DRTaint: A Dynamic Taint Analysis Framework Supporting Correlation Analysis Between Data Regions

Pan Yang¹, Fei Kang¹*, Yuntian Zhao¹, Hui Shu¹

¹State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, Henan 450001, China
²Corresponding author’s email: weijinben@sixents.com, 39323690@qq.com

Abstract. Taint analysis is a common technology for software analysis, which is widely used in the region of information security. Aiming at the problem that the existing binary program dynamic taint analysis framework does not support "replay" analysis and the analysis efficiency is low, a taint analysis framework called DRTaint is proposed. DRTaint is a universal taint analysis framework that supports correlation analysis between data regions. Experiments show that DRTaint shows good performance advantages when performing taint analysis many times on one program. On the basis of DRTaint, this paper studies the malicious code behavior extraction method based on taint analysis, and the research finds that DRTaint can provide better support for malicious code behavior extraction.

1. Introduction

Taint analysis is a common technology for software analysis, which has been studied and applied in the region of information security. Typical applications include malicious code analysis [2][3], software vulnerability analysis [4][5], network attack detection [6][7]. Taint analysis technology includes static taint analysis and dynamic taint analysis. This paper mainly studies the dynamic taint analysis technology of binary program.

The taint analysis frameworks have two problems mainly. On the one hand, it can only perform one taint analysis on the basis of once execution, which cannot deal with the problems of multiple taint labels, multiple source points and multiple sink points, and cannot support "replay" analysis, such as libdft [8] and TaintRabbit [9]. On the other hand, the analysis granularity is mostly byte level, which improves the accuracy to a certain extent, but reduces the efficiency of taint analysis greatly. These problems have affected the wide application of taint analysis in practical software analysis.

Aiming at these two problems, this paper designs a taint analysis framework DRTaint (Data Regions Taint Analysis Framework) that supports correlation analysis between arbitrary data regions. DRTaint is composed of three parts: dynamic recording program execution trace, offline simulation to restore the data flow propagation process, and taint propagation analysis between data regions. On the basis of offline data flow, the analysis of taint propagation between any tainted data, any Source point and Sink point is realized. At the same time, the correlation analysis between data regions can quickly determine whether there is a correlation between two data regions, which can effectively improve the efficiency of taint analysis.

In this paper, we use DRTaint to analyze malicious code API association through taint analysis technology, and try to extract high-level semantics of malicious code. Experiments show that DRTaint has achieved good results in malicious code API association analysis and behavior extraction. The taint
The relationship between all APIs can be obtained on the basis of one execution of the program, which is more convenient and efficient than traditional taint analysis tools.

2. The design of DRTaint

In order to alleviate the deficiencies of the existing taint analysis framework and realize the taint analysis between any data regions, this paper designs and implements a taint analysis framework that supports correlation analysis between data regions, we call it DRTaint.

2.1. Overall design

Compared with the existing taint analysis framework, the design of DRTaint needs to meet three characteristics: (1) Support for any taint data analysis between any two locations during program running; (2) Support taint correlation analysis between data regions; (3) Support multiple "replay" analyses based on the program running once. At the same time, DRTaint also needs to provide a common taint analysis interface.

The DRTaint's structure is shown in Figure 1. It mainly includes three parts: (1) a lightweight online recording module, which records the necessary information during program running; (2) an offline data flow restoration module, which uses the recorded program running information completely restores the data flow propagation process of the program; (3) universal taint analysis interface module, which provides a universal taint analysis interface and supports taint correlation analysis between data regions.

2.2. Lightweight dynamic binary instrumentation record

The instruction-level instrumentation of common taint analysis frameworks will generate huge performance overhead, and the delay in program execution caused by the additional overhead may lead to inaccurate program execution results. In addition, in order to perform multiple taint analyses for one program, if using the existing taint analysis framework, the program needs to be executed multiple times, and the load base address, stack and heap memory address, program execution process, etc. may be different each time the program is executed. It is not easy to comprehensively analyze the results of multiple taint analysis, which brings a lot of inconvenience to the software analysis process.

In order to solve this problem, this paper designs a combination of lightweight dynamic binary instrumentation recording and offline restoration of data flow. This method not only reduces the problems that may be caused by the huge performance overhead in the process of program dynamic binary analysis, but also satisfies the need to avoid unnecessary data integration processes when the same program is analyzed multiple times.

In order to provide necessary data support for offline analysis, the content of online recording includes four parts, (1) the memory image of each module loaded by the program, recording this content is to meet the information necessary for the initial execution of the program; (2) basic block instruction content, although the process space after the program is loaded can reflect most of the code information, since Shellocede, SMC (Self Modifying code), and shelling are common in malicious code, it is necessary to record the instruction content when the basic block first appears; (3) program execution trace, within
the basic block, the program execution trace is determined, but between basic blocks, due to the existence of conditional jumps, indirect jumps, etc., the program execution trace is uncertain, recording the first address of each executed basic block can reflect the execution trace of the program; (4) API call information, system calls may occur in the API, making the result of the API call uncertain, therefore, it is necessary to record API calls.

DRTaint uses the dynamic binary instrumentation platform Pin[10] as an online recording tool. During the recording process, it uses three Pin instrumentation methods: Image instrumentation, recording program loading memory mirroring; basic block (BBL) instrumentation, recording basic block code and basic block execution sequence; RTN instrumentation, recording API call information.

2.3. Data flow restoration based on Unicorn

Based on the data provided by the lightweight instrumentation record, in order to restore the real data flow propagation process, DRTaint uses the recorded code and data to simulate the code in the basic block one by one according to the basic block execution sequence and obtain the complete data flow, the specific process is as follows.

(1) Using capstone [11] to disassemble the basic block code, carry out the instruction analysis and generate the basic block data flow "template". The data flow template is used to avoid instruction level instrumentation during simulation execution. It only needs to fill in the corresponding memory address through the memory hook when the specific address cannot be determined. For the instructions without memory operation, there is no need to analyze again. This framework implements more detailed x86 instruction analysis than libdft, supports FPU floating-point instructions and SSE extension instructions. The instruction data flow propagation process can be divided into four types: mov (data transfer), operate (inter data operation), clear (clear data state), exchange (data exchange). Table 1 shows the data flow template of a basic block.

| Instructions          | Type   | Source   | Destination |
|-----------------------|--------|----------|-------------|
| push ebp              | mov    | ebp      | mem         |
| mov ebp, esp          | mov    | esp      | ebp         |
| mov eax, dword ptr [ebp + 0xc] | mov | mem | eax |
|                       | operate| eax, ebp| eax         |
| push ebx              | mov    | ebx      | mem         |
| xor ebx, ebx          | clear  | (null)   | ebx         |
| sub eax, ebx          | operate| ebx, eax| eax         |
| je 0x418426ce         | (null) | (null)   | (null)      |

(2) Basic block simulation execution. The framework uses Unicorn [14] simulator to simulate code execution. When the basic block is simulated, the recorded program image is first loaded to the specified location of memory space; then the necessary stack space is allocated; finally, the basic block is simulated one by one according to the sequence of recorded basic blocks. In the simulation of basic block execution process, the memory read-write hook is used. A data flow "template" generated in the first step fills the memory address to the corresponding location one by one, and records the data generated during simulation to the file after the basic block simulation.

2.4. Analysis of taint propagation based on data region

Existing taint analysis technologies focus on analyzing the taint propagation relationship between bits or bytes, but this brings about a serious problem—excessive performance overhead. In addition, from the perspective of software reverse analysts, software analysis usually focuses less on fine-grained tainted relationships and more on relationships between memory regions. For example, when analyzing malicious code, analysts usually want to know whether the memory region file_buf of the ReadFile is related to the send_buf sent by the send, and they are not very concerned about a certain byte in file_buf
and correspondence of a byte in send_buf. Therefore, we put forward the idea of taint correlation analysis among data regions.

The data region refers to a set of memory (including registers) addresses. Generally speaking, there is a certain logical relationship within the data region, otherwise, it is meaningless to study the data region which is regarded by randomly selected. The data region can be divided according to the actual analysis object requirements. For example, the data region can be the startup parameters of the program, the I/O buffer (including standard input and output, file reading and writing, network sending and receiving, etc.), API parameters and return values, etc. Simply put, the data region is a buffer that we need to study when we analyze the program.

The association relationship between the data regions satisfies the transitivity, that is, if the data region A is associated with the data region B, and the data region B is associated with the data region C, then the data region A is associated with the data region C. As shown in Code 1, the array a0 is the Source data region A0, and a1 is the Sink data region A1. The two data regions are located in different memories. Analysis can find that there is an intermediate data region T, which is contained in T is the variable t. Obviously T is associated with A0, and T is also associated with A1, so A0 is associated with A1.

```c
int a0[] = {1,2,3,4};
int a1[] = {0,0,0,0};
int t;
t = a0[0] + 1;
a1[0] = t;
```

**Code 1. Data regions analysis example**

3. **Malicious code analysis based on DRTaint**

Common malicious code analysis tools, such as sandboxes, display a series of lengthy API call sequences, or perform data mining on API sequences by pattern matching. Such analysis methods cannot fully and accurately reflect the internal association of malicious code. Through taint analysis technology, you can accurately obtain the relationship between the input and output data between these API calls. Based on the design and implementation of DRTaint, this paper uses taint analysis technology to study the association of malicious code API calls, and uses the association relationship of API calls to extract malicious code behavior.

In the process of malicious code analysis, all the data regions which were used by API functions were marked different labels. Such as the function ReadFile, which declaration is shown in Code 2, on the analysis process, five parameters and one return value are analyzed as data regions in turn to find out which data regions affect the data regions involved in these parameters before function call, and which data regions are affected after function call. Through the data region taint propagation relationship, the API with input-output relationship can be associated to generate API call association diagram (as shown in Figure 2, 3 and 4). DRTaint supports the ability of data region taint analysis and "replay" analysis, which can well meet the needs of malicious code analysis. If use libdft to analyze a ReadFile call, the analyses need to be repeated six times. However, in the real analysis environment, the number of API calls generated by malicious code execution is very large, and most APIs have multiple parameters and one return value, the analyser needs to repeatedly load and execute the same malicious code many times, which inevitably takes a lot of time.
BOOL ReadFile(
    HANDLE       hFile,
    LPVOID       lpBuffer,
    DWORD        nNumberOfBytesToRead,
    LPDWORD      lpNumberOfBytesRead,
    LPOVERLAPPED lpOverlapped
);

Code 2. ReadFile function declaration

The API call relationship graph generated by taint analysis reflects the relationship between malicious code API calls. By analyzing the API call relationship, malicious code behavior and analyze malicious code attack intentions can be identified.

For example, when analyzing the stealing behavior of malicious code as shown in Figure 2, (1) mark the SOCKET handle generated by the socket function as a tainted data region, then a group of network behaviors can be associated; (2) mark the file handle generated by CreatFile as a tainted data region, the file behavior of a same file operation can be associated; (3) the file buffer read by ReadFile is marked as a tainted data region, if it is found as the send data of the send function in the network behavior , the network stealing behavior of malicious code can be captured. In a similar way, a variety of malicious code behaviors can be identified, such as executing a remote shell command (Figure 3), writing the program itself into the registry autorun items (Figure 4), etc.

![Diagram of API call relationships]

Figure 2. Stealing behavior  Figure 3. Executing remote shell commands  Figure 4. Writing the program itself into the registry autorun item

4. Evaluation

There is a big difference of analysis framework between DRTaint and traditional taint analysis framework in design, so it is meaningless to directly compare the analysis efficiency of them. Most of the existing taint frameworks perform taint analysis while executing the binary program. DRTaint includes three parts: online recording program trace, offline restoration data flow propagation process, and taint analysis. Among them, the lightweight online recording program trace part usually takes less time; in offline restoration data flow propagation, because the unicorn-based instruction simulation execution process is usually inefficient, it will take more time; the taint analysis process is based on the preprocessed data, so the efficiency is relatively high. Although using DRTaint to perform taint analysis once takes more time, the efficiency is usually higher when performing multiple subsequent analyses. Therefore, when DRTaint needs to be analyzed multiple times for a program, the overall analysis efficiency is higher.

The experiment used real APT attack samples as the analysis object, analyzed the three cases of directly executing the program, using the libdft analysis program, and using the DRTaint analysis
program, recorded and compared the time-consuming analysis process. Among them, the original libdft framework is based on Linux platform, and this experiment uses the dft-win ported to the Windows platform based on the original libdft framework. Since the malicious code analysis process needs to analyze the taint propagation relationship between different APIs, and the analysis process needs to be executed multiple times, this article compares the total time consumption of different execution times, as shown in Figure 5, and the experimental environment is shown in Table 2. It can be seen that executing the program alone or using the dft-win analysis program, the execution time is basically the same each time, and the total time consumption increases linearly; when DRTaint is executed for the first time, it takes more time, but the subsequent taint analysis process takes less time. After reaching a certain number of times, the total time of DRTaint is lower than dft-win.

Table 2. Analysis environment

| Tools               | Information                      |
|---------------------|----------------------------------|
| Hardware Environment| VMWare Virtual Machine, 4 CPUs 4G Memory |
| Operating System    | Windows 7 x64                     |
| Sample              | A downloader sample of APT-C-12   |
|                     | MD5: 06589B18797F61F257A2E61F49B295B4 (Note: The sample is a dynamic link library dll, and its malicious behavior is triggered by calling the sample export function during testing.) |
| dft-win             | https://github.com/dingelish/dftwin |
| Pin                 | DBI tool, Version 2.12            |
| Unicorn             | Instruction simulation tool       |
| Capstone            | Disassembly tool                  |
| Python              | DRTaint taint analysis module, Version 3.7.7 |

Figure 5. Time comparison of three execution modes

In addition, DRTaint has two shortcomings. Firstly, the efficiency advantage of DRTaint is based on multiple analyses of the same program. If the analysis process requires less taint analysis, there is no obvious efficiency advantage. Secondly, for malicious code such as ransomware and mining virus, their I/O operations or instruction operations are intensive, which will generate a large data flow log. For example, when DRTaint is used to record WannaCry, dozens of GB of data are generated quickly, which is not convenient for subsequent analysis.

5. Conclusions

Aiming at the shortcomings of the existing taint analysis framework in both "replay" analysis and analysis efficiency, this paper proposes the DRTaint taint analysis framework, which uses online recording program execution trace, offline analysis of data flow restoration, and data region-based taint analysis. The combined design of these modules alleviated the shortcomings of the existing taint analysis framework to a certain extent. In addition, using the DRTaint framework, this paper studies the extraction of malicious code behavior based on taint analysis. The experiment found that DRTaint can effectively perform the correlation analysis of malicious code API calls, which provides good support
for malicious code behavior extraction. In view of the problem that the DRTaint data flow recording process may produce a large data flow log, follow-up research needs to further study the data flow recording strategy and optimize the recording algorithm.

References

[1] Schwartz E.J., Avgerinos T., Brumley D. (2010) All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In: IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, 2010, pp. 317-331.

[2] Ulrich B., Paolo M.C., Clemens H., Christopher K., Engin K. (2009) Scalable, behavior-based malware clustering. In: Network and Distributed System Security Symposium. Internet Society, 8–11.

[3] David K., Heng Y. (2017) Capturing Malware Propagations with Code Injections and Code-Reuse Attacks. In Computer and Communications Security. ACM, 1691–1708.

[4] Juan C., Gustavo G., Mark M., Antonio N. (2012) Undangle: Early Detection of Dangling Pointers in Use-after-free and Double-free Vulnerabilities. In International Symposium on Software Testing and Analysis. ACM, 133–143.

[5] Peng C., Hao C. (2018) Angora: Efficient fuzzing by principled search. In: IEEE Symposium on Security and Privacy (SP), San Francisco, CA, 2018, pp. 711-725.

[6] Dongseok J., Ranjit J., Sorin L., Hovav S. (2010) An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications. In Computer and Communications Security, Vol. 10. ACM, 270–283.

[7] Xiajing W., Rui M., Bowen D., Zefeng J., Hongzhou C. (2018) "OFFDTAN: A New Approach of Offline Dynamic Taint Analysis for Binaries", Security and Communication Networks, vol. 2018, Article ID 7693861, 13 pages.

[8] Kemerlis V.P., Portokalidis G., Jee K., et al. (2012) libdfi: Practical Dynamic Data Flow Tracking for Commodity Systems[J]. Acm Sigplan Notices, 47(7):121-132.

[9] Galea, J., Kroening D. (2020) The Taint Rabbit: Optimizing Generic Taint Analysis with Dynamic Fast Path Generation. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS ’20). Association for Computing Machinery, New York, NY, USA, 622–636.

[10] Chi-Keung L., Robert C., Robert M., Harish P., Artur K., Geoff L, Steven W., Vijay J.R., Kim H. (2005) Pin: building customized program analysis tools with dynamic instrumentation. SIGPLAN Not. 40, 6 (June 2005), 190–200.

[11] Nguyen A.Q., Coseinc. (2014) Capstone: Next-Gen Disassembly Framework. http://www.capstone-engine.org/BHUSA2014-capstone.pdf.

[12] NGUYEN A.Q., DANG H.V. (2015) Unicorn: Next Generation CPU Emulator Framework. https://www.unicorn-engine.org/BHUSA2015-unicorn.pdf.