Decompilation of Obfuscated Code with Machine Learning

Deniz Bölöni-Turgut [1] Advisor: Dr. Claire Le Goues [2] Graduate Mentor: Luke Dramko [2]

[1]Cornell University [2]Carnegie Mellon University

Introduction

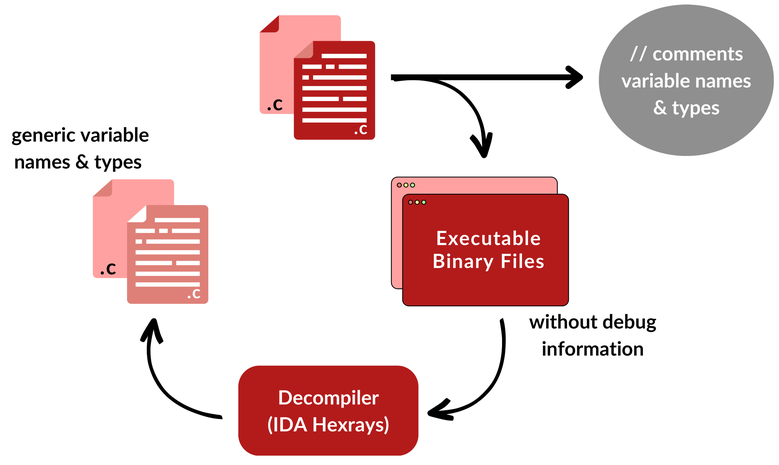

Figure 1. The process of decompiling compiled code back into high level language source code.

Motivation

- Decompilers are frequently used in the security sector to transform malware which is present on infected computers in its compiled, binary form.
- Readability of the decompiled malware is of utmost importance for human developers who combat the malware.

Research Contributions

- We generate a novel dataset of obfuscated and non-obfuscated binary files, which we hypothesize will better emulate actual malware which is often obfuscated by hand.
- We test the obfuscated dataset on DIRTY, a pre-trained model [2].
- We train and test a new model with the new obfuscated dataset to predict variable names and types of decompiled C code.

Decompiled Code Examples

Original Function and Struct

```
typedef struct point {
  float x;
  float y;
} pnt;

void fun() {
  pnt p1, p2;
  p1.x = 1.5;
  p1.y = 2.3;
  // ...
  use_pts(&p1, &p2);
}
```

Decompiled fun

```
void fun() {
  float v1[2], v2[2];
  v1[0] = 1.5;
  v1[1] = 2.3;
  // ...
  use_pts(v1, v2);
}
```

Figure 2. Example of struct implementation used in a C function and the decompiled function [2].

Original Function

```
int main(int argc, char *argv[])
{
  printf("Hello World!\n");
}
```

ADV Obfuscator Applied

```
int main(int argc, char *argv[])
{
  printf(OBFUSCATED("Hello World!\n"));
}
```

Obfuscated Function

Decompiled with HexRays

```
int main(int argc,char **argv)
{
 undefined8 local_16;
 undefined4 local_e;
 undefined2 local_a;

 local_16 = 0x6f57206f6c6c6548;
 local_e = 0x21646c72;
 local_a = 10;
 printf((char *)&local_16);
 return 0;
}
```

Figure 3. Simple example using the OBFUSCATED macro from ADV Obfuscator to obfuscate Strings, i.e. ”Hello World”.

Results

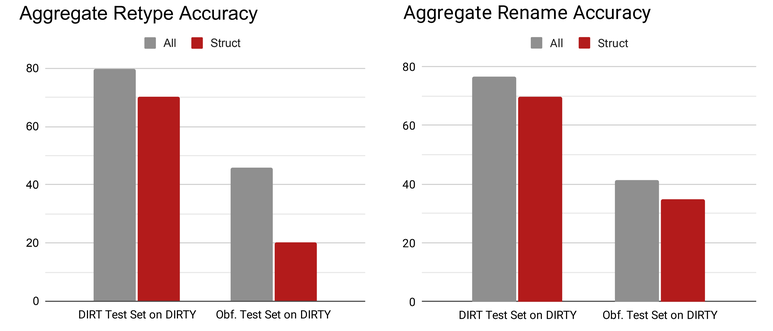

Table 1. Aggregate accuracies for renaming and retyping in % for different test sets on DIRTY Model.

	Rename Acc.		Retype Acc.	
Description	All	Struct	All	Struct
DIRT Test Set on DIRTY	76.6	69.7	79.6	70.3
Obf. Test Set on DIRTY	41.2	35.0	45.8	20.2

Table 2. Aggregate retyping accuracies in % for test set split by repository on model trained with obfuscated and non-obfuscated code.

	Overall		In Train		Not in Train	
Description	All	Struct	All	Struct	All	Struct
Full Test Set	56.4	52.4	97.1	96.4	53.7	42.8
Only Obf.	56.8	52.7	97.1	96.2	54.1	43.4
Only Non-Obf.	48.1	49.4	96.5	100	42.2	29.0

References

[1] Sebastien Andrivet. Advobfuscator, 2017.

[2] Qibin Chen, Jeremy Lacomis, Edward J. Schwartz, Claire Le Goues, Graham Neubig, and Bogdan Vasilescu. Augmenting decompiler output with learned variable names and types. In *31st USENIX Security Symposium*, Boston, MA, aug 2022.

[3] Christian Collberg. Tigress, 2017.

[4] Pascal Junod, Julien Rinaldini, Johan Wehrli, and Julie Michielin. Obfuscator-LLVM – software protection for the masses. In *Proceedings of the IEEE/ACM 1st International Workshop on Software Protection, SPRO'15, Firenze, Italy,*

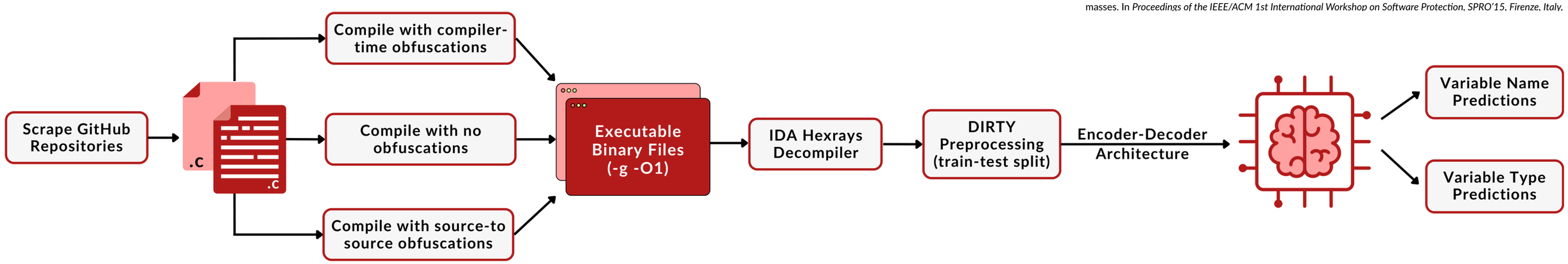

Figure 4. The Data Pipeline. Each binary was obfuscated with ADV Obfuscator [1], Tigress [3], and Obfuscator LLVM [4].