2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)

Hidden Path: Dynamic Software Watermarking Based on Control flow Obfuscation

Zhe Chen*, Chunfu Jia† and Donghui Xu
*College of Computer and Control Engineering
Nankai University, TianJin, China
Email: chenzhe501@foxmail.com
Email: cfjia@nankai.edu.cn

***Abstract*—Software watermarking is a general tool used to combat software piracy by embedding identifying information into a program. Most existing schemes of software watermarking have the weak point that the security only depends on the stealth of the watermark structure. Besides, the watermarking is independent of program in semantic level and can be destroyed via fairly straightforward semantics-preserving code transformations. To mitigate such problems, this paper introduces control flow obfuscation to build a novel dynamic software watermarking framework. By using control flow obfuscation, we skilfully connect several obfuscated branches to construct a hidden execution path. The watermarking is divided to fragments and each fragment is "stored" in each obfuscated branch. As a part of obfuscated branches ,watermarking is closely associated with program internal logic. The security can be reduced to the difficulty of reversing these obfuscated branches, rather than the stealth of watermark structure. We choose neural network as obfuscator; and deploy our watermarking in SPEC-2006int benchmarks. We provide the evaluation in overhead, data-rate, resiliency. Experimental results indicate that the proposed method has enough resilience against common attacks, yet practical in terms of performance.**

***Keywords*-software watermarking; code obfuscation; control-flow obfuscation; neural network;**

I. Introduction

It is estimated that 39% of the software in use around the world is illegitimate or unlicensed, with the commercial value of $10 billion [1]. Such illegitimate or unlicensed software caused serious damage to the economic interests of the software developers. As a matter of course, the software copyright protection needs to be effective. This means that the copyright owners should be able to claim the ownership of their products. Furthermore, it is possible to trace the unlicensed users who obtain the illegal distribution or pirate copy. *Software watermarking* can achieve these security goals.

According the definition given by Collberg and Clark [2], a complete software watermarking system has three components: embedder, extractor, and recognizer. There is a program P and a watermark w. The embedder generates the watermarked program P_w by introducing a secret key k. Assuming that the attacker knows the embedding algorithm, the secret key can prevent attacker extracting the watermark as easily as copyright owner. Meanwhile, the embedder must keep the program semantics: P and P_w must compute the same output for the same input. The extractor uses k extract the watermark w and then recognizer decides whether the w is correct.

The mainstream types of software watermarking are *static watermarking* and *dynamic watermarking* [3]. Static watermarking can be embedded and extracted easily; as a double-edged sword, attacker can easily locate the watermark structure. So static watermarking has been gradually abandoned. Different from the static watermarking, dynamic watermarking embeds watermark structure into program execution states. Simply the disassembling code blocks cannot locate the watermark structure. However, with the development of reverse engineering, inherent shortcoming of typical dynamic watermarking has become obvious. Because the semantic of watermark structure w and source program P are mutual independence, attacker can remove or distortive watermarking on the premise of ensuring the correctness of program. What's more, this defect may leak the secret key k under symbolic execution and dynamic taint analysis. To avoid such drawback, code obfuscation is a well "integrator" to combine watermark structure and program [3].

The primary contribution of this paper is a new approach to dynamic software watermarking, hidden path watermarking, which is closely associated with program logic. Unlike in the past, watermark structure has been"*stored*" as a special form into obfuscated branches, rather than inserting independent semantic code as watermarking. The implementation of obfuscator chooses the existing achievement: neural network control flow obfuscator. By using this obfuscation method, watermark structure will be embedded in program's control flow. This paper construct a hidden path by using obfuscation, which only can be triggered by the secret input. This way of embedding attaches fragility to watermark structure. Our watermarking has following three advantages.

- Watermarking and program logic are closely integrated in semantic level; this feature does not exist in classical studies.
- The security of watermarking is based on obfuscation strength and the fragility of watermark structure, rather than the stealth of watermark structure.
- Choosing a lightweight obfuscator to construct watermarking, the efficiency of program is affected lesser.

978-1-5386-3221-5/17 $31.00 © 2017 IEEE
DOI 10.1109/CSE-EUC.2017.280

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

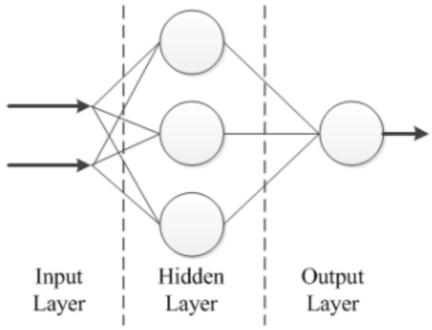

Figure 1. An example of neural network

The remainder of this paper is organized as follows. We introduce some useful preliminaries about neural network obfuscation in Section 2. The design of our watermarking scheme is introduced in Section 3. The implementation of our watermarking is detailed in Section 4. We analyze the security of our method against possible attacking strategies in Section 5, and show the evaluation results in Section 6. Section 7 discusses the related works on software watermarking. Finally we discuss the limitation of our scheme and draw a conclusion in Section 8.

II. Control flow obfuscation using artificial neural network

In this section, the basic knowledge of neural network will be introduced. Section 2.1 presents the basic knowledge. Section 2.2 introduces how to obfuscate control flow using neural network.

A. Background of neural network

Artificial neural network (e.g. feedforward neural network,like demonstrated in Fig. 1) is a connection model consisting of an interconnected group of artificial neurons. Its working principle is based on the human thinking process, mainly is used for data classification and linear regression. Neural network is known as a non-linear algorithm with powerful computational capabilities. During the 1990s, several researches claimed that a neural network of the above representative model is a universal function approximate and should be able to simulate arbitrary functions [4], [5]. In machine learning, a serious weakness of neural networks is that it is difficult to explain networks' internal rules in human-comprehensive models; this weakness partly perplex researchers. Several researchers attempt to propose effective algorithms to extract rules from neural network [6] [7]. A representative result gave by Golea [8] is that extracting rules from neural networks is a NP-hard problem. However, as a double-edged sword, the incomprehensibility of neural network could become security foundation in control flow obfuscation.

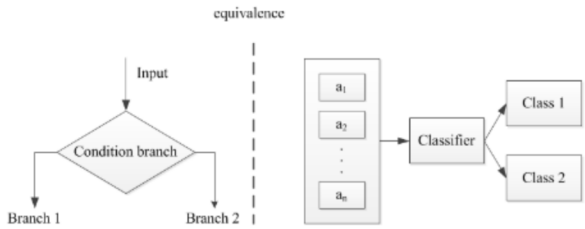

Figure 2. A condition branch and binomial classification

B. Neural-network-based control flow obfuscation

Before recommending the watermarking framework, we first present the main technique in our watermarking system, which we called that Neural-network-based control flow obfuscation. This obfuscation scheme is proposed by Ma [9] in 2014. In program execution, a condition branch is used to control program execution path by examining whether the input is conformed to transfer condition. From another perspective to see, a condition branch functionally looks like classification in machine learning. Fig. 2 shows that a condition branch can be considered as a *binomial classification* task, in which all possible values of the input space are classified into two categories, and each category triggers a corresponding path.

According this feature, neural network can replace a condition branch in functionality. Neural network is chose to build obfuscator, not only because it can provide strong ability of classification, but also its incomprehensibility could help providing resistance to combat state-of-art reverse engineering. Because of its incomprehensibility, attacker is difficult to accurately obtain knowledge which has been embedded in neural network. Likewise, distinguishing different networks is beyond attacker's ability. Ma [9] showed that neural network had provide strong resistance against concolic testing [10], the powerful hybrid software verification technique. The obfuscator significantly increase the difficulty of reverse analysis, and can against common attacks. This paper will construct watermarking based on this obfuscator.

III. Design overview

Typical dynamic watermark always have the problem that the semantic between watermarking and origin program are mutual independence. In fact, typical dynamic watermarking is not "embedded" into program; more like be "added" into program. The watermarked program feels like $P+w$, not P_w in semantic level. Note that, watermarked program must preserve *semantic* as the same as origin program; but it does not need preserve the *states* (such as execution trace) in program's runtime. In fact, program execution states can be explained as a variety of meaning. Our watermarking, hidden path watermarking, extends the execution states by using control flow obfuscation. The extra states can construct a oblivious execution path to express watermarking

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

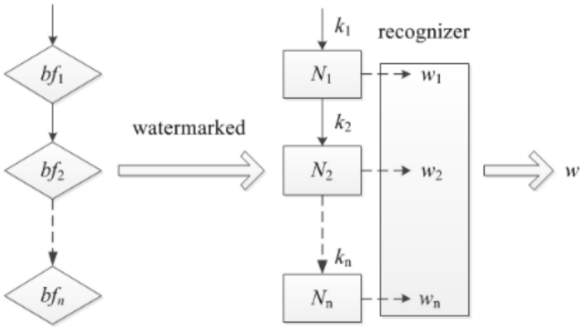

Figure 3. The framework of hidden path watermarking

information.

As mentioned in Section 2, neural network has been known as a robust data classifier and its internal rules are incomprehensibility. Therefore, the neural network can protect watermark structure by embedding it into neural network internal rules. Loosely speaking, our watermarking has three steps: firstly, the design of our watermarking is based on condition branch, so we choose some candidate `if-else` conditions. Secondly, the core is embedding watermarking pieces into these branches by obfuscating them. Finally, constructing a hidden path by using these obfuscated branches.

For explained the basic steps, the scheme we proposed is quite easy to describe. The framework will be given as follow and then it will be explained in detail. Given a secret key set $\boldsymbol{k}$ and watermark $\boldsymbol{w}$. Watermarking is triggered by the $\boldsymbol{k}$ and execute a hidden path as a watermarking path. As the Fig. 3 shown:

- Choosing n condition branches $\boldsymbol{b}$=(b_1, b_2, b_3 ..., b_n) (sequential execution are not necessary in origin program)from program control flow to construct hidden path.
- The watermark $\boldsymbol{w}$ is divided to n fragments $\boldsymbol{w}$=(w_1, w_2, w_3 ..., w_n)and be carried by obfuscated branches.
- The input set $\boldsymbol{k}$ is used to triggered the hidden path to extract watermark fragments.

To each chosen branch b_1, b_2, b_3 ..., b_n, we obfuscated them to corresponding neural networks models N_1, N_2, N_3 ..., N_n. These neural networks can not only realize the branch function, but also include the watermark piece. When extracting the watermark, we give the input set $\boldsymbol{k}$ to marked program P_w. All the watermarked branches are triggered sequentially from b_1 to b_n; we call this path *watermarking path*. The watermarking path cannot be seen unless needing being extracted.

In the following, we explain this framework in detail.

A. Pseudo-random sequence

In the watermarking framework, we introduce a function `generator(x)`. A pseudo-random sequence is generated using origin input as seed by this function. The sequence needs having good pseudo-randomness in cryptograph. All possible inputs `x` can be used as a seed. However, for each seed, it only corresponds a specific pseudo-random sequence. In other words, obfuscated branches use the chaotic pseudo-random sequences as there inputs. When program is running, origin input `x` first has been used to generate a pseudo-random sequence. In these sequences, only one can trigger the watermark path. That is to say, when we watermark a `if-else` condition branch, a secret key k has been chose to trigger watermarking. In order to intuitively expound our method, we assume that there are three chaotic pseudo-random sequences: s_{x_1}, s_{x_2} and s_k. Just as its name implies, these three sequences are generated by using input: x_1, x_2 and k.

B. Neural network module and watermarking embed

In this basic obfuscation framework, program execution path is decided by function `predict`. This function is a trained neural network and computes a virtual address to register `eax`. The unconditional jump instruction `jmp eax` completes control-flow transform according to this virtual address. In our watermarking framework, we train the watermark code and origin condition branch into neural networks. Figuratively speaking, we train the neural network using this train samples in table 1:

The training sample means that o_1 and o_2 correspond two origin execution paths. The special input s_k corresponds the special output o_k which includes the watermark information. In a similar way, other condition branch can be trained as this form. We will use the special output to store watermark in an ingenious way.

Considering a 32-bit operating system, the target address of jump instruction always has 24 bits length for long jump(even 8 bits for short jump). This means the output of neural network have 8 bits (24 bits for short jump) are redundancy. These redundancy is key solution of embedding the watermark. Taking the long jump as example, we divide this output into four parts for every 8 bits: **Byte0**, **Byte1**, **Byte2** and **Byte3**. As the fig.4 shows, we give the four bytes of different meaning. **Byte1** to **Byte3** is called *transfer section* that store the virtual address for control transfer. **Byte0** is *watermark section* that store the watermark bits. Training process of neural network ensure that the input k and corresponding watermark bits have been contained in neural network. For those inputs that goes to the origin branch, we fill the **Byte0** some meaningfulness random numbers.

C. Recognition

The recognition of watermark requires a dynamic analysis tool. Given the input set k to watermarked program P_w, we analyze the outputs of each obfuscated branch, and then

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

Table I
A CANDIDATE TRAINING SAMPLES

Output	Input
o_1	s_{x_1}
o_2	s_{x_2}
o_k	s_k

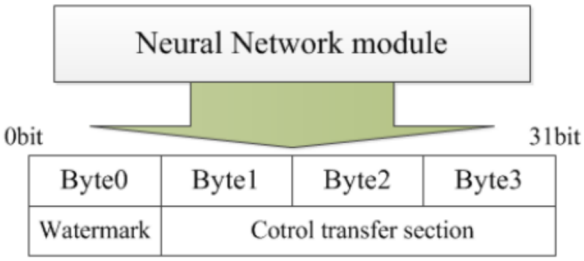

Figure 4. The way of watermark embed

extract the 0-7bits of each output. We combines these bits to verify whether these bits are watermark.

IV. IMPLEMENTATION

In this section, we will elaborate the implementation details. The watermark applies to the program written by C/C++ and compiled by common compiler.

A. Deploy pseudo-random sequence generator

The inputs of obfuscated branches are used to generate pseudo-random sequences. The generator must have following two features:

- Great randomness to resist brute-force attack
- Enough to distinguish between degrees

To satisfy such requirement, we introduce a pseudo-random sequence generator based on asymmetric Feistel network [11]. Compared with the symmetric Feistel network, asymmetric Feistel network can achieve more effective on diffusion and chaotic. Due to decryption is not need in our watermarking and asymmetric Feistel network is only used to encrypt branch inputs, tedious decryption calculation in symmetric Feistel network is skillfully evaded.

B. Neural network training and source codes rewriting

Without loss of generality, let us consider the following code as the a candidate `if-else` condition branches as eaxmple:

```
_asm{
nop nop nop nop
}
if(x>5)
  function1
else
  function2
```

We insert some `nop` as pretreatment placeholder to ensure the relative location between code blocks after being watermarked. After compiling this source code, we disassemble the executable file, the assembly codes is like follows:

```
text:00411A65 cmp [ebi+i], 5
text:00411A69 jle short loc_411A72
text:00411A6B call j_function1@@YAZZ
text:00411A70 jmp short loc_411A77
text:00411A72 call j_function2@@YAZZ
text:00411A78 retn
```

According to these disassemble code, we can extract some needful information to train the neural networks. In the first branch, we got the two virtual addresses `00411A6B` and `00411A72`.They correspond the two function `function1` and `function2` in the source code. For example, we want to embed the letter "*w*" as watermark into the sample program. The ASCII of *w* is 57 in hexadecimal. According to Section 3.2, we assign the training set with these virtual addresses (When we need watermark more branches the third output is the address of next branch entry point.

We apply practical swarm optimization (PSO), a sophisticated algorithm widely applied in neural network training, as the network learning algorithm [12] [13]. PSO is not only a effective algorithm, but also has better accuracy even in the boundary regions. In fact, no program input is outside of the train set. Consequently, so long as the selected training set is sufficient coverage of boundary values, program can never execute in wrong way. Although the trained model is overfitting, we still welcome for this.

After training process, a neural network model is get. We package the model into a function `predict`. This function use a pseudo-random sequence as its input and output a 32 bits integer. We choose a register `eax` to store the output and insert a uncondition transfer instruction `jmp eax` to complete the control flow transfer. Then we remove the placeholder and watermark the branch. After being marked, source code is rewrite like this:

```
  x=generator(x);
  address=predict(x) mod 0x1000000;
  _asm{
        mov eax, address
        jmp eax
        }
   function1;
  goto label;
   function2;
 label:
```

C. Watermark recognize

In the recognizing process, we select a dynamic simulator: TEMU component in Bitblaze [14] as recognizer.

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

TEMU can provide a dynamic analysis environment through whole-system emulation and dynamic binary instrumentation. When recognizing watermark, TEMU records the execute trace by running the program P_w with the input set $\boldsymbol{k}$. The we analyze the outputs that given by each neural network. If the outputs contain the watermark pieces, the watermark will be successfully extracted.

V. Security analysis

In this section, we have considered the four attack scenarios to watermarking: *brute-force* attack, *additive attack*, *subtractive attack* and *distortive attack*. In following, we assume that attacker have the *extract* algorithm; the goal of attacker is that making copyright owner cannot prove the copyright, and preserving the correct of program simultaneously.

1) brute-force attack: In a brute-force attack, attacker simply generates all possible secret keys and try to find the correct one by applying public recognizer algorithm. When attacker knows keys' distribution of possible, he may select more likely ones to test. In our watermarking, the secret keys is permutated to a chaotic pseudo-random sequence; the security of keys is based on the selected pseudo-random algorithm. Clearly, the difficult of brute-force attack is reduced to distinguishing the pseudo-random sequence. Brute-force attack need so considerable computational overhead that this attack is impractical.

2) additive attack: In an *additive attack*, attacker attempts to add his own watermarking w' into watermarked program P_w to generate $P_(w, w')$. If a program has two watermark, copyright owner cannot prove that his watermark is embedded before attacker's. In our watermark, the offset between code blocks needs strictly fixed in order to ensure program correctness. When attacker has generated $P_(w, w')$, the offset between code blocks has been changed. $P_(w, w')$ will crash unless attacker can recover original offset. The difficult of recover is equal to reverse the obfuscated branches. The incomprehensibility of neural network provides powerful resistance against *additive attack*.

3) subtractive attack: In a *subtractive attack*, attacker can (approximatively) point out where the watermark w is hidden and can try to crop it out. After attacking, copyright owner can recognize no watermark from P_w. This kind of attack must preserve the semantic(at least, preserve needful function). *Subtractive attack* is ineffective for the reason that all the watermark has been treated as knowledge in the trained neural networks; when attacker removes these networks, program cannot run correctly on account of incompleteness. Hence, it is impossible for attacker to remove the watermark in neural networks.

4) distortive attack: In a *distortive attack*, attacker applies semantic-preserving transform(i.e. obfuscation, code compression and encrypt et al.) to scramble program and watermark.The goal of *distortive attack*is to prevent copyright owner to extract watermark. This attack is no need to analyze the location of the watermark, because watermark can be always affected wherever the watermark is hidden. This kind of attack does not work as the same reason of *additive attack* scenario. We choose several candidate packer(i.e. UPX [15], ASPack [16]) to pack marked program P_w. Without exception, these packed program all crash in runtime.

VI. Evaluation

In this section we present our evaluation on C code implementations of the hidden path watermark. We have measured the cost of embedding the watermark in terms of the time and space overhead incurred by the inserted watermark code and the resiliency of the watermark to attacks. The following threat models have being considered:

- attackers disassemble the program to locate the position of watermark structure.
- attackers deploy automatic analysis tools to analyze the trigger of watermark structure.
- attackers attempt to remove suspicious code for destroying watermark structure.

A. Static disassemble

Once the attacker get the watermarked program, he may first disassemble the program to comprehend the internal logic. When attacker disassemble the program, he can get the following assemble code:

```
call _predict
add esp, 4
xor eax 0xFF000000
jmp eax
call _function1
jmp 004012F
call _function2
004012F:retn
```

The information attacker can only get is that the function `predict` and uncondition transfer instruction `jmp eax`. Attacker cannot find any suspicious code except the neural network module. Watermark structure has been embedded in neural network so that watermark path cannot be revealed in static control flow analysis. If attacker attempt to attack watermark, he must analysis the neural network module.

B. Remove the watermark

If the attacker realize that neural network modules include the watermark structure, he may try to remove the watermark structure from the neural network module. Firstly, attacker want to analysis which input is the watermark trigger in neural network. According to the Ma [9], neural network obfuscation can resist powerful reverse engineering: symbolic execution. Base on this advantage, we test a simple condition branch in section 4.2 to show whether Bitblaze can solve what secret key can trigger the watermark path.

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

Table II
THE RESULT OF CONSTRAINT SOLVER ON TEST BRANCH

Program	instruction	jump	STP size	STP result	constraint nodes
Origin	95	21	244458B	0x32	19
Watermarked	426	123	1097886B	N/A	117

Table III
THE TIME COST

program	origin	watermarked	execute times
specrand	20.761s	31.027s	48481
bzip2	17.328s	22.912s	268
mcf	5,715s	6.265s	710
hmmer	25.233s	26.092s	35
sjeng	15.367s	16.832s	268
lbm	3.775s	4.201s	272

Table IV
THE SPACE COST(UNIT:BYTE)

program	origin	watermarked	cost
specrand	32256	40762	8506
bzip2	178176	186296	8120
mcf	49664	57224	7560
hmmer	514048	522130	8082
sjeng	269312	277684	8372
lbm	58880	66102	7222

In experiment, TEMU model has record execution trace of test branch. And then we use VINE to compile the trace to intermediate language. Finally, constraint solver will give the result. Table 2 has given the result of constraint solver. The test input is 7. According the result, the complexity of watermarked branch is more complex than origin branch. The constraint solver cannot solve the value which can trigger the watermark path. It should be emphasized that the experiment only test a simple branch. Analyzing the watermarked branch is beyond ability of existing analysis tools. In a real world application, analyzing whole code blocks of watermarked program will meet more difficulties.

In addition, we evaluate our watermarking using dynamic taint analysis using Bitblaze. We taint the original input x and analyze the taint propagation. All the main added variables are tainted. That is to say, the dynamic analysis tool cannot distinguish the added code from the original code. These results show that our watermark structure is highly integrate with origin program. Therefore, even the attacker can point the location of watermark, he still cannot remove the watermark. In other words, the price of removing watermark is expensive enough.

C. Performance Overhead

In order to assess the practicability of the watermarking, we deploy our watermarking in 6 benchmarks from SPECint-2006 test suite. We embed 32bit watermark by using four neural networks. The experimental environment is as follows: Hardware platform is Intel(R) Core(TM) i7-2600 CPU 3.40GHz with 2.00GB RAM; The operating system is Windows XP SP3.

This paper evaluate our watermark based on time cost and space cost. The neural network model is trained by Matlab 2012, and its topological structure includes one input layer, two hidden layer(include 7 nodes) and one output layer. Training process is optimized by PSO algorithm. Table 3 shows the time cost of our watermark. The extra overhead mainly causes by the execution of watermarked branches, so we have detailed statistics on the execute times of the watermarked branches, under the test inputs. From table 3, we can conclude that our watermarking has affect the execution efficiency in a small range. The each run time of watermark structure is less than magnitude of $10^{-2}s$. The experimental results show a positive correlation between time cost and watermark structure execute times. In total, extra time overhead is in the acceptable range.

Space cost is another indicator for evaluation. It indicate the affection that the space watermark structure occupy. Table 4 shows the program size in the before and after being watermarked. From the table 5, the extra space cost for each benchmark is approximately 8KB. The space cost mainly caused by storing the weight matrix of neural network. Meanwhile, because watermark structure introduce pseudo-random sequence generator, it influences the program size in a certain extent. The space cost is constant.

VII. RELATED WORK

A. Software watermarking

A software watermarking is like a *copyright notice*; it can claim the *copyright ownership* to the program. According to the embed mode, watermark can divide into *static watermarking* and *dynamic watermarking*. Most of the previous works focus on static watermarking [17]. Cousot [18] proposed an abstract interpretation-based watermarking by inserting congruent integers in JAVA code; these big integers which do not often appear in common program may cause the attacker's suspicions. Microsoft [19] proposed a watermarking by reordering basic blocks. Regrettably, this watermarking can be easily destroyed by simply transform, such as inserting bogus code or obfuscating the program. These seemingly clever static watermarking design still had congenital defects. The stealth is "Achilles Heel" of the static watermarking. Although dynamic watermarking has solved this problem in static semantic environment, with the development of software testing, the same problem

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

still exists in dynamic semantic environment. A classical dynamic watermarking is Easter Egg Watermarks [20]. The main problem with Easter Egg watermarks is that they seem to be easily located. Watermark structure executes only when the right input sequence has been discovered. Therefore, watermark structure can be easily removed or destroyed and program correctness has not affected. Based on the design ideas of Easter Egg watermarks, researchers had proposed several dynamic watermarking. One kind is data structure watermarks [21] [22] [23]. Unfortunately, data structure watermarks cannot resist against distortive attack. Because these data structure often store in program by using point operations, several obfuscating transformations have effectively destroy the dynamic data structure and make watermark recognition impossible. Another kind of dynamic watermarking is path-based watermarking, whose idea is to embed condition branch structure into the program [24]. Based on such idea, several researchers have also proposed other similar schemes for execution trace watermarks. However, these watermarking frame still cannot work under distortive attack.

B. Control flow obfuscation

Control flow obfuscation is one of the major techniques of code obfuscation. The purpose of control flow obfuscation is that make the program logic difficult to be understood. Sharif et al. obfuscated `if-else` branch by using hash function and self-decrypting code [25]. The limitation of this work is that the application range is only to protect equal conditions (`if(x=5)`). If obfuscating unequal conditions (`if(x>5)`), it may cause too much time cost and space cost in deploying hash table. Wang et al. proposed a linear obfuscation scheme to combat symbolic execution [26]. Wang used the unsolved mathematical problems "Collazt conjecture" to encode branch inputs, so symbolic execution and constraint solver hardly to solve the origin inputs.

Ma [9] and Zong [27] introduced classification algorithms in machine learning to obfuscate program branches. Although there are differences between the algorithms to choose, these two obfuscator had the same principles: both neural network and SVM had complex internal logic. This feature of neural network and SVM can help obfuscator resist against common types of attack. However, being limited by properties of neural network and SVM, obfuscating non-linear branch conditions(`if` $(x > 5)$ `&&` $(y > 5)$) cause much extra overhead.

C. Symbolic execution and its application

Symbolic execution is a method in software testing for three decades [28]. It can solve the inputs which trigger the specific execution path by using symbol to analyze program. After more than 30 years of development, modern symbolic execution tools can automatically generate test cases, and can cover as many as possible execution paths [29]. Researchers had developed several powerful tools based on symbolic execution. Brumley developed a tools called MineSweeper to detect the trigger condition of malware [30]. Wang et al. removed malicious hidden-code in malware by using symbolic execution and taint analysis [31]. Zeng realized obfuscated code reuse through dynamic symbolic execution [32]. This efficient reverse analysis had caused serious damage to software intellectual property.

Although such techniques were powerful, some limitations still exist [33]. When path constrains cannot reduce to a certain symbol expression, analyzing such code was beyond ability of execution.

VIII. Conclusion

Typical dynamic watermarking has the problem that watermark structure and origin program have not been integrated in semantic level. Attackers can crack these watermarking by using some simply transformation to the watermarked programs. In this paper, we propose a novel dynamic watermarking: hidden path. We extend the control transfer of programs by using neural network control flow obfuscation. Watermark structure has been embedded in these extra states and highly combined with program control flow. The hidden path watermarking has effectively resistance against common attacks. And performance of execution efficiency suggests that our watermarking has a good practicability.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments. The author also would like to thank the assistant professor, Haoyu MA, the author of paper [9], give us the guidance about design of obfuscation module. This work is supported by the National Key Basic Research Program of China (No.2013CB834204), the National Natural Science Foundation of China (No. 61272423), and the Natural Science Foundation of Tianjin (No. 14JCYBJC15300).

References

[1] B.S.Alliance, "Bsa global software survey," [Online]. Available link: http://globalstudy.bsa.org/2016/index.html.

[2] C. Collberg and T. Clark, "Software watermarking: Models and dynamic embeddings," Proc. ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 1999), ACM, 1999, pp. 311-324.

[3] C. Collberg, and J. Nagra, "Surreptitious software: obfuscation, watermarking, and tamperproofing for software protection," Addison-Wesley Professional, 2009.

[4] G. Gybenko, "Approximation by superposition of sigmoidal functions," Mathematics of Control, Signals and Systems vol.2, no.4 ,pp. 303-314, 1989.

[5] K. Hornik, "Approximation capabilities of multilayer feedforward networks," Neural networks vol.4, no.2, pp. 251-257, 1991.

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.

[6] A. Tickle, R. Andrews, M. Golea, J. Diederich, "The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks," IEEE Trans. Neural Network, vol.9, no.6, pp. 1057-1068, 1998.

[7] G. Towell, J. Shavlik, "The extraction of refined rules from knowledge based neural networks," Mach. Learn. vol.13, no.1 ,pp. 71-101, 1993.

[8] M. Golea, "On the complexity of rule extraction from neural networks and network querying," Proc. the Rule Extraction From Trained Artificial Neural Networks Workshop, Society For the Study of Artificial Intelligence and Simulation of Behavior Workshop Series (AISB), pp.51-59, 1996.

[9] H. Ma, X. Ma, W. Liu, Z. Huang, D. Gao, and C. Jia, "Control flow obfuscation using neural network to fight concolic testing," Procs. the 10th International Conference on Security and Privacy in Communication Networks (SecureComm), 2014.

[10] S. Koushik, "Concolic testing," Procs the 22nd ACM international conference on Automated software engineering. ACM, 2007.

[11] Y. Dodis and P. Puniya, "Feistel networks made public, and applications," Proc. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, pp.534-554, 2007.

[12] R. C. Eberhart and Y. Shi, "Particle swarm optimization: Developments, applications and resources," Proc. the Congress on Evolutionary Computation (CEC), 2001, pp. 81C86.

[13] J.R. Zhang, J. Zhang, T.M. Lok, and M. R. Lyu, "A hybrid particle swarm optimizationCback-propagation algorithm for feedforward neural network training," Applied Mathematics and Computation, vol. 185, no. 2, pp. 1026-1037, 2007.

[14] D. Song et al, "BitBlaze: A new approach to computer security via binary analysis," Proc. the International Conference on Information Systems Security, Springer Berlin Heidelberg, 2008.

[15] Oberhumer, M. Franz, and L. Molnar, "The Ultimate Packer for eXecutables (UPX)," [Online]. Available link: http://upx. sourceforge. net/. Last accessed on 16 (2005).

[16] Aspack: Executable Packer, http://www.aspack.com.

[17] G. Myles and C. Collberg, "Software watermarking through register mallocation: Implementation, analysis, and attacks," Proc. the 6th International Conference of Information Security and Cryptology (ICISC), pp. 274-293, 2003.

[18] P. Cousot and R. Cousot, "An abstract interpretation-based framework for software watermarking," Proc. the 31th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages (POPL),ACM, pp. 173-185, 2004.

[19] Davidson , I. Robert , and N. Myhrvold. "Method and system for generating and auditing a signature for a computer program," U.S. Patent No. 5,559,884. 24 Sep. 1996.

[20] D. Nagy-Farkas, "The easter egg archive," [Online]. Available link: http://www.eeggs.com.

[21] C. Collberg, C. Thomborson, and G. M. Townsend, "Dynamic graph-based software fingerprinting," ACM Transactions on Programming Languages and Systems, vol. 29, no. 6, pp. 35-67, 2007.

[22] I. Kamel and A. Qutaiba, "A robust software watermarking for copyright protection," Computers and Security, vol. 28, no. 6, pp. 395-409, 2009.

[23] X. CHEN, D. FANG, J. SHEN, F. CHEN, W. WANG, and L. HE, "A dynamic graph watermark scheme of tamper resistance," Proc. the 5th International Conference on Information Assurance and Security, IEEE, pp. 3-6, 2009.

[24] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp, "Dynamic path-based software watermarking," Proc. the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation. ACM, pp. 107-118, 2004.

[25] M.I. Sharif, A. Lanzi and JT. Giffin, "Impeding Malware Analysis Using Conditional Code Obfuscation," Proc. the Network and Distributed System Security Symposium, 2008.

[26] Z. Wang, J. Ming, C. Jia and D. Gao, "Linear obfuscation to combat symbolic execution," Proc. the European Sympon Research in Computer Security (ESORICS 2011), Springer, pp. 210-226, 2011.

[27] N. Zong, and C. Jia, "Branch obfuscation using "Black Boxe"," Proc. the Theoretical Aspects of Software Engineering Conference (TASE 2014), IEEE, pp.114-121, 2014.

[28] C. Cadar, and S. Koushik, "Symbolic execution for software testing: three decades later," Communications of the ACM , vol. 56, no. 2 , pp. 82-90, 2013.

[29] S. Bugrara, and R. Engler, "Redundant State Detection for Dynamic Symbolic Execution," Proc. the USENIX Annual Technical Conference (USENIX ATC 2013), pp. 199-211, 2013.

[30] D, Brumley, C. Hartwig C, Z. Liang, J. Newsome, D. Song and H. Yin, "Automatically identifying trigger-based behavior in malware," Botnet Detection. Springer US, pp. 65-88, 2008.

[31] Z. Wang, C. Jia and K. Lu, "Malicious hidden-code extracting based on environment-sensitive analysis," Chinese Journal of Computers, vol. 35, no. 4, pp. 693-702, 2001.

[32] J. Zeng, Y. Fu, K.A. Miller, Z. Lin, X. Zhang and D. Xu, "Obfuscation resilient binary code reuse through trace-oriented programming," Proc. the 2013 ACM SIGSAC conference on computer and communications security (CCS 2013). ACM, pp. 487-498, 2013.

[33] X. Qu and B. Robinson, "A case study of concolic testing tools and their limitations," Proc. the 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM 2011), IEEE, pp. 117-126, 2011.

Authorized licensed use limited to: Universitatsbibliothek Erlangen Nurnberg. Downloaded on January 10,2021 at 00:05:13 UTC from IEEE Xplore. Restrictions apply.