Model theft attack against a tinyML application running on an Ultra-Low-Power Open-Source SoC

Model theft attack against a tinyML application running on an Ultra-Low-Power Open-Source SoC

Antonio Porsia
antonio.porsia@polito.it
Politecnico di Torino
Turin, Italy

Annachiara Ruospo
annachiara.ruospo@polito.it
Politecnico di Torino
Turin, Italy

Ernesto Sanchez
ernesto.sanchez@polito.it
Politecnico di Torino
Turin, Italy

ABSTRACT

With the advent of tinyML, IoT devices have expanded their range of operations from simple data gathering and transmission to full-fledged inference. This expansion has been further enabled by the rise in popularity of open-source hardware, with the RISC-V architecture being the most prominent example. TinyML's decentralization can solve the current privacy and security issues of IoT infrastructures. However, it also shifts the burden of security on already resource-constrained devices. Ultra-low-power devices, in particular, often sacrifice security features for energy and area efficiency. This work aims at showing that, in the context of edge computing based on open-source hardware, neglecting hardware security features for the sake of efficiency is not an acceptable trade-off with respect to AI security.

CCS CONCEPTS

• **Security and privacy** → **Embedded systems security**; • **Computer systems organization** → *System on a chip*.

KEYWORDS

Hardware Security, tinyML, Ultra-Low-Power Hardware, Open-Source Hardware

1 INTRODUCTION

Nowadays, special attention is posed to embedded systems security. In particular, when embedded devices belong to safety-critical applications, new standards, regulations and laws have been enacted by different governments, advisory and legislative boards. This guarantees a strong position regarding security that helps manufacturers to adopt a set of good manufacturing practices against possible attacks, and at the same time, benefits customers since the security of the final product is improved. This is the case of automotive industry regarding security, where last year regulations will become mandatory briefly: United Nations Regulation No. 155 (United Nations Regulation No. 155 (UNR155)). On the other hand, a lack of regulation, and correspondingly a lower attention to security is given to embedded systems not directly involved in safety critical applications. This is the case of most of the adopted applications labeled as tiny-ML, where the proposed application is low-power, low-cost and usually may run without human intervention. In this context, the Artificial Intelligence algorithm running in the embedded device may be vulnerable to different cyber attacks. In this context, this paper proposes a preliminary analysis regarding the possible security vulnerabilities presented by a tiny-ML application based on an open-hardware core. In particular, a model theft attack has been analyzed in a case study running a simple CNN addressing handwritten digit recognition in a microprocessor core based on an open source low-power RISC-V system. Thanks to the developed case study, it was possible to experimentally demonstrate that an attack such as model theft can be easily implemented by an attacker. Finally, the paper addresses some low-cost countermeasures to the presented attack. All materials used for the experiment are available in a GitHub repository[1]. The rest of the paper is organized as follows: section 2 provides some necessary background about RISC-V and tinyML. Section 3 introduces the case study, and section 4 describes the developed experiments. The last section concludes the paper.

2 BACKGROUND

2.1 RISC-V

RISC-V is an architecture that aims at providing a completely open and easily customizable Instruction Set Architecture (ISA) [12] targeting a range of devices that goes from ultra-low-power embedded systems [2] to high-performance clusters [12]. Openness and customizability have been, and still are, key factors in the proliferation of hardware based on RISC-V, from CPU cores, to System-on-Chips, to Deep Learning (DL) accelerators.

Security Features. Two optional security features are defined in the RISC-V Privileged Specification [13]:

- **Execution Privilege Separation** – From most to least privileged modes: *Machine mode* (M), *Supervisor mode* (S) and *User mode* (U). M-mode is the only mandatory privilege level
- **Physical Memory Protection** – The PMP unit enforces access control on memory accesses granting or revoking access privileges for each physical memory region. PMP configuration registers feature a lock bit that, when set, inhibits modifications to the corresponding register until the next hardware reset, regardless of the privilege level. Furthermore, it enables access control on M-mode accesses, which by default have full permissions

2.2 TinyML

TinyML is an emerging field of Machine Learning that aims at bringing ML capabilities to resource-constrained devices that up until now have been employed only for data collection at the extreme edge of the IoT network. These devices often consist of a battery-powered microcontroller unit (MCU) and various sensors, with power consumption in the range of 1 mW or lower [10]. TinyML enables these devices to analyze sensor data *on the spot* while keeping power consumption low, effectively eliminating various security,

[1] https://github.com/cad-polito-it/x-heep-tflite-cfoshw24

privacy and connectivity issues, since data almost never leaves the device [3]. TinyML use cases include safety- and security-critical systems, such as healthcare devices [11] and self-driving vehicles [1]. For this reason, securing tinyML models and devices is crucial.

2.3 AI security

The growing use of artificial intelligence in critical systems naturally raises concerns about the potential security issues that may arise during the lifecycle of an ML model.

2.3.1 Securing AI. According to the the European Telecommunications Standards Institute (ETSI) GR SAI 004 group report [4], securing an AI system means ensuring its confidentiality, integrity and availability at all stages of its life cycle: data acquisition, data curation, model design, software build, training, testing, deployment/inference and update. As for the deployment phase, the role of hardware in AI security is addressed by ETSI GR SAI 006 [5] and ETSI GR SAI 009 [6]. However, both reports focus on hardware as an inherently trusted defense mechanism, without fully investigating the impact of hardware design choices on the available attack surface of the AI application. As far as this work is concerned, the most interesting threats outlined by the group reports are:

- **Poisoning attacks** – can be further categorized into (1) data set poisoning, (2) algorithm poisoning and (3) model poisoning. The latter consists in manipulating the model's parameters or simply replacing it entirely to alter its behavior
- **Reverse engineering** – consists in reconstructing a model able to reproduce the behavior of the target model. Includes techniques such as model stealing, consisting of the exfiltration of the model's parameters and structure.

3 CASE STUDY

In this section, the tinyML device used as case study is presented. The device is based on an open-source microcontroller. Interestingly, a preliminary analysis regarding the secure weaknesses of the device are provided.

3.1 X-HEEP

X-HEEP (eXtendable Heterogeneous Energy-Efficient Platform) is an open-source RISC-V MCU developed specifically for edge-computing platforms [7]. X-HEEP allows to choose one of three RISC-V CPU cores from the OpenHW group: (1) *CV32E20* (a fork of the Ibex core), aimed at control-oriented tasks; (2) *CV32E40P* (formerly known as RI5CY) for processing-oriented tasks; (3) *CV32E40X*, a fork of CV32E40P featuring an external interface for coprocessors and lacking the floating-point unit and custom Xpulp ISA extensions. All of the cores only offer M-mode execution and have no PMP unit.

3.1.1 X-HEEP weaknesses. X-HEEP can be extended with a coprocessor or an accelerator to offer security functionalities or accelerate AI operations. However, such critical tasks require additional precautions and hardware support that X-HEEP currently lacks. While some shortcomings may be taken care of by extending X-HEEP with a security coprocessor, the platform itself presents some weaknesses pertaining to the RISC-V cores available by default that must be addressed:

- **Lack of memory protection mechanisms** – The default configuration of the RISC-V cores shipped with X-HEEP has no PMP unit, hence depriving the CPU of a basic memory protection mechanism. As a result, an attacker that gains unauthorized code execution gains also unrestricted access to memory. Besides, Data Execution Prevention (DEP), i.e., the prohibition of executing code located in data sections, cannot be enforced.
- **Lack of execution privilege separation** – All the RISC-V cores that can be integrated in X-HEEP implement only M-mode execution, making it impossible to adequately perform access control based on execution privileges. As a consequence, an attacker who gains code execution through a software exploit, automatically gains full access to the system. This weakness can lead to a breach of confidentiality and/or integrity even in the presence of a PMP unit, if the lock bit is unset or unimplemented.

It should be noted that these are no vulnerabilities *per se*, but rather they are weaknesses that may *enable* attacks by making some vulnerabilities, such as buffer overflows, more easily exploitable.

3.2 Threat model

The system upon which this analysis is based is composed of the X-HEEP platform, a tinyML application, the necessary runtime software for the application to interact with the hardware and a command protocol running over the UART port to allow external interaction with the application. The software is assumed to contain a vulnerability that can result in unauthorized code injection and execution.

3.2.1 Adversary model. For the purposes of this work, the attacker is assumed to have unrestricted physical access to the device and full knowledge of the device architecture and memory map, as well as the capability to interact with the device by sending commands and reading the output.

3.2.2 Critical assets and possible threats. The main asset to be protected is the ML model. The OWASP Machine Learning Security Top Ten offers a good overview of what are currently the critical security issues of AI systems [8]. The item of interest on the list is *model theft*, an attack that consists in gaining access to the model's parameters. This threat is enabled by the underlying hardware's weaknesses, since it requires the adversary to gain read access to the location where the model is stored. If there are no hardware facilities to control memory accesses, the confidentiality of the model completely relies on the assumption that software is and always will be vulnerability-free.

3.3 Attack outline

For the purposes of this work, the ML model is assumed to reside in the `.data` section of the application executable. Since by hypothesis Data Execution Prevention (DEP) cannot be enforced and no access control can be performed on memory accesses, it is possible to inject and execute code such that the entire data section of memory

can be dumped and analyzed to exfiltrate the ML model, hence resulting in a model theft attack.

4 EXPERIMENTAL RESULTS

Experiments were conducted on a TUL PYNQ-Z2 board, which includes a Xilinx Zynq Z7020 SoC. Additional details about the setup of the hardware and software components are provided in the following section.

4.1 Setup

Table 1: FPGA Resource Utilization for X-HEEP FEMU with 16 memory banks

Resource	Used	Available	Percentage
LUTs	37,486	53,200	70.46%
Flip-Flops	49,087	106,400	46.13%
Block RAM	585 KiB	630 KiB	92.86%

4.1.1 Hardware. The TUL PYNQ-Z2 board features a Programmable Logic (PL) side, i.e., the FPGA, and a Processor Subsystem (PS) side, which is capable of running Linux and communicating with the PL side.

The X-HEEP-based FPGA Emulation Platform (FEMU) was utilized to deploy X-HEEP on the PYNQ-Z2 and take advantage of features such as on-chip JTAG and UART virtualization for easy debugging.

The X-HEEP MCU has been resynthesized using Vivado 2023.2 with sixteen 32KB memory banks, providing a total of 512 KB of SRAM to accommodate the application and ML model. The rest of the hardware has been kept with the default configuration. The FPGA resource utilization is reported in Table 1.

4.1.2 Software. An example application was developed to highlight weaknesses in the underlying hardware. The application simply runs an infinite loop, waiting for commands from the user. The command interface uses the SCPI protocol, which is designed for communicating with measurement equipment such as oscilloscopes and multimeters and was standardized by IEEE in the IEEE/IEC 60488-2-2004 standard. SCPI commands are sent over the UART port.

Vulnerability. The application accepts custom inputs for the ML model using the arbitrary block feature of SCPI, which allows passing a data block of custom length as a command parameter. A buffer overflow vulnerability has been inserted here: instead of copying a fixed number of bytes into the input buffer that will be passed to the model, the application copies the number of bytes specified in the command, hence allowing an attacker to overwrite data on the stack, including the return address.

4.1.3 ML model. The deployed ML model is a simple Lenet5 Convolutional Neural Network trained on the MNIST dataset for handwritten digit recognition. The library used to deploy it is Tensorflow Lite for Microcontrollers (TFLM), one of the most popular tinyML frameworks. It is specifically targeted at low power resource-constrained MCUs, with the core runtime fitting into just 16KB on an ARM Cortex M3 [9]. The model is stored in a C array, hence it resides in the read-only data section of the executable.

4.2 Model theft attack

The first stage of the attack consists in finding the starting address of the buffer, in order to be able to include loops into the injected code. This has been achieved through a payload that executes the following operations: first it loads a canary, i.e., an easily recognizable value, into a register, then executes a series of nop instructions interleaved periodically with a load of a second canary into another register, and finally it prints the two canaries over the UART port. Once the execution reaches the end of the function, there are three possibilities: (1) if the injected return address points outside the buffer, the program likely crashes and restarts, allowing to try again; (2) if it points inside the nop section, only the second canary is printed; (3) if it points to the start of the buffer, both canaries are printed. The purpose of the nop section is to identify with less possible attempts an address inside the buffer. Once the second canary is printed, it is possible to fine-tune the address until the buffer start is found. Note that this is possible due to the executable being statically linked. Starting from the highest possible address, the process may require roughly 30 minutes.

The second stage consists in the proper model theft. The second payload loops over the whole data memory and writes it to the UART port, then it jumps to the first instruction of the boot ROM, resulting in a software reset. The data read from the UART port has been dumped to a file for further analysis.

4.2.1 Model recovery. TensorFlow Lite models are stored using FlatBuffers, an object serialization format designed specifically with mobile hardware in mind. A FlatBuffer contains a series of nested tables, structures and vectors that can be accessed using offsets stored in the FlatBuffer itself, so that the data structure may be traversed in place without prior loading and parsing. A FlatBuffer has to follow a precise schema defined *a priori*. In the case of TFLM, the schema is publicly available on the TFLM GitHub repository. In particular, TFLM's FlatBuffer schema contains the file identifier string "TFL3", which is placed exactly 4 bytes after the start of the model. Note that since offsets are stored into the FlatBuffer itself, knowing the exact size of the model is not necessary to successfully steal it and use it: it is possible to copy whatever comes after the "TFL3" string and TFLM will load it without a hassle.

4.2.2 Possible mitigations. This attack could be easily prevented if PMPs and U-mode were implemented. In particular, the TFLM code must be executed in M-mode, while the main logic of the program, including the vulnerable command parser, in U-mode. A PMP register must be configured to allow memory accesses to the model only in M-mode. Another mitigation could have consisted in configuring a PMP register to disable execution of the data section regardless of the privilege mode, using the Lock functionality of the PMP unit. As a further security measure, direct access to the UART port must be disabled in U-mode. Since none of these functionalities are available in X-HEEP, it is not possible to implement these mitigations. Instead, the user must hope that the software is and will always be vulnerability-free.

5 CONCLUSIONS

In this paper, a preliminary analysis of the security vulnerabilities related to the implementation of a tinyML application in an open-source microcontroller is provided. The implemented application runs a light version of a CNN for handwritten digit recognition in a RISC-V low-power SoC. As experimentally demonstrated, security issues are easily exploited by an attacker due to the missing security extensions in the open-source hardware. Finally, some countermeasures have been presented.

ACKNOWLEDGMENTS

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 118/2023.

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

REFERENCES

[1] Michael Bechtel, QiTao Weng, and Heechul Yun. 2022. DeepPicarMicro: Applying TinyML to Autonomous Cyber Physical Systems. arXiv:2208.11212 [cs.LG]

[2] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi, Antonio Pullini, Eric Flamand, and Luca Benini. 2017. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 1–8. https://doi.org/10.1109/PATMOS.2017.8106976

[3] Dr. Lachit Dutta and Swapna Bharali. 2021. TinyML Meets IoT: A Comprehensive Survey. *Internet of Things* 16 (2021), 100461. https://doi.org/10.1016/j.iot.2021.100461

[4] ETSI SAI ISG. 2020. *Securing Artifical Intelligence (SAI); Problem Statement.* Group Report DGR/SAI-004. ETSI.

[5] ETSI SAI ISG. 2022. *Securing Artifical Intelligence (SAI); The role of hardware in security of AI.* Group Report DGR/SAI-006. ETSI.

[6] ETSI SAI ISG. 2023. *Securing Artifical Intelligence (SAI); Artificial Intelligence Computing Platform Security Framework.* Group Report DGR/SAI-006. ETSI.

[7] Simone Machetti, Pasquale Davide Schiavone, Thomas Christoph Müller, Miguel Peón-Quirós, and David Atienza. 2024. X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators. arXiv:2401.05548 [cs.AR]

[8] OWASP Machine Learning Security Top Ten 2023. *OWASP Machine Learning Security Top Ten.* Retrieved March 1, 2024 from https://mltop10.info

[9] TensorFlow Lite for Microcontrollers 2024. *TensorFlow Lite for Microcontrollers.* Retrieved February 19, 2024 from https://www.tensorflow.org/lite/microcontrollers

[10] tinyML Foundation 2024. *tinyML Foundation.* Retrieved Feburary 19, 2024 from https://www.tinyml.org

[11] Vasileios Tsoukas, Eleni Boumpa, Georgios Giannakas, and Athanasios Kakarountas. 2022. A Review of Machine Learning and TinyML in Healthcare. In *Proceedings of the 25th Pan-Hellenic Conference on Informatics* (Volos, Greece) *(PCI '21)*. Association for Computing Machinery, New York, NY, USA, 69–73. https://doi.org/10.1145/3503823.3503836

[12] Andrew Waterman and Krste Asanović (Eds.). 2019. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213.* RISC-V Foundation.

[13] Andrew Waterman, Krste Asanović, and John Hauser (Eds.). 2021. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203.* RISC-V International.