# An Open-Source BinaryGame for Learning Reverse Engineering*

D'Angelo Gourdine[1] and Suzanne J. Matthews[2]
U.S. Military Academy
West Point, NY 10996
{dangelo.gourdine, suzanne.matthews}@westpoint.edu

**Abstract**

This paper introduces an open-source BinaryGame that assists students learning reverse engineering. The game consists of ten levels that increase in difficulty, help pages on GDB, and supports three flavors of assembly language. Work on the BinaryGame is ongoing; for our initial study, we used the BinaryGame to introduce students in a computer systems & organization course to Arm assembly. These students had prior knowledge of x64 assembly, but no prior knowledge of Arm assembly; our goal was to boost our students' confidence in learning unfamiliar assembly languages. Our results suggest that the BinaryGame increased student confidence in their a.) general reverse engineering abilities; b.) ability to reverse engineer programs in an unfamiliar assembly language, and c.) ability to reverse programs in Arm assembly. We believe that the BinaryGame can help students build their reverse engineering skillset.

## 1 Introduction

The BinaryGame is a open-source resource designed to help students learn reverse engineering at their own pace through an iterative, guided design. The notion of creating a game or other executable to teach reverse engineering has existed for nearly twenty years. The earliest known example is perhaps "Dr. Evil's Binary Bomb" (a.k.a., Bomblab) [5], initially developed by educators at

*Copyright © 2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Carnegie Mellon University, and distributed as part of the seminal text, *Computer Systems: A Programmer's Perspective* [4]. The original Bomblab was an executable compiled for x86 systems and was used as part of classroom laboratory assignment at CMU. The bomb consists of six phases. Students have to enter the correct string for each phase; if the incorrect string is entered, the bomb "explodes" (e.g., prints out a string that says "Boom!"), causing a student to lose points. In the intervening years, many other bomb-like executables/games were developed [3, 13, 15] and used to teach basic reverse engineering concepts at conferences across the country, including Defcon and Black Hat, the premier conferences for hackers. The source code (and compiled binaries) to these custom re-implementations are typically not shared across institutions to reduce the likelihood that solutions "get out into the wild", either through code-sharing utilities (e.g., GitHub, Pastebin) or detailed walkthroughs on YouTube. The proliferation of online solutions prevents instructors from reusing (or sharing) otherwise well-designed assessments that took considerable time and effort to create.

A key novelty of our work is that the BinaryGame is open-source, allowing instructors to modify and reuse the source code as a teaching aid in their own courses. The BinaryGame is *not* designed to replace Bomblab-like assessments, but to *supplement* them. In their 2016 paper, Shashidhar and Cooper [15] advocate for the use of hands-on lab activities to "level the playing field" when teaching malware analysis, prior to introducing a more free-form "grand challenge" assignment. In their course, Shashidhar and Cooper use Portable Executable (PE) files to first introduce students to the principles of malware analysis in a laboratory setting. The Bomblab assignment is then discussed as a potential "grand challenge" assignment to provide a "less structured learning environment... where leadership and direction comes from the student as much as the instructor" [15]. Our creation of the BinaryGame supports the model curriculum laid out by Shashidhar and Cooper by freely providing a hands-on lab activity to introduce students to reverse engineering prior to the introduction to a larger "grand challenge" assignment.

Wile the literature strongly supports the use of simple exercises to build reverse engineering skills, there are very few readily-available resources for instructors to use in their courses. Shashidhar and Cooper [15] do not describe nor provide any of the lab activities they use. Aycock et al. [1] discuss a series of exercises they developed to teach reverse engineering, though not with sufficient data to fully reproduce all of them. While Stricklan and OConnor [16] discuss a framework for creating "diversified challenges" for a reverse engineering course, the challenges themselves are not provided. Like Aycock et al. [1], the BinaryGame contains levels that build on each other and increase in difficulty as a user advances through the game. Like the method discussed by

Stricklan and OConnor [16], the BinaryGame includes options for compiling to distinct architectures and enables the generation of multiple unique binary executables. Like Bomblab, the goal for each level is to input a string that will allow the user "pass" to the next level. Unlike prior work, the BinaryGame also includes introductory guides on GDB and different flavors of assembly, and suggests readings in the free online textbook, *Dive into Systems* [12].

Work on the BinaryGame is ongoing and community contributions are welcome. For our initial evaluation of the BinaryGame, we focus specifically on its ability to introduce students to an unfamiliar assembly language, namely Arm assembly. The Arm architecture is one of the fastest growing and widely used computer architectures today; as of 2021, over 200 billion chips using the Arm architecture are used in a variety of mobile devices [14]. Unlike Stricklan and OConner [16] who use Qemu [2] to generate Arm binary executables, we deploy the BinaryGame on a Raspberry Pi. Our decision to use the Raspberry Pi in lieu of a hardware emulator was manifold: first, prior work strongly supports the use of the Raspberry Pi to teach cybersecurity concepts [6, 17, 7, 8]. Next, researchers have also used the Raspberry Pi to teach topics like C programming [18], Arm assembly [9, 10], and parallel programming [10], all which are common topics in a computer systems course, a natural place to use the BinaryGame. Lastly, using the Raspberry Pi allowed us to hide the code to generate new binaries on the Pi's file system, enabling the user to generate novel, playable binaries of the BinaryGame at will on their Pis.

## 2 Overview of Game

The BinaryGame includes ten levels that increase in difficult and scope, modeled after concepts covered in each of the first nine sections of the Assembly chapters in the free online textbook, *Dive into Systems* [11, 12]. The levels are: (0) predefined C functions; (1) basic string matching; (2,3) arithmetic operations; (4,5) loops; (6) recursion; (7) arrays; (8)matrices; and (9) structs. The goal of each level is to identify a unique string that will enable the user to continue on the following level; there is no penalty for incorrect answers. In the event that students are unable to pass the level and need assistance, they have the option to receive a hint, which encourage students to refer to specific portions of *Dive into Systems* and provide other clues for solving the level.

The BinaryGame is designed to be used in conjunction with the GNU Debugger (GDB), which typically represents a steep learning curve for students. As a result, the BinaryGame includes several help pages that provide an entry point to GDB and basic assembly concepts. In addition to GDB, there are separate help pages that provide basic information on Arm assembly, IA32 assembly, x86-64 assembly and a general introduction to the game.

The BinaryGame is written in C and includes a Python wrapper that generates multiple version of the binary executable. Giving students access to this variability allows them to harden their skills through increased repetitions. For the purposes of this study, the BinaryGame was compiled directly on a Raspberry Pi 4B; we have verified that the BinaryGame works with the Raspberry Pi 3B+ and 3B. We note that a 64-bit version of the operating system is preferable to ensure the assembly matches the coverage of *Dive into Systems*. To prepare the devices for classroom use, the instructor flashed a number of Raspberry Pis with an image that had the BinaryGame source code hidden in a directory on the file system. We wrote a custom shell script (called `refesh-game`) that generates a new version of the Binary Game executable, and replaces the BinaryGame directory in the user's home directory with this new version, deleting all solution files that students may have created in that directory. This allows the instructor to "refresh" the game directory between labs without reflashing the Pis. Instructors can access the BinaryGame source at: `https://github.com/suzannejmatthews/binaryGame`.

## 3 Methods

The surveyed students were primarily juniors majoring in computer science or cyber security. All students were enrolled in a required computer systems & organization course at West Point, which is the first course that students are introduced to reverse engineering. The BinaryGame was deployed as a lab exercise on Arm assembly fairly late in the semester. While the lab represents students' first exposure to Arm assembly and the Raspberry Pi, all students had previously completed a 10-lesson unit modeled after the first nine sections of the x86-64 assembly chapter in *Dive into Systems*. Our goal in having an Arm assembly lab in a course that otherwise covers x86 assembly was to build our students' confidence that their reverse engineering skills can translate to an ISA that they are unfamiliar with. We concentrated on two main research questions. First, RQ1: "Does the BinaryGame improve student confidence in their reverse engineering skills?". Second, RQ2: "Does BinaryGame improve student motivation to continue learning reverse engineering?"

To analyze the effect that the BinaryGame had on student's confidence in their reverse engineering skills, we asked participants to rate their current level of confidence on a Likert scale on three topics related to reverse engineering on the pre- and post-surveys:

- C1: Confidence reverse engineering programs in some ISA (e.g. x64)
- C2: Confidence learning an unfamiliar assembly language
- C3: Confidence reverse engineering programs in Arm assembly

Here, a score of 1 represented "very unconfident", 2 represented "somewhat unconfident", 3 represented "neither confident/unconfident", 4 represented "somewhat confident", and 5 represented "very confident".

To assess the impact that BinaryGame had on students' motivation to continue learning reverse engineering, we asked students to rate their current level of motivation on Likert scale on the pre- and post-surveys. For this question, a score of "1 represented ("very unmotivated") to 5 ("very motivated") to learn reverse engineering" on the pre- and post-surveys.

Since this lab was our students first exposure to the Raspberry Pi, we spent a few minutes at the beginning of the lab period discussing the ubiquity of Arm processors in mobile and embedded computing and discussed how malware may increasingly target the architecture in the future. We also discussed how cyber professionals need to be comfortable switching between multiple instruction set architectures, and how this lab was designed to help them gain confidence that they can quickly "pick up" a new ISA (namely Arm). After giving a quick overview of the features of the BinaryGame, the students were given the rest of the lab period to play through the game on their own. At the end of the lab period, we asked them to complete the post-survey which contained identical questions to the pre-survey, plus additional questions on various aspects of the game. We use a paired sample student $t$-test to measure the significance of the difference of the means of each population. For each question, we reject the null hypothesis when the $p$-value is less than 0.05. No identifiable data was collected on students; however, the pre- and post-surveys for students were distributed together to enable us to correlate student responses.

# 4 Results

We surveyed students over two semesters of the computer systems & organization course, which together had an enrollment of 58 students. Of these 58 students, only 33 students (56.9%) opted to participate. Of the 33 students, one student failed to complete the pre-survey and another failed to complete the post-survey. To ensure we could conduct a paired sample $t$-test between the two populations, we omitted these two students from our analysis. As a result, there are 31 students ($n = 31$) in our final analysis. We used the `ttest_rel` statistic in the Python SciPy stats module for the paired difference test.

## 4.1 Effect on Student Confidence

For RQ1, our null hypothesis is that students would not experience a significant improvement in their level of confidence in their reverse engineering skills. We reject the null hypothesis when the $p$-value is below 0.05 for each confidence-related topic. Table 1 depicts the mean scores and corresponding p-values of

Table 1: Summary of Confidence Results

| Question | Pre-Survey $n = 31$ | Post-Survey $n = 31$ | $p$-value |
|---|---|---|---|
| C1 | 2.71 | 3.452 | $4.519 \times 10^{-06}$ |
| C2 | 2.581 | 3.29 | $3.181 \times 10^{-06}$ |
| C3 | 1.823 | 3.323 | $8.053 \times 10^{-09}$ |

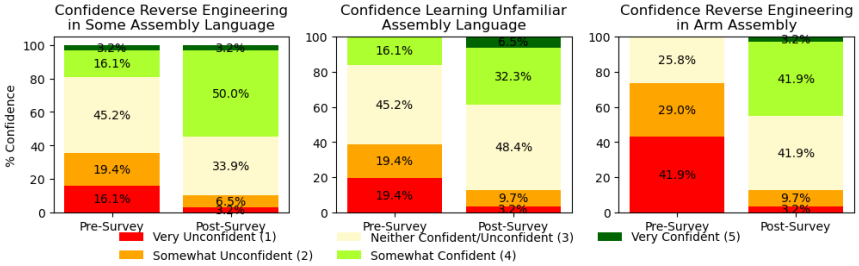


Figure 1: Distribution of Student Confidence on Pre- and Post-Surveys.

our three topics assessing student confidence related to reverse engineering. Figure 1 illustrates the distribution of scores from the pre- and post-surveys. Our data strongly suggests that our students experienced a statistically significant increase in confidence in in their ability to reverse engineer programs in some ISA, learn an unfamiliar assembly language, and reverse engineer programs in Arm assembly.

For example, only 19.3% of respondents expressed some level of confidence that they could reverse engineer in some assembly language on the pre-surveys, with a large number of students (45.2%) feeling neither confident or unconfident. In the post-surveys, the majority of students (53.2%) indicated some level of confidence in reverse engineering in some assembly language, with only 33.9% feeling neither confident or unconfident.

Similarly, students experienced a statistically significant increase in confidence in reverse engineering in an unfamiliar assembly language (C2). In the pre-survey, only 16.1% of students expressed some level of confidence that they could reverse engineering in an unfamiliar assembly language, while 38.8% expressed some level fo confidence in the post-survey. However, a large percentage of students expressed that they were neither confident or unconfident on both surveys. Perhaps unsurprisingly, students experienced the largest increase in confidence in reverse engineering in Arm assembly (C3). We do note that while a large number of students (45.1%) expressed confidence in the post-survey, a

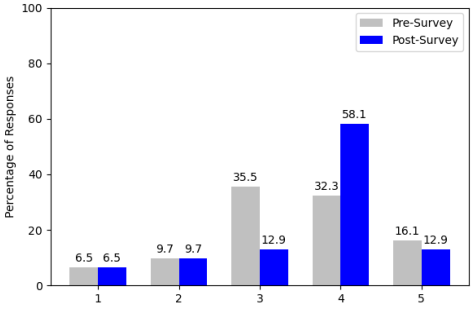


Figure 2: Distribution of Student Motivation on Pre- and Post-Surveys.

significant number (41.9%) still expressed ambivalence on the post-survey, with 12.9% expressing a lack of confidence.

While the increase in confidence across C3 is perhaps expected, the increase in confidence in "reverse engineering in some assembly language" (C1) is interesting. First, while our students had prior exposure to an in-course 10-lesson unit on reverse engineering in x86-64 prior to taking the Arm lab, nearly 35.5% of the course expressed that they felt some level of confidence about reverse engineering in some assembly language on the pre-survey. While the BinaryGame lab covered Arm reverse engineering, their overall confidence in reverse engineering in *some* assembly language increased, suggesting that process of using tools like GDB (even in the context of an unfamiliar assembly language) helped them improve their general skills in reverse engineering. Given our results, we reject the null hypothesis, and conclude that the BinaryGame had a statistically positive impact on the surveyed students' confidence levels in their reverse engineering abilities.

### 4.2 Effect on Student Motivation

Lastly, our pre- and post-game surveys asked students to assess their motivation for learning reverse engineering. For RQ2, our null hypothesis is that students would not experience a significant change in motivation in learning reverse engineering as a result of the BinaryGame lab. We reject the null hypothesis when the $p$-value is below 0.05.

Figure 2 shows the distribution of scores on the pre- and post-surveys. The y-axis indicates the percentage of students who indicated that they were: 1 ("very unmotivated"), 2 ("somewhat unmotivated"), 3 ("neither unmotivated or motivated"), 4 ("somewhat motivated") or 5 ("very motivated") to learn reverse engineering on the pre- and post-surveys. The pre-survey average was

3.419, while the post-survey average was 3.613. While 48.4% of students indicated that they were motivated to learn reverse engineering on the pre-survey, 71% of the students indicated some level of motivation on the post-survey. While students did experience a modest increase in motivation for learning reverse engineering between the pre- and post-surveys, the difference between the means was not statistically significant ($p = 0.2805$). Therefore, we accept the null hypothesis for RQ2.

We asked some additional questions of students on the post-survey to gauge the perceived usefulness of the BinaryGame to learn reverse engineering. While 81.27% of our students rated the game as being "useful" or "very useful" for learning Arm assembly, an additional 12.5% felt neutral about the game. Students who liked the game mentioned that the *"difficulty scales well"*, and that it was a *"great start"* for learning reverse engineering in Arm assembly. However, students also noted that *"the levels past level 3 felt almost the same"*, with one student complaining that problems could be easily solved by *"putting a break-point at the cmp instructions and reading out what the inputs had to be"* We feel this is valid criticism; it *is* hard to create meaningful reverse engineering exercises. However, we feel heartened that many students appreciated the level of difficulty.

## 5 Conclusion

This paper focused on introducing an unfamiliar (Arm) assembly language to a population of students who were already exposed to reverse engineering concepts. Our results suggest that the BinaryGame not only helps increase student confidence in using Arm assembly, but in their existing skillset as well; by seeing the same concepts applied to a novel architecture, students gained confidence in their general abilities to reverse engineer.

Future assessment will focus on the effectiveness of the BinaryGame to introduce reverse engineering concepts to students who are largely unfamiliar with any assembly language, and to include larger populations of students. The BinaryGame is still being actively developed, and we continue to refine the complexity of levels and modalities for deploying the game. Community contributions are welcome!

## 6 Acknowledgement

This project is supported by the National Science Foundation (DUE - 2141814). The views expressed in this article are those of the author and do not reflect the official policy or position of the Department of the Army, Department of Defense or the U.S. Government.

## References


[1] John Aycock et al. "Exercises for Teaching Reverse Engineering". In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018. Larnaca, Cyprus: Association for Computing Machinery, 2018, pp. 188–193. ISBN: 9781450357074. DOI: 10.1145/3197091.3197111. URL: https://doi.org/10.1145/3197091.3197111.

[2] Fabrice Bellard. "QEMU, a fast and portable dynamic translator." In: *USENIX annual technical conference, FREENIX Track*. Vol. 41. 46. California, USA. 2005, pp. 10–5555.

[3] Logan Brown, Gavin Hayes, and Tejas Rao. "Reinventing Bomblab". Major Qualifying Project. Department of Computer Science, Worcester, PA, USA: Worcester Polytechnic Institute, 2017. URL: https://digital.wpi.edu/downloads/s7526g02j4.

[4] Randal E Bryant, O'Hallaron David Richard, and O'Hallaron David Richard. *Computer systems: a programmer's perspective*. Vol. 2. Upper Saddle River, NJ USA: Prentice Hall, 2003.

[5] Randal E Bryant and David R O'Hallaron. "Introducing computer systems from a programmer's perspective". In: *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*. 2001, pp. 90–94.

[6] Andreea Cotoranu and Li-Chiou Chen. "Using Raspberry Pi as a Platform for Teaching Cybersecurity Concepts". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019, pp. 1237–1237.

[7] Sandra Gorka et al. "Tempting High School Students into Cybersecurity with a Slice of Raspberry Pi". In: *Journal of The Colloquium for Information Systems Security Education*. Vol. 8. 1. 2020, pp. 4–4.

[8] Stylianos Karagiannis et al. "PocketCTF: A fully featured approach for hosting portable attack and defense cybersecurity exercises". In: *Information* 12.8 (2021), p. 318.

[9] Jalal Kawash et al. "Undergraduate Assembly Language Instruction Sweetened with the Raspberry Pi". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE '16. Memphis, Tennessee, USA: Association for Computing Machinery, 2016, pp. 498–503. ISBN: 9781450336857. DOI: 10.1145/2839509.2844552. URL: https://doi.org/10.1145/2839509.2844552.

[10] Benjamin Levandowski, Debbie Perouli, and Dennis Brylow. "Using embedded xinu and the raspberry pi 3 to teach parallel computing in assembly programming". In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2019, pp. 334–341.

[11] Suzanne J. Matthews, Tia Newhall, and Kevin C. Webb. *Dive into Systems*. 1st ed. Free version available at: http://www.diveintosystems.org. San Francisco, CA: No Starch Press, June 2022. ISBN: 978-1718501362.

[12] Suzanne J. Matthews, Tia Newhall, and Kevin C. Webb. "Dive into Systems: A Free, Online Textbook for Introducing Computer Systems". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 1110–1116. ISBN: 9781450380621. DOI: 10.1145/3408877.3432514. URL: https://doi.org/10.1145/3408877.3432514.

[13] Elizabeth Patitsas and Daniel Levy. "Dr. Horrible's Fork Bomb: A Lab for Introducing Security Issues in CS2". In: *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '13. Canterbury, England, UK: Association for Computing Machinery, 2013, p. 318. ISBN: 9781450320788. DOI: 10.1145/2462476.2465577. URL: https://doi.org/10.1145/2462476.2465577.

[14] Simon Segars. *Arm Partners Have Shipped 200 Billion Chips*. Oct. 2021. URL: https://www.arm.com/blogs/blueprint/200bn-Arm-chips.

[15] Narasimha Shashidhar and Peter Cooper. "Teaching malware analysis: The design philosophy of a model curriculum". In: *2016 4th International Symposium on Digital Forensic and Security (ISDFS)*. 2016, pp. 119–125. DOI: 10.1109/ISDFS.2016.7473529.

[16] Christopher Stricklan and TJ OConnor. "Towards Binary Diversified Challenges For A Hands-On Reverse Engineering Course". In: *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE '21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 102–107. ISBN: 9781450382144. DOI: 10.1145/3430665.3456358. URL: https://doi.org/10.1145/3430665.3456358.

[17] Adam H Villa. "Hands-on computer security with a Raspberry Pi". In: *Journal of Computing Sciences in Colleges* 31.6 (2016), pp. 4–10.

[18] Michael Wirth and Judi McCuaig. "Making Programs With The Raspberry Pi". In: *Proceedings of the Western Canadian Conference on Computing Education*. WCCCE '14. Richmond, BC, Canada: Association for Computing Machinery, 2014. ISBN: 9781450328999. DOI: 10.1145/2597959.2597970. URL: https://doi.org/10.1145/2597959.2597970.