REPORT: USING DSBD IN PRACTICE: IS IT PIECE OF CHERI-CAKE?

MAYSARA ALHINDI, JOSEPH HALLETT AND BENJAMIN SHREEVE

MAY 4, 2023

This report documents work to date on the *ESRC Digital Security by Design Social Science Hub+ Connecting Capabilities* grant exploring the usability of current DSbD platforms. The grant commissioned a study looking at the usability of CHERI's APIs, with an aim to identifying what issues developers may face when portinng their software to CHERI's architectures—in particular developers who may not have ever encountered CHERI before. Within we describe:

- The study design for our experiment,
- Observations having run the experiment with 9 participants,
- Suggested improvements to CHERI's programming environment,

We find *preliminary* signs that digital security by design architectures may be falling into the same usability traps existing systems fell into. Namely that the developers struggle to work with code when the systems do not match their mental models and when the documentation isn't tailored to their needs.

Study design

Our study is designed around observing developers as they carry out two programming tasks, each taking an hour:

1. Porting an application to CHERI,
2. Reviewing pull requests containing changes related to CHERI.

Whilst carrying out theses tasks, participants were asked to narrate their thought processes and reasoning. A transcription of the narration is analysed using *grounded theory*[1] specifically looking for moments where participants mental models seemed confused and other usability smells[2]—signs that developers are struggling with a programming task.

[1] Christine Mattley, Anselm Strauss, and Juliet Corbin. Grounded theory in practice. *Contemporary Sociology*, 28(4):489, Jul 1999

[2] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. Usability smells: An analysis of developers' struggle with crypto libraries. In Heather Richter Lipford, editor, *Fifteenth Symposium on Usable Privacy and Security, SOUPS 2019, Santa Clara, CA, USA, August 11-13, 2019*. USENIX Association, 2019

The two tasks were performed in a random order to minimize learning effects. Prior to starting either task participants were asked to describe *what a capability is?* At the end of the study they were asked the same question again to look for a learning effect from working with CHERI for a short period of time. Developers had access to Google and any other documentation they felt they needed throughout the task.

Participant Recruitment

We recruited developers with experience in C programming. The participants had various backgrounds, some of them were students and the others were recruited from industry and academia.

Prior to the study, participants were provided with two technical reports introducing them to CHERI's architecture: *An Introduction to CHERI*[3] and *CHERI C/C++ Programming Guide*[4]. Following that, we asked the participants to take a short quiz and describe what a capability was. To ensure that participants had actually read the documentation only those who knew—even incorrectly—what a capability was were allowed to continue. Four participants were rejected in this manner.

[3] Robert N. M. Watson, Simon W. Moore, Peter Sewell, and Peter G. Neumann. *An Introduction to CHERI*. Number 941. University of Cambridge Computer Laboratory, 2019

[4] Robert N. M. Watson, Alexander Richardson, Brooks Davis, John Baldwin, David Chisnall, Jessica Clarke, Nathaniel Filardo, Simon W. Moore, Edward Napierala, Peter Sewell, and Peter G. Neumann. CHERI C/C++ programming guide. Technical Report 947, University of Cambridge Computer Laboratory, 2020

Porting an application to CHERI

Developers were given a simple C application and were asked to port it to CHERI BSD and harden it using the CHERI capability model. The given program is C program that writes to a file and output a file content to the standard output—a version of the classic UNIX `cat` program similar to the one in CHERI's own examples. In addition to the source code developers were given a compiler script to allow them to build it (and to record what warnings and errors they encountered) and a QEMU-based virtual machine running CHERI BSD.

Compiling the program produces two capability-related warnings; shown in Figure 1. In order for developers to resolve these issues and successfully port the program to run using the CHERI capability model, they have to change few lines of code and ensure that pointers and integers are not treated as synonymous, as they can be in traditional CPU architectures but not under CHERI.

This task aimed to explore how developers reacted in the face of CHERI specific programming errors and warnings; whether they understood what they meant and how they related to the underlying architecture and whether they could recall information from CHERI's to documentation to correctly fix the code.

Reviewing pull requests

In this task, developers had to review 4 merge requests made to port programs to CHERI. Reviewing pull requests is a daily task for many software engineers, and is occasionally a job in its own right: the so-called *merge master*. The job of the merge master is to review and

```
warning: binary expression on capability
types 'uintptr_t' (aka 'unsigned␣__intcap')
and 'uintptr_t'; it is not clear which
should be used as the source of provenance;
currently provenance is inherited from the
left-hand side [-Wcheri-provenance]
    return (offset + (uintptr_t)buffer);
            ~~~~~~ ^ ~~~~~~~~~~~~~~~~~
warning: cast from provenance-free integer
type to pointer type will give pointer that
can not be referenced
[-Wcheri-capability-misuse]
    FILE *fp = (FILE *)file;
               ^
```

Figure 1: Warnings after compling the program to capability mode.

test small changes made by individual developers before their code can be integrated with a project. Sometimes these changes are new features but often they are changes required to address known bugs.

The participants were asked to assume the role of a merge master and to answer the following questions for each pull request:

- Why this change has been made?
- Does this change provide a security benefit, and if so what benefit does it provide?
- Would they accept this merge request or not?

Three out of the four merge requests were taken from code changes made to port programs from FreeBSD to CHERI BSD. The fourth code change includes a function that calculates the pointer size in way that is not compatible with the CHERI architecture. Example of the code changes are shown in merge requsts 2, 3, 4 and 5. Whilst three of the changes (Figures 2, 3, 4) were taken from the porting work of CHERI BSD, Figure 5 was created from an example of a typical programming error with CHERI—assuming that pointer size matches CPU word size: but a CHERI pointer is *twice* the CPU word size to include the additional capability information.

Each of the merge requests covers different aspects of changes needed to port programs to CHERI model; including: pointers as an integer, pointers provenance, C coding styles and pointer size calculation.

The point of this task was to explore how developers understood the changes required to port applications to CHERI. Whilst the porting task focussed on whether the developers could recall and implement the changes; this task focussed on whether they could understand and descriminate *good* code from bad and understand the reasons why a change would have been made. Whilst we were not specifically interested in whether the changes were safe to merge, instead more interested in the developer's perceived rationale for why they were safe to merge: the first three were and the last one wasn't.

```
-    #define   PTR_WIDTH ((int)(sizeof(void *) * 2 + 2))
+    #define   PTR_WIDTH ((int)(sizeof(vaddr_t) * 2 + 2))
```

Figure 2: Part of the `kldstat` port

```
- ipsstp->iss_table = (void *)deadlist[18].n_value;
- ipsstp->iss_list = (void *)deadlist[17].n_value;
+ ipsstp->iss_table = (void *)(uintptr_t)deadlist[18].n_value;
+ ipsstp->iss_list = (void *)(uintptr_t)deadlist[17].n_value;
        ipsstp->iss_tcptab = ipstcptab;
```

Figure 3: Part of the `ipfstat` port

```
-int _arcfour_crypt(buf, len, desp)
-    char *buf;
-    int len;
-    struct desparams *desp;
+int _arcfour_crypt(char *buf, int len, struct desparams *desp)
```

Figure 4: Part of the `crypt` server port

```
+static int get_pointer_size_in_bits(){
+    // size in bytes
+    int size = sizeof(void*);
+    // check if pointer size is 32 bit or 64 bits
+    if(size*8==32){
+        printf("%s","This program supports 32 bit machines");
+        return 32;
+    }else{
+        printf("%s","This program supports 64 bits machines");
+        return 64;
+    }
+}
+
 int main(void) {
-    int pointer_size = 32;
+    int pointer_size = get_pointer_size_in_bits();
```

Figure 5: Change assuming that pointer size accurately describes the word size of the machine.

Results

The participation statistics are shown in Table 1. The study was run with 9 people, and a further 4 people were rejected at the pre-study task[5]

Once the sessions were completed, transcripts and recordings were analysed using an open coding approach to pull out interesting points and realisations that the participants demonstrated. Saturation was reached after 5 sessions, and 4 sessions are still yet to be coded[6]. The codes were then assigned to one of the four *usability whiff*'s identified by Patnaik et al.[7] to highlight the issues developers were struggling with.

In addition to Patnaik's four usability whiffs (Table 2), we have also added a fifth whiff that emerged from our analysis: *floundering*. We believe this fifth whiff was not covered in Patnaik et al.'s original paper and captures when developers are attempting to fix a problem by trying to stumble into a solution without any clear plan, approach or understanding what is actually going on.

[5] Having been given the CHERI documentation to read, they still could not say—even incorrectly—what a capability was.

Total participants	9
Participants rejected at pre-study task	4
Porting task	
Fixed first warning	9/9
Fixed second warning	6/9
Pull request task	
Correctly merged first patch	3/9
Correctly merged second patch	4/9
Correctly merged third patch	6/9
Correctly rejected fourth patch	7/9

Table 1: Participation figures.

[6] But will be imminently and described in subsequent publications.

[7] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. Usability smells: An analysis of developers' struggle with crypto libraries. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 245–257, 2019

Porting task

Whilst all 9 of the developers managed to resolve the first warning, only 6 managed to resolve the second and port the application. When attempting to fix the warnings, developers displayed signs that they were *confused*, trying to relate CHERI concepts to their own mental models and trying to understand the abstractions.

In addition developers also demonstrated a new whiff—*floundering*—by attempting to fix the issues through trial and error or by swapping parts of the code around without understanding why that might fix something. Some attempted to diagnose the issues in the code without actually reading what the code was and trying to understand it. Some tried to solve the warnings by changing all pointers types in the code. Others attempted to search for the error messages in the documentation, and online to see if someone could tell them what the individual errors actually meant. This particular new whiff is worrying because unlike the *confusion* whiff which suggests that developers are struggling to align their own mental models with CHERI's; the *floundering* whiff suggests that, despite being able to state what a capability is they do not understand them, or have any idea of how to work with them in practice. Instead they flounder at the code trying things that *might work* just in case they can make the errors go away.

Sleuthing.	Developer has issues with documentation or its clarity.
Confusion.	Developer is unsure how to apply core programming concepts
Post-Mortem.	Developer is trying to figure out what went wrong.
Doesn't play well.	Build, compatibility or performance issues.

Table 2: Patnaik et al.'s usability whiffs: high-level code smells for usability issues with cryptography libraries.

Task	Code	Usability Whiff
First warning	Because offset is different pointer	
First warning	Can be solved by changing all types to non cap types	
First warning	Casted the whole line to void pointer	
First warning	Casted the whole return line to intptr	
First warning	Change offset from int to int pointer	
First warning	Changed the data type to make buffer source of provenance	Floundering
First warning	Developer swapped sides of variables to solve first warning	Floundering
First warning	First warning might not be an issue	Confusion
First warning	Gave example of how CHERI solves buffer overflow	Confusion
First warning	Looked at the definition of provenance	Sleuthing
First warning	Noticed that there is a default case when getting capability source	Post-mortem
First warning	Solved without looking at docs	
First warning	Thought that int pointers do not carry capability	Confusion
First warning	To solve ambiguous cap case, we have to change *one* of the pointers	Confusion
Second warning	Changed a lot of unnecessary things	Floundering
Second warning	Changed many types and kept running the program to check	Confusion
Second warning	Developer unsure about the need to change all the way through	Confusion
Second warning	Developer didn't run the program and just worked based on warnings	Doesn't play well
Second warning	Developer figured the need to change in other parts to solve the issue	Post-mortem
Second warning	Developer tried to run the program after second warning was gone	Floundering
Second warning	Error is happening because its trying to use an address the pointer won't protect	Confusion
Second warning	Error is happening because long type can not maintain provenance	Confusion
Second warning	Not sure if warning is CHERI related	Sleuthing
Second warning	Second warning is the actual issue	Confusion
Second warning	Traced the execution of the code to find the issue	Post-mortem
Second warning	Tried to use malloc to store the string to solve the issue	Floundering
Second warning	Tried to use signed integer instead of unsigned to fix the issue	Floundering
Second warning	Started with the second warning	
Second warning	Trial and error	Floundering
Second warning	Tried to run the program to check if warning causing errors	Floundering
Second warning	Warnings are clear	
Overall	Changes in the wrong place	
Overall	Changes led to a new warning	
Overall	Changes to wrong data type	
Overall	Compile the code without reading it	Floundering
Overall	Decided to read the code after feeling stuck	Post-mortem
Overall	Developer gave up and moved on	Floundering
Overall	Developer not sure if changes will fix the code	Floundering
Overall	Developer solved warning by copying and pasting without understanding	Floundering
Overall	Did not run the program to check if warnings are affecting runtime	Floundering
Overall	Getting pointer size in CHERI is different in normal C	Confusion
Overall	Made changes based on common sense C knowledge	
Overall	Looked at man pages to understand what syscalls return	Floundering
Overall	Not sure which warning is causing the error	Floundering
Overall	Search the documentation for the error message	Floundering
Overall	Searched online for the error	Floundering

Table 3: Codes used when analysing porting task transcripts

Pull request task

Task	Code	Usability Whiff
PR1	Can not confirm if it is a necessary change	Floundering
PR1	Confused how CHERI handles size of pointers	Confusion
PR1	It has security implications since the bounds might be too large or small	
PR1	It is not clear from docs how to solve this	Sleuthing
PR1	It should be alignof instead of sizeof	
PR1	Made to increase compatibility with CHERI	
PR1	Mixed between vaddr and a pointer	
PR1	Not sure about the sizes of vaddr and pointer	
PR1	Unclear if it has security implications	
PR2	Added security since its CHERI complaint	
PR2	Change could cause an error	Floundering
PR2	Change made so it can be de-referenced in the future	
PR2	Change made to convert code to CHERI	
PR2	Change made to preserve capabilities	
PR2	Change made to silence a warning somewhere	Floundering
PR2	Confused about security implications	Floundering
PR2	It does not make sense to cast int to void	
PR3	It does have anything obvious with CHERI	Confusion
PR3	Made because CHERI requires being explicit with types	
PR3	Made because more CHERI compatible	
PR3	Not sure if it is CHERI related	Confusion
PR3	Not sure what it is doing	Floundering
PR4	It has security implications because bad size will lead to bad calculations	
PR4	It is made to make code CHERI compatible on other platforms	
PR4	It might be unnecessary	
PR4	Noticed that CHERI pointers are larger in size	
PR4	Will have security implications since it sets incorrect bounds	
PR4	Reject because it does not consider CHERI pointers size	
PR4	Reject because of strange implementation	Confusion

Table 4: Codes used when analysing pull-request task transcripts

When completeing the pull request task the developers were asked to decide whether a patch should be merged or not and whether, the merging of the patch had security implications. When completing this task our codebook seems to indicate less usability smells and instead more general confusion about the CHERI platform itself. Whilst there is evidence for *floundering* and *confusion*; most of the codes seem to represent developers being unsure. Rather than questioning whether they *should use something* or *how to use it* (signs of the confusion whiff), they instead seemed cautious and unclear.

Success rate with the task was mixed; with developers mostly incorrectly deciding against pulling the first and second patches and correctly pulling and rejecting the third and fourth—but not in overwhelmingly confident numbers. Given that patch three just rewrote some pre-ANSI standard C[8], and the caution in the codes identified: perhaps the developers just knew they didn't know enough about CHERI to make a definitive appraisal of the patches?

[8] Last seen in the 1970s, yet mentioned in the CHERI documentation so incorporated; K&R style C allows function arguments to be untyped in the usual argument parens and then optionally typed before the opening block. As untyped parameters are implictly an `int` until typed in K&R C they cannot be used as a capability later.

Overall success

Code	Usability Whiff
Developers have correct understanding of the issue but not able to fix it	Confusion
Need to understand CHERI model better	Confusion
Not sure about implications of changing types	Confusion
Not sure about the difference between void pointer and vaddr	Confusion
Not sure how CHERI handles size of void pointer	Confusion
Not sure how pointer width will change the behaviour	Confusion
Not sure if void types will increase in size with CHERI	Confusion
Not sure what's needed to be changed and what not	Confusion
Not sure what's the correct pointer size	Confusion
Not sure if problem is solved if warning was gone	Floundering
Not sure why error is there but warning is gone	Floundering
Hard to search the documentation	Sleuthing
It is not easy to read the docs	Sleuthing
Misunderstood docs	Sleuthing
Not knowing where to look for in the documentation	Sleuthing
There is a lot of information in the docs	Sleuthing
Trouble understanding examples in doc without context	Sleuthing
Understands the theory but needs more practical examples	Sleuthing

Table 5: Codes used when capturing developers struggles and comments

Throughout both tasks we noted when developers seemed to be struggling and why and captured their comments as codes (Table 5). The reasons varied between participants, but all seemed to display smells of *sleuthing*, and *confusion*. These smells indicate when developers are struggling to work with available documentation and either failing to find examples or understand what they are being told (sleuthing); and when they are unsure how to approach a problem and cannot reason about how to use something or what they need to do (confusion). In addition we also saw signs of *floundering* where developers were unsure whether a problem was solved just because a compiler warning had gone, or why an error persisted when the warning had gone. This seemed to add to developers struggles to successfully reason about their code.

Discussion

Something's fishy...

During several sessions we noticed the following when developers worked with CHERI code (Figure 6): developers would be presented with an error message or code they didn't immediately fully understand and would start searching the documentation for the literal pattern. When they found something similar enough they'd attempt to use the documentation to fix the error, but when their fix failed to work or left questions in their mind they became slightly panicked and would attempt solutions somewhat at random. Since this description doesn't fall neatly into Patnaik et al.'s code smells[9] we have created a new smell to capture it: *floundering.*

[9] It's sort of like *confusion* but rather than questioning what something does or how to do it, developers flounder around in search of something to give them a hint.

As two developers put it:

> "I might have to do a bit of trial and error here. . . "
>
> "I'm charging around just hoping to change things."

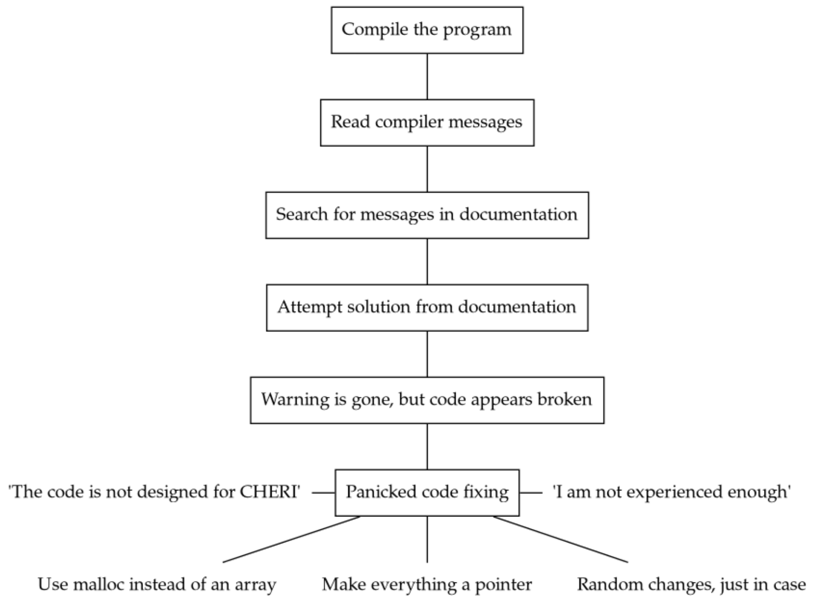
Compile the program
Read compiler messages
Search for messages in documentation
Attempt solution from documentation
Warning is gone, but code appears broken
'The code is not designed for CHERI'
Panicked code fixing
'I am not experienced enough'
Use malloc instead of an array
Make everything a pointer
Random changes, just in case

Figure 6: Floundering process as developers worked with the CHERI code, initially Googling for errors and then resorting to panic as the fixes didn't work.

Random solutions seemed to include switching arrays for malloc'd pointers and back and forth and making everything pointers (because CHERI is understood as having a different pointer architecture) and making random changes in the hope that something might fix the error or new errors in place. Alternatively they would blame either the code for being fundamentally incompatible with CHERI, or themselves for not being clever enough to understand it.

Despite the small sample size, our study has highlighted that there are usability issues with CHERI lurking. Several of our participants were competent C programmers and several had significant experience with CHERI itself; and yet the floundering smell was seen repeatedly throughout the study reducing developers to guessing and hoping that there programs were correct and secure. Not digital security by design; and a sign that CHERI needs better explanation for most developers.

Code	Category
Better way to get pointer size	Compiler
CHERI compilers should help the developer spot these problems	Compiler
Compilers should specifically say what types need to be fixed and changed	Compiler
Compilers should tell developers that other parts need to be changed	Compiler
Docs should show more examples that has context to them	Docs
Less writing and more simple examples	Docs
Need for a cheat sheet	Docs
Need for an easy conceptual model	Docs
Need for less verbose documentation	Docs
Simpler documentation of needed changes	Docs
Tutorial with a simple program then introduce changes to it	Docs

Table 6: Codes from sessions where participants suggested improvements to make their task easier.

Scope for improvement

When completing the tasks participants sometimes remarked on the usability challenges they were facing and what they felt could be done to improve them. Based on their suggestions we created codes to capture their suggestions. Broadly speaking these usability issues related to either the *compiler* or the *documentation* (Table 6).

The compiler. Programmers are well known to ignore warnings[10] yet when dealing with a new architecture this behavior can be especially dangerous. Several of our participants seemed confused about whether it was safe to ignore CHERI-specific warnings, and yet in every case it wasn't. The warnings were identifying the underlying issue, and yet the participants assumed they could ignore them:

> "I need to double check whether this warning is relevant or not."

> Interviewer: "So do you think these warnings generally will affect the programme if you run it?"
> Participant: "Oh, um. . . well, no, because they're warnings."

> "I don't think it will affect the programme running, because it's only a warning."

> "Well, yeah, I mean given it's a warning, it may not be a change that I need to do anything about."

Given that the warnings the developers faced were leading to the errors their programs encountered, perhaps it needs to be made more explicit to developers that these are issues that need fixing and should be treated as *errors* instead?

Additionally, since the compiler knows what it is compiling, additional warnings about particularly concepts which are likely to catch

[10] Peter Leo Gorski, Yasemin Acar, Luigi Lo Iacono, and Sascha Fahl. Listen to developers! A participatory design study on security warnings for cryptographic APIs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020

developers out when working with CHERI C (for example, surrounding the use of capabilities) could help developer spot issues ahead of time. Using empirically validated techniques to help ensure that compiler output is readable[11] may also help.

[11] Paul Denny, James Prather, Brett A Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B Powell. On designing programming error messages for novices: Readability and its constituent factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021

Furthermore, one of the issues that developers struggled with the most is that in some cases, the warnings will be gone, but the issue will still be there, and this happens because the developers made the correct change in one place, but that change is needed in other places as well. The compiler should be fixed to keep the warning until the correct change is implemented in all the needed places in the code.

The documentation. CHERI is a new—and changing—system, expecting the documentation to be polished at this early stage is perhaps unfair, but it is clearly an area programmers struggle with:

> "It's not fully clear what to look for. I have looked for these two types in here and I can't really spot like the main reason why it would change."
>
> "It's just very poor documentation, so it hasn't really got a basis actually, 'This is how you start,' there's no simple hello word example of how you can just use the pointers and other things."
>
> "Just taking the documentation as it is and taking the recommendation, that didn't work unfortunately."
>
> "I think the thing lacked any sort of tutorial. It jumped straight into fairly hardcore descriptions of it all, and there's no warm up."

Lack of tutorials, lack of CHERI expertise, and small errors led to programmers becoming frustrated. Other participants bemoaned how broad the documentation was and just wanted simpler examples and cheat sheets: providing more focused documentation for developers who just want to work with CHERI without understanding its full features and security architecture. Whilst we cannot expect all the tutorial material to be there before widespread adoption of digital security by design architectures—the calls for it will only get louder.

Barrier to entry

Finally, our study was carried out by a fairly limited number of participants: just 9. Despite a relatively high amount of reimbursement for partipants[12] we found recruitment extremely difficult. Despite approaching hundreds of students directly (the researcher was allowed to announce the study at the start of classes where students had an appropriate background, and approach TAs in labs), and many potential participants seeming initially keen; very few eventually took part, and a further 4 were rejected for not being able to demonstrate they'd read the introduction to CHERI.

[12] We offered £50 vouchers. A quick survey of the student noticeboard suggests that £5–10 is the normal rate for study participation.

Whilst the difficulties with recruiting developers are well known[13], in this particular study the problem seems to have been exasperated by the CHERI documentation. Informally, a few participants told us that they found the *Introduction to CHERI* and *CHERI C/C++ Programming Guide* intimidating and long. Participants reported them as being difficult to read. Enthusiasm for the study dropped and so participant numbers and rejection numbers seem to have suffered.

[13] Nikhil Patnaik, Joseph Hallett, Mohammad Tahaei, and Awais Rashid. If you build it, will they come? Developer recruitment for security studies. In *Recruitment of Participants for Empirical SE Studies (RoPES)*, 2022

Kind	Code	Usability Whiff
Praise	Found same warning	Sleuthing
Praise	Found similar examples	Sleuthing
Praise	Has good examples that compare traditional C to CHERI	Sleuthing
Praise	It explains things well	Sleuthing
Praise	It goes into details	Sleuthing
Criticism	Contradiction about no needed changes but a big API	Post-Mortem
Criticism	Error not found in documentation	Sleuthing
Criticism	Examples do not show context of the changes	Confusion
Criticism	Following the documentation does not help in fixing the bug	Post-Mortem
Criticism	Its in PDF format	
Criticism	No info about the need to do changes in other places not only the warning place	Sleuthing
Criticism	No information how to write CHERI compatible code	Sleuthing
Criticism	No simple guide or basics	Sleuthing
Criticism	No tutorial	Sleuthing
Criticism	There is no easy guide to code changes	Sleuthing
Criticism	Void pointers are not mentioned in the documentation	Sleuthing

Table 7: Codes used to capture participants thoughts when talking about the documentation.

When speaking about the documentation, whilst participants praised its examples and readability—helping to counter the *sleuthing* whiff (Table 7)—participants found it didn't cover every eventuality and error, and didn't help them to debug issues when things went wrong (adding to the *post-mortem* and *sleuthing* whiffs).

Speaking to CHERI developers it seems that whilst the documentation exists, the *normal* place most developers actually go for help is the CHERI developer chat channel; and that the official documentation is regarded as more of an academic reference. Whilst this may work acceptably when the number of CHERI boards is small; as adoption of digital security by design grows there is a need for more traditional reference documentation and guides as the number of developers, the languages they speak, and their questions grow.

Conclusion

As one developer put it:

> "CHERI stops your code from violating memory and security, but it doesn't stop the programmer from creating bad code."

The security possibilities of digital security by design offer a great opportunity to improve the safety of all software and hardware at a fundamental level. But early results from our study suggest that we are at risk of falling into the same usability pitfalls that plague conventional hardware: namely, poor documentation, and confusion about warnings and errors. These are all well known usability smells[14] for programmers, but to see them appearing with CHERI hardware suggests that there are still improvements to be made; which we will confirm with further study.

[14] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. Usability smells: An analysis of developers' struggle with crypto libraries. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 245–257, 2019

No matter what clever hardware we have, we *cannot* ignore the human element and the programmers building on these systems. It doesn't matter if we have secure by design hardware if we cannot build the software for it.

Key Recommendations

Whilst the CHERI architecture and software development environment is still *very* new; increased availability of Morello and awareness of DSbD means that CHERI is finally getting into the hands of developers. As CHERI matures, more usable documentation and support will be needed to ensure that developers find working with the platform a pleasure rather than a chore. To that end, we make the following recommendations:

Support developers when changes are required. The CHERI architecture is different to conventional computers that most developers have grown up using. Changes aren't often required *but when they are* make sure that developers can find what they need to change and understand why with the minimum of fuss. Don't assume that developers will have read the documentation or gone to the CHERI-project's Slack channel: make it easy to find.

It's about more than just technical reports. People learn differently. While technical reports and formal documentation might be right for some, for other developers it can be off putting. If developers can't find answers in the forms that work for them (be that Q&A style posts, tutorials, or videos) then they'll flounder instead of turning to the forms that don't work for them.

Digital Security by Design will ensure that sensible protections are built into every new technology from the get go; but if we want to ensure that these platforms are adopted quickly and the protections are utilised as effectively as possible then *the usability of Digital Security by Design cannot be an afterthought*.

Future Work

Our study has been run with a limited number of developers, and we still have a limited number of transcripts still to analyse (though our codebook has reached saturation). We plan to publish full results documenting our findings and the novel usability smell later at a conference venue.