REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
23-01-2018	Final Report	1-Aug-2015 - 31-Jul-2016

4. TITLE AND SUBTITLE

Final Report: Vulnerability detection using data-flow graphs and SMT solvers

5a. CONTRACT NUMBER
W911NF-15-1-0419

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES

University of Delaware
210 Hullihen Hall

Newark, DE 19716 -0099

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
67536-CS.1

12. DISTRIBUTION AVAILIBILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John Cavazos
UU	UU	UU	UU		19b. TELEPHONE NUMBER: 302-831-6052

RPPR Final Report
as of 24-Jan-2018

Agency Code:

Proposal Number: 67536CS **Agreement Number: W911NF-15-1-0419**
INVESTIGATOR(S):

Name: Jr. John Cavazos
Email: cavazos@udel.edu
Phone Number: 3028316052
Principal: Y

Organization: **University of Delaware**
Address: 210 Hullihen Hall, Newark, DE 197160099
Country: USA
DUNS Number: 059007500 EIN: 516000297
Report Date: 31-Oct-2016 Date Received: 23-Jan-2018
Final Report for Period Beginning 01-Aug-2015 and Ending 31-Jul-2016
Title: Vulnerability detection using data-flow graphs and SMT solvers
Begin Performance Period: 01-Aug-2015 **End Performance Period:** 31-Jul-2016
Report Term: 0-Other
Submitted By: Jr. John Cavazos Email: cavazos@udel.edu
Phone: (302) 831-6052
Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: **STEM Participants:**

Major Goals: Automated vulnerability detection in reverse engineered code.

Accomplishments: Please see "Upload" section for accomplished goals.

Training Opportunities: Nothing to Report

Results Dissemination: Nothing to Report

Honors and Awards: Nothing to Report

Protocol Activity Status:

Technology Transfer: Nothing to Report

PARTICIPANTS:

Participant Type: PD/PI
Participant: John Cavazos
Person Months Worked: 1.00 **Funding Support:**
Project Contribution:
International Collaboration:
International Travel:
National Academy Member: N
Other Collaborators:

Participant Type: Graduate Student (research assistant)
Participant: William Killian
Person Months Worked: 9.00 **Funding Support:**
Project Contribution:
International Collaboration:
International Travel:
National Academy Member: N

Other Collaborators:

Participant Type: Graduate Student (research assistant)
Participant: Robert Searles
Person Months Worked: 9.00 **Funding Support:**
Project Contribution:
International Collaboration:
International Travel:
National Academy Member: N
Other Collaborators:

Improved Vulnerability Detection for Cybersecurity in Army Computing Systems

Final Report

Army Research Office
Grant Number: W911NF-15-1-0419

PI: John Cavazos
Computer and Information Sciences Department
University of Delaware

This final report discusses progress on research made by the PI and his students on the research funded by this grant. Prof. John Cavazos and two Ph.D. students, William Killian and Robert Searles, were funded by the grant.

Motivation and Overview

Vulnerabilities in software need identified quickly and correctly. Developers rarely develop with consideration for eliminating vulnerabilities in source code. Source code is not always available for analysis; the code may be closed-source or contain market secrets. We introduce a framework for vulnerability detection of binaries to address these concerns. The framework is modular and pipelined to allow scalable analysis on distributed systems. Our vulnerability detection framework employs machine learning techniques. By using machine learning, the framework is quickly able to predict and identify vulnerabilities with not only existing vulnerabilities, but also with new vulnerabilities. Many machine learning algorithms are also resistant to obfuscation and noise. When considering binary files, this allows the framework to process optimized and non-optimized code, as well as ignore dead code contained in the binary file.

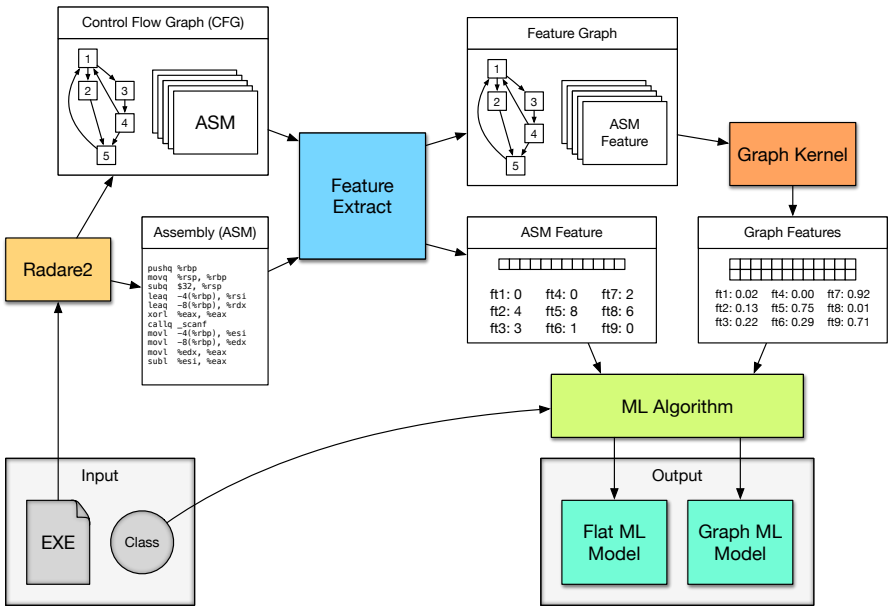

Figure 1: *The implemented learning framework for constructing binary vulnerability classification models. We generate two types of models based on the data format programs are represented as (flat vs. graph)*

Framework Design

We designed the framework to be modular to enable flexible reuse and extendibility. In its current form, our framework performs the following tasks: (1) convert the binary executable to a format that feature extractors can understand using Radare2. (1b) When doing graph-based learning, we use a graph kernel to map the control flow graph and features to a flat feature vector. (2) We then use the support vector machine learning algorithm to construct machine learning models. The known class of the binary executable is necessary to train a model.

Constructing Learners

Constructing a machine learning model to identify vulnerabilities in binary executables is challenging. Identifying buffer overflows requires additional restraints that need must be preserved: (1) The allocation of heap buffers do not always directly proceed the first use, and the size of the buffer may be unknown. (2) Because of these restrictions, window-based features may not necessarily yield accurate predictions. We leverage the pattern matching capabilities found with many machine learning algorithms to generate a model. Solving a classification problem makes support vector machine (SVM) learning algorithms favorable due to their distance maximizing property between classifications. Feature vectors provides some level of intuition on why the model classifies instances to classes. By extending our feature space to include graph-based program characterization, more complexity can be captured in the model and should yield better classification results.

Binary Analysis

Radare2 offers many independent command-line utilities that allow us to disassemble, analyze, and visualize entire datasets of binaries. We are currently able to emit two formats: (1) Call graph of the program as a JSON array where each node contains function names, arguments, types, and sizes. (2) An assembly dump of the program as an array of instructions. The first, graph-based,

format can encode more information about the structure of the program but this information encoding comes at a high cost for learning. The second, flat feature vector, format does not encode as much information but learning is much cheaper with a simplified data encoding format. Given one of these two formats, we can choose the instruction format that best fits the classification problem of identifying binary vulnerabilities. *Figure 2* shows a sample of feature extraction performed by Radare2 when converting assembly to a call graph with a feature vector.

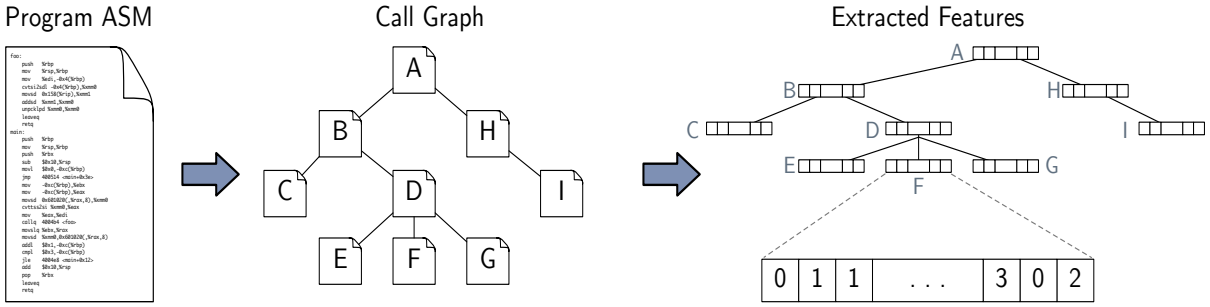

Figure 2: *Feature extraction of a binary using a call graph encoding for learning.*

We have three types of specification files that define which how features should be extracted from an assembly file: (1) Instruction type histogram corresponding to those that modify control flow, register values, and references to the stack pointer, (2) Instruction-level n-grams, and (3) Byte-level n-grams. Currently, we only use the n-gram feature extractors with vulnerability detection. Instruction-level n-grams enable pattern recognition based on the instruction order. Byte-level n-grams allow us to analyze the arguments (addresses, registers, and offsets) of the instructions to build a more advanced learner capable of recognizing more complex patterns. Our classifier should perform better using byte-level n-grams compared to instruction-level n-grams.

Identifying Vulnerabilities

We use two different types of n-gram feature vectors when constructing the classifiers. We use learning algorithms from the open source python library scikit-learn. We use the support vector machine (SVM) learning algorithm to construct a classifier and use k-fold cross validation to evaluate the generated model. There were four classes of vulnerabilities in our training set: (1) no vulnerability, (2) small buffer overflow, (3) medium buffer overflow, and (4) large buffer overflow. We highlight the results in *Figure 3*. Results show 60% accuracy with byte level n-gram and 70% accuracy with assembly level n-grams. The constructed instruction-level n-gram model can distinguish between benign and large overflow extremely well while the byte-level n-gram is able to differentiate between small and large buffer overflow.

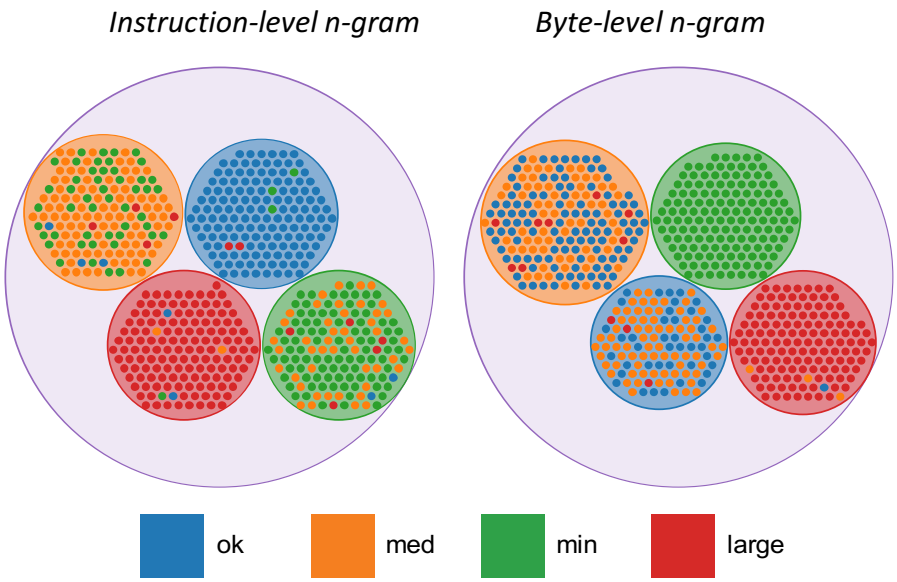

Figure 3: *Vulnerability classification is depicted as small circles. Prediction classes are shown as the encompassing circles. Predicting a vulnerability results in few false positives; however, predicting the size of the overflow is difficult. Additionally, there are many false-negatives*

Future Work

We plan to augment this work in the following ways: First, a call-graph representation captures what blocks of structured code call other blocks, but it cannot capture the relationship between other instructions. Radare2 can also emit a control-flow graph (CFG) which can capture the relationship between instructions. Using a more representative graph-based program representation should yield better learning models. Another component that can be expanded on this research is improving the end-to-end time of program ingestion to model generation. Many components of this framework are linearly dependent so pipelined-based acceleration can be applied. Additionally, the support vector machine (SVM) can be improved or replaced with a different machine learning algorithm which runs well on hardware accelerators (GPUs) such as deep neural networks. We also plan to define heuristics on what type of learners and features to use with identifying different vulnerabilities. These heuristics can help us tune model generation based on the objective classification. Finally, we plan to integrate online learners into the framework for discovering new vulnerabilities. Capturing program execution behavior would be necessary for this work, but the data flow graph (DFG) of the program, which can also be generated by Radare2, could assist with identifying *data invariants* for controlled execution and identifying unsafe areas in programs.