CS260 Binary Analysis
Daniel Stinson-Diess
Sean Richardson
Jason Zellmer

Project Proposal

1) Problem Statement (Motivation)

Many applications produced by corporations are closed source, meaning only the binary executable is distributed. With these applications security patch cycles can be as bad as a month to fix or never having updates rolled out meaning the user of the application is indefinitely running a vulnerable version without having a viable way of fixing the source and compiling it. The solution to this is to disassemble the binary and apply the security patches in the assembly code and reassemble this into a functioning binary. This approach has been worked on in two core papers: *Reassembleable Disassembling*, [3] and later *Ramblr: Making Reassembly Great Again* [1]. There are a few limitations to these approaches:

1. Control flow graph (CFG) Recovery: There has been a lack of testing when a CFG can't be recovered from compilers that add inline data into executable regions. This is done with the MSVC toolchain [4].
2. Infeasibility of Content Classification: Despite being shown as an issue that can't be done statically, [5] using dynamic techniques can be defeated with techniques like shifting of binary base addresses.
3. Abnormal Binary Techniques: Some binaries employ pointer encryption and decryption which causes problems with the symbolization techniques.
4. Targeting of Specific Architecture: These works have been focused on Intel's x86 and x86_64 platforms. These are the most common platforms for desktop computing but there has been a shift to RISC style architectures for reduced power usage and they are favored for their simpler ISA's. There has been progress in identifying the challenges by Ha et al in [6].

Of the problems identified, 1-3 are research oriented tasks where new and more novel techniques should be developed to addressed, whereas 4 is a more pressing engineering problem. We purport that challenge 4 is pressing because with the rise of IoT and "smart" devices becoming ubiquitous in households is made possible due to low levels of continuous support; this implies that software updates including these vital security patches.

2) Objective / Expected Outcome

Our initial goal is to evaluate some of the existing tools which perform reassembly on disassembled binaries to better understand their limitations and shortcomings. We would like to first recreate their approach of using Ramblr on the 106 real-world programs on Linux x86 and x86-64 from CoreUtils and the 143 programs collected from the Cyber Grand Challenge Qualification Event. We would like to implement this as a first step to better understand their approach and to ensure we can recreate their results.

Additionally, we would also like to find cases where the tools cannot properly disassemble and reassemble the code for a binary executable. Specifically, we would like to see if we can find identify the cases where their approach fails in hopes that, once they are identified, we might be able to better why they fail and possibly pursue these as further developments to their approach. Our overall goal is to try and better understand the approach that is taken by these tools and how the heuristics that they use affect the ability to modify the code along with attempt to further improve on their results by exploring the cases that fail. Another goal we hope to attempt to explore is to try to recreate the results they obtained from the Uroboros paper that they used to compare their results. We understand that this may not be feasible in a shorter project, but we would like to try to include it as part of the performance benchmarks in this project.

3) Approach / Evaluation Metrics / Dataset / etc

Datasets:

1. DARPA CGC + Linux CoreUtils
2. Coreutils compiled with MSVC

1. Validate the original results of the paper and collect more specific information on each case failure

For a data set we can test the GNU coreutils and also binaries from the DARPA Cyber Grand Challenge. As an evaluation metric we seek to recreate the test results and look at whether the reassembly process succeeded or failed. According to the paper, both Ramblr and Ramblr Fast do not cause generation failures when used on the binaries. However Ramblr has one test failure on the CGC binaries with Ofast optimization, while Ramblr Fast had 10 test failures with different optimization levels. We could then examine each of the failed test cases, dividing them up amongst the group, to see if we can determine a common case as to why each failed and use it as an avenue of improvement.

2. Attempt to recover the Control Flow Graph from executables compiled with MSVC.

In the Ramblr paper under section XI the authors mention some of the potential weak points that the reassembler has. The first is that static content classification is infeasible, so rebasing a binary's base address during linking can cause address collisions which lead to reassembly failure. The second is the recovery of the control flow graph. The recovery techniques used will work differently on binaries which have different features. The method used by Ramblr was tested on Linux executables which did not have inlined data. The inling of data into executable regions of a binary can interfere with content classification. Data bytes may be mixed in with instructions in a code section, which can cause false positive instructions if the data bytes are mistakenly interpreted as the start of a multibyte instruction.

The authors state that they believe that their CFG recovery and disassembly technique should work on these type of binaries with content classification, but more work is needed in this direction. For this reason we want to try and test Ramblr on a set of GNU Coreutils that have been compiled with a MSVC compiler to see how this affects the control flow graph recovery. The results of this can then be compared to the known results of the reassembler used on the set of Linux Coreutils compiled with either GCC or Clang.

We intend to work on getting the Ramblr system to run at the same time, then dividing the failed test cases amongst group members in an attempt to find commonality within the test cases that may reveal underlying causes that might yield the capability for improvements upon the current approach. After we have collected this set of data we would like to expand on the successes and failures section of the original paper with the added information about the specific extensions that caused failures and why. Ideally this would allow identifications of any trends or patterns that consistently cause failure and suggest potential spots for improvement.

References

[1] R. Wang, Y. Shoshitaishvili, A. Bianchi, A. Machiry, J. Grosen, P. Grosen, C. Kruegel, G. Vigna, “Ramblr: Making Reassembly Great Again”, Network and Distributed System Security Symposium, January 2017

[2] D. Ha, W. Jin and H. Oh, “REPICA: Rewriting Position Independent Code of ARM,” in IEEE Access, Vol. 6, pp. 50488-50509, 2018.

[3] S. Wang, P. Wang, and D. Wu, “Reassembleable Disassembling,” in 24th USENIX Security Symposium (USENIX Security ’15).USENIX Association, 2015, pp. 627–642.

[4] D. Andriesse, X. Chen, V. van der Veen, A. Slowinska, and H. Bos, “An In-Depth Analysis of Disassembly on Full-Scale x86/x64 Binaries,” in 25th USENIX Security Symposium (USENIX Security ’16). Austin, TX: USENIX Association, 2016, pp. 583–600.

[5] R. N. Horspool and N. Marovac, “An Approach to the Problem of Detranslation of Computer Programs,” Computer Journal, vol. 23, no. 3,pp. 223–229, 1980.

[6] D. Ha, W. Jin and H. Oh, "REPICA: Rewriting Position Independent Code of ARM," in *IEEE Access*, vol. 6, pp. 50488-50509, 2018. doi: 10.1109/ACCESS.2018.2868411