



University
of Glasgow

[Singer, J.](#) (2023) Towards Secure MicroPython on Morello (WIP). In: 24th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '23), Orlando, FL, USA, 18 Jun 2023, pp. 134-137. ISBN 9798400701740 (doi: [10.1145/3589610.3596272](https://doi.org/10.1145/3589610.3596272))

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

© 2023 Copyright held by the owner/author(s). This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in LCTES 2023: Proceedings of the 24th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems <https://dl.acm.org/doi/10.1145/3589610.3596272>

<https://eprints.gla.ac.uk/297964/>

Deposited on: 12 June 2023

Enlighten – Research publications by members of the University of Glasgow
<http://eprints.gla.ac.uk>

Towards Secure MicroPython on Morello (WIP)

Jeremy Singer

University of Glasgow
Glasgow, United Kingdom
jeremy.singer@glasgow.ac.uk

Abstract

The Arm Morello platform is a prototype system that supports hardware capabilities for improving runtime security. Although Morello is a server class compute component, there is ongoing work aimed at bringing architectural capabilities to embedded scale devices. For this reason, we are porting the MicroPython framework to Morello. Our intention is to understand the impact of hardware capabilities on lightweight runtime execution environments, like MicroPython, that target embedded devices. In this work-in-progress report, we describe the minimal modifications required to compile the C source code of MicroPython for Morello. We show that this approach gives a working, but not necessarily more secure, version of MicroPython. Our paper proceeds to outline how capabilities could be used to improve runtime system security for MicroPython runtime and hosted applications.

CCS Concepts: • Security and privacy → Software security engineering; • Software and its engineering → Interpreters.

Keywords: capabilities, CHERI, Python

ACM Reference Format:

Jeremy Singer. 2023. Towards Secure MicroPython on Morello (WIP). In *Proceedings of the 24th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '23)*, June 18, 2023, Orlando, FL, USA. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3589610.3596272>

1 Introduction

As part of the UK Digital Security by Design (DSbD) initiative, Arm has released a prototype platform code-named ‘Morello’ [2] which implements the CHERI hardware capability concept. Essentially, a capability is a fat pointer including inline metadata for bounds, permissions and validity. The promise of hardware supported capabilities is that whole classes of memory vulnerabilities can be eliminated, including out-of-bounds reads/writes and use-after-free bugs [6].

LCTES '23, June 18, 2023, Orlando, FL, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the 24th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '23)*, June 18, 2023, Orlando, FL, USA, <https://doi.org/10.1145/3589610.3596272>.

In this work-in-progress report, we describe our ongoing project to port the MicroPython framework to Morello, making adaptations to the C source code of MicroPython in order to support a minimal level of capability-awareness so the code will compile and run on the Morello platform. This is a necessary first step towards enhancing the security and resilience of MicroPython on Morello; however, the key contributions will come when we incorporate more advanced features of capabilities—we sketch a roadmap for this development in Section 4.

2 Background

2.1 What is Morello?

CHERI [8, 11] is an abstract set of processor extensions to support capabilities as pointers. With appropriate compression techniques [10], the pointer data and metadata can be stored in a double machine word, i.e. 128 bits on a 64 bit architecture. A further one bit (the 129th bit) is stored out-of-band to represent the capability validity. This mechanism prevents capability values from being forged by untrusted code. Dynamic checks take place in hardware on each memory access, to ensure:

1. the capability is valid (tag check)
2. the capability has appropriate access permission (permission check)
3. the capability is within bounds for this memory access (bounds check)

If any of the checks should fail, a SIGPROT signal is raised and code execution is interrupted.

Morello [2] is the Arm instantiation of the CHERI concept. It is an quad-core AArch64 server class system-on-chip, based on the Neoverse processor, enhanced to support 128-bit architectural capability values. There are appropriate new instructions, along with modifications to the register file, memory hierarchy and data paths.

The Morello platform runs CheriBSD, a capability-aware variant of the FreeBSD OS. Within CheriBSD, user code may run in *hybrid* or *purecap* mode. The hybrid mode involves executing unmodified AArch64 code, which has no capability awareness. Effectively, this code runs in a process-level sandbox, with all raw pointers being converted to capabilities via global base capabilities in the Morello register set (e.g. the default data capability, DDC). The purecap mode involves executing Morello code with capability support. All references within purecap applications are represented as capabilities directly. This is the preferred execution mode

for CheriBSD applications, since it provides finer-grained capability support.

While Morello is a large scale machine, there is ongoing research to create smaller, embedded variants of the Cheri concept, e.g. [1, 12].

The most common language for developing Cheri applications is C [9]. Cheri C is a variant of C, with intrinsics for capability support and additional semantic constraints around pointer accesses.

2.2 What is MicroPython?

MicroPython is a lean Python runtime, typically targeting microcontroller scale devices. Although such systems generally run software compiled from low-level languages like C, MicroPython is a radical alternative. Scripting languages enable rapid development for prototyping purposes; also, Python is useful for hands-on educational contexts.

MicroPython accepts almost the full set of Python 3 constructs, with one or two minor absences. Textual code is parsed and compiled to a compact bytecode format, which is then interpreted at runtime. There is no JIT compilation in MicroPython.

Efficiency-oriented features of MicroPython include interned strings, small integers embedded directly within tagged pointers, optimised method calls, Python stack frames hosted on the C stack, garbage collection without reference counting, and exceptions using `setjmp/longjmp`.

MicroPython is written in C; the project is 307 kSLOC. There are ports for many common microcontroller families including Arm CortexM, and ESP32. Further, there is a Unix process port so the MicroPython can be compiled and run as a hosted Unix executable. MicroPython has support for AArch64, as well as Arm32 / Thumb implementations.

2.3 Related Work

In terms of related work, several ongoing research projects are investigating the development of managed runtimes on capability platforms. The JavaScriptCore (JSC) runtime has been partially ported to Cheri [4]. The CPython interpreter compiles for Cheri, but is missing key functionality when we try to use it.

3 Initial Porting Process

There are various configurations of MicroPython, for different architectures and variants. For our proof of concept work, we chose to port the `minimal` configuration, which does not incorporate any Python library support and has no external dependencies. To build the minimal MicroPython, the following tools are required:

- GNU make
- C compiler
- Python 3

On the default CheriBSD OS running on Morello, there is no Python 3 interpreter. For this reason, we elected to go down the route of cross-compilation to Morello.

3.1 Hybrid Binary

As a first attempt, we built MicroPython natively for FreeBSD AArch 64 on a cloud instance (AWS t4g.micro) running a Graviton2 Arm processor, which is a near equivalent to the Morello platform (only Graviton does not have capabilities). We copied the compiled MicroPython binary to our Morello test server and it executes properly in *hybrid* mode. Only trivial source code modifications were required in this case:

1. add a 'fake' `alloca.h` header file, which simply includes the `stdlib.h` header
2. modify the configuration header file to add a macro block `#ifdef __FreeBSD__` that sets the initial heap size and enables standard output.

3.2 Purecap Binary

As a second attempt, we built MicroPython through the standard Cheri cross-compile process, using the Cambridge-supported LLVM toolchain for Morello and the `cheribuild` framework. This Morello cross-compiler is hosted on an x86-64 Linux server. In this case, more extensive source code modifications were required, in addition to those for the previous FreeBSD build. We describe these modifications in the remainder of this section.

Register Save Sequences. For non-local jumps (e.g. for `longjmp`, for exceptions, etc) sets of registers need to be saved onto the stack, following the procedure calling convention. These sequences need to be modified for morello, to use a capability base address for saves, and to update the size of the save block (since capabilities are twice as large as raw pointers).

Subword Bitfields in Structs. In the interests of conserving space, MicroPython in-memory data structures use subword bitfields in struct definitions. This feature is not supported by the Cheri LLVM toolchain, so we redefine the relevant structs so every field is a distinct word.

Capability Provenance Ambiguity. When multiple capabilities are used in an arithmetic calculation, the compiler is not clear how to handle provenance (tag validity). Cheri C has a *single-provenance semantics*, which means that every reference must be derived from exactly one other reference. Essentially, this means that only one set of metadata (bounds, permissions, validity) can be propagated to the result. We simplified the code by breaking up complex calculations into simpler, two-address style calculations. This prevents compiler errors (really only warnings) and allows the compiler to generate executable purecap code.

For the majority of source code modifications in this class, the capability metadata is irrelevant since the code is operating on tagged integer values that are embedded inside MicroPython references.

Extent of Modifications. We measured the number of lines of C source code that we needed to modify in order to enable MicroPython to compile for Morello in purecap mode. We emphasize that this is not a full port, only an initial proof-of-concept with negligible testing and no meaningful capability-based features.

In total, we modified 87 lines of code. This includes 49 altered lines and 38 lines of additional code. This is a vanishingly small proportion of the overall codebase—as a percentage we have modified 0.028%. Remarkably, this proportion corresponds almost exactly with the figure of 0.026% reported by other researchers for the open-source KDE desktop port [7].

4 Future Work

At present, we have a working, minimal MicroPython interpreter on Morello. However we are not taking advantage of any of the capability features. This section explores possible future development directions that will enhance our port and leverage the CHERI capabilities.

The first step will be to add *tight bounds to all memory allocations*. This will serve to make the system more secure. Further, we might be able to use CHERI hardware bounds checking to eliminate some of the software bounds checking performed by MicroPython.

Next we will study the *garbage collector (GC)* for MicroPython. This is a simple whole-heap tracing collector that implements the mark/sweep algorithm. While some research has been carried out regarding GC for capability systems [5] this will be helpful additional evidence for how to operate GC with capabilities, particularly in the context of a tightly integrated memory manager and language runtime.

The MicroPython framework uses the bottom few bits of object references to indicate whether this reference is to an in-heap object, a string or an integer value. Small integer values are encoded directly within the reference value itself. This superposition of values on object references is accomplished by *tagging*, so the interpreter understands how to evaluate each reference. As we consider the Morello port of MicroPython, we need to assess whether such tagged integers and strings make sense with capabilities (instead of raw pointers). Could we use the Morello capability metadata to handle tags instead of stealing bits from the pointer payload?

MicroPython in minimal configuration is missing many useful libraries, for instance, for regular expressions. Once we have a working GC, we intend to incorporate as many of these libraries as we can.

One of the most promising features of Morello is its support for *lightweight software compartmentalization*. Capabilities allow us to divide an executing program into mutually distrusting compartments with managed interfaces between them. We will explore the Morello compartmentalization primitives and determine how best to split up the MicroPython runtime. We will be looking to compartmentalize libraries and driver code in particular, since these are often supplied by third parties and may be untrusted [3]. Another interesting line of work would be to exploit the compartmentalization mechanism for the foreign function interface. Currently, foreign functions may be able to damage the Python heap if they behave improperly. We will further explore whether it makes sense for compartmentalization to be exposed to hosted Python applications running on MicroPython—can this Python software be split into compartments using a simple Python API?

Finally, we intend to explore the potential for running MicroPython in *baremetal mode* on Morello, rather than as a process within a conventional Unix OS like CheriBSD. Although this is not a feasible use case for server grade Morello platforms, we are aware that researchers are developing microcontroller devices with CHERI support [1]. Baremetal MicroPython is a typical microcontroller deployment scenario.

5 Conclusions

In this work in progress report, we have described our initial attempts at porting the MicroPython framework to the new capability-aware Morello platform. We have sketched out a roadmap of future work for improving capability support for MicroPython, which will become increasingly relevant as hardware capability support is adopted for microcontroller scale devices.

Acknowledgments

This work was partly funded by the Digital Security by Design (DSbD) programme delivered by UKRI (including grants EP/V000349/1 and EP/X015831/1), also by the UK Defence and Security Accelerator contract ACC6037520.

References

- [1] Saar Amar, Tony Chen, David Chisnall, Felix Domke, Nathaniel Filardo, Kunyan Liu, Robert Norton-Wright, Yucong Tao, Robert N. M. Watson, and Hongyan Xia. 2023. *CHERIoT: Rethinking security for low-cost embedded systems*. Technical Report MSR-TR-2023-6. Microsoft. <https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/>
- [2] Arm. 2021. Arm Architecture Reference Manual Supplement — Morello for A-profile Architecture. <https://developer.arm.com/documentation/ddi0606/>.
- [3] Adrien Ghosn, Marios Kogias, Mathias Payer, James R. Larus, and Edouard Bugnion. 2021. Enclosure: Language-Based Restriction of

- Untrusted Libraries. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 255–267. <https://doi.org/10.1145/3445814.3446728>
- [4] Brett Gutstein. 2022. *Memory safety with CHERI capabilities: security analysis, language interpreters, and heap temporal safety*. Technical Report UCAM-CL-TR-975. University of Cambridge, Computer Laboratory. <https://doi.org/10.48456/tr-975>
- [5] Dejice Jacob and Jeremy Singer. 2022. Capability Boehm: Challenges and Opportunities for Garbage Collection with Capability Hardware. In *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. 81–87. <https://doi.org/10.1145/3516807.3516823>
- [6] Nicolas Joly, Saif ElSherei, and Saar Amar. 2020. Security Analysis of CHERI ISA. <https://msrc.microsoft.com/blog/2020/10/security-analysis-of-cheri-isa/>.
- [7] Robert N. M. Watson, Ben Laurie, and Alex Richardson. 2021. Assessing the Viability of an Open-Source CHERI Desktop Software. https://www.capabilitieslimited.co.uk/_files/ugd/f4d681_e0f23245dace466297f20a0dbd22d371.pdf
- [8] Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, Graeme Barnes, David Chisnall, Jessica Clarke, Brooks Davis, Lee Eisen, Nathaniel Wesley Filardo, Richard Grisenthwaite, Alexandre Joannou, Ben Laurie, A. Theodore Marketos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alexander Richardson, Peter Rugg, Peter Sewell, Stacey Son, and Hongyan Xia. 2020. *Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8)*. Technical Report UCAM-CL-TR-951. University of Cambridge, Computer Laboratory. <https://doi.org/10.48456/tr-951>
- [9] 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. 2020. *CHERI C/C++ Programming Guide*. Technical Report UCAM-CL-TR-947. University of Cambridge, Computer Laboratory. <https://doi.org/10.48456/tr-947>
- [10] Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert M. Norton, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Marketos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, and Simon W. Moore. 2019. CHERI Concentrate: Practical Compressed Capabilities. *IEEE Trans. Comput.* 68, 10 (April 2019), 1455–1469. <https://doi.org/10.1109/TC.2019.2914037>
- [11] Jonathan Woodruff, Robert N.M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. 2014. The CHERI Capability Model: Revisiting RISC in an Age of Risk. In *Proceeding of the 41st Annual International Symposium on Computer Architecture*. 457–468. <https://doi.org/10.1145/2678373.2665740>
- [12] Hongyan Xia. 2021. *Capability memory protection for embedded systems*. Technical Report UCAM-CL-TR-955. University of Cambridge, Computer Laboratory. <https://doi.org/10.48456/tr-955>

Received 2023-03-16; accepted 2023-04-21