BEERR: Bench of Embedded system Experiments for Reproducible Research

Paul Olivier
EURECOM
paul.olivier@eurecom.fr

Xuan-Huy Ngo
EURECOM
xuan.ngo@eurecom.fr

Aurélien Francillon
EURECOM
aurelien.francillon@eurecom.fr

***Abstract*—Reproducing experiments is a key component to further research and knowledge. Testbeds provide a controlled and configurable environment in which experiments can be conducted in a repeatable and observable manner. In the field of system security, and binary analysis, several challenges hinder reproducible research, in particular when code is interacting tightly with low level hardware and physical devices. In those conditions, dynamic analysis techniques often require the physical device to correctly complete (hardware-in-the-loop). In recent years many rehosting techniques have been developed and evaluating their respective performance requires to compare them with an hardware-in-the-loop evaluation. However, it is challenging to share, acquire or maintain the original devices.**

In this paper, we tackle this problem by proposing a new infrastructure, and online service called "Bench of Embedded system Experiments for Reproducible Research" (BEERR). It aims to both make physical devices available remotely and facilitate the setup and reproduction of published experiments.

***Index Terms*—security, dynamic analysis, embedded systems, reproducible research, hardware-in-the-loop**

1. Introduction

A core aspect in the journey to expand the knowledge is experimenting. Experiments help to validate or refute hypothesis. One of experiments' most important attribute is reproducibility. The modification of experiments parameters is crucial in observing the impact of results and improvements. In this way, the experimenters better understand and study the phenomena at play. This is part of the reason why Science is an iterative process. By sharing reproducible experiments, other scientists can in turn add their own expertise into them and progress.

In particular, an emphasis has been recently put in making research more reproducible in computer science, and in system security in particular [3], [7], [44], [47]. For this purpose conferences promote the publication of the code and data that have produced the results along with the paper. Several conferences now award badges to publications whose artifacts have successfully been reproduced by an evaluation committee [6], [20], [31], [33], [45]. These badges characterize the way the artifacts have been audited and how reproducible they are. For instance, the ACM Artifact Review and Badging version 2.0 [6] defines three independent type of badges: *Artifacts Evaluated*, *Artifacts Available* and *Results Validated*. Each describes a qualitative set of requirements applied to the artifacts associated with the research.

According to the type of artifacts the experiment manipulates, different challenges may arise at the time of publication. First, it is often the source code that is the most easily published along the paper. This bring transparency on experiments by providing the exact operations executed. However, source code alone isn't sufficient to make an experiment reproducible. The second difficulty with reproducibility is if, and how, to share the data used during an experiment. Data copyright may raise concerns while malware studies face ethical issues to openly share datasets. User privacy also limits the scope of legitimate traffic, behavior and traces that can be collected and published. More trivially, the size of the dataset has a direct influence on its ease to be shared.

Third, the environment of the experiment has a great impact on its results. Thanks to virtualization technologies, sharing the exact environment for software is often straightforward. In contrast, trade-offs have to be made with hardware. For dynamic analysis, emulation is the primary solution to abstract hardware requirements. But it is not always feasible, as shown by Fasano et al. [15]: the task itself can be onerous and it exists a large variety of devices to support. To circumvent these obstacles, a different approach is to record the execution into traces that can be replayed later and somewhere else [13], [43]. It is however impossible to explore new execution paths with a replay. Both approaches suffer from limitations, highlighting the importance of using physical devices into the analysis loop.

Fourth, the experiment setup has an essential influence on the measured results. A wrong configuration can yield a different outcome from the original study. It is a crucial aspect for fair comparison between methodologies.

Experiments interacting with hardware devices introduce four challenges:

1) *the cost of the device:* How expensive is it? Is the budget available for purchasing it?
2) *availability:* Is the device still produced and sold? Is the device available and legal to operate in user's region of the world?
3) *storage:* Where to keep the device? How to make it accessible to use it?
4) *safety:* Can the device be dangerous when manipulated?

In this paper, we want to address these last questions by proposing the *Bench of Embedded systems Experiments for Reproducible Research*, BEERR. It is an infrastructure aiming to ease the access to physical devices used in system security publications by making them available remotely. In addition, we want to congregate and propose a collection of experiment code and data which can be easily used with these devices. We focus on dynamic firmware analysis with a strong emphasis on studies looking at interactions with the hardware. We believe BEERR can help future researchers by simplifying the access to published experiments, removing the burden of hardware management, promoting fair comparison between studies and be a key to foster new ideas. The website is accessible at *https://beerr.s3.eurecom.fr*.

The remainder of this paper is structured as follows. Section 2 describes relevant background information on computer testbeds and their potential applications in the context of system security and embedded systems. Section 3 proposes a high level survey of publications and their artifact status for hardware-in-the-loop papers. In Section 4, we present an overview of our testbed and its workflow. Section 5 describes the software and hardware implementation of our infrastructure. Lastly, section 6 discusses the security aspect of the proposed infrastructure and sketches future works.

2. Background

2.1. Terminology

To clear any confusion within the rest of the publication with the terms *repeatable*, *reproducible* and *replicable*, we refer to the definition provided by ACM artifact review and badging [6]:

- ***Repeatability:*** *[...] a researcher can reliably repeat her own computation.*
- ***Reproducibility:*** *[...] an independent group can obtain the same result using the author's own artifacts.*
- ***Replicability:*** *[...] an independent group can obtain the same result using artifacts which they develop completely independently.*

2.2. Computer Testbeds

To overcome the difficulties in reproducing, sharing and comparing experiments, scientific communities are building testbeds. They take a wide variety of forms depending on their objectives and the problems to be studied. They can be small robots placed in an arena to study swarm intelligence as done in the Robotarium [36]; a grid of radio transceiver to analyze wireless interference and performance on different topology (R2Lab [34]); inspect IoT devices interactions within an environment (FIT IoT-LAB [9] and CorteXLab [25]); or to study distributed systems networking and services such as Emulab [46], CloudLab [14], PlanetLab [35] and more recently EdgeNet [40].

To simplify administrative work, management and to be accessible to as many people as possible, front end projects have been created to federate multiple testbeds [2], [10] and offer a common interface. A recent example of this case is the Fed4FIRE+ project which ran for 5 years. It gathered 20 different testbeds in Europe and led to an large number of publications [2].

In the context of system security, building a testbed is an effective approach to study a system [16], [32], [39], [49]. A diverse range of applications are reasonable: benchmarking [27], training [8], [26], or investigating multi-stage attacks on distributed systems. In other words, all situations where the reproduction of the environment is crucial to understand the events that are at stake. Examples of such case are often related to networking attacks like DDoS [19], industrial control systems such as SCADA [8], [26], [28], [37], or Internet-of-Things [39] where physical devices communicate with external services often located on the Internet.

Fuzzing has recently experienced a considerable interest in software testing and vulnerability research because of its efficiency in bug finding. Yet establishing good methodologies and metrics to compare fuzzing techniques remain a challenge for researchers. For this reason, Google proposes a service called FuzzBench [27] to evaluate and compare fuzzers against a set of benchmarks.

2.3. Hardware-in-the-loop

Hardware-in-the-loop (HIL) testing consists in the integration of physical components within a simulation [12], [21], [22], [24], [28], [29], [32], [38], [42]. In this way, the physical devices are fed with inputs from the simulation while its outputs are monitored. This technique brings the benefit to only require an access of the interface with the physical component to join. This interface is often standardized (JTAG, I2C, MMIO) which removed the burden to have knowledge of the internal structure of the device: it is considered as a black box.

Cars are increasingly filled with embedded systems in complex and sensitive architectures. The automotive sector has strong incentives from regulations to extensively test their systems. HIL helps to build realistic testbeds to establish the reliability of the system [16], [32].

In the context of system security, HIL gained a lot of interest recently with the dynamic firmware analysis technique called *rehosting* [15], [48]. The core idea of rehosting is to move the firmware into a virtualized environment where it is easier to instrument and apply other general testing techniques such as debugging, tracing, fuzz testing and symbolic execution [5], [18].

Another essential advantage of HIL lies in its high fidelity of outputs returned to the simulation. In contrast, the approach presents several drawbacks: it only scales with the number of physical devices employed and requires an access to debug them. Depending on the approach taken, the execution overhead introduced by forwarding accesses may impact the analysis speed significantly.

2.4. Continuous Integration

Continuous Integration (CI) helps developers to automatically merge their changes into the main development tree. However, before merging, the changes need to be tested to minimize new bugs or regressions. Embedded systems are known to be a very heterogeneous ecosystem. It is therefore difficult to know upfront if a specific code change would work on all the hardware the project

TABLE 1. EXPERIMENTS DESCRIPTION IN SURVEYED PAPERS.

Publication	Experiment	Hardware	Artifact availability	
			hardware[1]	source code
Wycinwyc [30]	study effects of memory corruption on different class of embedded systems	Beaglebone Black Linksys EA6300v1 Foscam FI8918W STM32 L152RE	✓ ✗ ✗ ✓	✗
	measure mitigation execution overheads with fuzzing	STM32 L152RE	✓	✓
Avatar² [29]	reproduce existing study	PLC Allen Bradley 1769-L16ER-BB1B	✓	✓
	state transfer between concrete and symbolic execution modes			
	record firmware execution	STM32 L152RE		
Avatar² examples	forward memory accesses between emulators		✓	✓
	state transfer between different targets	STM32 L152RE		
	state transfer & peripheral modeling	nRF51-DK		
Avatar [50]	backdoor detection in a masked ROM bootloader	Seagate ST3320413AS HDD	✗	✓
	vulnerability research in a commercial Zigbee device	Redwire Econotag	✗	
	helping reverse engineering the GSM stack of a phone	Motorola C118	✓	
Charm [42]	feasibility *(how long it takes to port a new driver)* performance *(driver fuzzing with Syzkaller, driver initialization)* record-and-replay *(record bug PoC, measure execution overhead)* bug finding *(fuzzing with Syzkaller, sanitizing with KASAN)* analyzing vulnerabilities with GDB *(CVE-2016-3903, CVE-2016-2501, CVE-2016-2061)* build driver exploit using GDB	Nexus 5X	✓	✓
Prospect [21]	performance impact on forwarding driver accesses using strace	324MHz embedded Linux MIPS with 16MiB RAM	✗	✗
	case study on proprietary fire alarm system *(network fuzzing)*	not disclosed		
Surrogates [22]	measure performance impact of MMIO forwarding	Pico Computing E17FX70T custom JTAG adapter board custom JTAG breakout / debug board FriendlyARM Mini2440	✗	✗
Inception [12]	measure vulnerability detection via synthetic tests *(Klocwork Test Suite)*		✓	✓
	validation tests *(53200 tests)*	LPC1850-DB1 & STM32 L152RE		
	comparison with binary-only approaches measure timing overhead *(Dhrystone benchmark and real world applications)* compare recovering semantic from a binary to the source code with libopencm3 security flaw detection with Juliet Test Suite 1.3 on FreeRTOS	Xilinx ZedBoard FPGA STM32 L152RE		
	analysis of products during development phase *(bootloader, chip SDK, payment terminal)*	not disclosed	✗	✗
Mousse [24]	performance evaluation measure coverage bugs and vulnerabilities research	Pixel 3 Nexus 5X Nexus 5	✓	partially
Pretender [18]	generate models for hardware peripherals *(record, build and emulate)*	STM32 L152RE STM32 F072RB Maxim MAX32600MBED	✓	✓
Conware [41]	generate models for hardware peripherals *(record, build and emulate)*	Arduino Due (Atmel SMART SAM3X/A)	✓	✓
Frankenstein [38]	heap overflow in device inquiry *(CVE-2019-11516)* heap overflow in the reception of BLE PDUs *(CVE-2019-13916)* heap overflow on ACL packets buffer *(CVE-2019-18614)*	CYW20735 CYW20819	✓	✓
FirmCorn [17]	accuracy between virtual execution approaches efficiency *(benchmark nbench)* stability effectiveness	DLink (DIR-816, DIR-629, DIR-859, DIR-823G) TPLink (WR940N, WR941N) Ezviz C6CDahua (HFW5238M, HFW3236M)	partially	partially
Incision [43]	correctness *(control flow extraction, region inference, database improvement and error correction)* real-world usability *(emulate Renault BCM, analysis the cryptography of Huawei R216h)* human effort *(qualitative measure of complexity of manual intervention in database correction)*	Huawei LTE R216h (ARMv7) Renault BCM (Renesas V850ES)	✓	✗

[1] The availability of the devices has been checked on google.com, amazon.com, digikey.com and ebay.com at the date 2021/12/15.

wants to support. An example of such case is the Linaro Automated Validation Architecture (LAVA) [23], which aims to test deployments on Linux-based systems for the ARM architecture.

3. Survey

In this section we survey papers related to embedded device security testing, using HIL, and inspect their artifacts to estimate their reproducibility. We will also discuss which experiments we integrate in the initial BEERR setup. Table 1 reports all experiments from the surveyed papers while table 2 presents a summary of their artifacts status.

Muench et al. [30] address the state of memory corruptions in embedded devices and the lack of mechanisms to mitigate silent memory corruptions. For this purpose they insert multiple vulnerabilities with independent trigger conditions on different class of embedded systems. They observe different behavior ranging from crash, reboot, hang, malfunctioning to no effect. They propose mitigations against these vulnerabilities and measure their performance costs on a fuzz testing. It is these last experiments which are available to reproduce.

Emulation facilitates the use of generic dynamic analysis techniques on firmware, but suffers from limited support in device emulation. Therefore, many publications explore the idea of dynamically forwarding I/O operations to the physical device to improve emulation.

Avatar² [29] is a framework written in Python which aims to facilitate the interoperability between different dynamic binary analysis tools. In particular, it offers the power to use HIL techniques to plug devices into an emulator with the help of a debugger. Three use cases are presented within the publication. The first experiment reproduces the analysis of the HARVEY rootkit, while the second shows the ability to move firmware execution state between concrete and symbolic execution modes. The third experiment demonstrates the capabilities of avatar² to forward peripherals accesses on the physical device from an emulator. This helps recording traces to replay and analyze them later without the device. We decide to focus

TABLE 2. Artifacts status in hardware in the loop papers surveyed
📄: SOURCE CODE 📦: CONTAINER 🐧: VIRTUAL MACHINE

Publication	Artifacts Packaging		Hardware	Link
	Tool	Dataset		
Wycinwyc [30]	📄 🐧	📄 🐧	STM32 L152RE	https://github.com/avatartwo/ndss18_wycinwyc
Avatar² [29]	📄 📦	📄 🐧	PLC Allen Bradley 1769-L16ER-BB1B STM32 L152RE	https://github.com/avatartwo/bar18_avatar2
Avatar² examples	N/A	📄	STM32 L152RE nRF51-DK	https://github.com/avatartwo/avatar2-examples
Avatar [50]	📄 📦 🐧	📄	Seagate ST3320413AS HDD Econotag development board Motorola C118	https://github.com/avatarone
Charm [42]	📄	📄	Nexus 5X	https://trusslab.github.io/charm
Inception [12]	📄 📦	📄	Xilinx ZedBoard FPGA STM32 L152RE LPC1850-DB1	https://github.com/Inception-framework
Mousse [24]	📄	📄	Pixel 3	https://github.com/trusslab/mousse
Pretender [18]	📄	📄	STM32 L152RE STM32 F072RB Maxim MAX32600MBED	https://github.com/ucsb-seclab/pretender
Conware [41]	📄	📄	Arduino Due	https://github.com/ucsb-seclab/conware
Frankenstein [38]	📄	📄	CYW20735 partially CYW20819	https://github.com/seemoo-lab/frankenstein
FirmCorn [17]	📄	📄	DLink (DIR-816, DIR-629, DIR-859, DIR-823G) TPLink (WR940N, WR941N) Ezviz C6CDahua (HFW5238M, HFW3236M)	https://github.com/FIRMCORN-Fuzzing/FIRMCORN
Incision [43]			Huawei LTE R216h (ARMv7) Renault BCM (Renesas V850ES)	https://github.com/UoBAutoSec/INCISION

on the first and third experiments because they operate on physical devices.

As an ancestor of avatar², Avatar [50] shares a similar objective and characteristics. Experiments gather around three case studies: backdoor detection in a masked ROM bootloader from a hard drive, vulnerability research in a commercial Zigbee device, the Econotag and helping reverse engineering the GSM stack of a Motorola C118 phone.

Charm [42] focuses on device drivers for smartphones. It claims to support four different device drivers on different smartphones: camera and audio for LG Nexus 5X, GPU for Huawei Nexus 6P and IMU sensors for Samsung Galaxy S7. Experiments try to answer several questions on its feasibility, performance and capability to perform dynamic analysis techniques such as interactive debugging, fuzzing and record-and-replay on a Nexus 5X smartphone.

Prospect [21] targets embedded Linux systems by intercepting accesses to character devices in the filesystem and forwarding them to the physical device. The performance of the system is evaluated against an unknown 324 MHz embedded Linux MIPS system with 16MiB RAM using `strace`. In addition an undisclosed proprietary fire alarm system is fuzzed as a security audit. The source code of the Prospect has not been made public.

Surrogates [22] leverages specialized hardware to enable low latency communication between the emulator and the system under test. It uses a custom FPGA to bridge the devices's JTAG interface to the host's PCI Express bus. The implementation uses a Pico Computing E17FX70T with Xilinx Virtex5 FX70T FPGA because of its included ready-to-use PCI Express card. Unfortunately this product is not commercially available anymore. The experiments measure Surrogates' performance impact on MMIO forwarding and its ease to port it to two new target devices.

Inception [12] introduces symbolic execution to embedded systems. Similarly to Surrogates, it includes a FPGA based debugger to provide high speed and low latency access to peripherals. But it differs in interfacing itself with the host via USB3 instead of PCI Express. Experiments focus on validation of the design, benchmarking the performance, vulnerability detection and several use cases on proprietary systems.

Mousse [24] brings selective symbolic execution to environments that are too difficult to emulate because of specific hardware. The proposed system is evaluated around three aspect: performance, code coverage and vulnerability discovery; and against three smartphones: Pixel 3, Nexus 5X and Nexus 5.

Pretender [18] and Conware [41] focus on the challenges to automatically model hardware peripherals to enable better firmware emulation. Both follow a similar logic: first record traces of peripheral interactions, then use these traces to generate a model, and finally plug the model into an emulator to allow the firmware to execute. Their contributions differ in the way of modeling the peripheral behavior from a recorded trace. Pretender uses machine learning while Conware employs automata representations. Both firmware datasets focus on 32-bit ARM Cortex-M processor with a wide range of peripherals (timer, button, GPIO, I2C, UART, radio, etc).

Frankenstein [38] leverages emulation with HIL to fuzz wireless firmware, mainly for Bluetooth and WiFi. The framework is used to discover three heap overflow vulnerabilities in implementations of the Bluetooth standard using the CYW20735 evaluation board.

FirmCorn [17] is a framework to fuzz IoT firmware. A collection of different firmware contexts are captured from the physical devices to be used as starting point for the fuzzing phase. Experiments target seven routers and three cameras. Multiple aspects of proposed system are evaluated such as accuracy, efficiency, stability and effectiveness.

Incision [43] tackles the challenge of combining static with dynamic analysis to help with the task of reverse engineering complex embedded systems. Execution traces are recorded in order to improve the static firmware analysis. The evaluation targets two physical devices, a LTE baseband unit and an automotive Body Control Module. The artifacts are not available. But at the time of writing, the authors plan to reimplement the source code with similar functionalities in a new open source framework called *Fugue*[1].

We observe that most publications release the source code of their proposed system and collected dataset. However, packaging their artifacts within a container or virtual machine images may improve their usability. In addition, scripts used to process generated data and plot figures for papers are rarely shared within the artifacts.

It is worth noting an initiative has been created to collect monolithic firmware used in publications (*https://github.com/ucsb-seclab/monolithic-firmware-collection*).

4. BEERR Overview

We now describe a high level overview of the proposed testbed for enabling reproducible research on dynamic security analysis with embedded system.

4.1. BEERR Objective

BEERR aims to help researchers with their experiments by offering an access to environments where dynamic security analysis can be performed on embedded systems. The devices and their experiments are mainly selected from scientific publications.

The infrastructure can be used for several purposes. First, it makes a variety of embedded systems available remotely with their physical components. Secondly, it helps researchers to reproduce diverse experiments from published papers. It removes the effort and responsibility to setup complex bleeding edge prototypes. Therefore, we believe this would improve the state of fair comparison between systems. Thirdly, researchers are free to modify existing experiments to better study influence of various settings and improvements. The platform can also be used by binary analysis projects to perform continuous integration with hardware in the loop.

4.2. BEERR Architecture

BEERR is divided in two parts. The first part is called the front node. It hosts the website with the scheduler and stores the experiment codes and data. The website lets the user register an account and submit tasks to the scheduler. The latter takes care of allocating and setting up the resources as well as opening the connection to the selected experiment node. Code and data for experiments are combined in portable container images which are stored in a local registry.

The second part is composed of independent experiment nodes. Each of these nodes holds a gateway which is the main access to the experiment node for the user. From it, the user can interact with the available devices, upload files, download container image from the local registry and run its analysis. Other components such as debuggers and power switches help in controlling the device state under study.

The initial design was strongly inspired from existing testbed, in particular R2Lab [34] and FIT IoT-LAB [9], because they also target embedded systems. However, the type of analysis we want to perform is very different. Unlike R2Lab with its anechoic chamber, we don't aim to study radio propagation across different nodes in a controlled environment. Similarly, in contrary to FIT IoT-LAB, there is no need to work with sensor networks, routing protocols or distributed applications on distinct typologies. As shown in the survey (Section 3), the analysis we target focuses on the firmware and code closely coupled with low level hardware rather than across a pool of different devices. It may require powerful computing resources to handle a partial emulation of the system and its analysis. That is the reason we made the trade-offs to divide the testbed in independent bookable experiment nodes with a more or less powerful gateway. This design still offers the possibility to build systems composed of multiple devices behind a single gateway.

4.3. User Workflow

The typical workflow for the user first involves creating an account on the website and submitting its SSH public key. Then the user has to wait for the validation of his account from the person in charge of its affiliation. To create such group of users (e.g., per university or company), an application explaining their motivations has to be sent to the administrators. The access is free of charge and mainly target scientific activities.

The creation of an experiment requires to select an experiment node, select a time slot and a duration, select an image to boot the gateway and optionally fill a public link to a git repository. This repository is a way for the user to prepare the experiment code upstream its time slot. The repository may contain a `Dockerfile` which would be build and stored in the local registry.

When the booked time comes, the gateway is powered with the selected image and an SSH connection is opened between the front node and the gateway. The user can use this access to start a shell on the bare-metal gateway with root permissions. He has the possibility to upload files, install and configure the gateway as needed. No direct Internet access is provided on the gateway from our infrastructure, users can use SSH tunneling to share their own Internet connection.

Finally, the front node closes the SSH connection, resets the devices, cleans up the gateway disk and powers them off. Status of previous experiments is displayed on the website.

5. BEERR Implementation

5.1. Front Node

The front node hosts the website and the database with user and experiment data such as credentials, SSH keys, affiliation, experiment scheduled time, repository link. The scheduler is integrated into the back end via the *Advanced Python Scheduler library*.

[1] https://github.com/UoBAutoSec/INCISION

A local docker registry stores the experiments under the form of docker images. They are built locally from the git repository link provided by the user on experiment submission.

The gateways boot on the network to facilitate the image selection and management. We use *dnsmasq* because of its advantage to offer a complete and lightweight solution with DNS caching, DHCP and TFTP servers and its ease of configuration. The root filesystem is mounted using an union filesystem, OverlayFS. The read-only bottom layer is on an NFS server on the front node while the read-write upper layer is on the local disk. In this way, images are easily added and updated by administrators while users have the possibility to modify the system. Modifications are cleaned at the end of the experiment by unlinking files.

5.2. Experiment Nodes

Experiments are assembled in individual container images. We chose this solution because it facilitates the packaging, sharing, setup and resetting of experiments and parts of their environment. The essential advantage to use precise software versions from the time of publication offers the possibility to circumvent the hard task of maintaining experiments in different environments. A corner case occurs when a special kernel version is required. In this case, because the user has a root shell access on the bare-metal gateway, he is free to use a virtual machine. Nevertheless, we did not observe such situation in our survey (Section 3).

BEERR provides a set of pre-built images in the local registry. This allows the user to use them directly or use them as a base to build new images.

The first set of experiments we propose to reproduce are from the paper *Avatar²: A Multi-target Orchestration Platform* [29]. The second set targets the paper *What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices* [30] (see Section 3). In addition, we make available the already existing avatar² examples [1].

The experiments have been grouped in categories of nodes according to their nature and the devices they use. A node is composed of a gateway where the analysis is executed, and one or multiple devices which are subjects of the analysis. The user has a root shell access on the bare-metal gateway where he can freely interact with the devices, build and run container experiments.

The gateways are either a Raspberry Pi 4 Model B 4GB or an Intel NUC BOXNUC8I5BELS1 with CPU Intel Core i5-8260U (Quad-Core 1.6/3.9 GHz, 8 threads, 6M cache), 16GB DDR4, 250Go NVMe SSD. The Raspberry Pi has the advantage to be relatively cheap, space and power efficient. Yet its processor architecture is based on the ARM instruction set, which might cause compatibility problems with some experiments. Indeed, most research experiments work in a limited environment where portability is not always a main objective. For example, the PANDA emulator only supports host machines with x86_64 architecture. In this situation, the experiment needs to be carry out within the second node type using Intel NUC computers. We therefore use two categories of experiment nodes:

- **Alpha nodes**. The Alpha nodes are designed as a low cost entry node. They also allow new users to familiarize themselves with the environment through a set of basic experiences. Each are composed of a Raspberry Pi 4 with Nucleo boards and an USB switch.
- **Bravo nodes**. The Bravo nodes are more powerful and have for theme rehosting with powerful emulation. Those experiments rely on an Intel NUC (instead of raspberry PI) to be able to run more complex experiments, e.g., those that rely on the PANDA emulator.

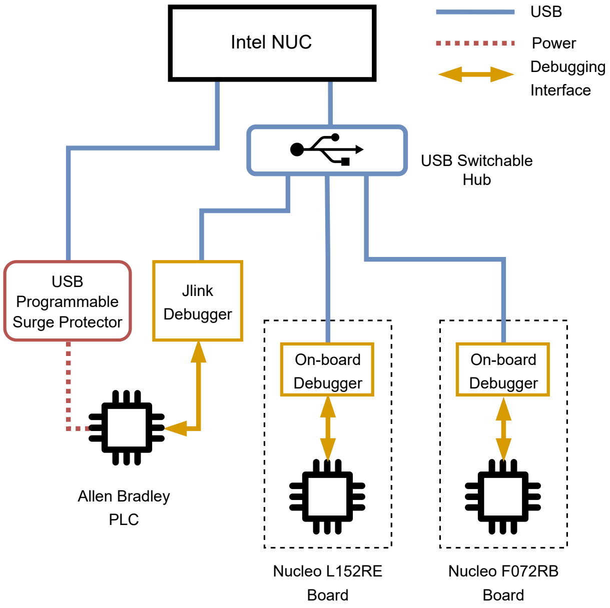

Figure 1. An example layout for a Bravo node

5.3. Devices Overview

The following devices are available on the nodes. We refer to Table 2 in Section 3 to show which experiments they allow to reproduce.

- STM32 Nucleo L152RE (ARM Cortex-M3) and F072RB (ARM Cortex-M0) boards,
- Nordic semiconductor nRF51-DK for BLE (ARM Cortex-M3),
- Cypress CYW920735Q60EVB-01 BLE Evaluation Kit (SoC CYW20735 with ARM Cortex-M4),
- Allen Bradley 1769-L16ER-BB1B CompactLogix

In addition, other components are present to control the state of devices:

- Yepkit YKUSH, an USB Switchable Hub to turn on and off an USB link,
- an USB programmable surge protector to power devices using power plug,
- SEGGER J-Link Debug Probes to interface with JTAG ports

6. Discussion

6.1. Infrastructure Security

BEERR is mainly intend for the scientific community, without excluding other potential collaborations. The affiliation link makes the user accountable for its conduct. It is a best effort approach where we rely on good behavior from the users. We plan to treat malicious behavior on a case-by-case basis and terminate the corresponding user or affiliation accesses.

We nevertheless implemented minimal mechanisms to protect a fair access to the service and preserve the integrity of the system. Time spent using BEERR is divided in 55 minutes time slots and daily quotas to avoid monopolizing all resources. To preserve a clean state at the start of a session, we use an overlay filesystem with the read layer stored in the network while the modifications are written locally on disk. At the end of the booked session, files are unlinked from the disk.

We do not protect against any attack attending to modify the gateway bootloader. Systems under tests are also not protected against modifications. This is a desired behavior as the user might need to flash different firmware. However, nothing prevents from permanent modifications such as a blowing fuse. We have chosen in a first step to give freedom to the user before restricting its capabilities. We trust the user to do their best to not intentionally brick the devices. If users need specific requirements, we encourage them to contact us to discuss its feasibility. In the future as more expensive or hard to operate devices are incorporated, we might consider implementing group policy for access to categories of experimental nodes.

6.2. Future work

In the future, we plan to include more experiments from available publications and artifacts. In particular, we refer to Table 2. We want to first focus on experiments around firmware analysis and their interactions with hardware components. Then, we consider extending to larger type of analysis involving both hardware and software such as fault injection and side channel attacks [11].

We provide a largely unrestricted remote access to devices, but we do not currently offer a way to control and monitor their physical environment. However embedded systems typically interface with physical world via sensors and actuators. We cannot reproduce all realistic physical environments, but some preliminary steps are possible. For example, when physical interaction is needed on a specific device, such as pressing buttons, we will implement this using a relay box (e.g., [4]). Similarly a webcam can be added to be able to monitor the physical status of a device (such as LEDs, or screen status). Some experiments may use radio transceivers such as the nRF51-DK. Caution must be taken to avoid interference with the environment, both to prevent wrong measurements during the experiments and degrading other wireless systems. Isolation with Faraday shield can be used.

We also see the benefit in the future to include BEERR into continuous integration of research projects, notably avatar².

7. Conclusion

In this paper, we discussed the challenges and the significance of reproducible research in system security. Testbeds are a reliable way to tackle this topic. We highlighted the importance to reproduce dynamic analysis with embedded systems.

We presented a new testbed BEERR seeking to help researchers to reproduce experiments involving physical devices. For this purpose we collected experiment code and data from publications and integrated them into ready-to-use containers. The experiment hardware is set up together with control nodes. Those nodes are made remotely accessible via SSH and can be reserved with a booking system. Our goal is to provide researchers the possibility to have an easier access to research artifacts, and to perform better evaluations and comparisons with related work.

8. Acknowledgements

The author would like to thanks the anonymous reviewer for their feedback and interesting suggestions. This work was partially funded by a research contract with Siemens AG. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA8655-20-1-7048.

References

[1] avatar² examples repository. [Online]. Available: https://github.com/avatartwo/avatar2-examples

[2] Fed4fire+ website. [Online]. Available: https://www.fed4fire.eu/

[3] Sharing expertise and artifacts for reuse through cybersecurity community hub (searcch) project. [Online]. Available: https://searcch.cyberexperimentation.org/about

[4] Usb relay module 4 channels, for home automation - v2. [Online]. Available: https://denkovi.com/usb-relay-board-four-channels-for-home-automation-v2

[5] "Fuzzware: Using precise mmio modeling for effective firmware fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[6] ACM. Artifact review and badging - version 2.0. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-badging

[7] ACSAC. Paper artifacts. [Online]. Available: https://www.acsac.org/2020/submissions/papers/artifacts/

[8] S. Adepu, N. K. Kandasamy, and A. Mathur, "Epic: An electric power testbed for research and training in cyber physical systems security," in *Computer Security*, 2018.

[9] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "Fit iot-lab: A large scale open experimental iot testbed," 2015. [Online]. Available: https://hal.inria.fr/hal-01213938

[10] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, 2014.

[11] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, "Screaming channels: When electromagnetic side channels meet radio transceivers," in *Proceedings of the 25th ACM conference on Computer and communications security (CCS)*, 2018.

[12] N. Corteggiani, G. Camurati, and A. Francillon, "Inception: System-wide security testing of real-world embedded systems software," in *USENIX Security Symposium*, 2018.

[13] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan, “Repeatable reverse engineering with panda,” in *Program Protection and Reverse Engineering Workshop*, 2015.

[14] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, “The design and operation of cloudlab,” in *USENIX Annual Technical Conference (USENIX ATC 19)*, 2019.

[15] A. Fasano, T. Ballo, M. Muench, T. Leek, A. Bulekov, B. Dolan-Gavitt, M. Egele, A. Francillon, L. Lu, N. Gregory *et al.*, “Sok: Enabling security analyses of embedded systems via rehosting,” in *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2021.

[16] D. S. Fowler, M. Cheah, S. A. Shaikh, and J. Bryans, “Towards a testbed for automotive cybersecurity,” in *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2017.

[17] Z. Gui, H. Shu, F. Kang, and X. Xiong, “Firmcorn: Vulnerability-oriented fuzzing of iot firmware via optimized virtual execution,” *IEEE Access*, 2020.

[18] E. Gustafson, M. Muench, C. Spensky, N. Redini, A. Machiry, Y. Fratantonio, D. Balzarotti, A. Francillon, Y. R. Choe, C. Kruegel *et al.*, “Toward the analysis of embedded firmware through automated re-hosting,” in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019.

[19] A. Hussain, D. DeAngelis, E. Kline, and S. Schwab, “Replicated testbed experiments for the evaluation of a wide-range of ddos defenses,” in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, 2020.

[20] IEEE. Submitting a paper to tpds. [Online]. Available: https://www.computer.org/csdl/journal/td/write-for-us/15085

[21] M. Kammerstetter, C. Platzer, and W. Kastner, “Prospect: peripheral proxying supported embedded code testing,” in *ACM symposium on Information, computer and communications security (AsiaCCS)*, 2014.

[22] K. Koscher, T. Kohno, and D. Molnar, “SURROGATES: Enabling near-real-time dynamic analyses of embedded systems,” in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[23] Linaro. Linaro automated validation architecture (lava). [Online]. Available: https://validation.linaro.org/

[24] Y. Liu, H.-W. Hung, and A. A. Sani, “Mousse: a system for selective symbolic execution of programs with untamed environments,” in *Fifteenth European Conference on Computer Systems*, 2020.

[25] A. Massouri, L. Cardoso, B. Guillon, F. Hutu, G. Villemaud, T. Risset, and J.-M. Gorce, “Cortexlab: An open fpga-based facility for testing sdr amp; cognitive radio networks in a reproducible environment,” in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014.

[26] A. P. Mathur and N. O. Tippenhauer, “Swat: A water treatment testbed for research and training on ics security,” in *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*, 2016.

[27] J. Metzman, L. Szekeres, L. Maurice Romain Simon, R. Trevelin Sprabery, and A. Arya, “FuzzBench: An Open Fuzzer Benchmarking Platform and Service,” in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.

[28] S. Mocanu, M. Puys, and P.-H. Thevenon, “An Open-Source Hardware-In-The-Loop Virtualization System for Cybersecurity Studies of SCADA Systems,” in *C&esar 2019 - Virtualization and Cybersecurity*, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02371133

[29] M. Muench, D. Nisi, A. Francillon, and D. Balzarotti, “Avatar²: A multi-target orchestration platform,” in *Workshop on Binary Analysis Research (colocated with NDSS Symposium)(February 2018), BAR*, 2018.

[30] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, “What you corrupt is not what you crash: Challenges in fuzzing embedded devices.” in *NDSS*, 2018.

[31] NISO. NISO RP-31-2021 reproducibility badging and definitions. [Online]. Available: https://groups.niso.org/apps/group_public/download.php/24810/RP-31-2021_Reproducibility_Badging_and_Definitions.pdf

[32] P. S. Oruganti, M. Appel, and Q. Ahmed, “Hardware-in-loop based automotive embedded systems cybersecurity evaluation testbed,” in *ACM Workshop on Automotive Cybersecurity*, 2019.

[33] OSF. Center for open science badges. [Online]. Available: https://osf.io/tvyxz/wiki/1.%20View%20the%20Badges/

[34] T. Parmentelat, M. N. Mahfoudi, T. Turletti, and W. Dabbous, “A step towards runnable papers using r2lab,” 2019.

[35] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, “A blueprint for introducing disruptive technology into the internet,” *ACM SIGCOMM Computer Communication Review*, 2003.

[36] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, “The robotarium: A remotely accessible swarm robotics research testbed,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[37] Q. Qassim, N. Jamil, I. Z. Abidin, M. E. Rusli, S. Yussof, R. Ismail, F. Abdullah, N. Ja’afar, H. C. Hasan, and M. Daud, “A survey of scada testbed implementation approaches,” *Indian Journal of Science and Technology*, 2017.

[38] J. Ruge, J. Classen, F. Gringoli, and M. Hollick, “Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets,” in *USENIX Security Symposium*, 2020.

[39] S. Siboni, V. Sachidananda, Y. Meidan, M. Bohadana, Y. Mathov, S. Bhairav, A. Shabtai, and Y. Elovici, “Security testbed for internet-of-things devices,” *IEEE transactions on reliability*, 2019.

[40] M. Simioni, P. Gladyshev, B. Habibnia, and P. R. N. de Souza, “Monitoring an anonymity network: Toward the deanonymization of hidden services,” *Digital Investigation*, 2021.

[41] C. Spensky, A. Machiry, N. Redini, C. Unger, G. Foster, E. Blasband, H. Okhravi, C. Kruegel, and G. Vigna, “Conware: Automated modeling of hardware peripherals,” in *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2021.

[42] S. M. S. Talebi, H. Tavakoli, H. Zhang, Z. Zhang, A. A. Sani, and Z. Qian, “Charm: Facilitating dynamic analysis of device drivers of mobile systems,” in *USENIX Security Symposium*, 2018.

[43] S. L. Thomas, J. Van den Herrewegen, G. Vasilakis, Z. Chen, M. Ordean, and F. D. Garcia, “Cutting through the complexity of reverse engineering embedded devices,” in *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021.

[44] USENIX. Usenix security ’20 artifact evaluation information. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/artifact-evaluation-information

[45] ——. Usenix security ’22 call for artifacts. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/call-for-artifacts

[46] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, “An integrated experimental environment for distributed systems and networks,” in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, 2002.

[47] WOOT. Woot ’19 artifact evaluation information. [Online]. Available: https://www.usenix.org/conference/woot19/artifact-evaluation-information

[48] C. Wright, W. A. Moeglein, S. Bagchi, M. Kulkarni, and A. A. Clements, “Challenges in firmware re-hosting, emulation, and analysis,” *ACM Computing Surveys (CSUR)*, 2021.

[49] M. M. Yamin, B. Katt, and V. Gkioulos, “Cyber ranges and security testbeds: Scenarios, functions, tools and architecture,” *Computers & Security*, 2020.

[50] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, “Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems’ Firmwares,” in *Network and Distributed System Security (NDSS) Symposium*, 2014.