);
...
```

Updates in version 0.2 of this document:

- The variable name changed from `_identity` to `stored_identity`.
- Added code sample.

Updates in version 0.2a of this document:

- Issue reviewed and closed

3.4. Possible memleaks in `myself()`: `keylist` after return

- Status: fixed
- Severity: low
- Tracker ID: #31
- Added with document version: 0.1

The variable `keylist` may not be freed in case of return.

Listing 9.17: keymanagement.c

```
233 status = find_keys(session, identity->address, &keylist);
```

Updates in version 0.2 of this document:

- Closed issue after major rewrite.

3.5. Memleak `*pair` in `message_api.c` in error case

- Status: fixed
- Severity: medium
- Tracker ID: #35
- Added with document version: 0.1

The variable `pair` is not freed in the error case `field == NULL` (line 65).

Listing 9.18: message_api.c

```
61 stringpair_t *pair = new_stringpair(name, value);
62 if (pair == NULL)
63     return;
64
65 stringpair_list_t *field = stringpair_list_add(msg->opt_fields, pair);
66 if (field == NULL)
67     return;
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

3.6. Possible memleak: message_api.c: encrypt_PGP_MIME()

- Status: fixed
- Severity: low
- Tracker ID: #41
- Added with document version: 0.1

The variable `ctext` is never freed.

Listing 9.19: message_api.c: encrypt_PGP_MIME()

```
char *ctext;
...
status = encrypt_and_sign(session, keys, mimetext, strlen(mimetext),
    &ctext, &csize);
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

3.7. Possible memleak: message_api.c: encrypt_PGP_in_pieces ctext not freed before reuse

- Status: fixed
- Severity: low
- Tracker ID: #42
- Added with document version: 0.1

The variable `ctext` is not freed before reuse and not before the function returns.
`ctext` is used several times in this function like so:

Listing 9.20: message_api.c: encrypt_PGP_in_pieces()

```
status = encrypt_and_sign(session, keys, ptext, strlen(ptext), &ctext,
    &csize);
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

3.8. Memleak ptext not freed before reuse in message_api.c: _decrypt_message()

- Status: fixed
- Severity: low
- Tracker ID: #44
- Added with document version: 0.1

The variable `ptext` is allocated and later reused without `free()` and still not being freed.

Listing 9.21: message_api.c: _decrypt_message()

```
status = cryptotech[crypto].decrypt_and_verify(session, ctext,
                                              csize, &ptext, &psize, &_keylist);
...
status = decrypt_and_verify(session, attctext, attcsize,
                           &ptext, &psize, &_keylist);
```

Updates in version 0.2 of this document:

- Added better description.

Updates in version 0.2a of this document:

- Issue reviewed and closed

4. Other memory management issues

4.1. Double free(ptext) in message_api.c:encrypt_PGP_MIME in case of enomem: or pep_error

- Status: fixed
- Severity: high
- Tracker ID: #40
- Added with document version: 0.1

Updates in version 0.2 of this document:

- Closed issue after review.

4.2. Double free - Incorrect memory allocation in stringpair_list_append()

- Status: fixed
- Severity: high
- Tracker ID: #95
- Added with document version: 0.2

The inline documentation for `stringpair_list_append()` (in `stringpair.h`) states: *“all values are being copied before being added to the list. the original values are still being owned by the caller”*

Contrary to the documentation only `stringpair_list_t` elements are allocated for each element of the list to be appended (second list). After appending stringpair lists, the corresponding `stringpair_t` values are shared between the original second list and the result list.

The following code sample will always lead to double free:

```
// given stringpair_list_t *s1 and *s2
stringpair_list_append(s1, s2);
free_stringpair_list(s1);
free_stringpair_list(s2); // double free during free_stringpair();
```

Updates in version 0.2a of this document:

- Issue reviewed and closed

5. Recursion in C

Recursion in C results in the stack to fill up and may eventually lead to memory problems which in turn may lead to unexpected behaviour and errors. So, for a guaranteed or checked small number of calls, recursion is perfectly fine. If the number of recursive calls is unknown or is derived from user input, iterative programming should always be preferred.

A modern C compiler, e.g. GCC, is able to apply code optimization for tail recursion as in `foo() { ...; return foo(); }`; when using the compiler flag `-O2` or `-O3`.

Unless otherwise stated, the following issues all refer to recursive functions without tail-recursion.

5.1. Unnecessary recursion: `stringlist_dup()`

- Status: fixed
- Severity: low
- Tracker ID: #11
- Added with document version: 0.1

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.2. Unnecessary recursion in `bloblist.c`: `free_bloblist()`

- Status: fixed
- Severity: low
- Tracker ID: #16
- Added with document version: 0.1

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.3. Unnecessary recursion: `bloblist_dup()`

- Status: fixed
- Severity: low
- Tracker ID: #18
- Added with document version: 0.1

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.4. Tail-recursion: bloblist_add()

- Status: fixed
- Severity: low
- Tracker ID: #19
- Added with document version: 0.1

This is a tail-recursion that may be ok with -02 or more.

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.5. Unnecessary recursion: identity_list_dup()

- Status: fixed
- Severity: low
- Tracker ID: #23
- Added with document version: 0.1

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.6. Unnecessary recursion: free_identity_list()

- Status: fixed
- Severity: low
- Tracker ID: #25
- Added with document version: 0.1

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.7. Unnecessary recursion: identity_list_add()

- Status: fixed
- Severity: low
- Tracker ID: #27
- Added with document version: 0.1

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.8. Unnecessary recursion: stringpair_list_dup()

- Status: fixed
- Severity: low
- Tracker ID: #96
- Added with document version: 0.2

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.9. Unnecessary recursion: stringpair_list_add()

- Status: fixed
- Severity: low
- Tracker ID: #97
- Added with document version: 0.2

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.10. Unnecessary recursion: stringlist_add()

- Status: fixed
- Severity: low
- Tracker ID: #98
- Added with document version: 0.2

Updates in version 0.2a of this document:

- Issue reviewed and closed

5.11. Unnecessary recursion: free_stringlist()

- Status: fixed
- Severity: low
- Tracker ID: #99
- Added with document version: 0.2

Updates in version 0.2a of this document:

- Issue reviewed and closed

6. Other issues

6.1. Sqlite: old version 3.8.6 instead of 3.8.11.1

- Status: fixed
- Severity: medium
- Tracker ID: #13
- Added with document version: 0.1

6.2. Missing NULL checks in free_bloblist

- Status: closed
- Severity: low
- Tracker ID: #17
- Added with document version: 0.1

Updates in version 0.2 of this document:

- Set status to *closed*, because `free()` allows NULL values.
- This issue may be removed with the next document version.

6.3. Sqlite: embedded instead of just linked statically

- Status: closed
- Severity: low
- Tracker ID: #12
- Added with document version: 0.1

Embedding external libraries is perfectly fine. However these libraries may get security related updates, so the question remains how these updates are applied to the embedded code.

- Is there an update strategy?
- Would it be possible to check for new versions at compile time?
- Would it be possible to check the embedded library's version against a version installed on the system?
- Would it be possible to link statically or dynamically against the library installed by the system package manager?

Updates in version 0.2a of this document:

- This issue is set to *closed*, because there is an update strategy. It may remain open in Trac in order to track further development.

Questions have been answered, in particular the first question about the update strategy:

- Is there an update strategy? *Yes, we'll patch and send out a pEp security update when security-related updates occur.*
- Would it be possible to check for new versions at compile time? *Difficult, because we have deal with multiple platforms, all of which react differently. We will have to keep abreast of the security issues explicitly.*
- Would it be possible to check the embedded library's version against a version installed on the system? *Maybe, but this would be different for every system and would be a lot of system-specific code. So this is not a preferred option.*
- Would it be possible to link statically or dynamically against the library installed by the system package manager? *Yes, this is how it is intended.*

6.4. Improvement: use `strl*()` instead of `str*()` style functions

- Status: fixed
- Severity: medium
- Tracker ID: #36
- Added with document version: 0.1

The pEp engine makes use of `strcpy()` and `strcat()`. Using their restricting equivalents `strncpy()` and `strncat()` respectively protects the output buffer from accidental buffer overflows.

Examples:

Listing 9.22: `etpan_mime.c`

```
757 strcpy(_type, _main_type);
758 strcat(_type, "/");
759 strcat(_type, content->ct_subtype);
```

Listing 9.23: `message_api.c`

```
89 strcpy(pTEXT, "Subject: ");
90 strcat(pTEXT, shortmsg);
91 strcat(pTEXT, "\n\n");
92 strcat(pTEXT, longmsg);
```

Listing 9.24: `message_api.c`

```
519 strcpy(filename, _s->filename);
520 strcpy(filename + len, ".pgp");
```

Listing 9.25: `pEpEngine.c`

```
1280 strcpy(result + len1 + 1, str2);
```

Updates in version 0.2 of this document:

- generalised issue from strcat to all str*() style functions
- added examples

Updates in version 0.2a of this document:

- Updated recommendation to use strl*() instead of strn*() style functions.
- Reopened issue in Trac

Updates in version 0.2c of this document:

- Issue reviewed and closed

6.5. Possibly unintended sideeffect: seperate_short_and_long()

- Status: closed
- Severity: comment
- Tracker ID: #37
- Added with document version: 0.1

Several cases come to mind that may not be handled as intended:

- Multiline shortmsg cannot be parsed at all.
- Any number of \n or \r at the beginning of longmsg will be removed.
- shortmsg and longmsg separated by only one \n are still handled as if there were two \n.

Updates in version 0.2 of this document:

- Added better description.

Updates in version 0.2a of this document:

- Issue comments reviewed and issue closed
- The project commented as follows: *This is a user-interface issue for a static internal function which is not exposed to the outside. Hiding the subject is defined for email only. There is no “multiline subject”. Longmsgs cannot start with \n\r (because in email this is the line ending, that means they can start with an empty line, but that will do no harm), so the only point will be having a mail body starting with Subject. It's intended that this will replace the subject if the subject is set to pEp.*

6.6. Unchecked return value: update_identity() in message_api.c

- Status: fixed
- Severity: low
- Tracker ID: #45
- Added with document version: 0.1

Several invocations of `update_identity()` in `message_api.c` do not check the return value:

Listing 9.26: outgoing_message_color()

```
update_identity(session, il->ident);
```

Listing 9.27: _update_identity_for_incoming_message()

```
update_identity(session, src->from);
```

Updates in version 0.2 of this document:

- Updated description.

Updates in version 0.2a of this document:

- Issue reviewed and closed

6.7. Missing out-of-memory check after malloc()

- Status: fixed
- Severity: medium
- Tracker ID: #10
- Added with document version: 0.1

Listing 9.28: message_api.c

```
1144 char *copy = malloc(_s->size);  
1145 memcpy(copy, _s->value, _s->size);
```

Updates in version 0.2 of this document:

- Closed issue after bugfix.

6.8. 31 bit random number generator with static seed (Android)

- Status: fixed
- Severity: low
- Tracker ID: #100
- Added with document version: 0.2

The 31 bit random number generator is using a static seed on Android.

This issue is rated low because this function is only ever used for mime boundary generation. In order to avoid name collisions with real random functions or future problems with random numbers, this function should be renamed, e.g. to `really_unsafe_random()`.

Listing 9.29: platform_unix.c

```
53 long int random(void)
54 {
55     static unsigned short xsubi[3] = { 'p', 'E', 'p' };
56     return nrand48(xsubi);
57 }
```

Updates in version 0.2a of this document:

- The random seed is derived from `time()`, now, which is ok for applications other than cryptography.
- Issue reviewed and closed

7. Comments

7.1. Improvement: keymanagement.c: update_identity(): strndup instead of strdup

- Status: fixed
- Severity: comment
- Tracker ID: #28
- Added with document version: 0.1

strndup() should be preferred over strdup, especially here because the string size is known anyway.

Updates in version 0.2 of this document:

- Closed issue after bugfix.

7.2. bloblist_add() does not allow empty blobs

- Status: fixed
- Severity: comment
- Tracker ID: #20
- Added with document version: 0.1

bloblist_add() does not allow empty blobs as in `value = NULL`. This behaviour is not documented.

Updates in version 0.2a of this document:

- Issue reviewed and closed

7.3. Possible caveat: is_fileending() returns false if strings are equal

- Status: closed
- Severity: comment
- Tracker ID: #34
- Added with document version: 0.1

This may not be the desired behaviour.

Listing 9.30: message_api.c

```
36 static bool is_fileending(const bloblist_t *bl, const char *fe)
37 {
38     assert(fe);
39
40     if (bl == NULL || bl->filename == NULL)
41         return false;
42 }
```

```

43     assert(bl && bl->filename);
44
45     size_t fe_len = strlen(fe);
46     size_t fn_len = strlen(bl->filename);
47
48     if (fn_len <= fe_len)
49         return false;
50
51     assert(fn_len > fe_len);
52
53     return strcmp(bl->filename + (fn_len - fe_len), fe) == 0;
54 }

```

Updates in version 0.2a of this document:

- Apparently this behaviour is intended.
- Issue closed

7.4. decrypt_message() auto-imports attached keys

- Status: fixed
- Severity: comment
- Tracker ID: #43
- Added with document version: 0.1

Automatically importing attached keys may be a desired feature. However, depending on how keys are selected for encryption, an attacker may preemptively generate a number of keys for the victim's peer group which may then be selected for encryption.

Also, importing many keys may have a performance impact and enable denial-of-service attacks.

Updates in version 0.2a of this document:

- The project commented, that importing keys as implemented is a "desired behaviour". Key trust is implemented separately.
- A limit of 20 key imports per message was implemented against DoS.
- Issue reviewed and closed

7.5. Typo in function name separate_short_and_long()

- Status: fixed
- Severity: comment
- Tracker ID: #101
- Added with document version: 0.2

It should probably be renamed to `separate_short_and_long()`.

Updates in version 0.2a of this document:

- Issue reviewed and closed

7.6. Potentially incorrect logic in fread() wrapper

- Status: closed
- Severity: comment
- Tracker ID: #102
- Added with document version: 0.2

The read operation may not complete if `fread()` returns before EOF for other errors than `EINTR`.

Listing 9.31: wrappers.h line 119

```
do {  
    ... fread(...);  
} while (!feof(stream) && nitems && ferror(stream) == EINTR);
```

Updates in version 0.2a of this document:

- Explanation given by pEp: *If another error than `EINTR` occurs, then `(ferror(stream) == EINTR)` will be false; therefore the function will return an error state and `errno` in the same way that `fread()` does. . . . The wrapper is here for completing read on `EINTR` only.*
- Issue closed.

7.7. Stringpair struct naming prone to errors

- Status: closed
- Severity: comment
- Tracker ID: #103
- Added with document version: 0.2

As defined in `stringpair.h` struct `_stringpair_t` has a member named `value` as in the value of a key/value pair. The struct `_stringpair_list_t` has a member named `value`, too, which points to a `stringpair_t`. As these structs are to be commonly used together, simple typos may easily cause errors. Although the compiler will warn about pointer incompatibilities, it is recommended to rename the value pointing to the stringpair to something like `pStringpair`.

Updates in version 0.2a of this document:

- Comment by pEp: *stringpair_list_t is already used by a lot of the adapter code, and we feel the context makes node-value vs. key-value-pair clear enough that adjusting all of the adapter code accordingly may cause more problems than it solves in this case.*
- Issue closed.

7.8. Confusing behaviour: `stringpair_list_add(. . . , NULL)`

- Status: fixed
- Severity: comment
- Tracker ID: #104
- Added with document version: 0.2

Given a `stringpair` list with one `NULL` element as `stringpair` (value) with at least one element following, the function `stringpair_list_add` would return the `stringpair_list_t` of the first `NULL` element rather than the last element of the chained list.

The behaviour should reflect the documentation. However this issue is particularly unlikely to occur, because a `stringpair` list as described cannot be constructed with the defined `stringpair` list API.

Note: A similar confusion can be triggered with `stringpair_list_append()` when appending a list with a `NULL` element as first element, and with `stringlist_append()`.

Updates in version 0.2a of this document:

- Issue reviewed and closed

Notes

p≡p Switzerland/Luxembourg - as written on the front page - is comprised of the following legal entities:

- pEp Foundation
Oberer Graben 4
8400 Winterthur
Schweiz
- pEp Security AG
Aeschstrasse 4
8834 Schindellegi
Schweiz
- pEp Security SA
24 rue Léon Kauffman
1853 Luxembourg
Luxembourg