PA193 - Secure coding principles and practices

Protecting integrity of modules and external components - LABS

Petr Švenda svenda@fi.muni.cz

CROCS

Centre for Research on Cryptography and Security

Secure temporary files handling - lib

- Write library for secure usage of temporary files
  - just *.h, *.cpp (no dll is required)
- Design suitable API
  - open, write/read, close, information (set/get tmp directory)
- It should be as easy as possible to refactor existing code based on tmpfile / tmpfile_s functions
  - minimal changes to existing source code
- Library implementation should follow security checklist for temporary files, but hide complexity from user
  - e.g., file name generation, directory and ACL setting, encryption...

Original code for refactor

```c
int main() {
  FILE * pTmpFile;
  // Open temporary files
  tmpfile_s(&pTmpFile);

  char buffer[100] = "Test";
  for (size_t i = 0; i < 50; i++) {
       fwrite("Test", strlen("Test"), sizeof(char), pTmpFile);
  }

  rewind(pTmpFile);
  fseek(pTmpFile, 30, SEEK_SET);

  memset(buffer, 0, sizeof(buffer));
  fread(buffer, sizeof(char), sizeof(buffer) - 1, pTmpFile);
  printf("%s", buffer);

  fclose(pTmpFile);

  // Remove still opened tmp files (only these opened by tmpfile / tmpfile_s)
  _rmtmp();

  return 0;
}
```

TODO: hidden manipulation

- obtain temporary directory
- create subdirectory, set proper ACL
- generate long random file name
- open file exclusively with absolute path
- generate random key
- encrypt data before write, decrypt on read
- shred content (overwrite) before close, erase key
- close file after use
- handle abnormal termination (signal, exception)

TODO: hidden manipulation – this lab

- obtain temporary directory
- create subdirectory, set proper ACL
- generate long random file name
- open file exclusively with absolute path
- generate random key
- encrypt data before write, decrypt on read
- shred content (overwrite) before close, erase key
- close file after use
- handle abnormal termination (signal, exception)

How to start

- Create new project, change Charatcter set to ANSII
  - Project properties->General->Character set->No set
- Write function for file open
  - use same function interface like tmpfile or tmpfile_s
  - fix directory to “C:\\Temp\\” for start (but change later)
  - use rand() for generating random name for file (but change later to robust random number generator)
  - fopen with “wb” mode for start (but change later to include exclusivity mode or use CreateFile(CREATE_NEW...)
  - store file handle in global array of opened handles
  - return opened file handle

Notes

- Functions from standard C library are using FILE* handle for file manipulation
  - when you start refactoring, you may end up having some functions from C library being changed to your improved functions, but not all (e.g., not all fread() will be replaced)
- By changing FILE* to your own structure TMPFILE*, you will force user to switch all functions he is using to your functions – otherwise code will not compile (fread() cannot work with TMPFILE*)
  - TMPFILE may contain original FILE* inside

Assignment 6 – Password hashing

- Write 2 functions in C
  - To hash a password and store it in a hashed form (into file)
  - To verify supplied password against the stored password
- Notes
  - First read: http://www.codeproject.com/Articles/704865/Salted-Password-Hashing-Doing-it-Right
  - Use password salting
  - To generate the random salt use your code from the previous seminars (on random data)
  - Write both UNIX and Windows variants
  - In Unix use the crypt() function [use e.g. sha256]
  - In Windows implement yourself PBKDF2; use MS crypto API for hash functions
- Deadline: 30.11.2014 23:59

Project reports – Dec 11, 14:00 in A319

- Join both the seminar groups!
- Presentations: 5 minutes (sharp)
- No PPT/PDF. Bring notes written on paper.
- Deadline Dec 11 by 11:11 to put files into the IS, bring 2 paper copies for teachers
  - 1-2 pages A4 for report from part 1
    - What project you reviewed, what does the SW do, which tools you used, what are your results, did you provide feedback to the developers?
  - 1 page A4 from part 2
    - What you implemented, including details (restrictions); your result (how many lines of code), what was difficult, …
  - 1 page A4 from part 3
    - What tests did you perform (automated tests, manual review), what did you focus on, what did you find out.