An Analysis of Emulation based Fuzz Testing for NRF52 Embedded Software Verification

Rowan Saunders

January 2, 2024

1 Problem Description

Embedded systems are becoming more prevalent in today's society, due to the rise of the Internet of Things (IOT) (Aloseel et al., 2021). Embedded software is often written in low level languages such as C where errors and security vulnerabilities are easy to introduce (Svoboda, 2021), and have many many attack surfaces where these mistakes could be exploited (Aloseel et al., 2021). Furthermore, many embedded systems have stringent reliability requirements, such as safety critical software in automotive devices. As such, verification and testing of embedded systems is paramount to prevent the presence of security exploits and critical bugs in these devices.

In the non-embedded space, many software programs use a methodology called fuzz testing to automatically verify the software. Google has used fuzzing to great effect, detecting over 10000 security vulnerabilities in over 1000 open source projects through its OSS-Fuzz project (Google, 2023). In its most basic form, fuzz testing, or fuzzing, consists of generating some input test data and monitoring the response of the software/system under test (SUT) to this input. If the fuzzer detects the program has crashed, it saves the generated test data for analysis by an engineer, and mutates the input data in some way to produce a different result. This brute force approach is known as black-box fuzzing, and given enough time to execute, will uncover many exploitable bugs with the system, such as memory leaks, buffer overflows, and off by one errors. Basic black-box fuzzing on an embedded system will require some fuzzing harness (Eisele et al., 2022) and some method of detecting a fault, such as performing a liveness check (Yun et al., 2022). Borsig et al. do not consider black-box fuzzing as a promising methodology for fuzzing an esp32 based IOT device, and conclude that white-box and grey-box fuzzing techniques are more capable (Borsig et al., 2020).

Compared to black-box fuzzing, white-box fuzzing involves targeted test case generation through static program analysis techniques, generating a test case to match every statically identified code path. Microsoft have had success using white-box fuzzing to identify difficult bugs that were missed by black-box fuzzing (Godefroid, Levin and Molnar, 2012). However, symbolic execution techniques used in white-box fuzzing can become unfeasably computationally expensive for larger projects (Krishnamoorthy, Hsiao and Lingappan, 2010).

Grey-box fuzzing is a middle ground between black-box and white-box fuzzing. Grey-box fuzzing relies on the fuzzer receiving feedback regarding code coverage for each generated test-case to mutate and generate further test cases. This approach allows it to find code paths faster than black-box testing, but without requiring any static code analysis (Yun et al., 2022).

Grey-box fuzzing of a program can be easily implemented for application (i.e non-embedded) software using popular fuzzers like AFL. AFL provides a library that is linked into the program at compile time, along with several sanitisers, to instrument the program. This allows AFL to receive coverage information from the running program during fuzzing (Google, 2019). Embedded systems often run on constrained environments, where running the fuzzer on the target hardware would be unfeasible. As such, the fuzzer and the SUT are run in different environments, and it is difficult to instrument and forward runtime information for grey-box fuzzing (Muench et al., 2018).

Attempts to solve this problem focus either on fuzzing a program running on target hardware, or in an emulated environment (Eisele et al., 2022). Running in an emulated environment allows easy inspection of the SUT by the fuzzer, and potentially faster running and parallel tests (Eisele et al., 2022). However, emulating embedded systems correctly is a hard problem, and due to the diversity in operating systems, hardware, peripherals etc. often means that development effort spent on emulating one system cannot be easily transferred to another. Attempts have been made to automate emulator development, such as Clements et al. HALucinator, which allows Hardware Abstraction Layer (HAL) functions in a compiled binary to be stubbed and fuzzed (Clements et al., 2021). Chen et al. investigate a novel approach to fuzzing Real Time Operating Systems (RTOS) which involves slicing a single program into its constituent tasks and fuzzing them separately on an emulator based on the call graph of the program (Chen et al., 2022).

Yun et al. note that most embedded systems fuzzers rely on emulation (Yun et al., 2022). However, most bugs found through emulation still need to be validated on hardware, and so being able to run the fuzz tests directly on the target hardware is preferable (Eisele et al., 2022). Several methods have been discussed to improve instrumentation and feedback from running fuzz tests on target hardware. Beckmann et al. propose making use of the tracing facilities on modern Arm Cortex-M microcontrollers to stream coverage information back to the fuzzer (Beckmann and Steffan, 2023). Eisele proposes using a debugger as a method of inspecting the SUT and providing coverage feedback to a fuzzer (Eisele, 2022).

During the development of an embedded system, engineers often require access to representative development hardware to effectively design and write software. Often, waiting for prototype hardware to start software development does not align with business deadlines. The use of emulators improves enables engineers to write embedded software without access to development hardware, improving productivity and reducing development time.

Existing research into embedded systems fuzzing often focuses on its cybersecurity applications and benefits. However, fuzz testing can also be an effective method for general verification of software. AdaCore provide a fuzz testing tool called GNATfuzz, which enables subprogram level fuzz testing as a supplement to unit testing for Ada and SPARK embedded software (AdaCore, 2023). By isolating subprograms and building isolated fuzz test executables in a manner similar to a unit testing framework, GNATfuzz removes the need to be able to run the full program under a fuzzer, and thus the need for an emulator or target hardware. However, the effectiveness of this approach is relient on Ada's extended runtime constraint checking when compared to C (AdaCore, 2023). Furthermore, subprogram level fuzzing is less able detect more complex issues resulting from interactions with hardware peripherals and state.

While research into the fuzz testing of embedded systems has been increasing year by year (Yun et al., 2022), there are few generic solutions (Eisele et al., 2022). The two main challenges to embedded system fuzzing when compared to desktop or server systems remain the variety in CPU architectures in embedded systems, and the lack of an operating system such as linux for bare metal systems (Eisele et al., 2022). Currently, the availability and ease of use of fuzzing tooling for embedded systems does not match that of desktop applications. However, while much research outlines the challenges with current embedded systems fuzzing techniques, it is unclear how these challenges compare with the effort required to verify embedded software using traditional techniques. In order to prove that embedded fuzz testing is a viable alternative to existing verification methods, a comparison of the cost effectiveness of conducting embedded fuzzing with other verification techniques is required.

The design and implementation of embedded software and its verification is highly coupled to hardware. The nordic semiconductor NRF52 family of microcontrollers are a common choice for IOT devices. NRF52s are based on the ARM Cortex-M architecture, and include embedded radio peripherals, such as bluetooth low energy and wifi (amongst others). Another common IOT radio microcontroller is the ESP32, for which Borsig et al. previously developed a fuzzing framework (Borsig et al., 2020). Furthermore, Yun suggested the need for further research into emulation of specific architectures and devices (Yun et al., 2022). An NRF52 was used by Beckmann for on-target coverage guided fuzzing of a bare metal ARM Cortex-M system using instruction tracing over a single wire output (SWO) interface (Beckmann and Steffan, 2023). Additionally, Behrang outlines a method for using the unicorn cpu emulator to execute NRF52 based bare-metal radio firmware (Behrang, 2023). The desktop fuzzer AFL++ includes an implementation called ”unicorn_mode” which allows the unicorn engine to be built with AFL++ support (Maier, Voss and Fioraldi, 2023). Maier develops the Unicorefuzz framework and presents an example use of AFL++ Unicorn Mode for fuzzing kernel modules, which have similar challenges to embedded systems (Maier, Radtke and Harren, 2019).

2 Project Objectives and Deliverables

This research project aims to investigate the perceived difficulty of fuzzing an embedded system with currently available tooling, and make a comparison between different fuzzing and other verification methods with regards to that difficulty. The project will outline the procedure for configuring and executing the different verification methodologies, and gather data about the effectiveness, time/effort, and reuseability. The objective of the study is to prove that applying fuzz testing methodologies during embedded systems development provides a strong benefit, despite any time/effort costs. The expectation is that fuzz testing may outperform traditional verification methods with respect to cost, such as manual systems testing, and prove to have higher reuseability.

To do this, an example bare-metal embedded system will be developed based on the NRF52 chip, with sufficient features to be representative of a real device. Test harnesses required for manual and fuzz testing will be developed, and the time and tasks to produce these will be tracked. The example firmware will then be tested with each verification method, and the number of errors discovered, and time taken to discover them, will be recorded. These quantitive data will be compared and discussed, along with qualititative anecdotal information regarding challenges encountered with each proposed method.

To meet this objective, and based on the work outlined in *Section 1*, a unicorn based emulator for the example NRF52 based system will be developed, with test harnesses for fuzzing with AFL++ UnicornMode. Required NRF52 hardware peripherals will be modeled in the unicorn based emulator. Manual test scripts derived from software requirements for the embedded system will be created, and also run on the emulator. Furthermore, a simple black box fuzzer will be developed to run on the emulator. The core deliverables will be:

- A literature review outlining and comparing embedded fuzz testing methodologies
- The design and implementation of an example NRF52 based embedded system
- A set of system test scripts for manually testing the example system
- The implementation of a unicorn NRF52 emulator, with appropriate peripherals modeled
- A set of error reports and bug investigations resulting from testing the example system with a black-box fuzzer on the unicorn emulator
- A set of error reports and bug investigations resulting from running the test scripts against the example system on the unicorn emulator

- A set of error reports and bug investigations resulting from testing the example system with unicorn-afl
- A discussion comparing each emulation based verification methodology, considering cost effectiveness

The main objective outlined will be supplemented with additional extension objectives. One potential extension to the main objective would include an additional investigation and comparison with hardware based verification techniques. Coverage guided fuzzing based on Beckmann's tracing based work (Beckmann and Steffan, 2023) could be implemented, along with developing a test harness to execute the manual test scripts and apply the black-box fuzzer on the target hardware. The deliverables for this extended objective would be as follows:

- The development of test harnesses for on target verification of the embedded system
- Tooling, Instrumentation and Harnesses required for on target coverage guided fuzzing of the example system
- A set of error reports and bug investigations resulting from testing the example system with a black-box fuzzer on the target hardware
- A set of error reports and bug investigations resulting from running the test scripts against the example system on the target hardware
- A set of error reports and bug investigations resulting from testing the example system with a coverage guided hardware fuzzer
- A discussion comparing each hardware based verification methodology, considering cost effectiveness

An alternative extension objective would be to measure the effect of system complexity on the cost effectiveness of each verification methodology. The example system can be updated to use common IOT peripherals, such as wifi or bluetooth, and model these in the unicorn emulator. The ease of emulating and fuzzing an embedded system may be correlated with the complexity of that system, and using a wireless protocol should improve the relevence of the results of the study to IOT devices. The deliverables for this extended objective would be as follows:

- The design and implmentation of an expanded NRF52 based embedded system, including the use of a wireless communication peripheral
- The implementation of a unicorn NRF52 emulator with wireless peripherals modeled
- A set of expanded system test scripts for manually testing the updated example system
- A set of error reports and bug investigations resulting from testing the expanded example system with a black-box fuzzer on the unicorn emulator
- A set of error reports and bug investigations resulting from running the test scripts against the example system on the unicorn emulator
- A set of error reports and bug investigations resulting from testing the expanded example system with unicorn-afl
- A discussion comparing each emulation based verification methodology, considering cost effectiveness and software complexity

3 Project Plan

Figure 1 shows a top level gantt chart for the duration of the project. The timeframe in *Figure 1* is scoped in terms of week increments (i.e. 5 working days each). The expected duration of the project of 13 working weeks is equivilent to 65 working days. Given an expected average working rate of 8 working days per calendar month, an overall estimate of the expected timeline can be calculated at around 8 months. This aligns with the chosen completion period.

The project will initially begin with an extended literature review, gathering and investigating more literature and developing on ideas outlined in *Section 1.* This will lead into a pilot study, where a basic version of the study will be completed. A detailed work breakdown for the pilot study is shown in *Figure 2.*

The focus of the pilot study will be a comparison of emulation based fuzzing with manual testing and black box fuzzing techniques. The initial example system will be a simplistic IOT device, using few hardware peripherals. A serial interface will be used to communicate to the device, which will be simple to integrate with the unicorn-afl fuzzer.

The work conducted in the pilot study will then be revised and built upon for a more indepth extended study. There are several potential avenues for investigation in the extended study, and the direction taken at this point will depend on the output of the literature review and pilot study. The extended study will aim to meet at least one of the extension objectives in *Section 2.* Two alternative detailed plans are shown for the extended study in *Figure 3* and *Figure 4.*

The extended study detailed in *Figure 3* expands the firmware developed in the pilot study to use more peripherals. The serial interface will be modified to use a wireless protocol, such as wifi or bluetooth, which is more common for IOT devices. Comparing test methodologies for a simple and a more complex system may also show that cost effectiveness of a test methodology is a function of cost effectiveness, as well as exploring the ease of modification and reusability of the emulator based test techniques.

The extended study detailed in *Figure 4* configures a suite of hardware based verification techniques with the example system developed in the pilot study. It is anticipated that tooling and harnesses developed in the pilot study for emulated black-box and manual testing will be able to be easily adapted to their hardware based equivilents. As such, the main focus of this work will be on implementing the coverage guided hardware fuzzing tooling and instrumentation. Currently, time estimates for these tasks are somewhat uncertain, and so performing the coverage guided hardware fuzzing last out of the three techniques will at least allow a comparison with the hardware based manual and black box fuzzing if the coverage guided fuzzing cannot be completed in the time frame.

The work outlined in this plan can be completed with a linux based computer. The outlined work can be fully completed with open source tools, and where tooling is not available, this will be created as part of the study. Running the fuzz tests may require leaving a computer for an extended period, or may require access to a computer with more powerful hardware (e.g. more RAM). If this is the case, a Virtual Private Server (VPS) can be purchased with the required capability for executing the fuzz tests. Some tasks in the extended study may require access to an NRF52 development kit. This has already been procured, as it may also be useful for experimenting with during the pilot study.

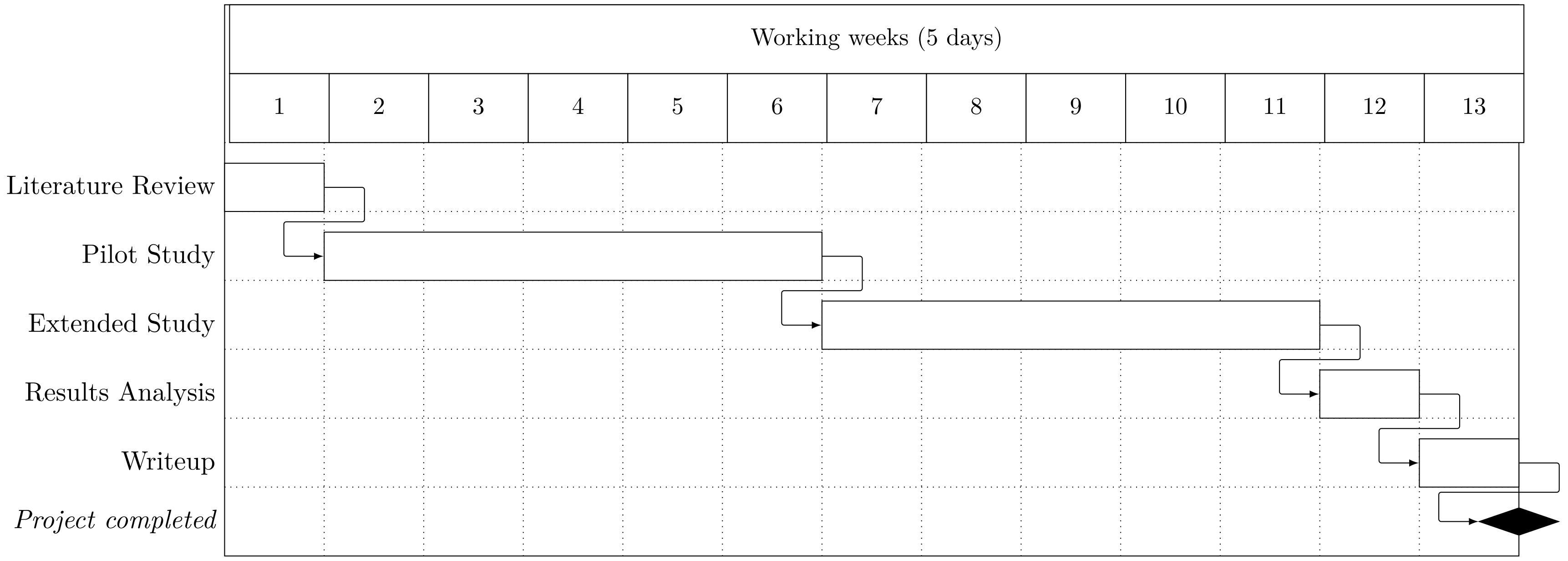

Figure 1: Top Level Gantt Chart

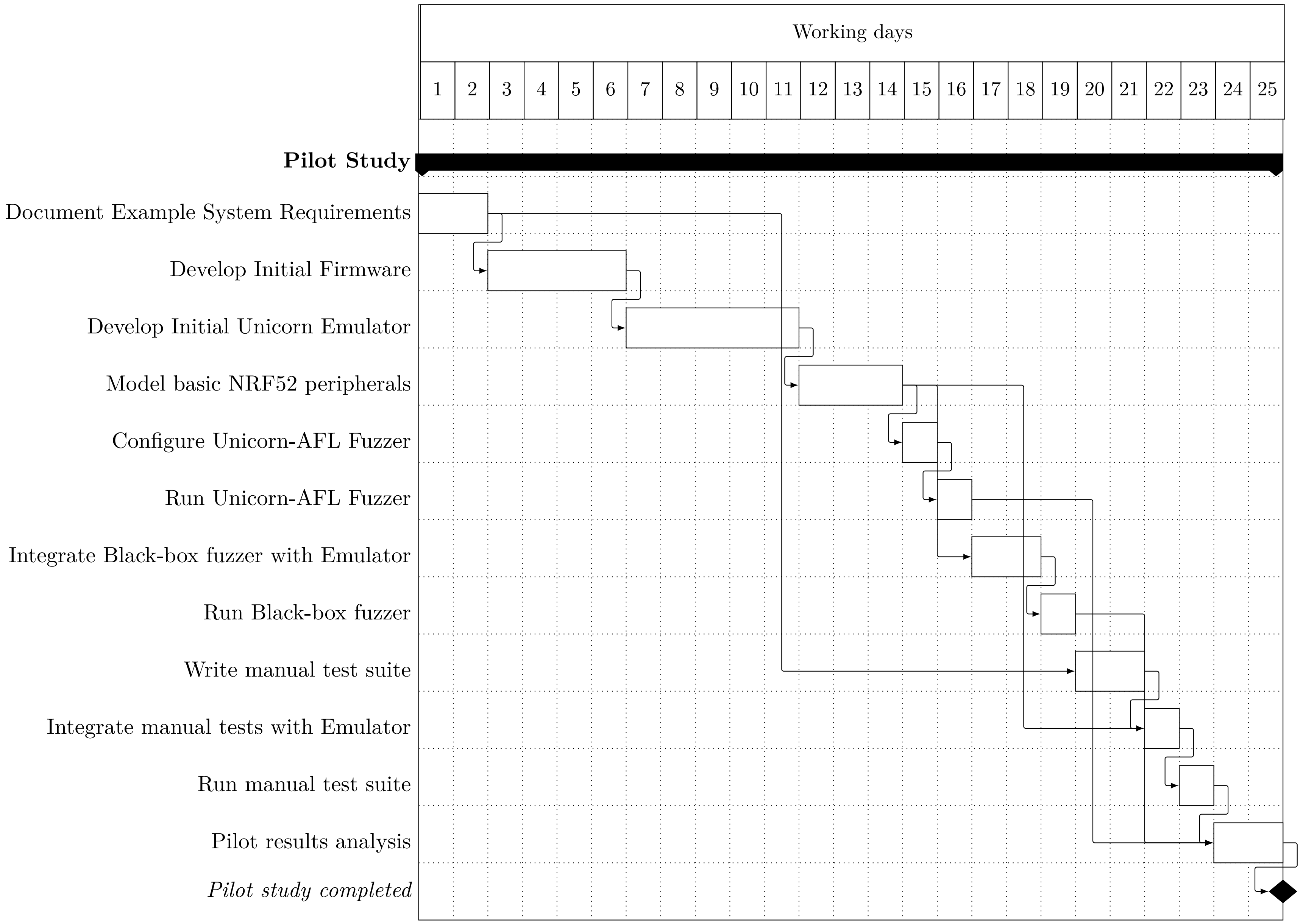

Figure 2: Detail level Gantt Chart for Pilot Study

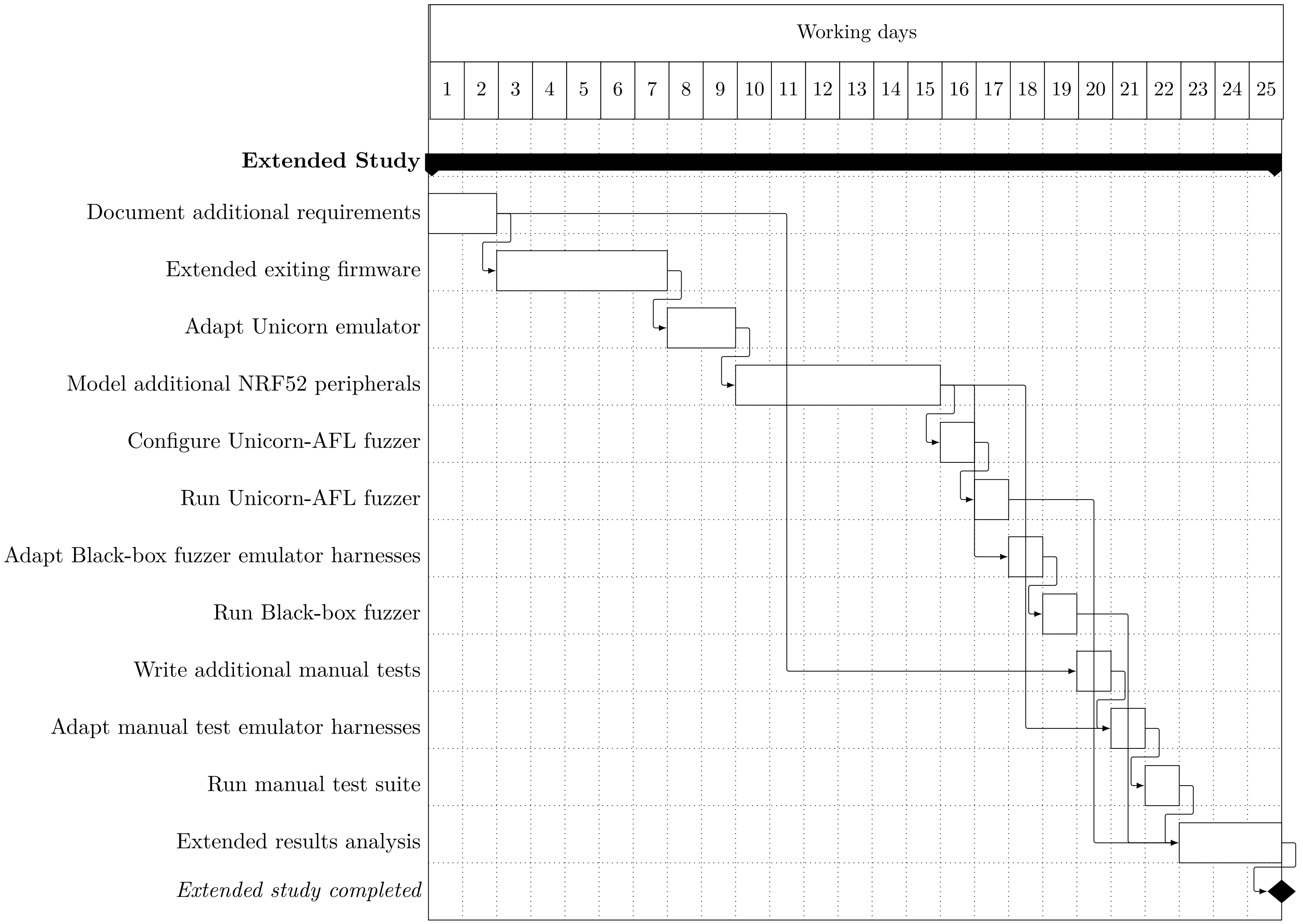

Figure 3: Detail level Gantt Chart for Extended Study where the pilot example system is modified to increase system complexity

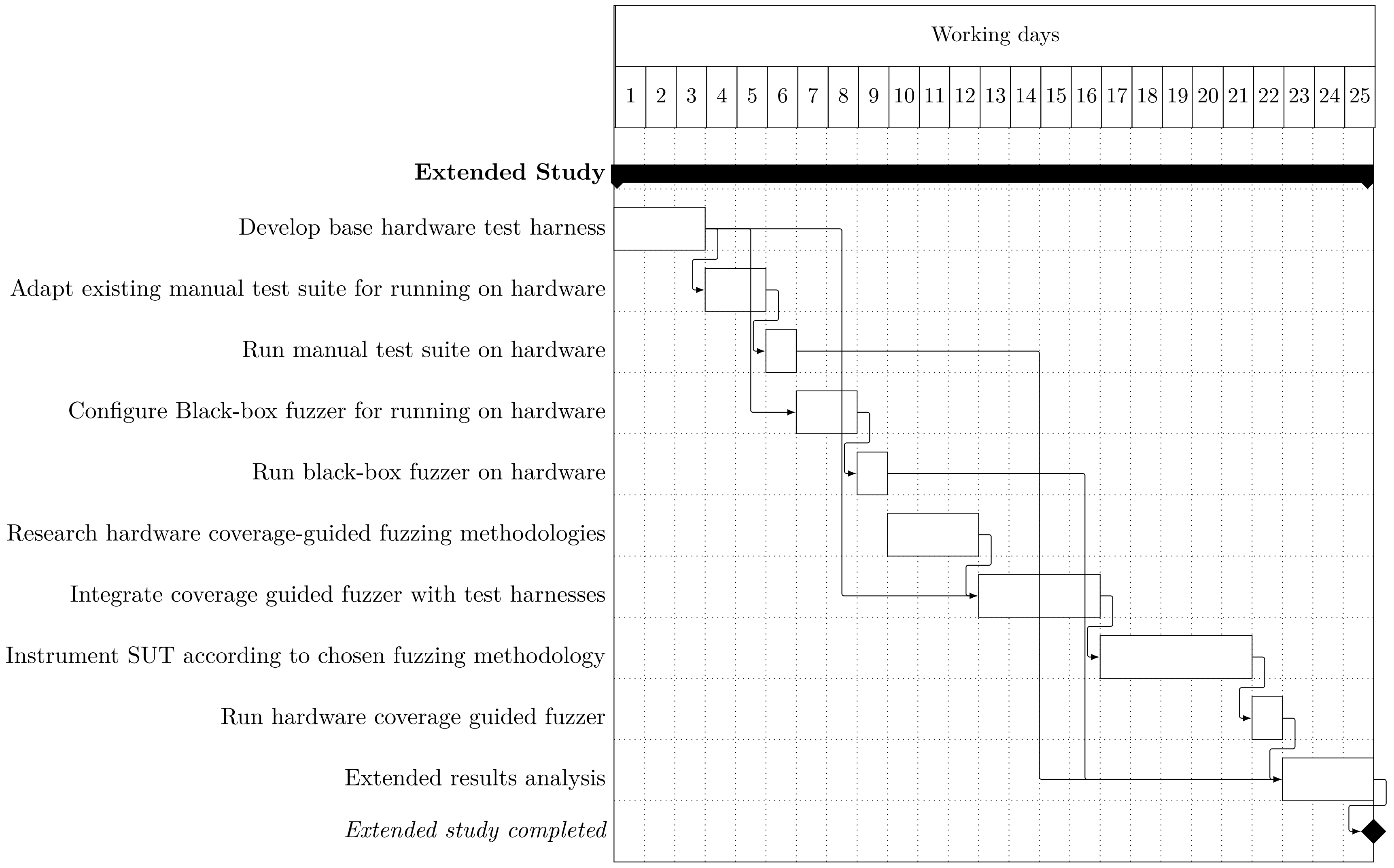

Figure 4: Detail level Gantt Chart for Extended Study where the pilot example system is tested with additional verification techniques

References

AdaCore, 2023. *Gnatfuzz user's guide* [Online]. Available from: https://docs.adacore.com/live/wave/gnatdas/html/gnatdas_ug/gnatfuzz/gnatfuzz_part.html# [Accessed 2024-01-02].

Aloseel, A., He, H., Shaw, C. and Khan, M.A., 2021. Analytical review of cybersecurity for embedded systems. *Ieee access* [Online], 9, pp.961–982. Available from: https://doi.org/10.1109/ACCESS.2020.3045972.

Beckmann, M. and Steffan, J., 2023. Coverage-guided fuzzing of embedded systems leveraging hardware tracing. *Computer security. esorics 2022 international workshops* [Online], p.362–378. Available from: https://doi.org/10.1007/978-3-031-25460-4_21.

Behrang, F., 2023. *nrf52 radio emu* [Online]. Available from: https://github.com/befoulad/nrf52_radio_emu [Accessed 2023-12-28].

Borsig, M., Nitzsche, S., Eisele, M., Groll, R., Becker, J. and Baumgart, I., 2020. Fuzzing framework for esp32 microcontrollers. *2020 ieee international workshop on information forensics and security (wifs)* [Online]. Available from: https://doi.org/10.1109/wifs49906.2020.9360889.

Chen, L., Cai, Q., Ma, Z., Wang, Y., Hu, H., Shen, M., Liu, Y., Guo, S., Duan, H., Jiang, K. and Xue, Z., 2022. Sfuzz: Slice-based fuzzing for real-time operating systems. *Proceedings of the 2022 acm sigsac conference on computer and communications security* [Online]. New York, NY, USA: Association for Computing Machinery, CCS '22, p.485–498. Available from: https://doi.org/10.1145/3548606.3559367.

Clements, A.A., Carpenter, L., Moeglein, W.A. and Wright, C., 2021. Is your firmware real or re-hosted? a case study in re-hosting vxworks control system firmware. *Proceedings 2021 workshop on binary analysis research* [Online]. Available from: https://doi.org/10.14722/bar.2021.23006.

Eisele, M., 2022. Debugger-driven embedded fuzzing. *2022 ieee conference on software testing, verification and validation (icst)* [Online]. Available from: https://doi.org/10.1109/icst53961.2022.00062.

Eisele, M., Maugeri, M., Shriwas, R., Huth, C. and Bella, G., 2022. Embedded fuzzing: A review of challenges, tools, and solutions. *Cybersecurity* [Online], 5(1). Available from: https://doi.org/10.1186/s42400-022-00123-y.

Godefroid, P., Levin, M.Y. and Molnar, D., 2012. Sage: Whitebox fuzzing for security testing. *Queue* [Online], 10(1), p.20–27. Available from: https://doi.org/10.1145/2090147.2094081.

Google, 2019. *Afl (american fuzzy lop)* [Online]. Available from: https://afl-1.readthedocs.io/en/latest/index.html [Accessed 2023-11-28].

Google, 2023. *Oss-fuzz* [Online]. Available from: https://google.github.io/oss-fuzz/ [Accessed 2023-11-28].

Krishnamoorthy, S., Hsiao, M.S. and Lingappan, L., 2010. Tackling the path explosion problem in symbolic execution-driven test generation for programs. *2010 19th ieee asian test symposium* [Online]. Available from: https://doi.org/10.1109/ats.2010.19.

Maier, D., Radtke, B. and Harren, B., 2019. Unicorefuzz: On the viability of emulation for kernelspace fuzzing. *13th usenix workshop on offensive technologies (woot 19)* [Online]. Santa Clara, CA: USENIX Association. Available from: https://www.usenix.org/conference/woot19/presentation/maier.

Maier, D., Voss, N. and Fioraldi, A., 2023. *Unicorn-based binary-only instrumentation for afl-fuzz* [Online]. Available from: https://github.com/AFLplusplus/AFLplusplus/blob/stable/unicorn_mode/README.md [Accessed 2023-12-28].

Muench, M., Stijohann, J., Kargl, F., Francillon, A. and Balzarotti, D., 2018. What you corrupt is not what you crash: Challenges in fuzzing embedded devices. *Proceedings 2018 network and distributed system security symposium* [Online]. Available from: `https://doi.org/10.14722/ndss.2018.23166`.

Svoboda, D., 2021. Hands-on tutorial: How exploitable is insecure c code? *2021 ieee secure development conference (secdev)* [Online]. pp.7–8. Available from: `https://doi.org/10.1109/SecDev51306.2021.00015`.

Yun, J., Rustamov, F., Kim, J. and Shin, Y., 2022. Fuzzing of embedded systems: A survey. *Acm comput. surv.* [Online], 55(7). Available from: `https://doi.org/10.1145/3538644`.

CHECKLIST FOR RESEARCH WITH HUMAN PARTICIPANTS THAT MAY REQUIRE FULL COMMITTEE REVIEW

This checklist should be completed for every research project involving human participants in order to identify whether a full application for ethics approval may be needed to be submitted to one of the University Ethics Committees. If you tick yes to any of the boxes, please consult with your supervisor or the Department Research Ethics Officer (DREO).

(A) Research that may need full review by a University of Bath Ethics Committee	**YES**	**NO**
Do you and/or your supervisor intend to submit your results for publication to the wider research community (other than in your dissertation) where evidence of ethical approval is required?	☐	☒
Does the project involve the collection of material that could be considered of a personal, biographical, medical, psychological, social or physiological nature?	☐	☒
Does the research involve vulnerable groups: e.g. children; those with cognitive impairment; or those in unequal relationships (e.g. your own students)?	☐	☒
Will the study require the cooperation of a gatekeeper for initial access to the groups or individuals to be recruited (e.g. headmaster at a School; group leader of a self-help group)?	☐	☒
Will it be necessary for participants to take part in the study without their knowledge and consent at the time? (e.g. covert observation of people in non-public places?)	☐	☒
Will the study involve discussion of sensitive topics (e.g. sexual activity; drug use; criminal activity)?	☐	☒
Is pain or more than mild discomfort likely to result from the study?	☐	☒
Could the study induce psychological stress or anxiety or cause harm or negative consequences beyond the risks encountered in normal life?	☐	☒
Will the study involve prolonged or repetitive testing?	☐	☒
Will the research involve organisational administrative or secure data that requires permission from the appropriate organisation/authorities before use (data that is not in the public domain)?	☐	☒
Does the research involve participants carrying out any of the research activities themselves (i.e. acting as researchers as opposed to just being participants)?	☐	☒
Is there a possibility that the safety of the researcher may be in question (e.g. international research; locally employed research assistants)?	☐	☒
Will personal data be transferred to/from the UK (including to/from the EEA?)	☐	☒
Will the outcome of the research allow respondents to be identified either directly or indirectly (e.g. through aggregating separate data sources gathered from the internet)?	☐	☒
Will research involve the sharing of data or confidential information beyond the initial consent given?	☐	☒
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?	☐	☒
Will the proposed findings be controversial or are there any conflicts of interest?	☐	☒
Will the study involve the publication, sharing or potentially insecure electronic storage and/or transfer of data that might allow identification of individuals, either directly or indirectly? (e.g. publication of verbatim quotations from an online forum; sharing of audio/visual recordings; insecure transfer of personal data such as addresses, telephone numbers etc.; collecting identifiable personal data on unprotected** internet sites.) [**Please note that Qualtrics provides adequate data security]	☐	☒

(B) Security Sensitive Material	**YES**	**NO**
Does your research involve access to or use of material covered by the Terrorism Act? (The Terrorism Act (2006) outlaws the dissemination of records, statements and other documents that can be interpreted as promoting and endorsing terrorist acts. By answering ‘yes’ you are registering your legitimate use of this material with the Research Ethics Advisory Group. In the event of a police investigation, this registration will help you to demonstrate that your use of this material is legitimate and lawful).	☐	☒

(C) Prevent Agenda	**YES**	**NO**
Does the research have the potential to radicalise people who are vulnerable to supporting terrorism or becoming terrorists themselves?	☐	☒

This form must be attached to the dissertation as an appendix.

Department of Computer Science

12-Point Ethics Checklist for UG and MSc Projects

Student	Rowan Edward Jon Saunders
Academic Year or Project Title	2024
Supervisor	

Does your project involve people for the collection of data other than you and your supervisor(s)? ~~YES~~ / NO

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Will you prepare a Participant Information Sheet for volunteers?* YES / NO

   This means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO

   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Will there be any intentional deception of the participants?* YES / NO

   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO

   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. *Will participants voluntarily give informed consent?* YES / NO

Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete. If your data collection is entirely anonymous and does not include collection of personal data you do not need to collect a signature. Instead, you should include a checkbox, which must be checked by the participant to indicate that informed consent has been given.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* YES / NO

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Will you be offering any incentive to the participants?* YES / NO

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Will you be in a position of authority or influence over any of your participants?* YES / NO

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Will any of your participants be under the age of 16?* YES / NO

Parental consent is required for participants under the age of 16.

10. *Will any of your participants have an impairment that will limit Their understanding or communication?* YES / NO

Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES / NO

All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. Will you have a data management plan for all recorded data? YES / NO

Personal data is anything which could be used to identify a person, or which can be related to an identifiable person. All personal data (hard copy and/or soft copy) should be anonymized (with the exception of consent forms) and stored securely on university servers (not the cloud).