Everything you wanted to know, but were too afraid to ask about...

Mbed TLS and TF-PSA-Crypto unit tests

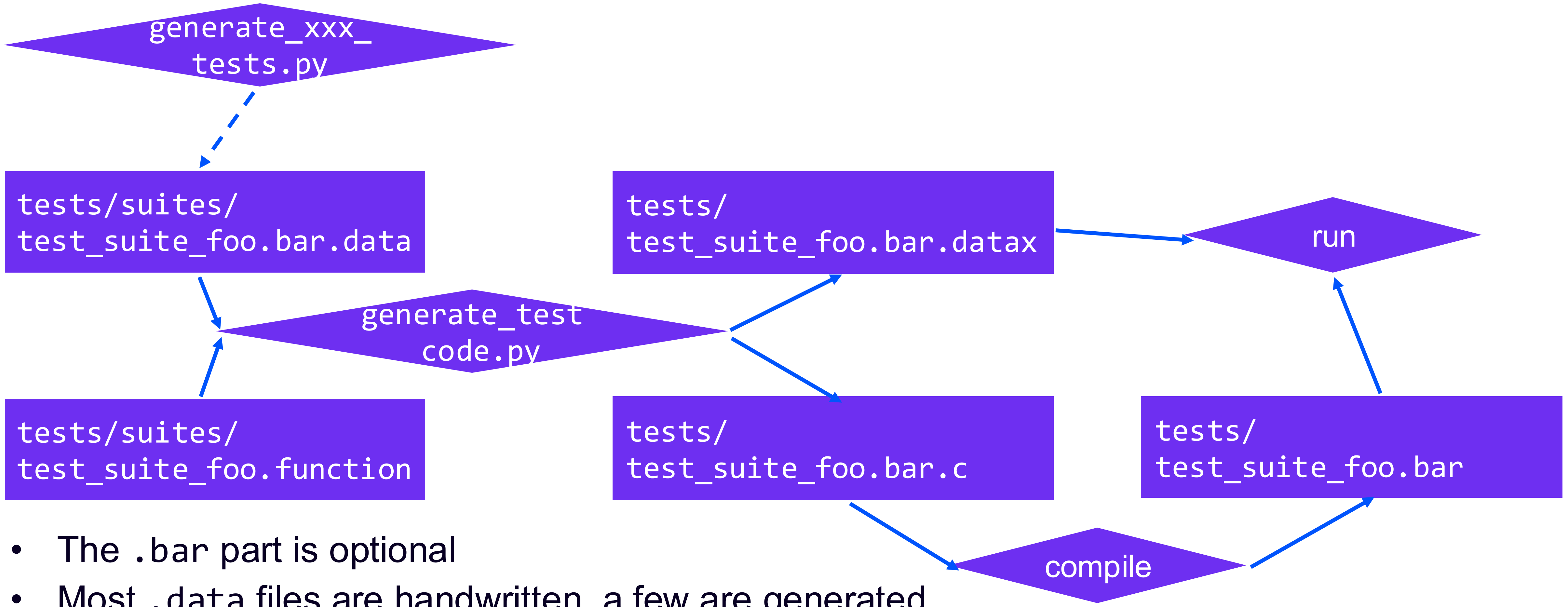
Gilles Peskine and others

2025-07-04

Writing unit tests

Compilation

Doc: kb: Mbed TLS tests guidelines



- The `.bar` part is optional
- Most `.data` files are handwritten, a few are generated
- No need to declare new files in build scripts
    - But you need to re-run cmake

Test assertions

- Effects of `TEST_ASSERT(condition)`:
  - Mark test case as failed
  - `goto exit`
    - Not `return` because we often need to clean up
    - `generate_test_code.py` adds `exit:;` if there's no `exit` label
  - Display the failure location when the test exits
- Fancier assertions
  - `TEST_EQUAL(a, b)` asserts `a==b`
    - Displays the values on failure
    - Also TEST_LE_S(a, b) to assert `a<=b` with `a`, `b` signed
    - Also TEST_LE_U(a, b) to assert `a<=b` with `a`, `b` unsigned
  - `TEST_MEMORY_COMPARE(buffer1, length1, buffer2, length2)`
    - asserts the same length and content
  - `TEST_CALLOC(pointer_variable, size)`
    - Allocate memory (free it with `mbedtls_free()`)
    - `pointer_variable` must be null
    - The size is in elements, not bytes (e.g. `int *p = NULL; TEST_CALLOC(p, 3);` → 3 integers)
  - More: see [`tests/include/test/macros.h`]()

Limitations on information displayed on failure

- Failure information is displayed when the test case returns
  - Only a limited space to store information between mark-as-failed and test-returns
  - Static strings: function, file name, message
  - Integers: line, lhs, rhs, step
- Tip: some complex conditions can be restructured for better display

Basic	**Nicer**
`TEST_ASSERT(cond1 && cond2);`	`TEST_ASSERT(cond1);` `TEST_ASSERT(cond2);`
`TEST_ASSERT(ret == 0 ||` `ret == ACCEPTABLE_ERROR);`	`if (ret != 0) {` `TEST_EQUAL(ret, ACCEPTABLE_ERROR);` `}`
`TEST_ASSERT(x < max);`	`TEST_LE_S(x, max - 1);`

Test step

- You only see where the failure happened, no stack trace and no local state
- `mbedtls_test_set_step(n)`
  - Shows “at step n” in the failure message

Example: loop

```
for (i = 0; i < count; i++) {
    mbedtls_test_set_step(i);
    do_stuff(i);
    TEST_ASSERT(check(i));
}
```

Example: auxiliary function called several times

```
mbedtls_test_set_step(1);
aux_may_fail(…);
mbedtls_test_set_step(2);
aux_may_fail(…);
mbedtls_test_set_step(3);
aux_may_fail(…);
```

- Limitation: does not nest
  - The step is a global variable
  - For nested loops, you can use e.g. `mbedtls_test_set_step(outer_i * 100000 + inner_i)`

Running unit tests

Skip boring output

- To run all test cases but omit skip/pass lines:
  - `tests/test_suite_foo |& grep -Ev '(PASS|SKIP|----)'`
  - If you don't like huge test suites, split them!
- To skip slow test suites:
  - With Make, set the environment variable `SKIP_TEST_SUITES`, e.g.

    ```
    SKIP_TEST_SUITES=constant_time_hmac,lmots,lms,gcm,psa_crypto.pbkdf2,ssl_decrypt make test
    ```
  - With CMake, set the CMake parameter `SKIP_TEST_SUITES` (you have to re-run cmake to change it), e.g.

    ```
    cmake -B build-debug -DCMAKE_BUILD_TYPE=Debug
          -DSKIP_TEST_SUITES=constant_time_hmac,lmots,lms,gcm,psa_crypto.pbkdf2,ssl_decrypt
    ```

Run only one test case

- *I'm debugging one particular test case. I set a debugger breakpoint but it's triggered by many previous test cases. How can I skip over them?*
- Hack: delete the previous cases in `*.data`
  - Assumes you haven't modified them!
  - Remember to undelete before committing!
- Hack: comment out the previous cases in `*.data`
  - Remember to uncomment before committing!
- Hack: copy the test case to the top of `*.data`
  - If you change the test case, remember to update both copies!
  - Remember to remove the extra copy before committing!
- Hack: copy to a new file `test_suite_foo.temp.data`
  - If you change the test case, remember to update both copies!
  - make picks it up automatically, but cmake and mbedtls-prepare-build must be re-run
- Hack: copy the test case in `*.datax` to a new file and run `tests/test_suite_foo my.datax`
  - Difficult to change the data

Debugging unit tests

We use bullets on level one

Here is our second level bullet

Here is our next level bullet

Here is our next level bullet

Which function failed first?

- `mbedtls_foo()` *returned -1234, how do I find which of the 50 functions it might have called detected the error?*
- Use a debugger that supports reverse stepping (a.k.a backward debugging a.k.a. time travel a.k.a...)
  - E.g. Visual Studio (Windows only) or gdb (Linux)
    - macOS: warpspeed?
  - Set a breakpoint after `mbedtls_foo()` returns
  - Step back to the first `return MBEDTLS_ERR_EXAMPLE`
- *Gdb? Really? I never got it to work!*

Reverse debugging made simple on Linux

- Initial setup
  1. Install the Mozilla Record and Replay framework (rr) https://rr-project.org/ (`apt install rr`).
  2. If needed, give yourself debugging permission: `sudo sysctl kernel.perf_event_paranoid=1` (the Ubuntu default is 4 which is too paranoid).
  3. To make this survive across reboots: `echo 'kernel.perf_event_paranoid = 1' >>/etc/sysctl.d/zz-local.conf`
- Debug a program
  1. Build with debugging symbols as usual (`-O0 -g3` or `-Og -g3`).
  2. `rr record ./test_suite_ssl` saves a full trace of the execution.
  3. `rr replay` gives you a gdb interface where reverse execution actually works.
     - Use `reverse-xxx` commands
     - `rs` (`reverse-step`) steps into functions
     - `rn` (`reverse-next`) steps over function calls
     - `reverse-finish` goes back to where the current function was called
  4. If you use a frontend, configure it to run `rr replay` instead of `gdb myprogram`.
     - If the frontend uses gdb's machine interface: `rr replay -i=mi …` instead of `gdb -i=mi …`

Meaning of numerical values

- *When `TEST_EQUAL(a, b)` fails, it shows numerical values. What about symbolic names for enum-like types?*
- Guess the type from the assertion and search the source code.
- For Mbed TLS error codes: `programs/util/strerror` (in `mbedtls`)

```
$ mbedtls-strerror –28160
Last error was: -0x6e00 - SSL - The handshake negotiation failed
```

- If you want the macro name:

```
$ git grep --recurse-submodules '#define MBEDTLS_ERR_.*-0x3F00'
tf-psa-crypto/include/mbedtls/pk.h:#define MBEDTLS_ERR_PK_TYPE_MISMATCH       -0x3F00
```

- For PSA constants: `programs/psa/psa_constant_names`

```
$ psa_constant_names alg 100664841
PSA_ALG_ECDSA(PSA_ALG_SHA_256)
```

- Tip: constants change rarely so you can install `psa_constant_names` and `strerror` from any version

Debugging builds with PSA drivers

- Hackish: edit an `all.sh` component to get a build with debug symbols:
  - Add `ASAN_CFLAGS='-O0 -g3'` at the beginning
  - Replace `make test` by `false`
  - Run `all.sh` (without `-k`) and you'll get the drivers and library built with debug symbols
  - You can't do incremental builds. Tip: use ccache for faster rebuilds:
    ```
    CC="ccache ${CC:cc}" ASAN_CC="ccache clang"
    ```
- mbedtls-prepare-build has some code for driver builds but it's still unstable and clumsy
- ???

SSL unit tests

- *If an SSL test doesn't enable debug logs, can I see them anyway when debugging? I'd like something as simple as adding* `debug_level=4` *to* `ssl-opt.sh`*.*
  - As of June 2025:
    - Set up the test for debugging:
      ```
      options.cli_log_fun = mbedtls_test_ssl_log_analyzer;
      options.srv_log_fun = mbedtls_test_ssl_log_analyzer;
      ```
    - In `tests/src/test_helpers/ssl_helpers.c`, change `#if 0` to `#if 1`
    - All debug logs go to stdout
    - Run `tests/test_suite_ssl -v` (without `-v`, stdout is suppressed)
- Simpler (just set a variable and run with `-v`)
  - coming in https://github.com/Mbed-TLS/mbedtls/pull/10273

Unit tests on the CI

Which components have failed?

- If test cases have failed, this is recorded in `failures.csv`
  - Just the `FAIL` lines of `outcomes.csv`
  - Available under “Artifacts”
  - Tip: if the list on BlueOcean only shows `all*.log.gz`, copy one URL and edit it!
  - Note: the file is compressed if it's large, so try `failures.csv.xz`
- The outcome file only records test case failures, not e.g. build failures
  - Improvement pending: Mbed-TLS/mbedtls-framework#129
  - Pipeline Steps + userscript: see CI failure FAQ “PR tests failed. Which components failed?”
  - Anonymous tip “*it can be found by looking for “Failed jobs” in the pipeline log on the artefacts page, but I can never remember that “Failed jobs” is the string to look for - something more obvious would be helpful.*”
  - Tip from Bence: add `/wfapi` to the classic (non-BlueOcean) main page (`/job/mbed-tls-pr-head/job/PR-NNNN-head/1/wfapi`) for a JSON list of failed jobs

Analyzing outcome files

- Outcome file format (documented in [test-framework.md]())
  - `platform;component[-configuration];test_suite;test case description;FAIL;message`
  - E.g. in which configurations did this test case fail and why?
    - `grep ';test_suite_foo;test case description;FAIL;' failures.csv | cut -d';' -f2,6`
- *Which test cases are failing, across all components?*
  - `<failures.csv cut -d\; -f3,4 | sort -u`
  - `xzcat failures.csv.xz |` … if the file is compressed
  - … `| sort | uniq –c | sort -n` to sort by the count of failing components
- *A test case is failing only in some configurations. What's special about them?*
  - First, determine in which components the test case is passing vs failing.

```
xzcat outcomes.csv.xz | grep ';test_suite_foo;Description of the test case;PASS;' | cut -d\; -f2
xzcat outcomes.csv.xz | grep ';test_suite_foo;Description of the test case;PASS;' | cut -d\; -f2
```

  - Next you can try to analyze the configurations.
    - The SKIP/PASS status for `test_suite_config*` tells you about each component's configuration.
    - `framework/scripts/search_outcomes_config.py` queries the outcome file for config properties.

Reproduce an all.sh build with debugging

- General tip: to see the products from an `all.sh` build, add the command `false` before running the tests and run `all.sh` without `-k`. It will stop on the failure without cleaning up.
  - `all.sh` cleans up on Ctrl+C. Use Ctrl+Z to suspend it. Use Ctrl+\ or `kill -9` to kill it without cleanup.
- If it's an ASan build using make (not CMake), add `ASAN_CFLAGS='-Og -g3'` at the beginning of the component, and replace `make test` by `false`
- With CMake, add/change the build type: `cmake -DCMAKE_BUILD_TYPE=Debug` or `cmake -DCMAKE_BUILD_TYPE=ASanDbg`
- ???

Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm