Poster Abstract: Shingled Graph Disassembly: Finding the Undecidable Path*

Richard Wartell[1], Yan Zhou[2], Kevin W. Hamlen[2], and Murat Kantarcioglu[2]

[1] Mandiant

[2] Computer Science Department, The University of Texas at Dallas

{rhw072000,yan.zhou2,hamlen,muratk}@utdallas.edu

Abstract. A probabilistic finite state machine approach to statically disassembling x86 executables is presented. It leverages semantic meanings of opcode sequences to infer similarities between groups of opcode and operand sequences. Preliminary results demonstrate that the technique is more efficient and effective than comparable approaches used by state-of-the-art disassembly tools.

1 Introduction

Static disassembly of binaries for Intel-based architectures is particularly challenging because of the heavy use of variable-length, unaligned instruction encodings, dynamically computed control-flows, and interleaved code and data. Most state-of-the-art disassembly tools, such as IDA Pro [1], decode instructions by recursively traversing the static control flow of the program, thereby skipping data bytes that may punctuate the code bytes. However, not all control flows can be predicted statically. Recent work has introduced a machine learning-based disassembler [2] developed using modern statistical data compression models. The experimental results demonstrate substantial improvements over IDA Pro's traversal-based approach, but it has the disadvantage of extremely high memory usage.

This poster paper presents an improved machine learning-based technique that uses a finite state machine with transitional probabilities to infer likely execution paths through a sea of bytes. Our disassembler is simple, effective, and much more efficient than alternative approaches with comparable accuracy.

2 Disassembler Design

Our disassembler includes three primary components: (1) a *shingled disassembler* that recovers the (overlapping) building blocks (*shingles*) of all possible valid execution paths, (2) a finite state machine trained on binary executables, and (3) a graph disassembler that traces and prunes the shingles to output the maximum-likelihood execution path.

Shingled Disassembler: The shingled disassembler conservatively considers every byte as a potential instruction starting point, eliminating paths that reach invalid opcodes. This is a major benefit of the approach, since the shingled disassembly encodes a superset of all the possible valid disassemblies of the binary.

* The research reported herein was supported in part by NSF award #1054629, AFOSR award FA9550-10-1-0088, and ARO award W911NF-12-1-0558.

Opcode State Machine: The state machine is constructed from a large corpus of pre-tagged binaries, disassembled with IDA Pro v6.3. The byte sequences of the training executables are used to build an opcode graph, consisting of opcode states and transitions from one state to another. For each opcode state, we label its transition with the probability of seeing the next opcode in the training instruction streams.

Maximum-Likelihood Execution Path: We find the maximum-likelihood execution path by tracing the shingled binary through the opcode finite state machine. At every receiving state, we check which preceding path (predecessor) has the highest transition probability. The transition probability of each valid shingle-path $s \in S$ resulting in trace $r_0, \ldots, r_i, \ldots, r_k$ is:

$$Pr(s) = Pr(r_0)Pr(r_1) \cdots Pr(r_i) \cdots Pr(r_k)$$

and the optimal path is $s^* = \arg\max_{s \in \mathcal{S}} Pr(s)$.

3 Evaluation

Our disassembler was developed in Windows using Microsoft .NET C#, and was tested on an Intel Xeon processor with six 2.4GHz cores and 24GB of physical RAM. We disassembled 24 difficult binaries with very positive results. The preliminary results show that our disassembler identifies 99.9% of instructions that IDA Pro labels as code while avoiding its mistakes—for example, misclassification of large, non-executed data blocks as code; confusion of common opcode sequences with code addresses; and omission of various direct branch instructions. Furthermore, our disassembler runs in linear time in the size of the input binary. It is therefore increasingly faster than IDA Pro as the size of the input grows.

4 Conclusion

We present an extremely simple yet highly effective static disassembly technique using probabilistic finite state machines. Compared to the current state-of-the-art IDA Pro, our disassembler runs in linear time in the size of a given binary. We achieve both greater efficiency and greater accuracy than IDA Pro. More details can be found in our technical report [3].

References

1. Hex-Rays: The IDA Pro disassembler and debugger. `www.hex-rays.com/idapro`
2. Wartell, R., Zhou, Y., Hamlen, K.W., Kantarcioglu, M., Thuraisingham, B.: Differentiating code from data in x86 binaries. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD). Volume 3. (2011) 522–536
3. Wartell, R., Zhou, Y., Hamlen, K.W., Kantarcioglu, M.: Shingled graph disassembly: Finding the undecideable path. Technical Report UTDCS-12-13, The University of Texas at Dallas (2013)