Recovering Meaningful Variable Names in Decompiled Code

Introduction

Conventional wisdom tells us that when a compiler transforms a program from source code to executable, some information is lost and cannot be recovered. For example, variable names are not included in a compiled executable, and have been assumed to be lost. Although state of the art decompilers can recover the presence of variables, they make attempt to recover their original names. Instead, they name the variables v1, v2, and so on. This is unfortunate since several studies have shown that programmers carefully select variable names to make the program easier to understand.

In this project, we showed that the conventional wisdom that variable names cannot be recovered is wrong. Specifically, we showed that variable names can largely be predicted based on the context of code in which they are used and accessed. We trained a neural network to predict variable names on a large corpus of C source code that we collected from GitHub.

Corpus

To generate our corpus, we scraped GitHub for projects written in C. We then automatically built 164,632 binaries from these projects and extracted 1,259,935 functions. For each function, we generated a corpus entry that consisted of the original source code with placeholder variables, as shown in the code figure to the right. Each corpus entry also included a mapping from placeholder variable to the original identifier in the source code and the decompiler's identifier.

We can exactly predict **74.3%** of variable names in decompiled executable code by training a neural network on a large corpus of C source code from GitHub.

```
void *file_mmap(int v1|fd|fd, int v2|size|size)
{
void *ptr|ret|buf;
ptr|ret|buf = mmap (0, v2|size|size, 1, 2, v1|fd|fd, 0);
if (ptr|ret|buf == (void *) -1)
{ perror ("mmap"); exit(1); }

return ptr|ret|buf;
}
```

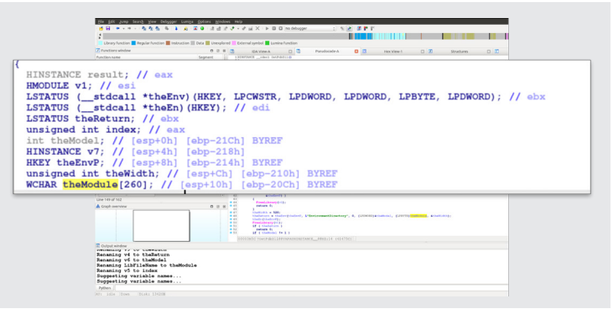
Key

Decompiled Original Recovered

Results

Experiment	Accuracy
Overall	74.3
Function in Training	85.5
Function not in Training	35.3

An important consideration when evaluating a solution based on machine learning such as ours is the construction of the training and testing sets. Each binary was randomly assigned to either the training or testing set. As in real reverse-engineering scenarios, library functions may be present in multiple binaries, and thus may be present in both the training and testing sets. To better understand the effect of this on our system, we partitioned our testing set into the set of functions that were also in the training set, and those that were not in the training set. As demonstrated in the table below, DIRE achieves 85.5% accuracy on functions it has been trained on, compared to 74.3% overall. On functions that it has not seen in training, it yields 35.3% accuracy.



Plug-in for Hex-Rays decompiler showing recovered names.

Bogdan Vasilescu | Dr. Edward Schwartz | info@sei.cmu.edu

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.

DM20-0862