CHERITech’24 - Welcome

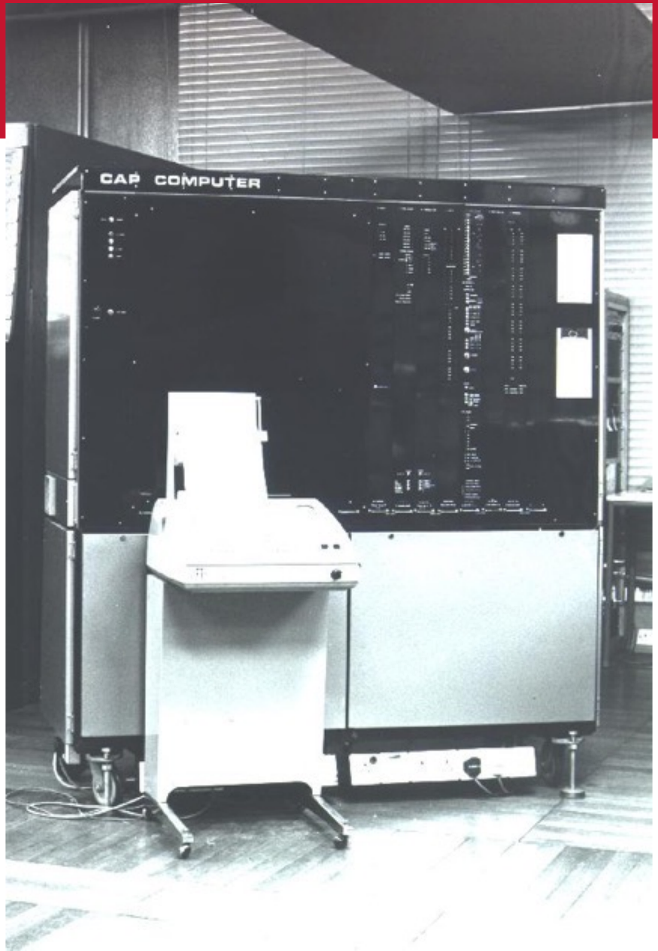
The William Gates Building
University of Cambridge
23 April 2024

Professor Robert N. M. Watson
Professor Simon W. Moore
Franz A. Fuchs

Welcome!

- The CHERI project kicked off in this building in late 2010 with the generous support of DARPA
- Since joined by Arm, Google, Microsoft, and of course funders such as InnovateUK, and others in supporting a massive expansion of scope and interest
- We thank these and other sponsors for their support for this event!

- Capability systems are, of course, an old idea, even if our application of the concept is very contemporary. Do make sure you take a look at the CAP Computer in The Street, completed in 1977!



The CAP computer project ran from 1970-1977 at the University of Cambridge, led by R. Needham, M. Wilkes, and D. Wheeler.

Some administrative things

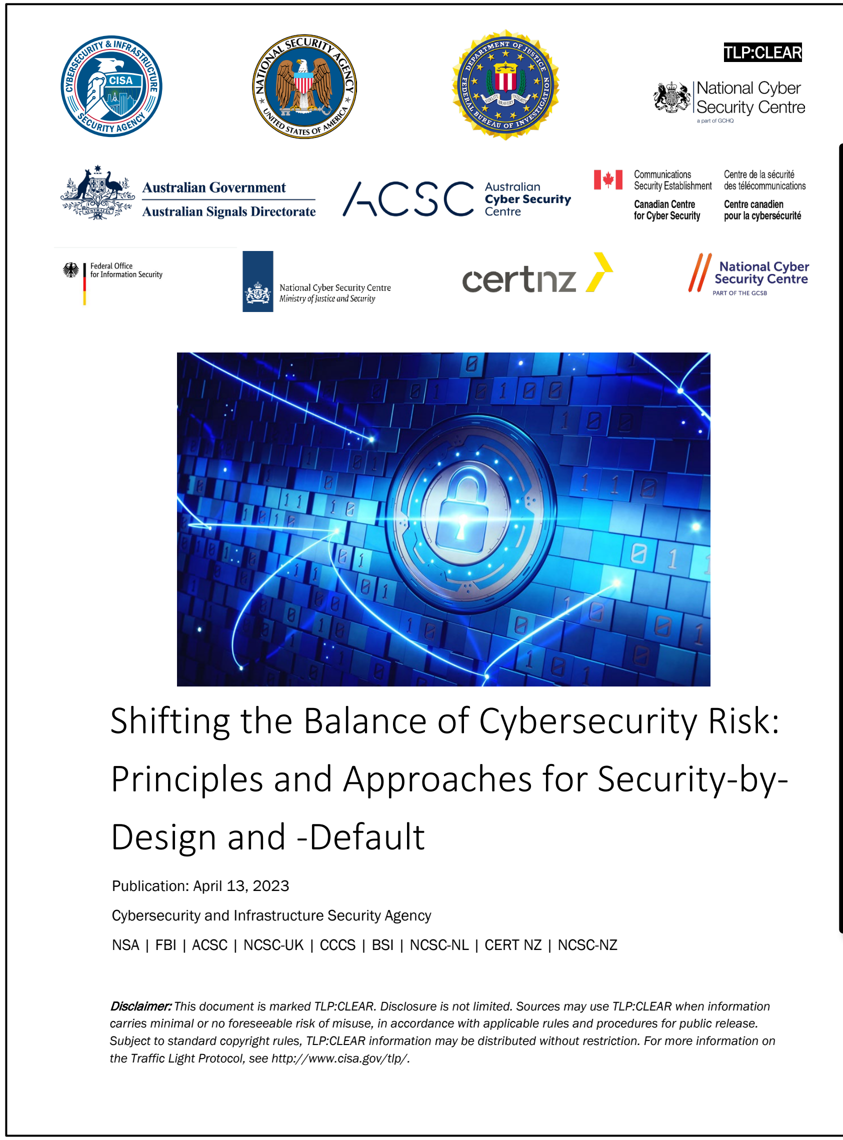
- You are in Lecture Theatre 1 (LT1), which will be where all of our talks are
- Just outside is The Street, where we will have poster sessions, coffee, and lunch
- Bathrooms can be found off The Street, opposite the stairs
- If you need help with a taxi or similar matters, the building's Reception can surely lend a hand
- There are no planned fire drills today … …

Enunciating visions for CHERI adoption

- CHERI is a very challenging technology to transition
  - Hardware and software R&D cycles that differ enormously
  - Supply-chain challenges make deploying any disruptive change hard
  - Evaluating and selling security to generalist markets very difficult
- But the potential wins are also huge
  - Achieve strong memory safety without ground-up rewrite of all software (nearly infinitely expensive, and so will not happen in less than 30 years)
  - Enable compartmentalized software designs capability of resisting arbitrary code execution, software supply-chain adversaries
- To succeed, we need to identify and engage with
  - Clearly enunciated visions for use and deployment
  - Both technical and non-technical obstacles to use and adoption

NCSC, CISA, NSA, FBI, and U.S.-ally cybersecurity agencies recommend CHERI

April 2023



TLP:CLEAR

National Cyber Security Centre

Australian Government
Australian Signals Directorate

ACSC Australian Cyber Security Centre

Communications Security Establishment
Centre de la sécurité des télécommunications
Canadian Centre for Cyber Security
Centre canadien pour la cybersécurité

Federal Office for Information Security

National Cyber Security Centre
Ministry of Justice and Security

certnz

National Cyber Security Centre
PART OF THE GCSB

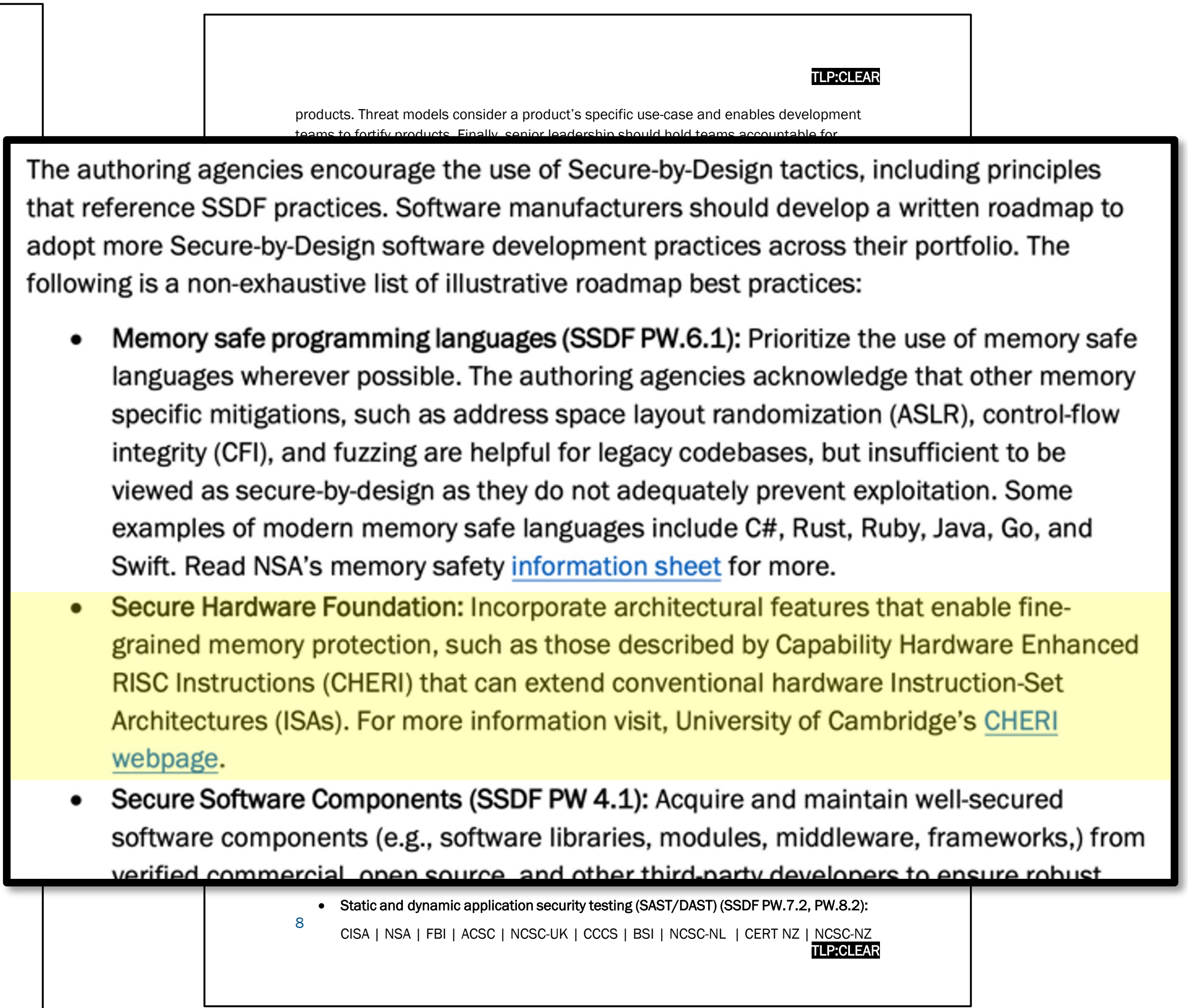
Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Security-by-Design and -Default

Publication: April 13, 2023

Cybersecurity and Infrastructure Security Agency

NSA | FBI | ACSC | NCSC-UK | CCCS | BSI | NCSC-NL | CERT NZ | NCSC-NZ

Disclaimer: *This document is marked TLP:CLEAR. Disclosure is not limited. Sources may use TLP:CLEAR when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release. Subject to standard copyright rules, TLP:CLEAR information may be distributed without restriction. For more information on the Traffic Light Protocol, see http://www.cisa.gov/tlp/.*



TLP:CLEAR

products. Threat models consider a product's specific use-case and enables development teams to fortify products. Finally, senior leadership should hold teams accountable for

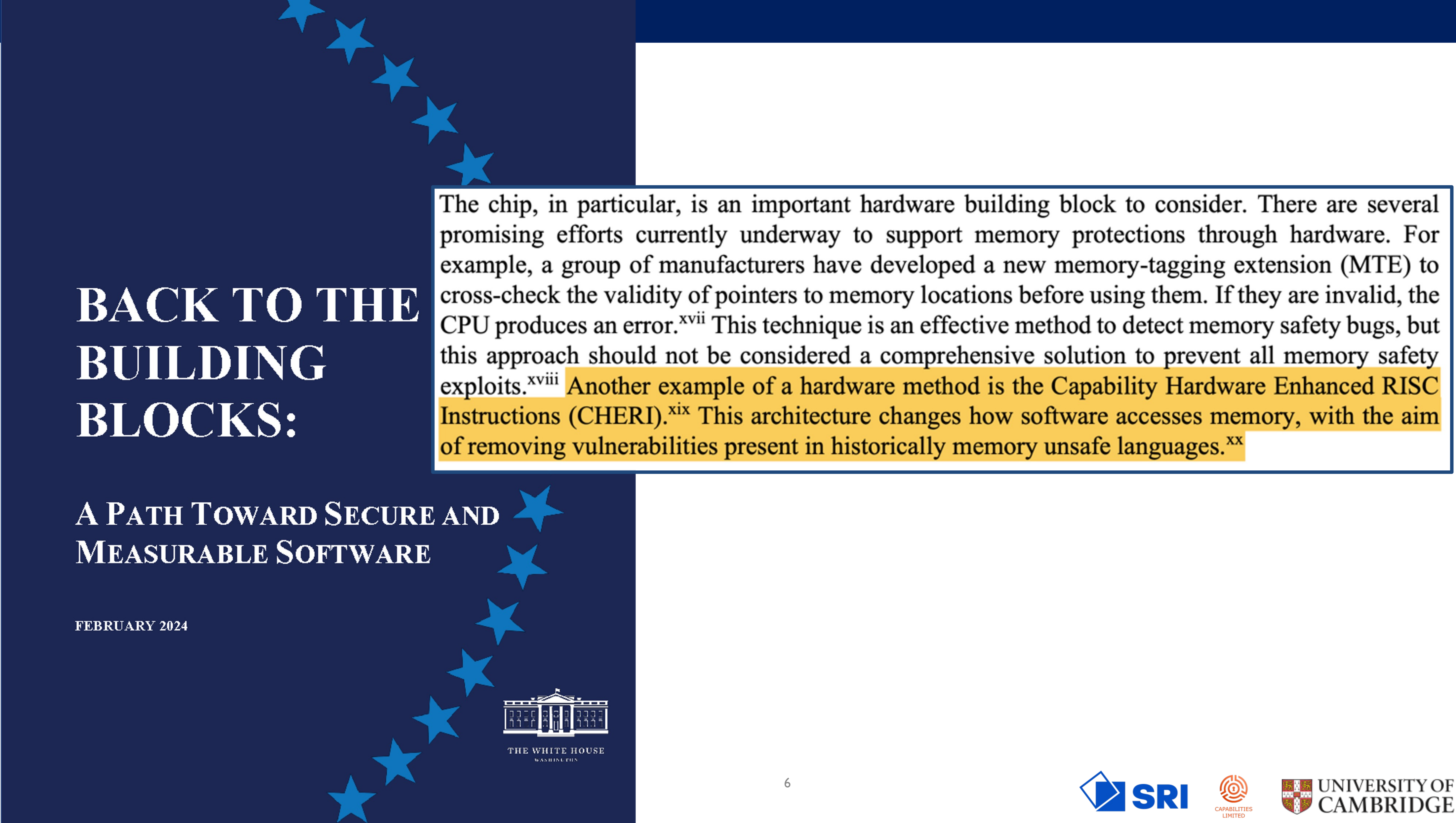
The authoring agencies encourage the use of Secure-by-Design tactics, including principles that reference SSDF practices. Software manufacturers should develop a written roadmap to adopt more Secure-by-Design software development practices across their portfolio. The following is a non-exhaustive list of illustrative roadmap best practices:

- **Memory safe programming languages (SSDF PW.6.1):** Prioritize the use of memory safe languages wherever possible. The authoring agencies acknowledge that other memory specific mitigations, such as address space layout randomization (ASLR), control-flow integrity (CFI), and fuzzing are helpful for legacy codebases, but insufficient to be viewed as secure-by-design as they do not adequately prevent exploitation. Some examples of modern memory safe languages include C#, Rust, Ruby, Java, Go, and Swift. Read NSA's memory safety information sheet for more.
- **Secure Hardware Foundation:** Incorporate architectural features that enable fine-grained memory protection, such as those described by Capability Hardware Enhanced RISC Instructions (CHERI) that can extend conventional hardware Instruction-Set Architectures (ISAs). For more information visit, University of Cambridge's CHERI webpage.
- **Secure Software Components (SSDF PW 4.1):** Acquire and maintain well-secured software components (e.g., software libraries, modules, middleware, frameworks,) from verified commercial, open source, and other third-party developers to ensure robust

- **Static and dynamic application security testing (SAST/DAST) (SSDF PW.7.2, PW.8.2):**

8 CISA | NSA | FBI | ACSC | NCSC-UK | CCCS | BSI | NCSC-NL | CERT NZ | NCSC-NZ

TLP:CLEAR



BACK TO THE BUILDING BLOCKS:

A PATH TOWARD SECURE AND MEASURABLE SOFTWARE

FEBRUARY 2024

THE WHITE HOUSE
WASHINGTON

The chip, in particular, is an important hardware building block to consider. There are several promising efforts currently underway to support memory protections through hardware. For example, a group of manufacturers have developed a new memory-tagging extension (MTE) to cross-check the validity of pointers to memory locations before using them. If they are invalid, the CPU produces an error.[xvii] This technique is an effective method to detect memory safety bugs, but this approach should not be considered a comprehensive solution to prevent all memory safety exploits.[xviii] Another example of a hardware method is the Capability Hardware Enhanced RISC Instructions (CHERI).[xix] This architecture changes how software accesses memory, with the aim of removing vulnerabilities present in historically memory unsafe languages.[xx]

From **drumbeat** to **standardisation**

- An idea gaining increasing currency to help answer the question:
  "How can consumers ask for systems with memory safety?"
- Complex path including technical consensus building ("what is memory safety?") and vast tricky tradeoffs and potential pitfalls
- Want an inclusive definition across methodologies: At least (perhaps multiple) of CHERI, Rust, formal methods, ... but accept:
  - Differing capabilities, adoption tradeoffs, adversary models, limitations
- Reward early adoption (exploit mitigations) while motivating ratcheting up of ambition over time (e.g., from PAC to CHERI)
  - From secrets-based/probabilistic to deterministic protection
  - Mature adversary models including attackers with arbitrary code execution
- Be aware that this discussion is coming ..
  - May go nowhere, but likely essential to produce the supply-chain pull to get beyond widely lauded but actually token Rust deployments with limited impact

CHERI Alliance CIC

- A UK-based Community Interest Corporation (CIC)
- Provide a space enabling companies, universities, and governments to pool resources to promote and enable CHERI
- CIC now created legally, and in framework development
- Planned Autumn launch with initial membership
- Support efforts across CHERI-enabled architectures, such as:
  - Common marketing material
  - Standardisation and certification activities
  - Software ecosystem enablement
  - Efforts such as memory-safety standardization
- Your feedback and participation very much invited!

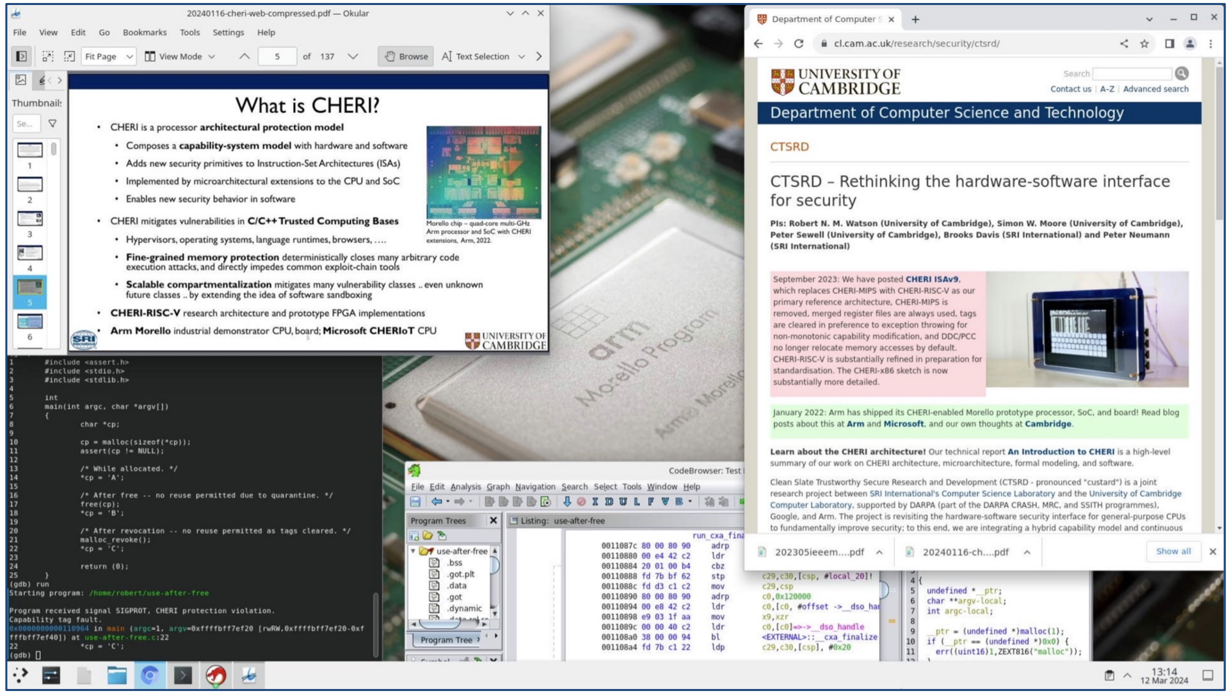
CheriBSD 2024.05 software release coming soon

Reference design for CHERI integration into a mainstream, open-source OS and application stack

Approaching 100MLoC of memory-safe, compartmentalised C/C++ on a shipping prototype Arm Morello board today:

- CheriBSD kernel with DRM + Panfrost drivers
- CheriBSD userspace with libraries and tools
- OpenGL, Wayland display server
- Desktop: Plasma, KDE base applications including Dolphin, Okular, Kate, Konsole, …
- Server: nginx, Postgres, gRPC, …
- "Ubiquitous" Library compartmentalization of all memory-safe userlevel components
- Complete software development environment including Clang/LLVM, Git, GDB, Ghidra, …
- Roughly 10K memory-safe third-party software packages, and 20K aarch64 packages

Some more complex, un-adapted applications (e.g., Chromium, OpenJDK) running with 64-bit Arm support


20240116-cheri-web-compressed.pdf — Okular
File View Edit Go Bookmarks Tools Settings Help
Fit Page
View Mode
5
of 137
Browse
Text Selection
Thumbnails
Se...
1
2
3
4
5
6

What is CHERI?

- CHERI is a processor **architectural protection model**
  - Composes a **capability-system model** with hardware and software
  - Adds new security primitives to Instruction-Set Architectures (ISAs)
  - Implemented by microarchitectural extensions to the CPU and SoC
  - Enables new security behavior in software
- CHERI mitigates vulnerabilities in **C/C++ Trusted Computing Bases**
  - Hypervisors, operating systems, language runtimes, browsers, ….
  - **Fine-grained memory protection** deterministically closes many arbitrary code execution attacks, and directly impedes common exploit-chain tools
  - **Scalable compartmentalization** mitigates many vulnerability classes .. even unknown future classes .. by extending the idea of software sandboxing
- **CHERI-RISC-V** research architecture and prototype FPGA implementations
- **Arm Morello** industrial demonstrator CPU, board; **Microsoft CHERIoT** CPU

Morello chip – quad-core multi-GHz Arm processor and SoC with CHERI extensions, Arm, 2022.

Department of Computer Science and Technology

CTSRD

CTSRD – Rethinking the hardware-software interface for security

PIs: Robert N. M. Watson (University of Cambridge), Simon W. Moore (University of Cambridge), Peter Sewell (University of Cambridge), Brooks Davis (SRI International) and Peter Neumann (SRI International)

September 2023: We have posted **CHERI ISAv9**, which replaces CHERI-MIPS with CHERI-RISC-V as our primary reference architecture, CHERI-MIPS is removed, merged register files are always used, tags are cleared in preference to exception throwing for non-monotonic capability modification, and DDC/PCC no longer relocate memory accesses by default. CHERI-RISC-V is substantially refined in preparation for standardisation. The CHERI-x86 sketch is now substantially more detailed.

January 2022: Arm has shipped its CHERI-enabled Morello prototype processor, SoC, and board! Read blog posts about this at **Arm** and **Microsoft**, and our own thoughts at **Cambridge**.

Learn about the CHERI architecture! Our technical report **An Introduction to CHERI** is a high-level summary of our work on CHERI architecture, microarchitecture, formal modeling, and software.

Clean Slate Trustworthy Secure Research and Development (CTSRD - pronounced "custard") is a joint research project between SRI International's Computer Science Laboratory and the University of Cambridge Computer Laboratory, supported by DARPA (part of the DARPA CRASH, MRC, and SSITH programmes), Google, and Arm. The project is revisiting the hardware-software security interface for general-purpose CPUs to fundamentally improve security; to this end, we are integrating a hybrid capability model and continuous

```
1       #include <assert.h>
2       #include <stdio.h>
3       #include <stdlib.h>
4
5       int
6       main(int argc, char *argv[])
7       {
8               char *cp;
9
10              cp = malloc(sizeof(*cp));
11              assert(cp != NULL);
12
13              /* While allocated. */
14              *cp = 'A';
15
16              /* After free -- no reuse permitted due to quarantine. */
17              free(cp);
18              *cp = 'B';
19
20              /* After revocation -- no reuse permitted as tags cleared. */
21              malloc_revoke();
22              *cp = 'C';
23
24              return (0);
25      }
(gdb) run
Starting program: /home/robert/use-after-free

Program received signal SIGPROT, CHERI protection violation.
Capability tag fault.
0x0000000000110964 in main (argc=1, argv=0xffffbff7ef20 [rwRW,0xffffbff7ef20-0xf
fffbff7ef40]) at use-after-free.c:22
22              *cp = 'C';
(gdb) 
```

```
                          run_cxa_fina
0011087c 80 00 80 90      adrp
00110880 00 e4 42 c2      ldr
00110884 20 01 00 b4      cbz
00110888 fd 7b bf 62      stp       c29,c30,[csp, #local_20]!
0011088c fd d3 c1 c2      mov       c29,csp
00110890 80 00 80 90      adrp      c0,0x120000
00110894 00 e8 42 c2      ldr       c0,[c0, #offset ->__dso_han
00110898 e9 03 1f aa      mov       x9,xzr
0011089c 00 00 40 c2      ldr       c0,[c0]=>->__dso_handle
001108a0 38 00 00 94      bl        <EXTERNAL>::__cxa_finalize
001108a4 fd 7b c1 22      ldp       c29,c30,[csp], #0x20
```

```
{
  undefined *__ptr;
  char **argv-local;
  int argc-local;

  __ptr = (undefined *)malloc(1);
  if (__ptr == (undefined *)0x0) {
    err((uint16)1,ZEXT816("malloc"));
```

Demonstration

CheriBSD 2024.05 development snapshot running on Arm Morello desktop system