

Towards Security Defect Prediction with AI

In this study:

- We investigate the limits of the current state-of-the-art AI system for detecting buffer overflows and compare it with current static analysis tools.
- We develop a code generator, sa-bAbl, capable of producing an arbitrarily large number of code samples of controlled complexity.

Static analysis tools considered:

- Frama-C – “A collection of scalable, interoperable, and sound software analyses” for ISO C99 source code. Uses abstract interpretation.
- Clang – Based on symbolic execution and, by default, uses unsound heuristics such as loop unrolling to contend with state space explosion.
- Cppcheck – We believe it also uses unsound heuristics, though little has been published about its specific approach.
- Anonymized commercial tool – Well known to be unsound.

sa-bAbl generator

- Modeled after bAbl from Weston et al. 2015, [1]
- Intentionally very simple
 - Valid C code
 - Conditionals
 - Loops
 - Unknown values such as rand()
- Complements existing software assurance datasets for training AI
- Will be included in NIST SARD

A memory network based on Choi et al., 2016 [2]

Input:

- A program code $X [N \times J]$, consisting of N lines X_1, \dots, X_N , where each line X_i is a list of integer tokens w_i^1, \dots, w_i^J
- A query line $q [1 \times J]$, equal to one of the lines X_i encoding a buffer write

Embedding: We fix an embedding dimension d and establish two learnable embedding matrices E_{val} and E_{addr} , both of dimension $V \times d$. Letting A represent both E_{val} and E_{addr} , we encode each integer token twice, letting $Aw_i^j [1 \times d]$ be the w_i^j -th row of A . For $i = 1, \dots, N$, define $m_i [1 \times d]$ by

$$m_i = \text{Dropout}_{0.3} \left(\sum_{j=1}^J l_j \cdot Aw_i^j \right)$$

$$l_j^k = (1 - j/J) - (k/d)(1 - 2j/J)$$

We store the lines m_i encoded by E_{val} in a matrix $M_{\text{val}} [N \times d]$, and store the lines encoded by E_{addr} in a matrix M_{addr} . We embed the query line q by E_{addr} and store the result in $u^1 [1 \times d]$.

Memory search: For each “hop number” $h = 1, \dots, H$ in a fixed number of “hops” H :

$$p [N \times 1] = \text{softmax}(M_{\text{addr}}u^T)$$

$$o [1 \times d] = \sum_{i=1}^N p_i(M_{\text{val}})_i$$

$$(*) r [1 \times d] = R_h o$$

$$(*) s [1 \times d] = \text{Norm}_h(r)$$

$$u^{h+1} [1 \times d] = u^h + s$$

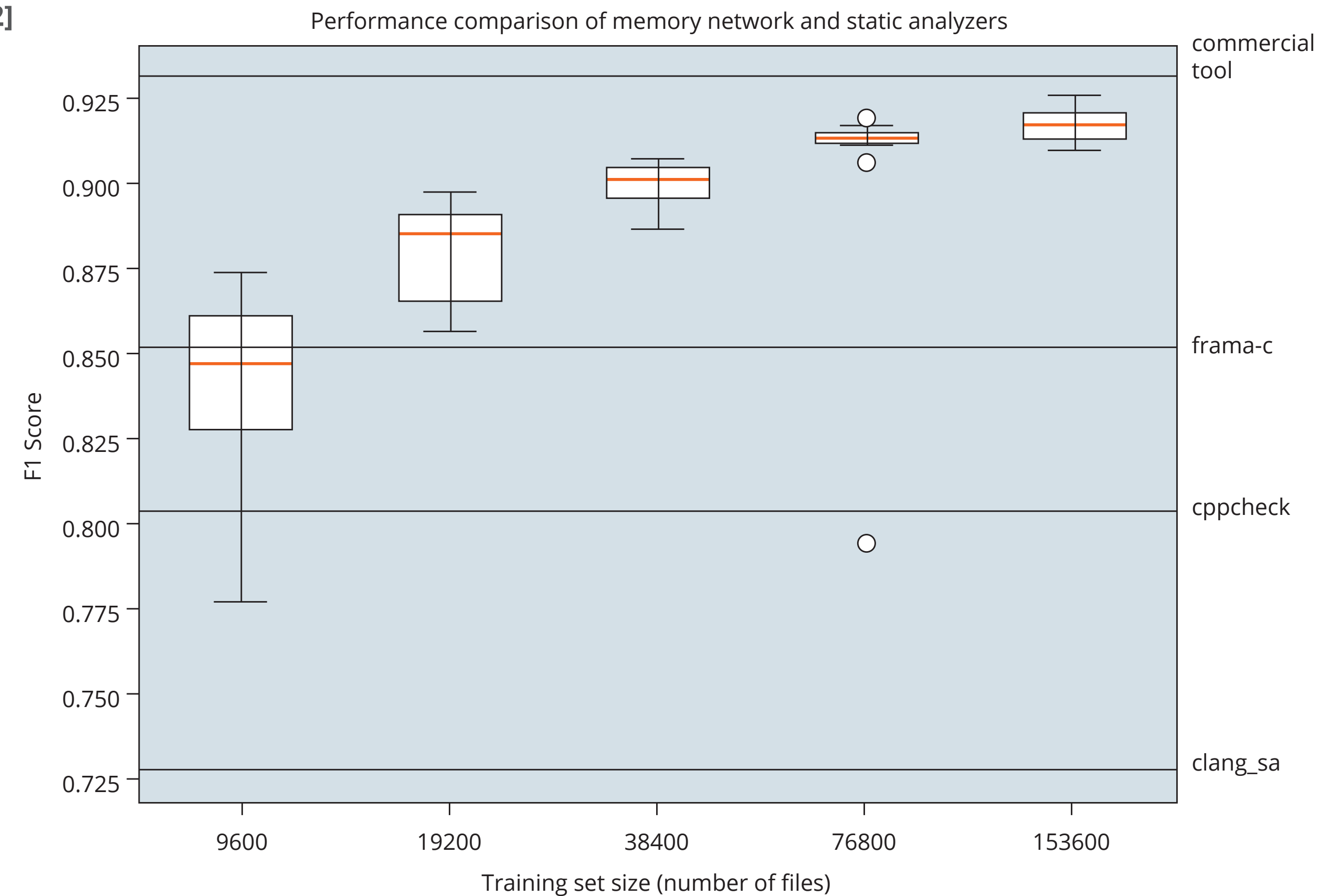
where $R_h [d \times d]$ is an internal learnable weight matrix

Classification:

$$\hat{y} [2 \times 1] = \text{softmax}(W(u^H)^T)$$

where $W [2 \times d]$ is a learnable weight matrix.

The forward pass is effectively an iterative inner-product search matching the current query line u^h , which changes with each processing hop, against each line m_i of the stored memory, which remains fixed.



Example Code

```

1 #include <stdlib.h> // OTHER
2 int main() // OTHER
3 { // OTHER
4     int entity_4; // BODY
5     char entity_8[11]; // BODY
6     int entity_3; // BODY
7     int entity_0; // BODY
8     entity_0 = 9; // BODY
9     entity_3 = rand(); // BODY
10    entity_4 = 42; // BODY
11    if (entity_3 < entity_0){ // BODY
12    } else { // BODY
13    entity_3 = 69; // BODY
14    } // BODY
15    while(entity_4 < entity_3){ // BODY
16    entity_4++; // BODY
17    } // BODY
18    int entity_9; // BODY
19    char entity_7[50]; // BODY
20    entity_8[entity_4] = 's'; // BUFWRITE_COND_UNSAFE
21    entity_9 = 75; // BODY
22    entity_7[entity_9] = 'S'; // BUFWRITE_TAUT_UNSAFE
23    return 0; // BODY
24 } // OTHER

```

We found:

- Static analysis engines have good precision but poor recall on our dataset.
- The state-of-the-art AI system can achieve similar performance to the static analysis engines, but it requires an exhaustive amount of training data to do so.

Our future work:

- Using representations of code that can capture appropriate scope information.
- Using deep learning methods that are able to perform arithmetic operations.

[1]. Weston et al., “Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks,” arXiv:1502.05698 [cs.AI], 19-Feb-2015.

[2] M.-J. Choi, S. Jeong, H. Oh, and J. Choo, “End-to-End Prediction of Buffer Overruns from Raw Source Code via Neural Memory Networks,” arXiv:1703.02458 [cs.SE], 07-Mar-2017.

Copyright 2018 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM18-1147