Enhance Information Flow Tracking with Function Recognition

Kan Zhou[1], Shiqiu Huang[1], Zhengwei Qi[1], Jian Gu[2], and Beijun Shen[1]

[1] School of software, Shanghai JiaoTong University
Shanghai, 200240, China
[2] Key Lab of Information Network Security, Ministry of Public Security
Shanghai, 200031, China
{zhoukan,hsqfire,qizhwei,bjshen}@sjtu.edu.cn,
gujian@mail.mctc.gov.cn

Abstract. With the spread use of the computers, a new crime space and method are presented for criminals. Computer evidence plays a key part in criminal cases. Traditional computer evidence searches require that the computer specialists know what is stored in the given computer. Binary-based information flow tracking which concerns on the changes of control flow is an effective way to analyze the behavior of a program. The existing systems ignore the modifications of the data flow, which may be also a malicious behavior. Function recognition is introduced to improve the information flow tracking, which recognizes the function body from the software binary. And no false positive and false negative in our experiment strongly prove that our approach is effective.

Keywords: function recognition, information flow tracking.

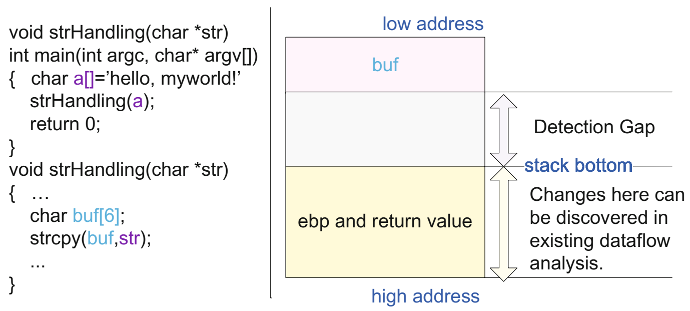
1 Introduction

With the spread use of the computers, the number of crimes with computers has been increasing rapidly in recent years. Computer evidence is useful in criminal cases, civil disputes, and so on. Traditional computer evidence searches require that the computer specialists know what is stored in a given computer. Information Flow Tracking (IFT) [7] is introduced and applied to our work to analyze the behavior of a program specially the malicious behavior.

Given program source code, there are already some techniques and tools that can perform IFT [5]. While the source code is not always available to the computer forensics, the techniques have to rely on the binary to detect the malicious behaviors [4]. Existing binary-based IFT systems ignore the modifications of the data flow, which may be also a malicious behavior [2]. Thus the Function Recognition (FR) [6] is applied to improve the accuracy of IFT.

We enhance IFT with FR for computer forensics. Our contributions include:

– We implement FR which recognize the functions from the software binary. A method of enhancing IFT in executables with FR is proposed.
– IFT with FR is applied into the computer forensics area.



```
void strHandling(char *str)
int main(int argc, char* argv[])
{   char a[]='hello, myworld!'
    strHandling(a);
    return 0;
}
void strHandling(char *str)
{   …
    char buf[6];
    strcpy(buf,str);
    ...
}
```

Fig. 1. An example of an overflow and the detection gap. In function *strHandling*, the size of the buffer is smaller than the size of the string assigned to it. When an overflow happens, it can be discovered only when it modified the value of *ebp* and the return address in the regular systems.

2 Motivation

When operations that results in a value greater than the maximum value which causes the value to wrap-around, the overflow happens as same as the one shown in Figure 1. This example is in C for clarity but our tool works with the binary code. The existing systems only concern on whether the control flow is modified or not, while the modifications of data flow are ignored. Thus a detection gap between the existing systems and our tool comes up just like Figure 1 shows. In our work, FR is introduced to address the detection gap. Take the Figure 1 as example, by comparing the lengths of the two parameters of *strcpy*, this kind of overflow can be easily detected.

3 Key Technique

3.1 Challenges

Memory Usage. The sheer quantity of functions and the size of the memory they occupy is a obstacle in FR [3]. If all versions of all libraries produced by all compiler vendors for different memory models are evaluated, the tens of gigabytes range is easily to wrap around. When we try to consider *MFC* and similar libraries, the size of the memory needed is huge. The requirement is beyond what the present personal computer can afford [3]. Thus a strategy is implemented to diminish the size of the information needed to recognize the functions. Not all the functions are recognized, only the functions related to the program behavior recognition are recognized and analyzed.

Signature Conflict. The relocation information of the call instruction will be replaced with “00”. If most of two functions are same except for one call instructions, the two functions will have the same signature, which we call signature conflict. To resolve this, the original general signature will be linked by the special symbol "&" with the machine code of callee functions, the addresses of which can be found in the corresponding *.obj* file. After that a new unique signature is generated.

Algorithm: *General Signature Extraction*

```
Input: the set of signature samples:
        A {f1,f2,f3,f4……},
Output: the general signature fr,

procedure GeneralExtract
fr=Func(f1,f2)
while (A≠NULL)
{
fr=Func(fr,fn);
n++;
}

procedure  Func(fr,fn)
GetSuperSequence(fr,fn);
//Get the most related general subsequence.
fr=RestructSig();
//Restructure the signatures to a new one.
```

3.2 Steps

Generation of General Signatures. The common parts of the machine code are extracted as a general signature. The algorithm using in our work has been presented below. The signature is separated to several subsequence with special symbols like "HHHH" and "&&". That the original signatures produced for different parameter types may have different lengths should been taken in account. Thus symbols like "00"s will be inserted into the shorter one where difference in successive bytes are detected, and different bytes of them are also replaced with "00"s to extract the common parts of original signatures. The procedure of the generation is as follows. Firstly a *.cpp* file that contains all the related functions are compiled by compilers with options, and a series of *.obj* files are generated. Then each *.obj* file will be analyzed and the machine codes of the functions are taken to generate the signatures.

Function Recognition with Signatures. When function calls (FCs) happen, FCs will be compared with the signatures, and the matched result is considered as an identified function. FC is identified by comparing the machine code with the signatures. It needs to match all these signatures to identify a FC.

3.3 Enhanced Information Flow Tracking

IFT usually tracks the information flow to analyze the modifications of the control flow by the input. Generally this technique labels the input from unsafe channels as "tainted", then any data derived from the tainted data are labeled

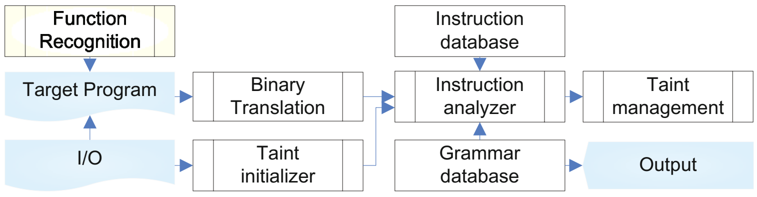


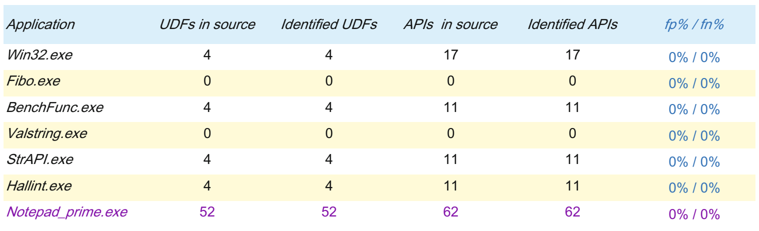
Fig. 2. The structure of the enhanced IFT system. Function Recognition is the module to recognize the functions. Taint Initializer initializes other modules and starts up the system. Instruction Analyzer analyzes the propagation and communicates with Taint Management module.

as tainted. In this way the behavior of a program can be analyzed and presented. General IFT focuses on the changes of the control flow, while the changes of data flow are always ignored. In our work, FR is introduced into IFT to solve the problem referred in the section 2, and the structure of the tool has been shown in Figure 2. Most of the structure is the same as the regular binary IFT. FR is the an important part different from other systems.

4 Experimental Results

4.1 Accuracy

To test our work, we have used 7 applications listed in Figure 3, also the results of FR are shown. All the functions appeared in the code can be divided into 2 types, *User-Defined Function* (*UDF*) and *Windows API*. In our experiments, the false positive rate and the false negative rate are both 0%. Experiment results prove that our work can recognize the functions accurately.



Application	*UDFs in source*	*Identified UDFs*	*APIs in source*	*Identified APIs*	*fp% / fn%*
Win32.exe	4	4	17	17	0% / 0%
Fibo.exe	0	0	0	0	0% / 0%
BenchFunc.exe	4	4	11	11	0% / 0%
Valstring.exe	0	0	0	0	0% / 0%
StrAPI.exe	4	4	11	11	0% / 0%
Hallint.exe	4	4	11	11	0% / 0%
Notepad_prime.exe	52	52	62	62	0% / 0%

Fig. 3. The results of the FR. *fp%* and *fn%* interprets the false positive rate and false negative rate. *UDFs in source* is the number of *UDFs* in the source code, and *Identified UDFs* shows the number of *UDFs* our tool identified. *APIs in source* and *Identified APIs* demonstrates the number of *APIs* in the source code and *APIs* identified by our tool respectively. *Notepad_prime* is a third-party program, which has the same functions and a similar interface with *Microsoft notepad.exe*.

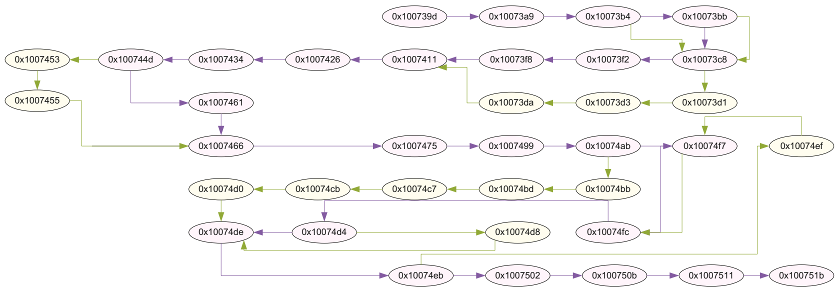


Fig. 4. A behavior graph that presents how *Microsoft notepad.exe* works. Different colors mean the different execution paths.

4.2 Behavior Graph

The behavior graph is a graph to illuminate the behavior of a program. It is useful for the computer specialists to understand the behavior of a program. Figure 4 shows the behavior graph of *Microsoft notepad*. Different colors of the ellipses and the lines mean they are from different execution paths separately. For example we track the information flow and get the purple path, while the malicious behaviors are not included in this path. Then we could change the input according to the behavior graph, and another path like the green one can be tracked and labeled in the graph.

4.3 Performance

Figure 5 demonstrates the performance of the tool when it is used in *SPEC CINT2006* applications. The results show that FR incurs the low overhead. *DynamoRIO*[1] is the binary translation our tool based on. In the results, FR does not significantly increase the execution time of IFT. The main reason is that we only track the functions related to the program behavior.

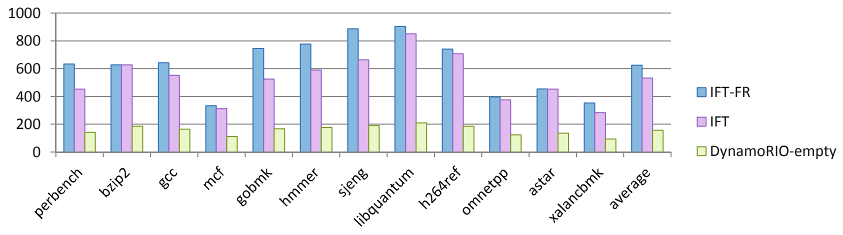


Fig. 5. The comparison of normalized execution time. "*DynamoRIO*-empty" bars show the *dynamoRIO* without any extra function. "*IFT*" bars interpret IFT system without FR. "*IFT-FR*" means the technique with FR.

[1] `http://dynamorio.org/`

5 Conclusion

In this paper we provide a way to analyze the behavior of a program to assist people to understand the program. The accuracy is an important issue in computer forensics, thus we implement the FR to improve IFT. FR is the strategy we applied to address the detection gap problem. And experimental results prove the method we implement the FR is effective. Zero false positive and zero false negative in our experiment illuminate the accuracy. Also the experiment results on the performance demonstrate that our tool is practical.

Acknowledgement. This work is supported by Key Lab of Information Network Security, Ministry of Public Security, and the Opening Project of Shanghai Key Lab of Advanced Manufacturing Environment (No. KF200902) and National Natural Science Foundation of China (Grant No.60773093, 60873209, and 60970107), the Key Program for Basic Research of Shanghai (Grant No.09JC1407900, 09510701600), and the foundation of inter-discipline of medical and engineering.

References

1. Baek, E., Kim, Y., Sung, J., Lee, S.: The Design of Framework for Detecting an Insiders Leak of Confidential Information. e-Forensics (2008)
2. Pan, L., Margaret Batten, L.: Robust Correctness Testing for Digital Forensic Tools. e-Forensics (2009)
3. Guilfanov, I.: Fast Library Identification and Recognition Technology, `http://www.hex-rays.com`
4. Song, D.X., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Saxena, P.: BitBlaze: A new approach to computer security via binary analysis. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 1–25. Springer, Heidelberg (2008)
5. Clause, J.A., Li, W., Orso, A.: Dytan: a generic dynamic taint analysis framework. In: ISSTA 2007 (2007)
6. Cifuentes, C., Simon, D.: Procedure Abstraction Recovery from Binary Code. In: CSMR 2000 (2000)
7. Clause, J.A., Orso, A.: Penumbra: automatically identifying failure-relevant inputs using dynamic tainting. In: ISSTA 2009 (2009)
8. Mittal, G., Zaretsky, D., Memik, G., Banerjee, P.: Automatic extraction of function bodies from software binaries. In: ASP-DAC 2005 (2005)