0x41414141: Avatar² Artifacts, Advances & Analysis

Paul L. R. Olivier
LAAS CNRS
Toulouse, France
olivier.paul.lr@gmail.com

Marius Muench
University of Birmingham
Birmingham, UK
m.muench@bham.ac.uk

Aurélien Francillon
EURECOM
Sophia Antipolis, France
aurelien.francillon@eurecom.fr

Abstract

Avatar² is an open-source orchestration framework with a focus on dynamic analysis of embedded devices' firmware. Since its release in 2017, *avatar²* has made significant contributions to the field of embedded systems security research. In particular, it has facilitated numerous studies and tools in areas of automated re-hosting, firmware fuzzing, and vulnerability discovery. This short paper presents an overview of *avatar²*'s key features, its life as a project, and its impact both on academic research and beyond.

The framework is publicly hosted under the Apache-2 License on GitHub at https://github.com/avatartwo.

Keywords

Binary Analysis, Vulnerability Discovery

ACM Reference Format:
Paul L. R. Olivier, Marius Muench, and Aurélien Francillon. 2024. 0x41414141: Avatar² Artifacts, Advances & Analysis. In *Proceedings of ACSAC's Cybersecurity Artifacts Competition and Impact Award (ACSAC 2024).* ACM, New York, NY, USA, 5 pages.

1 Introduction

The *avatar²* framework [28] is the worthy successor of AVATAR [41]. It is developed to facilitate the *integration* and *interoperability* between various binary analysis tools such as debuggers, emulators, disassemblers, symbolic execution engines, and fuzzers.

The framework has a particular focus on dynamic binary-only analysis. This is an essential method for ensuring the security, compliance, and reverse engineering of software. For example, it offers valuable insights into the final product delivered to clients when used at the end of the Software Development Life Cycle (SDLC). Binary-only analysis includes all the dependencies, such as third-party libraries, without relying on assumptions about the compiler used. Moreover, the developed analysis brings the benefit of being independent of the source code language. Another area where binary-only analysis becomes crucial is situations where source code is unavailable. This is a common scenario for studying existing software as part of reverse engineering or conducting in-depth security research on malware, backdoors, and vulnerability detection. Debugging at the binary level allows developers to diagnose and fix issues in software when source code access is restricted or impractical, as could be the case for legacy software on outdated or specialized hardware platforms.

Binary-only analysis includes a wide range of static and dynamic techniques. Typically, the process starts with disassembling and decompiling. This allows the conversion of the binary code into assembly or higher-level language that approximates the original source code. More powerful static analysis can then take place to better understand the program structure, such as signature and function identification or binary diffing. Beyond these, dynamic analysis observes runtime execution. For example, patching the binary program helps to modify the code's behavior. Profiling and instrumentation insert hooks to monitor execution. This helps in understanding the program's performance and interaction with its environment, including file systems, networks, and system calls. More advanced techniques include fuzzing, which tests the binary with unexpected inputs; symbolic and concolic execution, which explore multiple execution paths simultaneously; and taint analysis, which tracks the flow of tainted data through the program.

It has often been a challenge to extend and improve the use of these tools and techniques to more diverse and heterogeneous targets. Embedded systems represent a typical case [29]. They offer unique challenges due to their constrained resources, specialized hardware, and often proprietary software environments. These factors make the analysis more complex and necessitate adaptable tools capable of interfacing with a wide range of architectures and operating systems. Traditional analysis tools are typically designed with general-purpose computing environments in mind. They may struggle to cope with the particularities of embedded systems.

The importance of combining state-of-the-art techniques was illustrated during the DARPA Cyber Grand Challenge (CGC) in 2016. In this high-stakes competition, teams were tasked with autonomously identifying vulnerabilities in complex software systems. To tackle this, many teams opted to combine fuzzing with symbolic execution [5, 15, 31, 36]. The two techniques offer a powerful approach to vulnerability detection as it is recognized today. Fuzzing excels at quickly generating large volumes of test cases to explore a program's behavior, but may have difficulties finding specific inputs to uncover new execution paths. In contrast, symbolic execution leverages the program structure to explore different execution paths but suffers from an exponential path explosion problem, limiting its scalability. The competition revealed diverse approaches to how teams integrated these techniques together. This richness underscores the value of improving integration and interoperability to craft tailored solutions to a specific problem.

However, the integration of multiple tools into a cohesive workflow brings its own set of challenges. One of the primary difficulties lies in managing these tools effectively: deciding how and when to switch from one environment to another and ensuring that the tools communicate with each other in a meaningful way. This requires a robust system for orchestrating the various tools and ensuring that they operate in concert rather than in isolation.

Avatar² addresses these challenges by providing a unified platform facilitating the integration of diverse tools and offering the necessary control mechanisms to manage their interactions.

ACSAC 2024, December 09–13, 2024, Waikiki, Hawaii, USA
2024.

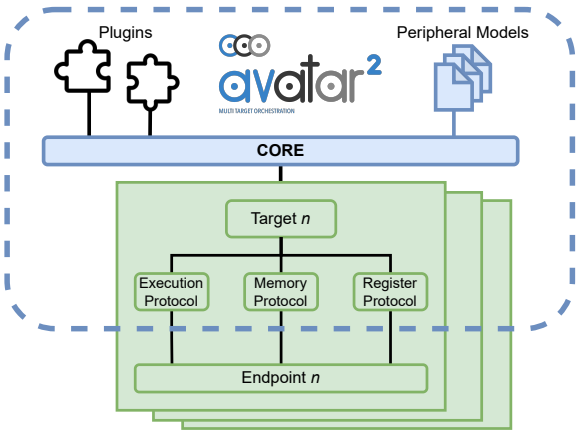

Figure 1: *Avatar²* Overview. The core of the framework orchestrates multiple *targets*, which are Python abstractions of binary analysis tools, physical devices, or emulators. The targets utilize different unified *protocols* to communicate with their respective endpoint (i.e., tool, device, or emulator).

2 History

The design principles of *avatar²* are rooted in, and improving upon, the first iteration of AVATAR [41] published in 2014. AVATAR enables partial emulation of firmware by integrating an emulator with physical hardware. The main dynamic analysis technique was selective symbolic execution with S²E [6]. Following are the key concepts introduced by its predecessor:

- **Target Orchestration.** The ability to control the program execution over multiple targets (S²E and the physical device) is crucial for partial emulation. This also allows the framework to automatically respond to specific events triggered by the environment (e.g., interruptions) or defined by the analyst.
- **State transfer and synchronization.** Transferring execution states (i.e., CPU register, memory content) across different analysis environments allows for greater flexibility in dynamic analysis. This is particularly valuable for scenarios where specific stages of firmware execution must occur on the physical device to accurately capture its behavior. For instance, during firmware initialization, many hardware-specific interactions take place, making it advantageous to start the firmware on the actual hardware.
- **Separation of Execution and Memory.** This allows to use *remote memory* where the execution can proceed on one target while some memory accesses are handled by another. A common application is when the main execution occurs in an emulator, but accesses to memory-mapped peripherals are forwarded to the actual hardware.

While AVATAR laid the groundwork, *avatar²* expands upon and generalizes these concepts, enhancing usability and applicability across a broader range of scenarios.

3 *Avatar²*: Concepts and Design

Avatar² is built around four main components as illustrated by Figure 1: core, targets, protocols, and endpoints.

- **Core.** The core serves as the main interface for the analyst, orchestrating the interaction between different components and responding to various events. It acts as the central control unit, managing the overall analysis process.
- **Targets.** These represent abstracted interfaces to different analysis tools (endpoints). They do not directly communicate with the tools but instead rely on protocols to manage the interactions. This abstraction allows *avatar²* to support a wide range of tools and easily integrate new ones.
- **Protocols.** These are responsible for handling specific types of interactions, such as memory operations, execution control, and register access. By decoupling the protocols from the targets, *avatar²* allows for the reuse of communication patterns across different tools, making it easier to prototype new targets and integrate additional analysis environments.
- **Endpoints.** Endpoints are the actual tools or devices being controlled by *avatar²*. These can be emulators, debuggers, or physical devices, which are orchestrated together to perform complex analysis tasks.

Moreover, *avatar²* offers unique features that make dynamic analysis easier on embedded systems by being adaptable to their unique characteristics.

- **Architecture Independence.** *avatar²* is designed to support multiple architectures, making it adaptable to the diverse range of instruction sets found in software and embedded systems. It uses a modular approach to describe architectures, which allows for easy extension to new architectures.
- **Memory Orchestration.** To synchronize analysis across different platforms, *avatar²* provides a consistent internal memory representation. Unlike other tools that use page-granularity memory representations, *avatar²* can handle memory ranges of arbitrary sizes, which is particularly useful for embedded devices with memory-mapped peripherals.
- **Peripheral Modeling.** *Avatar²* allows the modeling of custom peripherals that might not be implemented or representable in endpoints. This feature allows analysts to create simple models in Python that can respond to memory reads and writes, simulating the behavior of hardware peripherals.
- **Plugin System.** The framework's minimalistic core is complemented by a rich event-driven plugin system that automates repetitive tasks and extends functionality. Plugins can hook into various events during analysis, enabling custom callbacks or adding new features to the framework.
- **Configurable Machine.** Although not directly integrated into *avatar²*, the Configurable Machine is a custom full-system QEMU machine developed as part of the avatar² project. It combines QEMU's high-speed emulation with a flexible platform (memory layout, cores, peripherals, etc) configuration system. This design pairs well with the peripheral modeling and the hardware I/O forwarding.

4 Project Life

Avatar² was initially released in June 2017, with a publication describing the framework only following later in 2018 [28]. This initial release comprised multiple repositories, including the QEMU fork

that implements the Configurable Machine and examples showcasing typical usage of the framework. Additional repositories, including artifact releases for publications leveraging the framework, were added over time. As of September 2024, the main repository of *avatar²* had four additional minor releases and received 387 commits spanning 27 different contributors.

The project initially supports ARM 32-bit, x86, and x86_64 architectures, later received additional MIPS 32-bit support, and external forks include aarch64 support (e.g., [21, 22]). The number of compatible targets has doubled since the launch of the framework. Initially supporting four targets (GDB, OpenOCD, QEMU, and PANDA), we have expanded the framework to include targets for the JLink debugger, Inception debugger [8], Unicorn emulator and PyPanda emulator [10]. All of these implement abstractions required to interact with a specific endpoint, and we observed that third-party tools often borrow and adapt these abstractions to fit their specific needs [7, 14, 16].

Beyond academic publication expanding *avatar²*, covered in the next Section, three master's theses were carried out in close collaboration with the core *avatar²* development team. AFLtar [3] combines *avatar²* with AFL and AFL-Unicorn to enable coverage-guided fuzzing of devices with complex peripherals. Visser [19] provides a comprehensive survey and analysis of automated firmware re-hosting approaches for security analysis of embedded systems. Albrecht [1] extends *avatar²* with new plugins to enable hardware-in-the-loop rehosting of asynchronous, interrupt-driven firmware using complex peripherals like CAN and WiFi controllers.

Examples. The initial release of *avatar²* included two examples: a fully virtualized orchestration showcasing the concept of remote memory and a hardware-in-the-loop setup demonstrating state transfer. Later, we released three further examples with the artifact of *avatar²*'s main publication: a reimplementation of a PLC rootkit, a concolic execution analysis for finding a vulnerability in the Firefox browser, and an orchestration setup for tracing MMIO interactions with a physical device and later replaying it against the emulated firmware without the device attached[1].

To further ease usage of the framework, we later developed and publicly released three further example analysis projects targeting additional widely deployed embedded systems, such as a Fitbit watch, an nRF52 Bluetooth chip, and the Raspberry Pi Pico[2].

5 Impact

5.1 Academic Impact

As of September 2024, the papers for *avatar²* [28] and AVATAR [41] have been cited 172 and 462 times, respectively, according to Google Scholar. Besides this, and more importantly, the *avatar²* framework has been the foundation for several influential extensions and research projects. The main use cases of *avatar²* have been in different areas of firmware re-hosting [13], vulnerability discovery, research replication, and reproduction, as well as enabling advanced dynamic analysis of embedded system's firmware. Table 1 highlights the components of *avatar²* utilized by later work, and we discuss the different areas of applications in the following.

[1] https://github.com/avatartwo/bar18_avatar2.
[2] https://github.com/avatartwo/avatar2-examples.

Peripheral Modeling based Rehosting. Pretender [17] builds on *avatar²* to address the challenges of firmware re-hosting [13] by automatically creating interactive, stateful models of hardware peripherals from recorded I/O interactions. Conware [38] extends this approach, further abstracting these models to enable pluggable and combinable peripheral emulation across different firmware and hardware configurations. Both approaches leverage *avatar²*'s peripheral modeling mechanisms extensively. Laelaps [4] utilizes *avatar²*'s capabilities to run symbolic execution on firmware, inferring the expected behavior of peripherals from the firmware itself. DICE [27] enables firmware analyzers to emulate and manipulate Direct Memory Access (DMA) inputs without needing actual hardware or firmware modifications.

Abstraction based Rehosting. HALucinator [7] leverages *avatar²* to decouple firmware from its hardware dependencies using High-Level Emulation (HLE). It enables dynamic analysis by emulating Hardware Abstraction Layers (HALs) and identifying vulnerabilities in firmware without requiring specific hardware. SYMBION [16] combines symbolic and concrete execution to efficiently analyze programs. It applies symbolic execution to targeted code sections while relying on concrete execution for the rest of the program's interactions with its environment. It reemploys the notions of the target from *avatar²* for orchestrating between its concrete and symbolic execution contexts. SyncEmu [22] focuses on re-hosting the Trusted Applications (TA) running on TEEs using *avatar²*'s Configurable Machine.

Fuzzing. WYCINWYC [29] studied the challenges encountered during fuzzing embedded systems and used *avatar²* to highlight the advantages of rehosting-based approaches. Unicorefuzz [25] extends *avatar²* by utilizing CPU emulation to fuzz kernel components and device drivers in environments where traditional userland fuzzing is impractical. Fuzzware [35] automates and improves fuzz testing of embedded device firmware. It uses *avatar²* for peripheral modeling, allowing execution of additional dynamic analysis on the re-hosted firmware. Firmwire [18] adapts *avatar²* for the analysis of baseband processors in smartphones. It offers a scalable full-system emulation and fuzzing platform focused on discovering vulnerabilities in cellular protocols. FirmHybridFuzzer [37] extends Laelaps and *avatar²* with hybrid fuzzing to uncover vulnerabilities in microcontroller firmware. Sizzler [14] relies on *avatar²* to manage the emulation environment for fuzzing Programmable Logic Controllers (PLC). EL3XIR [21] extends *avatar²* to enable effective re-hosting and fuzzing of the secure monitor firmware (EL3) in TrustZone-based Trusted Execution Environments (TEEs).

Overall, publications leveraging *avatar²* reported identifying various CVEs [4, 7, 14, 18, 21, 35], with the majority classified as memory corruption vulnerabilities.

Facilitating Research and Reproducability. The configurable machine has been reused and extended by several other frameworks [10, 11, 21], and we observed that *avatar²* helps to build systems allowing reproducible experiments as illustrated in [2, 32]. Moreover, *avatar²* has been used in qualitative [8, 9, 12, 16, 39, 40, 42] and quantitative [20, 23, 24] comparisons.

Publication	Year	Target Orchestration	State Transfer	Peripheral Modeling	Configurable Machine
WYCINWYC [29]	2018	✔	✔	✔	✔
Unicorefuzz [25]	2019	✔	–	–	–
Pretender [17]	2019	–	–	✔	–
HALucinator [7]	2020	✔	–	–	–
Laelaps [4]	2020	–	✔	–	–
SYMBION [16]	2020	✔	–	–	–
Conware [38]	2021	–	–	✔	–
DICE [27]	2021	–	–	✔	–
PyPANDA [10]	2021	–	–	–	✔
Fuzzware [35]	2022	–	–	✔	–
Firmwire [18]	2022	–	–	✔	✔
FirmHybridFuzzer [37]	2023	–	–	✔	✔
Sizzler [14]	2023	✔	–	–	✔
EL3XIR [21]	2024	–	–	✔	✔
SyncEmu [22]	2024	–	–	✔	✔

Table 1: *Avatar²*'s features usage accross publications.

5.2 Impact beyond Academia

AVATAR originated from a research contract with Google, with subsequent funding for *avatar²* provided by Siemens AG - Technology. By now, multiple companies confidentially reported the use of the framework or its derivatives for internal purposes. As of September 2024, the main repository of *avatar²* accumulated 518 stars and 98 forks, showcasing the wider community created by the framework.

Additionally, *avatar²* has been discussed by third parties (e.g., [33]) and is featured in an educative book on vulnerability research with emulated IoT devices [30]. Parts of *avatar²* are further included in a large-scale source code dataset maintained by Hugging Face[3].

6 Discussion

6.1 Lessons Learned

Designing and maintaining *avatar²* over half a decade provided us with multiple insights:

(1) **Value of Software Design Practices.** We believe that one driving factor for *avatar²*'s adoption was the design decisions made early on in the project's life. While, in our experience, a lot of system security research code grows organically, we dedicated a considerable amount of time to creating, discussing, and revising the initial design of *avatar²*, based on insights from the original AVATAR. These design decisions, leading to clean abstractions and extensability, benefit the framework up to this day.

(2) **Orchestration and Integration with other frameworks.** A key benefit of *avatar²* was to allow flexible orchestration of other frameworks. This idea resonated with the community. However, when integrated with other frameworks, it was not always clear which one should take over the core orchestration tasks. While this allowed different communities to work with *avatar²*, it also led to code duplication and fragmentation. We believe that better synchronization could lead to better synergies between different frameworks.

(3) **Packaging Artifacts.** When we released the initial version of *avatar²*, artifact evaluation possibilities were not widely deployed yet, with ACSAC being one of the exceptions. Despite this, we put additional focus on making the experiments of *avatar²* replicable, with a separate experiments repository, basic documentation, and a pre-packaged environment for running the framework. We believe this tremendously increased the impact of our work, as it allowed other teams to build on top of the provided examples and documentation. Thus, we highly appreciate the shift in the community toward more widely used artifact evaluations but would, regardless, recommend packaging artifacts in replicable environments (e.g., via containers or virtual machines).

[3]https://huggingface.co/datasets/Samip/Scotch/viewer

6.2 Outlook & Future Work

We plan to continue maintaining the *avatar²* framework in the future. In the short-term, we aim to mainline the community version of the `aarch64` architecture [21, 22] extension. In the long run, we plan to enhance *avatar²*'s capabilities by developing and integrating additional targets. Possible examples include enabling the orchestration of advanced fuzzing campaigns via a `LibAFL QEMU` [26] target or enhanced concolic execution capabilities via integration with `SymQemu` [34].

Looking into the extended future, we believe a rewrite of *avatar²*'s core would overcome the current limitations of the framework. At the moment, users of the framework occasionally experience issues due to the multi-threading and multi-process approach of *avatar²*. We believe that modern languages, such as Rust or Zig, have the potential to increase the overall robustness and reliability of the framework while exposing Python bindings, allowing integration with the existing target, protocol, and plugin code bases.

7 Conclusion

Avatar² is a multi-target orchestration framework for dynamic binary analysis. Since its inception in 2017, it had a significant impact, facilitating the analysis of various embedded systems and forming the basis for follow-up research.

Avatar² served as the basis for a variety of additional research tooling, and selected parts of it have been integrated into other high-profile frameworks, such as `angr` and `PANDA`.

Acknowledgements

We would like to thank the various individuals who made the *avatar²* framework possible and extended its feature set over time. This includes, in no particular order, Dario Nisi, Eric Gustafson, Grant Hernandez, Andrea Biondo, Dominik Maier, Fabian Freyer, Chad Spensky, and Florian Albrecht. We further thank the wider community for their contributions.

References

[1] Florian Albrecht. 2023. Dynamic security analysis of interconnected embedded devices using state-of-the-art rehosting approaches.

[2] Claudio Anliker, Giovanni Camurati, and Srdjan Čapkun. 2023. Time for Change: How Clocks Break UWB Secure Ranging. In *32nd USENIX Security Symposium (USENIX Security 23)*.

[3] Andrea Biondo. 2019. Coverage-guided fuzzing of embedded firmware with avatar2.

[4] Chen Cao, Le Guan, Jiang Ming, and Peng Liu. 2020. Device-agnostic firmware execution is possible: A concolic execution approach for peripheral emulation. In *Proceedings of the 36th Annual Computer Security Applications Conference*. 746–759.

[5] Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, and David Brumley. 2012. Unleashing mayhem on binary code. In *2012 IEEE Symposium on Security and Privacy*.

[6] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. 2012. The S2E platform: Design, implementation, and applications. *ACM Transactions on Computer Systems (TOCS)* (2012).

[7] Abraham A Clements, Eric Gustafson, Tobias Scharnowski, Paul Grosen, David Fritz, Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer. 2020. HALucinator: Firmware re-hosting through abstraction layer emulation. In *29th USENIX Security Symposium (USENIX Security 20)*. 1201–1218.

[8] Nassim Corteggiani, Giovanni Camurati, and Aurélien Francillon. 2018. Inception:System-Wide security testing of Real-World embedded systems software. In *27th USENIX security symposium (USENIX security 18)*.

[9] Nassim Corteggiani and Aurélien Francillon. 2020. HardSnap: Leveraging hardware snapshotting for embedded systems security testing. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[10] Luke Craig, Andrew Fasano, Tiemoko Ballo, Tim Leek, Brendan Dolan-Gavitt, and William Robertson. 2021. PyPANDA: taming the pandamonium of whole system dynamic analysis. In *NDSS Binary Analysis Research Workshop*.

[11] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. 2015. Repeatable reverse engineering with PANDA. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*.

[12] Max Eisele, Marcello Maugeri, Rachna Shriwas, Christopher Huth, and Giampaolo Bella. 2022. Embedded fuzzing: a review of challenges, tools, and solutions. *Cybersecurity* (2022).

[13] Andrew Fasano, Tiemoko Ballo, Marius Muench, Tim Leek, Alexander Bulekov, Brendan Dolan-Gavitt, Manuel Egele, Aurélien Francillon, Long Lu, Nick Gregory, et al. 2021. Sok: Enabling security analyses of embedded systems via rehosting. In *Proceedings of the 2021 ACM Asia conference on computer and communications security*.

[14] Kai Feng, Marco M Cook, and Angelos K Marnerides. 2023. Sizzler: Sequential fuzzing in ladder diagrams for vulnerability detection and discovery in Programmable Logic Controllers. *IEEE Transactions on Information Forensics and Security* (2023).

[15] Peter Goodman and Artem Dinaburg. 2018. The Past, Present, and Future of Cyberdyne. *IEEE Security & Privacy* (2018).

[16] Fabio Gritti, Lorenzo Fontana, Eric Gustafson, Fabio Pagani, Andrea Continella, Christopher Kruegel, and Giovanni Vigna. 2020. Symbion: Interleaving symbolic with concrete execution. In *2020 IEEE Conference on Communications and Network Security (CNS)*.

[17] Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Yanick Fratantonio, Davide Balzarotti, Aurélien Francillon, Yung Ryn Choe, Christophe Kruegel, et al. 2019. Toward the analysis of embedded firmware through automated re-hosting. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 135–150.

[18] Grant Hernandez, Marius Muench, Dominik Maier, Alyssa Milburn, Shinjo Park, Tobias Scharnowski, Tyler Tucker, Patrick Traynor, and Kevin Butler. 2022. FIRMWIRE: Transparent dynamic analysis for cellular baseband firmware. In *Network and Distributed Systems Security Symposium (NDSS) 2022*.

[19] Glenn iain Visser. 2020. Automated Firmware Re-hosting for Security Analysis.

[20] Wenqiang Li, Jiameng Shi, Fengjun Li, Jingqiang Lin, Wei Wang, and Le Guan. 2022. muAFL: non-intrusive feedback-driven fuzzing for microcontroller firmware. In *Proceedings of the 44th International Conference on Software Engineering*. 1–12.

[21] Christian Lindenmeier, Mathias Payer, and Marcel Busch. 2024. EL3XIR: Fuzzing COTS Secure Monitors. In *33rd USENIX Security Symposium (USENIX Security 24)*.

[22] Christian Lindenmeier, Matti Schulze, Jonas Röckl, and Marcel Busch. 2024. SyncEmu: Enabling Dynamic Analysis of Stateful Trusted Applications. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE.

[23] Changming Liu, Alejandro Mera, Engin Kirda, Meng Xu, and Long Lu. 2024. CO3: Concolic Co-execution for Firmware. In *33rd USENIX Security Symposium (USENIX Security 24)*.

[24] Yingtong Liu, Hsin-Wei Hung, and Ardalan Amiri Sani. 2020. Mousse: a system for selective symbolic execution of programs with untamed environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*.

[25] Dominik Maier, Benedikt Radtke, and Bastian Harren. 2019. Unicorefuzz: On the viability of emulation for kernelspace fuzzing. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*.

[26] Romain Malmain, Andrea Fioraldi, and Francillon Aurélien. 2024. LibAFL QEMU: A Library for Fuzzing-oriented Emulation. In *BAR 2024, Workshop on Binary Analysis Research, colocated with NDSS 2024*.

[27] Alejandro Mera, Bo Feng, Long Lu, and Engin Kirda. 2021. DICE: Automatic emulation of DMA input channels for dynamic firmware analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1938–1954.

[28] Marius Muench, Dario Nisi, Aurélien Francillon, and Davide Balzarotti. 2018. Avatar 2: A multi-target orchestration platform. In *Proc. Workshop Binary Anal. Res.(Colocated NDSS Symp.)*.

[29] Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon, and Davide Balzarotti. 2018. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices.. In *NDSS*.

[30] Antonio Nappa, Eduardo Blazquez, Nikias Bassen, and Javier Lopez-Gomez. 2023. *Fuzzing Against the Machine: Automate vulnerability research with emulated IoT devices on QEMU*. Packt Publishing Ltd.

[31] Anh Nguyen-Tuong, David Melski, Jack W Davidson, Michele Co, William Hawkins, Jason D Hiser, Derek Morris, Ducson Nguyen, and Eric Rizzi. 2018. Xandra: An autonomous cyber battle system for the Cyber Grand Challenge. *IEEE Security & Privacy* (2018).

[32] Paul Olivier, Xuan-Huy Ngo, and Aurélien Francillon. 2022. BEERR: Bench of Embedded System Experiments for Reproducible Research. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE.

[33] Nishal Pasham. 2019. Competing with 'diversity'- Firmware analysis and its challenges. https://nihal-pasham.medium.com/competing-with-diversity-firmware-analysis-and-its-challenges-f224c52a3b2

[34] Sebastian Poeplau and Aurélien Francillon. 2021. SymQEMU: Compilation-based symbolic execution for binaries. In *NDSS 2021, Network and Distributed System Security Symposium*. Internet Society.

[35] Tobias Scharnowski, Nils Bars, Moritz Schloegel, Eric Gustafson, Marius Muench, Giovanni Vigna, Christopher Kruegel, Thorsten Holz, and Ali Abbasi. 2022. Fuzzware: Using precise MMIO modeling for effective firmware fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*.

[36] Yan Shoshitaishvili, Antonio Bianchi, Kevin Borgolte, Amat Cama, Jacopo Corbetta, Francesco Disperati, Audrey Dutcher, John Grosen, Paul Grosen, Aravind Machiry, et al. 2018. Mechanical phish: Resilient autonomous hacking. *IEEE Security & Privacy* (2018).

[37] Lingyun Situ, Chi Zhang, Le Guan, Zhiqiang Zuo, Linzhang Wang, Xuandong Li, Peng Liu, and Jin Shi. 2023. Physical devices-agnostic hybrid fuzzing of IoT firmware. *IEEE Internet of Things Journal* (2023).

[38] Chad Spensky, Aravind Machiry, Nilo Redini, Colin Unger, Graham Foster, Evan Blasband, Hamed Okhravi, Christopher Kruegel, and Giovanni Vigna. 2021. Conware: Automated modeling of hardware peripherals. In *Proceedings of the 2021 ACM Asia conference on computer and communications security*.

[39] Miao Yu, Jianwei Zhuge, Ming Cao, Zhiwei Shi, and Lin Jiang. 2020. A survey of security vulnerability analysis, discovery, detection, and mitigation on IoT devices. *Future Internet* (2020).

[40] Joobeom Yun, Fayozbek Rustamov, Juhwan Kim, and Youngjoo Shin. 2022. Fuzzing of embedded systems: A survey. *Comput. Surveys* (2022).

[41] Jonas Zaddach, Luca Bruno, Aurelien Francillon, Davide Balzarotti, et al. 2014. AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares.. In *NDSS*.

[42] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. 2019. FIRM-AFL:High-Throughput greybox fuzzing of IoT firmware via augmented process emulation. In *28th USENIX Security Symposium (USENIX Security 19)*.