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-bAbI, 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-bAbI generator

- Modeled after bAbI 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 $X_1, \ldots, X_N$, where each line X_i is a list of integer tokens $w_i^1, \ldots, w_i^J$
- 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 $i = 1, \ldots, N$, 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_{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" $h = 1, \ldots, H$ in a fixed number of "hops" H:

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

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

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

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

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

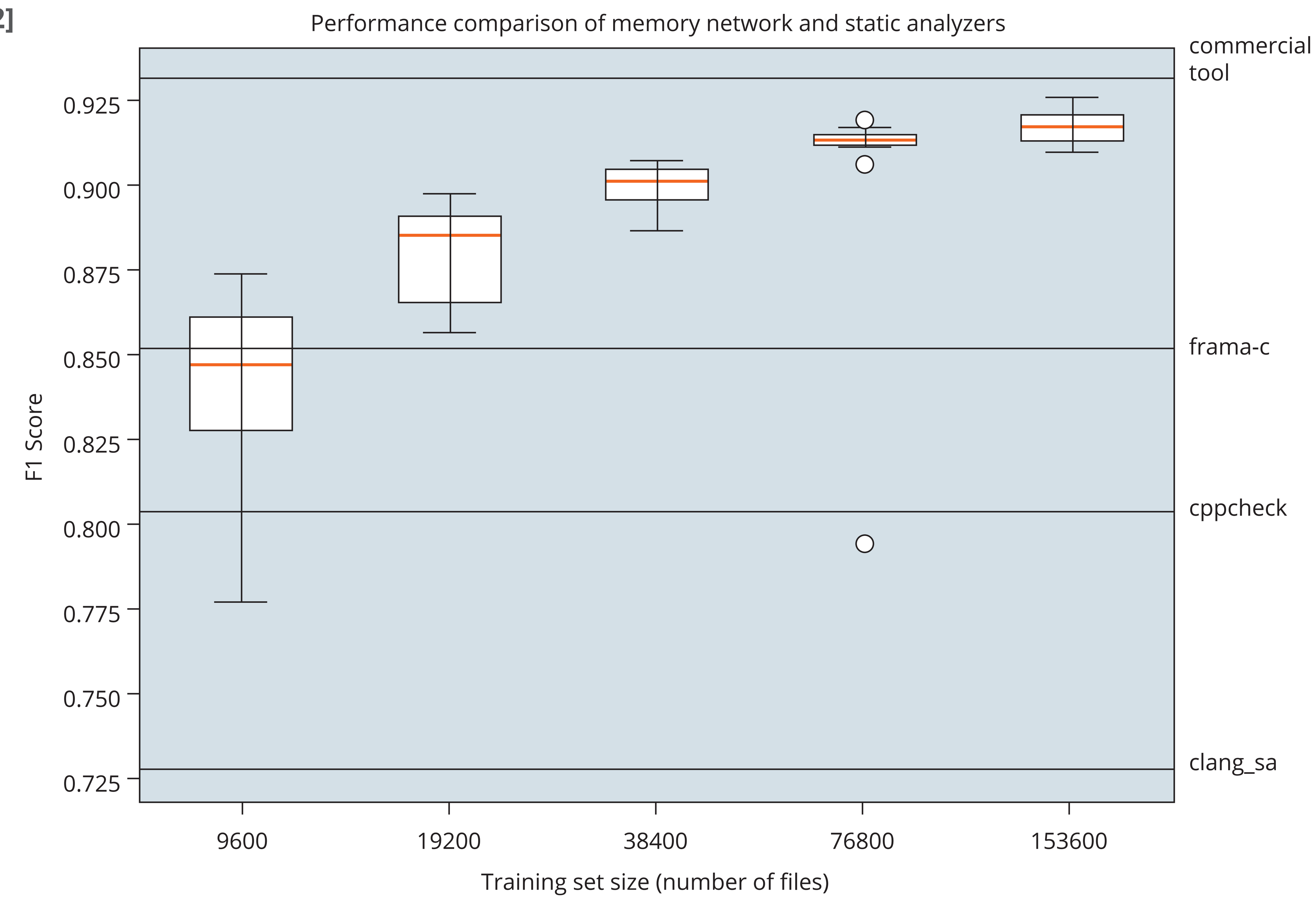
where R_h $[d \times d]$ is an internal learnable weight matrix

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

```
#include <stdlib.h>                 // OTHER
int main()                          // OTHER
{                                   // OTHER
    int entity_4;                   // BODY
    char entity_8[11];              // BODY
    int entity_3;                   // BODY
    int entity_0;                   // BODY
    entity_0 = 9;                   // BODY
    entity_3 = rand();              // BODY
    entity_4 = 42;                  // BODY
    if (entity_3 < entity_0){       // BODY
    } else {                        // BODY
    entity_3 = 69;                  // BODY
    }                               // BODY
    while(entity_4 < entity_3){     // BODY
    entity_4++;                     // BODY
    }                               // BODY
    int entity_9;                   // BODY
    char entity_7[50];              // BODY
    entity_8[entity_4] = 's';       // BUFWRITE_COND_UNSAFE
    entity_9 = 75;                  // BODY
    entity_7[entity_9] = 'S';       // BUFWRITE_TAUT_UNSAFE
    return 0;                       // BODY
}                                   // 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] J. 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.

Dr. Nathan VanHoudnos | nmvanhoudnos@cert.org