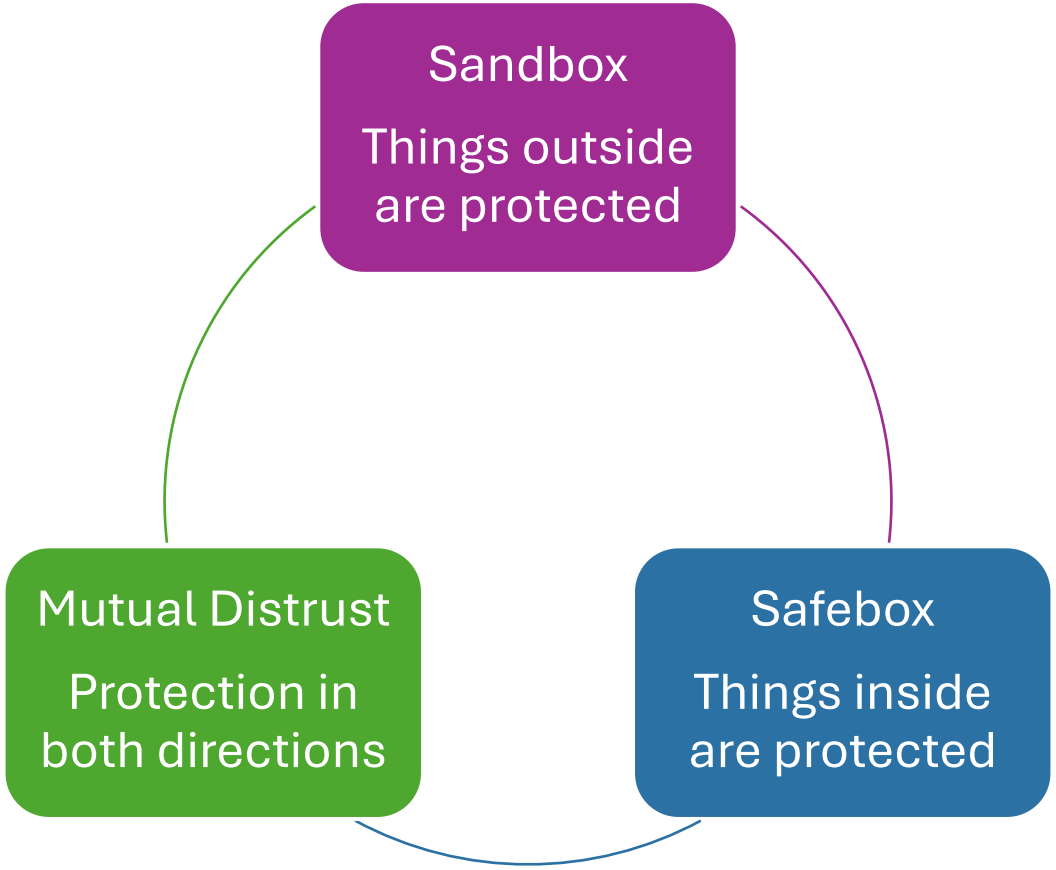
Compartmentalisation Workshop

Ben Laurie, David Chisnall

Reducing blast radius

- What breaks if this crashes?
- What breaks if this is compromised?
- What is leaked in either case?

Trust models for compartments

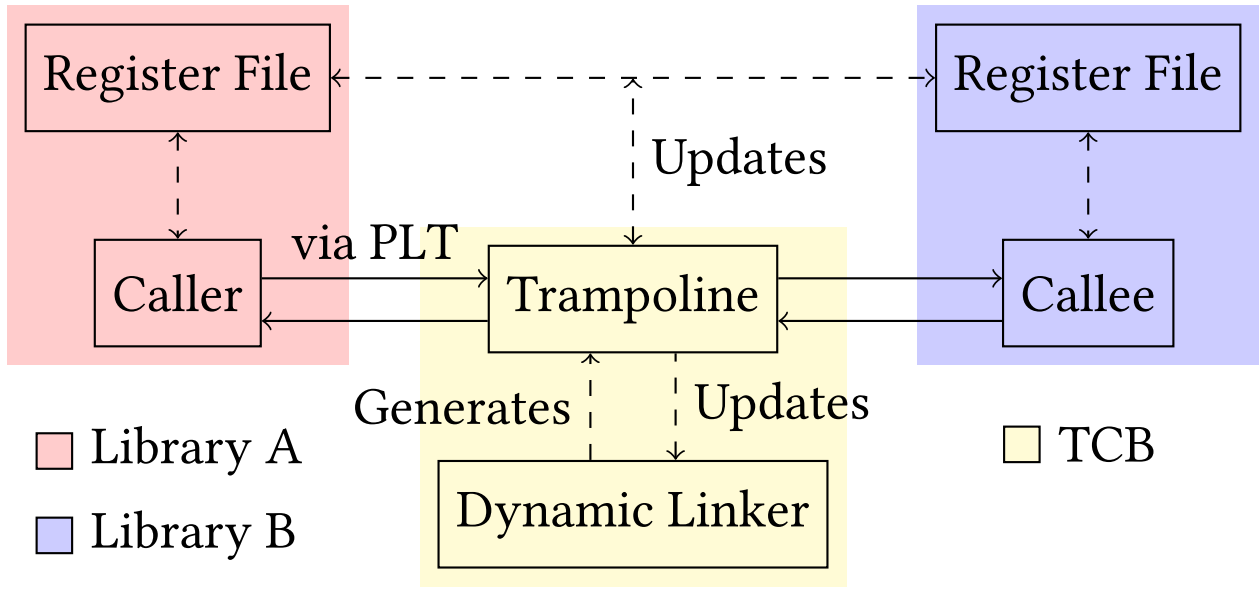


The principle of least privilege

No compartment has rights that it doesn’t need

Library-based compartmentalisation

- Each dynamic library is a compartment
- Dynamic linker sets up *trampolines* between cross-compartment calls
  - Clean up unused registers
  - Switch the execution stack
  - Etc.



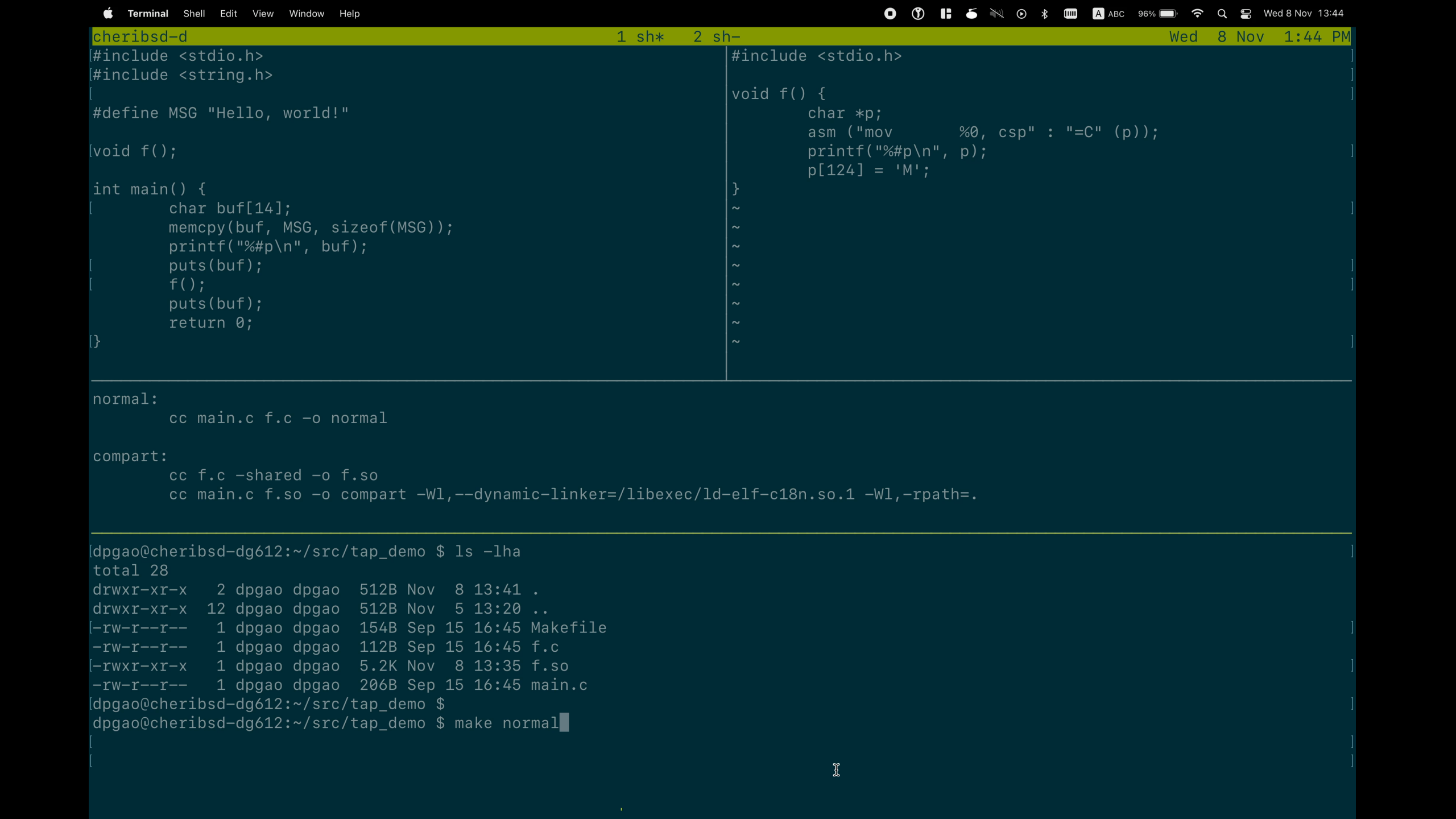
Demo: Protecting against stack corruption

A `pure_computation` function is supplied by some untrusted third-party library.

```
int pure_computation();
```

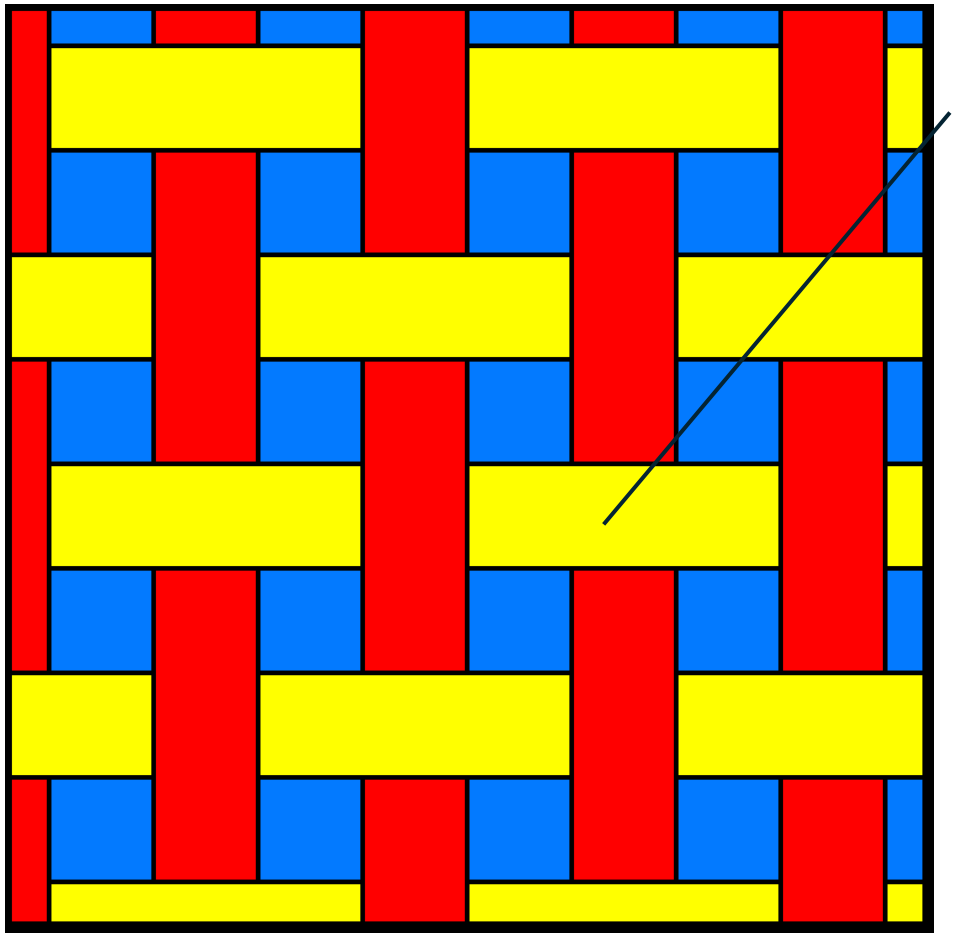
This function is supposed to have no side effects, but how do we guarantee that?

The concrete implementation of C implicitly confers many more powerful capabilities to the untrusted function. For example, it could overwrite the stack frame of its caller…


Terminal Shell Edit View Window Help
Wed 8 Nov 13:44
cheribsd-d 1 sh* 2 sh- Wed 8 Nov 1:44 PM
#include <stdio.h>
#include <string.h>
#define MSG "Hello, world!"
void f();
int main() {
char buf[14];
memcpy(buf, MSG, sizeof(MSG));
printf("%#p\n", buf);
puts(buf);
f();
puts(buf);
return 0;
}
#include <stdio.h>
void f() {
char *p;
asm ("mov %0, csp" : "=C" (p));
printf("%#p\n", p);
p[124] = 'M';
}
normal:
cc main.c f.c -o normal
compart:
cc f.c -shared -o f.so
cc main.c f.so -o compart -Wl,--dynamic-linker=/libexec/ld-elf-c18n.so.1 -Wl,-rpath=.
dpgao@cheribsd-dg612:~/src/tap_demo $ ls -lha
total 28
drwxr-xr-x 2 dpgao dpgao 512B Nov 8 13:41 .
drwxr-xr-x 12 dpgao dpgao 512B Nov 5 13:20 ..
-rw-r--r-- 1 dpgao dpgao 154B Sep 15 16:45 Makefile
-rw-r--r-- 1 dpgao dpgao 112B Sep 15 16:45 f.c
-rwxr-xr-x 1 dpgao dpgao 5.2K Nov 8 13:35 f.so
-rw-r--r-- 1 dpgao dpgao 206B Sep 15 16:45 main.c
dpgao@cheribsd-dg612:~/src/tap_demo $
dpgao@cheribsd-dg612:~/src/tap_demo $ make normal

CHERIoT has two-dimensional isolation

Compartments own code and globals



Running code can access the current compartment's state, on behalf of the current thread

Threads are scheduled independently and call through compartments

Compartmentalisation with CHERIoT

- Compartments are invoked as function calls / returns
- Safe return is guaranteed by a *trusted stack*
- Callees can access explicit arguments
- Callees cannot access any other caller state

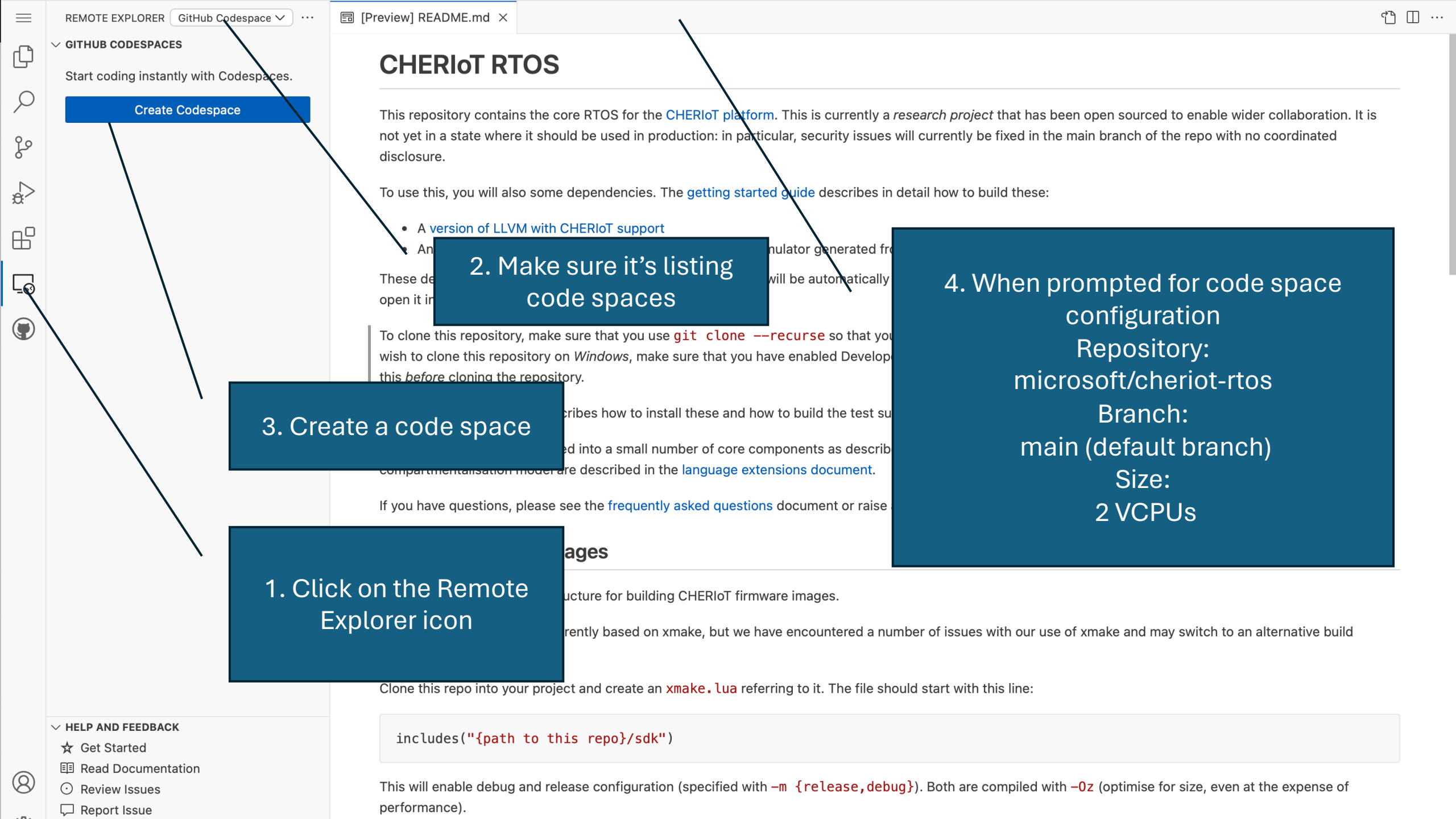
Add compartmentalization to C/C++

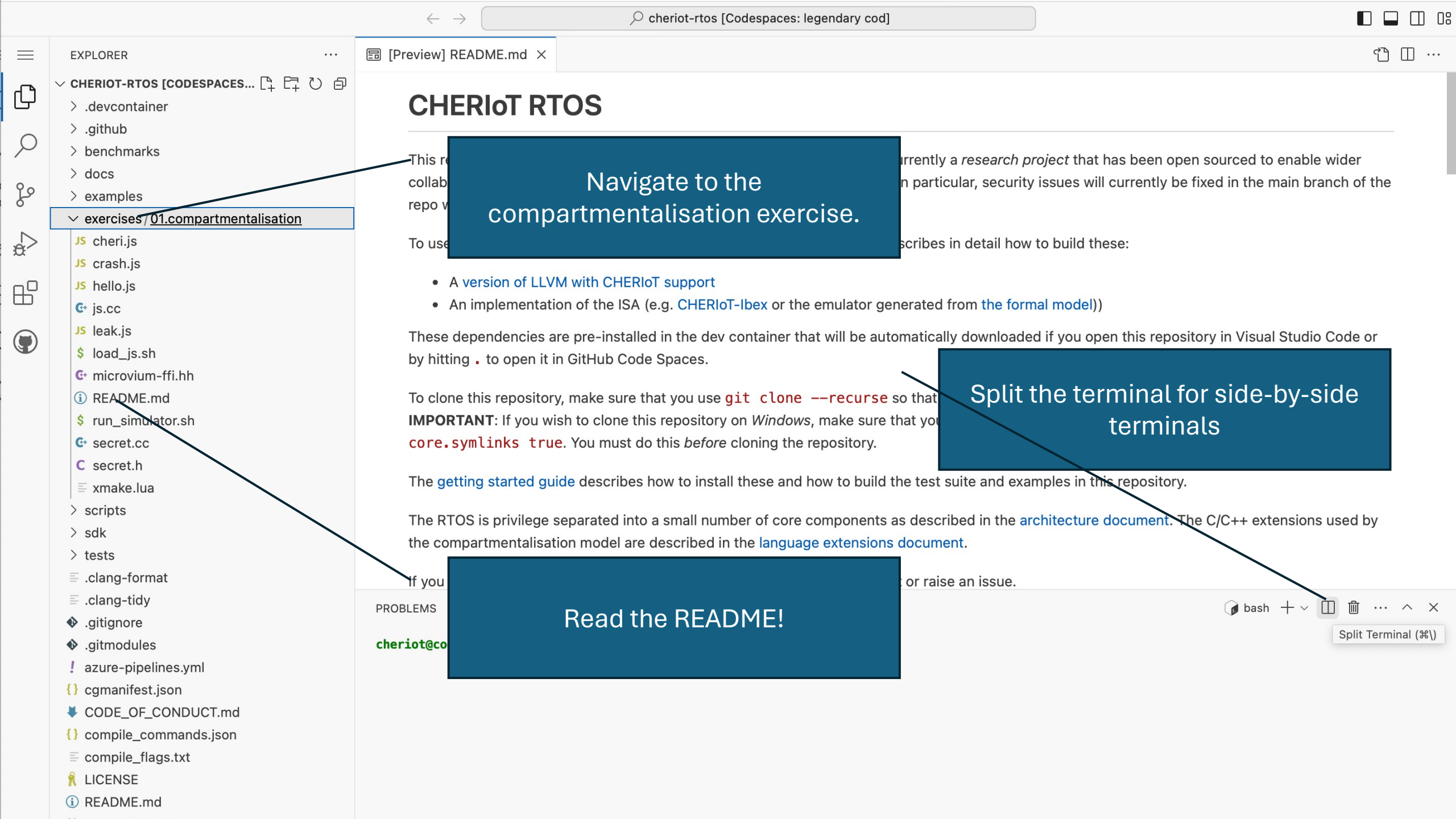
```
// Declaration adds an attribute to indicate
// the compartment containing the implementation
void __cheri_compartment("kv_store_sdk")
publish(char *key, uint8_t *buffer, size_t size);
```

```
-- Make sure it's compiled in the right
-- compartment in xmake.lua
compartment("kv_store_sdk")
      add_files("publish.cc")
      ...
```

Starting the exercise

https://github.dev/microsoft/cheriot-rtos


REMOTE EXPLORER
GitHub Codespace
GITHUB CODESPACES
Start coding instantly with Codespaces.
Create Codespace
HELP AND FEEDBACK
Get Started
Read Documentation
Review Issues
Report Issue
[Preview] README.md
CHERIoT RTOS
This repository contains the core RTOS for the CHERIoT platform. This is currently a research project that has been open sourced to enable wider collaboration. It is not yet in a state where it should be used in production: in particular, security issues will currently be fixed in the main branch of the repo with no coordinated disclosure.
To use this, you will also some dependencies. The getting started guide describes in detail how to build these:
A version of LLVM with CHERIoT support
To clone this repository, make sure that you use git clone --recurse so that you
wish to clone this repository on Windows, make sure that you have enabled Develop
this before cloning the repository.
If you have questions, please see the frequently asked questions document or raise
Clone this repo into your project and create an xmake.lua referring to it. The file should start with this line:
includes("{path to this repo}/sdk")
This will enable debug and release configuration (specified with -m {release,debug}). Both are compiled with -Oz (optimise for size, even at the expense of performance).
1. Click on the Remote Explorer icon
2. Make sure it's listing code spaces
3. Create a code space
4. When prompted for code space configuration
Repository:
microsoft/cheriot-rtos
Branch:
main (default branch)
Size:
2 VCPUs



CHERIoT RTOS

This r… …rrently a *research project* that has been open sourced to enable wider collab… …n particular, security issues will currently be fixed in the main branch of the repo w…

To use… …scribes in detail how to build these:

- A version of LLVM with CHERIoT support
- An implementation of the ISA (e.g. CHERIoT-Ibex or the emulator generated from the formal model))

These dependencies are pre-installed in the dev container that will be automatically downloaded if you open this repository in Visual Studio Code or by hitting `.` to open it in GitHub Code Spaces.

To clone this repository, make sure that you use `git clone --recurse` so that…

IMPORTANT: If you wish to clone this repository on *Windows*, make sure that yo… `core.symlinks true`. You must do this *before* cloning the repository.

The getting started guide describes how to install these and how to build the test suite and examples in this repository.

The RTOS is privilege separated into a small number of core components as described in the architecture document. The C/C++ extensions used by the compartmentalisation model are described in the language extensions document.

If you… …or raise an issue.

cheriot@co…

Run commands in the two terminals

Run the simulator in one terminal

```
$ cd exercises/01.compartmentalisation
$ xmake f --sdk=/cheriot-tools/
...
$ ./run_simulator.sh
...
JavaScript compartment: Secret stored at
0x2004cc8c (v:1 0x2004cc8c-0x2004cc90
l:0x4 o:0x0 p: G RWcgm- -- ---)
JavaScript compartment: Read 0x1ac bytes
of bytecode
JavaScript compartment: 0xdf8 bytes of
heap available
Hello world
```

Compile JavaScript and send it to the UART from the other

```
$ cd exercises/01.compartmentalisation
$ ./load_js.sh hello.js
Loading JavaScript:
...
Output generated: /dev/null
428 bytes
```

The exercise structure

- JavaScript code simulates an attacker with arbitrary code execution.
- Attacks from JavaScript can:
  - Leak a secret
  - Crash the system
  - Exhaust compartment memory

Each exercise will improve compartmentalisation to prevent one attack.

Three exercises

Move the code that owns the secret into a compartment.

Move the JavaScript interpreter into a compartment.

Prevent crashes from leaking memory in the JavaScript compartment.