Reconstruction of Data Type in Obfuscated Binary Programs

Wei Ding\textsuperscript{ab}, ZhiMin Gu\textsuperscript{c}, Feng Gao\textsuperscript{a}

\textsuperscript{a} School of Computer Science Technology, Beijing Institute of Technology ,Beijing 100081,China
\textsuperscript{b} College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001,China

\texttt{orangeding@163.com,dingwei@bit.edu.cn}

Abstract- Recently, research community has advanced in type reconstruction technology for reverse engineering, but emerging with obfuscate technology, data type reconstruction is difficult and obfuscated code is easier to be monitored and analyzed by attacker or hacker. Therefore, we present a novel approach for automatic establish data type inference rules and reconstruct type from obfuscated binary programs using machine learning algorithm.

Keywords- Type reconstruction, Obfuscated Binary, Deobfuscation, Disassembly, Inference Rules

I. INTRODUCTION

Recent years have seen an increase in research of disassemble for reverse engineering and forensics. They offer variety techniques to lift low-level code to high level code. However, they are typically weak in reverse engineering data structures. Recover high-level program data type is a recurring step in process of reverse engineering and decompile. Source codes are translated down to operations on registers and one globally addressed memory region, a number of data types of high-level without symbol tables during compilation. In executable file there are no symbol tables and type information, therefore, reconstruction of data type depends on memory allocation access and how memory is used.

Data type reconstruction aim to transform binary code without data type symbol into meaning typing language easier to understand and recover explicit type of anonymous byte blocks in order to improve the readability and understandability and be easier to program analysis.

More and more research community is aware of importance and advances in the reconstruction of data structure have led to significant research, including well-known type inference algorithms Hindley-Milner\textsuperscript{[1]}, cartesian product\textsuperscript{[2]}, iterative analysis\textsuperscript{[3]}, abstract type inference\textsuperscript{[4]}. Yet they work on source codes, we need analysis binary code.

The most common type reconstruction approaches are based on static analysis techniques in binary code, like IDA Pro\textsuperscript{[5]}, OllyDbg\textsuperscript{[6]}. VSA\textsuperscript{[7]} (Value Set Analysis) attempts to identify location of like-variable and evaluate possible value set, which use a-loc to find possible value set and track value of data object. ASI\textsuperscript{[8]} (Abstract Structure Identification) tries to statically partition array and variable in memory block according to memory access. It use system call and famous library function information types, the types of called parameters are known, which are marked with according types and propagate them. Then, Balakrishnan\textsuperscript{[9]} combine VSA and ASI to identify simple structure, array and the nest of array and structure. But static analysis method is difficult in basic aggregation structure.

REWAEDS\textsuperscript{[10]} is a dynamic analysis method based on PIN analysis technology, which infer variable type by means of function parameters, return value and type signature instruction. In other words, it marks each location with timestamp type attribute and propagates it to other memory addresses, registers with program executed data flow, yet it can’t deal with control flowing limited to executed path and can’t deal with obfuscated code. Howard \textsuperscript{[11]} is complementary with Rewards, it is more powerful. It supplies assembler and debugger with data structure and type to relieve reverse engineering. It can reveal data structure layout according to memory access patterns and generates automatically debugger symbol. Howard can recover fields of aggregation structure, nested arrays, yet its results depend on runtime path coverage like any other dynamic analysis instruments.

TIE \textsuperscript{[12]} develops a novel type inference system based on type reconstruction rules, which can be applied in static and dynamic analysis. The core of TIE is to infer type according how the codes use, for example, in arithmetic operation, SF flag is detected and it can infer two operands are signed int.

Laika\textsuperscript{[13]} detects data syntactic structure through unsupervised learning during program execution, but accuracy of this technology is not enough for reverse engineer, it can identify part of obfuscated code with virus detectors, which is worthy for our reference.

Nevertheless, along with advance of data type reconstruction, a number of obfuscation techniques \textsuperscript{[14-16]} are very effective against state-of-the-art disassembles, preventing a substantial fraction of a binary program from being disassembled correctly and reconstruct data structure.
In [16], Linn and Debray describe a novel obfuscation technique that can be used to thwart binary static analysis. Their techniques are independent of and complementary to previous approaches to enhance software security by making it harder for an attacker to steal intellectual property. On the other hand, obfuscation technology could also be used by virus writers to hide malicious code such as Trojan Horses from virus scanners. The reason is that if relevant program structures were incorrectly extracted and identified obfuscated data type, malicious code could be classified as benign.

This paper presents a novel model for automatic establishment data type inference rules to reconstruct type from obfuscated binary programs using machine learning algorithms.

The remainder of this paper is organized as follows. Section II discusses background material on data type reconstruction and decision tree model. Section III discusses decision tree models for data type reconstruction rules.

II. KEY TECHNOLOGY AND DEFINITION

In this section we review background material on type inference rules, relevant definitions and decision tree model.

A. Inference Rules Definitions

Lattices. A lattice is a partial order among the values in a domain, which have a lowest bound element 0 and a highest bound element T. Lattices conclude two operations. One is the “join” operator ∨, which is the least upper bound, the other is the “meet” operator ∧, which is the greatest lower bound.

Inference Rules. General inference rules form is following:

\[ P_1, P_2, \ldots, P_n \rightarrow C \]

The upper of inference rule formula is the premises \( P_1, P_2, \ldots, P_n \). If all the premises are satisfied, then we can conclude results C. For example, 

\[ \Gamma \vdash t: S \quad S <: T \quad (T-Sub) \]

shows rule relation of type and subtype, in where each element in S is all in T, if S <: T.

Typing. Every term t, whether it is a variable, value or expression, has a type T. It is denoted by \( \Gamma \vdash t: T \), which means “t has type T under context \( \Gamma \).”

Subtyping[12]. Formula \( S <: T \), where is read as “Type S is a subtype of Type T”. Subtyping is transitive and reflexive, such as \( S <: U \land U <: T \Rightarrow S <: T \).

Array subtyping relation. \( S <: T \Rightarrow ArrayS <: ArrayT \), which indicates that subtyping S1 and subtypeing T1 are equivalent.

B. Decision Trees

Decision trees are common classifier algorithms, which are a technique for supervised machine learning, for example, learning an instruction from a training data set.

II. IDENTIFYING OF DATA TYPE OF OBfuscated BINARY

In this section we expand [17] algorithm to deconfuscation and use the decision tree to record the instruction information, so that we can correctly reconstruct data type.

A. Instruction Feature Extraction

It is necessary to identify features of the disassembled instructions so as to construct a decision tree for obfuscated binary disassembly. Different disassemble instructions are defined as different values and meaning, i.e. ‘opcode, source addressing mode, destination addressing mode’, which infer data type according instruction features. For each feature, the operation mnemonic and repeat prefix are used to features set. For each feature, the operation mnemonic and repeat prefix are used to features set.

<table>
<thead>
<tr>
<th>Algorithm Decision tree Algorithm Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECISION-TREE-LEARNING(T, default)</td>
</tr>
<tr>
<td>INPUTS: T, set of training cases</td>
</tr>
<tr>
<td>if T is empty return default;</td>
</tr>
<tr>
<td>else if all cases in T have the same class return the class;</td>
</tr>
<tr>
<td>else {</td>
</tr>
<tr>
<td>text=PARTITION(T);//belong to different class and attribute</td>
</tr>
<tr>
<td>if N-TEST(test)=1 return MAJORITY-CLASS(T);</td>
</tr>
<tr>
<td>else{</td>
</tr>
<tr>
<td>default= MAJORITY-CLASS(T);</td>
</tr>
<tr>
<td>tree=a new decision tree with the root test;</td>
</tr>
<tr>
<td>for each condition i{</td>
</tr>
<tr>
<td>Ti=elements in T which satisfy test:</td>
</tr>
<tr>
<td>Subtree= DECISION-TREE-LEARNING(Ti, default);</td>
</tr>
<tr>
<td>Add a branch with label i{ to the tree and the subtree;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

A collection of items \( S={ \{ S_1, \ldots, S_i \} } \) is given, \( S_i \in S \) happens with probability \( P_i \), the entropy of S is denoted as \( \text{Entropy}(S) = \sum_{i=1}^{n} p_i \log p_i \).

For a collection of training items S, which along with features \( F_i, \ldots, F_i \), the information obtained from a feature \( F \) is specified by \( \text{Gain}(S,F) = \text{Entropy}(S) - \sum_{i=1}^{n} (|S_i|/|S|)\text{Entropy}(S_i) \), where \( Vi \) means the value subset undertaken by features \( F_i \), similarly, \( S_i \) is the subset of S, which have the value v for features \( F_i \).

III. IDENTIFYING OF DATA TYPE OF OBfuscated BINARY

In this section we expand [17] algorithm to deconfuscation and use the decision tree to record the instruction information, so that we can correctly reconstruct data type.

A. Instruction Feature Extraction

It is necessary to identify features of the disassembled instructions so as to construct a decision tree for obfuscated binary disassembly. Different disassemble instructions are defined as different values and meaning, i.e. ‘opcode, source addressing mode, destination addressing mode’, which infer data type according instruction features. For each feature, the operation mnemonic and repeat prefix are used to features set with a vector of features for each operands. The features we consider for operands are shown in [12].
We aim to train sets and identify the sets of features and feature values of interest. As shown in Figure 1.

1) Lift BIL: At first, we translate the binary code into BIL (Binary Intermediate Language), which lift assembly by BAP[12] and can be easier to understand.

2) Training Data Sets: We can extract feature of instruction, operation, operands and other information, such as jmp, call. Operand types are register, pointer, memory address, immediate addresses, etc.

3) Classification and Report: Sequential report shows features of a fraction of sequence, which will help reconstruct data type.

B. Instruction n-Graph

In general, the common approach is to use adjacent instructions as construct n-Graph. However, it can not work well on transfer instruction, i.e. address that jmp or call instruction would transfer is not sequence. So we take a novel approach to set succes(P) denoted as the collections of all possible control-flow successors of an instruction P. For their possible successors are not known, we define succe(P)=\phi.

\[ \text{ng}(P) = \begin{cases} \{P\} & \text{if } n = 1 \text{ or succs}(P) = \phi \\ \bigcup_{I \in \text{InstAddrs}(P)} \text{ng}(I) \cup \text{ng}(J) & \text{otherwise} \end{cases} \]

Given a n-Graph S. \( \phi(S) = \{ \phi(\alpha) | \alpha \in S \} \), \( \phi(S) \) means feature vector set for element of S.

C. Construct Decision Tree

In vector paces, the similarity of features can be expressed by geometric. We use the appropriate feature vectors as training input to construct a decision tree. How to identify the obfuscated instruction is a difficult problem. Given an arbitrary binary code and without any other auxiliary information, we can only judge it with instructions that are executed at different addresses.

In order to gain total assembly, rather than an executed path instructions, we use gcc-compiler to gain binaries. A mechanism is presented specifically tailored against the tool implemented by Linn and Debray [16]. Therefore, we must recognize code addresses, the targets of indirect jumps and calls. For binary P, we extract the set of addresses of all the instructions in P, denoted as InstAddrs(P) using BAP[12]. Assume binary section B span interval of addresses A, then we compute the set of all n-graphs for P at the beginning of addresses A.

\[ \text{NG}(P) = \bigcup_{a \in A} \text{ng}(a) \]

Let addr(I) denote the address of an instruction I. For the set of n-Graph for P, we set flag \{1, 0\} to indicate whether the P is obfuscated. When flag is 1, then each instruction in the n-Graph is at an address in InstAddrs(p):

\[ \text{NG}(P | \text{flag} = 1) = \{ a | a \in \text{NG}(P) \} \]

\( \forall I \in a : \text{addr}(I) \in \text{InstAddrs}(P), \text{flag} = 1 \)

For flag=1, training inputs set is given by the set of feature vectors for good n-graphs:

\[ \text{PosInputs}(P) = \phi(\text{NG}(P | \text{flag} = 1)) \]

In the same way,

\[ \text{NG}(P | \text{flag} = 0) = \text{NG}(P) - \text{NG}(P | \text{flag} = 1) \]

NegInputs(P) = \phi(\text{NG}(P | \text{flag} = 0)) - \text{PosInputs}(P)

D. Constraints rules and Type Reconstruction Algorithm

The type constraints are computed using properties of assembly instructions.

Type reconstruction Algorithm

\[ \text{INPUTS: DECISION-TREE-LEARNING } T \]

\[ \text{OUTPUT: } \text{Type} \]

\[ \text{MATCH_{type}(T)}, \text{OUTPUT type} \]

IV. EXPERIMENTS

In experiments, to evaluate our approach, firstly, we use BAP[12] (Binary Analysis Platform) to gain BIL(Binary Intermediate Language ). An experiment is example as obfuscated binary code including control construct. Then we will compare its result with IDA Pro and new type inference. The following section source code and obfuscated assembly.
decision tree. For TIE, if call instruction is obfuscated, it can not detect the call instruction, so it cannot reconstruct the function type in other fun. We can take advantage of decision tree to deobfuscate and gain correct data type.

V. CONCLUSIONS

In this paper, we present a novel approach for automatic establishment of data type inference rules using machine learning algorithm. On the other hand, we can remove obfuscation during constructing the decision tree. Then we reconstruct type operating on the type lattice.

Our approach can be applied to transfer transformation obfuscation, however, it doesn’t work well on reconstruction of obfuscated data type. In the future we can devote to research how to reconstruct splitting arrays, matrix and so on.

ACKNOWLEDGMENT

This work has been supported by the National Natural Science Foundation of China 61370062, the National High Technology Research and Development Program of China 2012AA101608 and the Twelfth Five-year National Key Technology Support Program 2013BAD17B04 and Commonweal Research Project for Grain Industry 201313008.

REFERENCES


Wei Ding was born in Anhui province, China. After graduated from Henan Institute of Technology in 2002, she entered into Zhengzhou University, and gained Master Degree. Then she worked into Henan University of Technology and became a doctoral student of Beijing Institute of Technology in 2009. Her research area of interest is binary analysis and software security.