Effectiveness Evaluation of Software Protection based on MPN

Wang Huai Jun
School of Information Science and Technology, Northwest University,
NWU-Irdeto IoT-Information Security Joint Laboratory
Xi'an, China
hjwang006@163.com

Fang Ding Yi
School of Information Science and Technology, Northwest University,
NWU-Irdeto IoT -Information Security Joint Laboratory
Xi'an, China

Wang Ni
School of Information Science and Technology, Northwest University,
NWU-Irdeto IoT -Information Security Joint Laboratory
Xi'an, China

Gu Yuan Xiang
Irdeto, Holland
NWU-Irdeto IoT -Information Security Joint Laboratory
Ottawa, Canada

***Abstract*—There are many technologies of software protection to prevent software from being attacked, such as software piracy, tampering, and reverse engineering, but what about the effectiveness of them is hard to answer. This paper presents a new evaluation method considering attack cost as the metric to evaluate the effectiveness of software protection under the circumstances of fulfilling assumptions that all software can be attacked successfully. We propose a new attack model, called MPN, and deduce an approach to calculate attack cost with it. Moreover, we also verify the approach.**

Keywords-software protection; attack model; Petri Net; effectiveness evaluation

I. INTRODUCTION

Software protection technologies were born to avoid threats of software piracy, tampering, and reverse engineering. But whether and how these protection technologies are effective are still an unsolved problem. Our motivation derives from how to evaluate the effectiveness of software protection by measuring attack cost. Now, there is no accepted approach was widely accepted as a common metric. But many researchers have done lots of relevant works, and we can classify them in two groups. ① Evaluation based on Theoretical Analysis[1-4], which measures or prove the effectiveness of software protection through theoretical analysis. ② Evaluation based on Attack[5-9] which measures or proof from the view of attack.

Technically, the evaluation approach in this paper belongs to the second group. Rather than doing manual attack experiment or designing common attack tools, we describe software attacks with an attack model. In this paper: Section 2 introduces an attack model called MPN. In Section 3, discuss the method to evaluate the effectiveness of software protection with the metric "attack cost". We verify the evaluation method through a case in Section 4. Section 5 concludes this paper.

II. EFFECTIVENESS EVALUATION WITH PETRI NET

Attack Net model based on Petri Net was first proposed by McDermott[10]. The basic Petri Net[11] is a three-tuple (*P, T, F*) . The flexibility of Petri Net makes that it can abstract attack process and refine it. It can indicate more information. So, we use Petri Net to model software attacks.

A. Primary elements in Software Attack

In an informal attack description, there are six types of elements are listed by Steffan[12]. According to that list, we made new description in following Table.

TABLE I. Primary elements in the attack process

Name	Meaning
Attacker	Attacker in Software Attack.
Object	Object in Software Attack.
Goal	The purpose of a software attack process.
Method	The way to achieve Goal. There are many Methods in a process.
State	Intermediate process of software attack.
Technique	The attack technique to attack software in a attack process.
Sub-goal	The purpose of to achieve States.
Action	Performing Technique.
Precondition	The condition to perform the attack technique.
Influence	The consequence after performing the attack technique.

There are many Methods to finish a Goal, and each Method includes many States and Techniques.

B. Mapping Primary elements into Marked Petri Net

Fig. 1 is an example of Petri Net. $P=\{p_0, p_1, p_2, p_3, p_4\}$ is the set of States, $T=\{t_0, t_1, t_2, t_3, t_4\}$ is the set of Techniques.

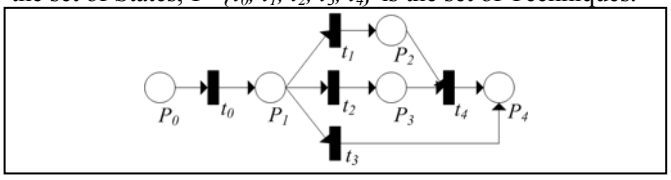

Figure 1. Example

Then we can mapping the primary elements in Table I into the counterpart in Fig.1 as follows.

1). Attacker and Object

Attacker is the people executing attack based on Petri net. Object is the software that Attacker want to attack. And they both can be seen from the Petri net.

2). Goal, Sub-goal, State and Technique

*I*n Fig.1, p_0 is the original *State*, and p_4 is the last *State* which is the *Goal* of the attack. t_0, t_1, t_2, t_3 and t_4 are *Techniques* in the software attack process. p_1, p_2, and p_3 are *Sub-goals* of Technique.

3). Method

© 2013. The authors - Published by Atlantis Press

One *Method* contains *States* and *Techniques*, which form a path from original *State* to last *State*. In Fig.1,we can find three *Methods*. $path_1$={p_0, t_0, p_1, t_3, p_4}; $path_2$ = {p_0, t_0, p_1, t_1, p_2, t_4, p_4}; $path_3$ = {p_0, t_0, p_1, t_2, p_3, t_4, p_4}.

Obviously, a *Method* is a path in Petri Net from original *State* to last.

4). Precondition, Action and Influence

The elements introduced above belong to static properties. The difference between Marked Petri Net[13] and the basic one is that the Marked Petri Net has a Token, which is a dynamic object. In the attacking process, Token is moved from one *State* to next one. If the condition is satisfied, an event will be enabled, that means in its input there are enough Tokens. Firing event will change the number and position of Token. Enabling event is just the Precondition; firing event is just the Action; changing Token is the influence of the Action.

C. Software Attack Modeling

Definition 1 MPN: which is software attack modeling based on Marked Petri Net, including six elements (*P, T, F, Path, Rate, Cost)*, among which, first three elements come from the basic Petri Net; *Path* is a finite set of Methods. *Rate* and *Cost* respectively stand for attack rate and attack cost.

Based on the software attack model MPN and software attack process, we give the following definition.

Definition 2 Attack Path: Some *Methods* are included in a *MPN*, and each method is called an *Attack Path*. (See the description of "***Method***" in detail from Table I) .

Definition 3 Attack Rate: Be used to quantify the rate of attack progress in one *Attack Path*. It stands for Precondition of each *Technique*, and it will be changed dynamically when attacking.

We quantify attack process with *Attack Rate*, and it is instantiated as Token in *MPN*. Let $R(p_i)$ stands for *Attack Rate*'s value of reaching State p_i, and then $R(p_i)$ is more than or equal to 0, also less than or equal to 1. While $R(p_i)$ is equal to 1, the Goal of attack will be achieved, and p_i is the last State in SMMPN. Meanwhile, R(t_i) stands for increased attack rate after activating technique t_i, and we can get a conclusion: $R(p_{i+1}) \geq R(t_1) + R(t_2) + \ldots + R$.

Definition 4 Attack Cost: Reflect the cost of *T(Technique)* in *Attack Path*. In the attack process, *Cost*(p_i) is used to calculate the *attack cost* produced when reaching the State p_i. And if p_i is the last State, *Cost*(p_i) is the attack cost of the Method. *Attack Cost* is quantifiable, and we use *sac* as its unit.

D. The Token in MPN

Since *MPN* = (*P, T, F, Path, Rate, Cost*) is the process model to describe software attack in detail. *In* Attack Path $Path_a$= {P_0, T_0, ..., T_i, P_{i+1}, T_{i+1}, ..., T_{m-1}, P_m}, every *T* in *Attack Path* one by one, and ends at P_m. After performing T_i, the number of *Token* included in P_{i+1} decides whether T_{i+1} will be enabled. So, in the attack process, the Sub-goal of each technique is to get enough *Token* needed in following State. Next we analyze t_i in $Path_a$ in the view of *Attack Rate*. The condition to enabling t_i is whether the value of $R(p_i)$ after firing t_i is big enough. Obviously, when firing *t*, the value of $R(p_i)$ will be increased.

Then, in *MPN* Token can be instantiated with *Attack Rate*. Next, we give a definition:

Definition 5 Threshold Value (μ): it stands for the condition of enabling t_i. Where is $\mu_{i+1} > \mu_i$.

So Precondition of performing T_i is: Token in P_i >=μ_i; and the Influence of performing T_i increase the Token in P_{i+1}. At first, Token is assigned 0 in P_i. After performing Technique t_i , the Token= $R(P_{i-1})+ R(t_i)$ in P_i. Fig.2 presents the progress of attack along $Path_a$. $R(t_i)= R(P_{i+1}) - R(P_i)$ is the increment of Rate caused by firing t_i in actual attacking.

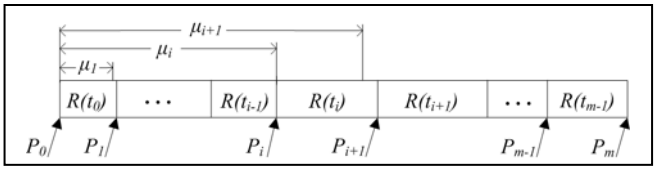

Figure 2. The transition of Token in Patha

According to Fig.2, there is: $\mu_{i+1} \geq R(p_i) \geq \mu_{i+1}$.

So, $\mu_{i+1} - R(p_i)$ is the minimum increment of *Attack Rate* caused by firing t_i, besides it is also the minimal Token to reach P_i. Because first Technique fired in $Path_a$ is t_0, $\mu_0 = 0$; In addition, because the last Technique is t_{m-1}, after t_{m-1} fired, $R(p_m) = \mu_m = 1$.

Also, we can get: $\triangle\mu = \mu_{i+1} - \mu_i$, and it can be seen as the importance of t_i according to $\triangle\mu$. If t_i is more important, $\triangle\mu$ will be larger. In practice, it is affected by many factors, for example, attack scene, attack experience. So, we need to determine it through analyzing t_i in depth.

Above all, we can get following conclusion: the effect of Method displayed by an *Attack Path* $Path_a$ consist of firing every *t* in $Path_a$.

As the only dynamic behavior in *Attack Path*, firing *t* is the reason that *Attack Cost* is produced. Then, in *Attack Path* $Path_a$, *Cost*($Path_a$) is made up of the attack cost produced by firing each *t*.

III. Effectiveness Evaluation with SAPMMON

From the above mentioned, when evaluating effectiveness the metric can use the attack cost.

A. Metric Calculation

Let c_i be the attack cost produced by firing Technique t_i. Sub-goals of different *t* may be the same, but the *c* of these *t* may be yet different. Determining *c* is difficult, because we need to analyze and understand the Technique in depth. However, there are also many methods to acquire *c*, most of which based on expert or attack experience. In *Attack Path* $Path_a$ = {p_0, t_0, ..., t_i, p_{i+1}, t_{i+1}, ..., t_{m-1}, p_m}, there is,

$$Cost(P_i) = \sum_{i=0}^{m-1} \cos t(t_i) \quad (1)$$

To get more accurate results, we decompose the technique based on Software Security Patterns[14] into some smaller attack units. We named the small attack units as *ae*, main attack units are shown in Table II.

TABLE II. Main attack units

ae	Class	Meaning	Attack Information	Rank
ae_1	Disassembling	by tools.	disassembly code	*A*
ae_2	Memory spying	Spy the operation to memory of PE file.	Number of memory blocks being spied	*B*
ae_3	Differential attack	Compare difference	Number of difference points	*D*
ae_4	Collusion attack	Multiple attackers cooperate to attack.	------	*D*
ae_5	Process snooping	Run program to analyze character .	snooping time	*E*
ae_6	Interactive debugging	interact with debugger.	Time of user interference	*E*
ae_7	Control flow analysis	Analyze control flow	numbers of control flow instructions	*D*
ae_8	Instruction replacement	Modify instruction in the program.	Numbers of replacement instructions	*E*
ae_9	Branch interference	Modify the condition jump.	Modify time	*D*
ae_{10}	Control flow modifying	Modify control flow.	Numbers of modified control flow	*C*
ae_{11}	Code lifting	Code reuse.	Times (code size）	*C*
ae_{12}	Data lifting	Data reuse.	Times（data size）	*C*
ae_{13}	Data file replacement	Replace data file.	Time of data file replacement	*D*
ae_{14}	Program file replacement	Replace code file.	Time of program file replacement	*D*

Then calculate *cost()* of Technique by calculating cost of *ae*. If Technique t_i is performing by $\{ae_0, ..., ae_n\}$, we can get

$$\cos t(t_i) = \sum_{j=0}^{n} c(ae_i^j) \quad (2)$$

These data are obtained from attack log. For example, technique "DUMP" is decomposed by "Memory spying" and "Data lifting", The attack log is as follows.

DUMP:	ae_2:memory operation times
	ae_{11}:code size

So, $cost(t_{DUMP}) = c(ae_2) + c(ae_{11})$. Next, divided *ae* into five ranks, and value of *ae* is set according to attack experience.

Rank A: 1(sac). attack automatically absolutely.

Rank B: 5(sac).it is feasible to attack automatically absolutely by designing a special tool.

Rank C: 10(sac). attack manually with some assistant tools, which could provide much help.

Rank D: 15(sac). attack manually with some tools which could only provide less help.

Rank E: 20(sac). attack manually, there is no tools to provide help.

In which, *sac* is the unit of software attack cost. Then, calculate $c(ae)$ by (3) in following.

$$c(ae) = Grade(ae) \times Value(ae) \quad (3)$$

In (3), Rank(*ae*) is the function to get *ae*'s Rank and Value(*ae*) is the function to calculate *ae*'s information value. But, it needs to normalize *Value*(*ae*) based on the data normalization theory, and *Value(ae)*=lg(*Value(ae)*).

According to (1),(2),(3), we can get:

$$Cost(P_i) = \sum_{i=0}^{m-1} \sum_{j=0}^{n} Grade(ae_i^j) \times \lg(Value(ae_i^j)) \quad (4)$$

This attack modeling will be perfected with appearance of new *Techniques* in future and the Rank of attack units may be changed.

IV. A CASE STUDY

A. Background

1). Serial Numbers

This is often used to make user use the software legally [15]. If user had paid for the software, the developer will provide a unique Serial Number. Of course, in software there is an algorithm to certify the input number.

2). Control Flow Flattening

This[4] is a control obfuscation technique, which can obfuscate original control logic into a flatten control flow. The original software is "*SerialNum.exe*" , and "*SerialNum-Obf.exe"* is the protected software protected by the tool introduced in [16].

3). Attacks on Serial Numbers

First, we find some programs like "Crackme", then attack them manually. Then, we modeled the attack introduced in [15], adding other methods with new codes into the program. At last, the MPN can be constructed for attacking against checking program by Serial Number.

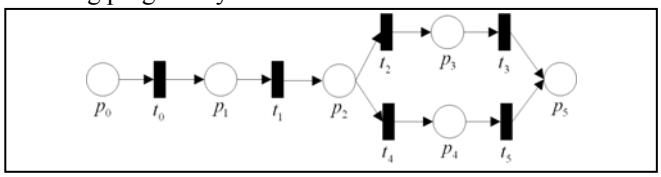

Figure 3. MPN for attacking against Serial Number

The meaning of each *State* and *Technique* is shown in Table III.

TABLE III. meaning of each P and T in Fig. 3

P/T	Meaning
p_0	Original program
p_1	The codes had been assembled
p_2	The *Basic Segments* have been obtained
p_3	Obtain the performed *Basic Segments*
p_4	Obtain the *Key Segment*
p_5	Obtain the cracked program
t_0	Disassemble the program
t_1	Partition codes in some *Basic Segments*
t_2	Mark performed *Basic Segments*
t_3	locate the *Key Segment* and modify *Key Instruction* among it
t_4	Monitor memory change and terminate when generating the address of *Fail block*, and *Key Block* is the terminated block.
t_5	Add *New Codes* into the program; *Key Block* always jumps to *New Codes*.

Obviously, two attack paths are included in Fig. 3.

- The first path is: $\{p_0, t_0, p_1, t_1, p_2, t_2, p_3, t_3, p_5\}$, which attacks the software by destroying Serial Number checking based on modifying the key instruction.
- The second path is: $\{p_0, t_0, p_1, t_1, p_2, t_4, p_4, t_5, p_5\}$, it stands for the attack which destroys Serial Number checking by adding new codes into the program.

Some researches or experiments have prove the Control Flow Flattening is effective[4, 16, 17]. Next, we will evaluate the technique with our proposed method.

B. Effectiveness Evaluation of Control Flow Flatenning

Firstly, determine *ae* in each Technique t_i, which are as follows:

Technique	t_0	t_1	t_2	t_3	t_4	t_5
ae	ae_1	ae_7	ae_9	ae_6	ae_6	ae_8+ae_{10}
Rank	*A*	*D*	*D*	*E*	*E*	*C*

Then, calculate Value of t_i. PEiD[18] is used to analyze *SeriaNuml.exe* and *SerialNum-Obf.exe* to obtain sizes of .text sections. Some scripts and plug-ins in IDA[19] to collect data, which includes: amount of jmp and conditional jump instruction; amount of the re-performed instructions; amount of *Basic segments* performed; checking the redundant space in *Object.* The value of each *ea* is shown in Table IV.

TABLE IV. Results of attack units *ae*

Object	ae_1	ae_6	ae_7	ae_8	ae_9	ae_{10}
SerialNum.exe	21371	206	417	1026	1566	1
SerialNum-Obf.exe	23315	263	657	2938	3938	1

With the data in Table II and V, calculate *Cost*(*Path1*) and *Cost*(*Path2*), the results are shown in Table V.

TABLE V. Value of attack cost calculation

Object	***Cost*(*Path1*)**	***Cost*(*Path2*)**
SerialNum.exe	134.3 (*sac*)	116.3 (*sac*)
SerialNum-Obf.exe	148.4 (*sac*)	130.4 (*sac*)

C. Analysis

In *Path1* and *Path2,* both of attack costs of attacking on *SerialNum.exe* are lower than *SerialNum-Obj.exe*. Obviously, we can quantize the effectiveness of the Control Flow Flattening by attack cost which is the evaluation metric we proposed. *D*iff is defined as the difference of attack Cost between unprotected and protected program.

Diff(Path2)= 130.4-116.3 =14.1(sac)

So we can evaluate the protection by compare the *Diff*, the bigger, the better. But if we consider the performance of the software, the result may be different, because some protection technologies bring little performance penalty, but can improve security of the software too much.

V. CONCLUSION

In the paper, a new approach is presented to evaluate effectiveness of software protection, and in which attack cost is considered as the evaluation metric. An attack model is introduced with Marked Petri Net, and give a way to calculate the attack cost in attack process with MPN. At last, the feasibility and reasonability of our method are tested and verified through evaluating effectiveness of Control Flow Flattening. We can see that attack cost can quantize the effectiveness of the protection according to the experiment results. In the way, we can evaluate effectiveness of software protection according to *Diff*.

ACKNOWLEDGMENTS

The paper was supported by the National Natural Science Foundation of China (No.61070176, No.61170218, No.61272461), Research and Industrial Project of Shaanxi Province under Grant (No. 2011K06-07).

REFERENCES

[1] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," The University of Auckland, New Zealand 1173-3500, 1997.

[2] G. Myles, C. Collberg, Z. Heidepriem, and A. Navabi, "The evaluation of two software watermarking algorithms," Software: Practice and Experience, vol. 35, pp. 923-938, 2005.

[3] M. Dalla Preda and R. Giacobazzi, "Semantic-based code obfuscation by abstract interpretation," in In Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05, 2005.

[4] C. Wang, J. Hill, J. Knight, and J. Davidson, "Software tamper resistance: Obstructing static analysis of programs," University of Virginia2000.

[5] I. Sutherland, G. E. Kalb, A. Blyth, and G. Mulley, "An empirical examination of the reverse engineering process for binary files," Computers & Security, vol. 25, pp. 221-228, 2006.

[6] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in Proceedings of the 10th ACM conference on Computer and communications security, 2003, pp. 290-299.

[7] S. K. Udupa, S. K. Debray, and M. Madou, "Deobfuscation: Reverse engineering obfuscated code," in In: 12th Working Conference on Reverse Engineering, Pittsburgh, PA, USA, 2005, pp. 45-54.

[8] J. Hamilton and S. Danicic, "An Evaluation of the Resilience of Static Java Bytecode Watermarks Against Distortive Attacks," IAENG International Journal of Computer Science, vol. 38, pp. 1-15, 2011.

[9] Y.-J. Zhao, Z.-Y. Tang, N. Wang, D.-Y. Fang, and Y.-X. Gu, "Evaluation of code obfuscating transformation," Ruanjian Xuebao/Journal of Software, vol. 23, pp. 700-711, 2012.

[10] J. P. McDermott, "Attack net penetration testing," in Proceedings of the 2000 workshop on New security paradigms, New York, NY, USA, 2001, pp. 15-21.

[11] J. L. Peterson, "Petri nets," ACM Computing Surveys (CSUR), vol. 9, pp. 223-252, 1977.

[12] J. Steffan and M. Schumacher, "Collaborative attack modeling," in Proceedings of the 2002 ACM symposium on Applied computing, New York, NY, USA, 2002, pp. 253-259.

[13] T. Murata, "Petri nets: Properties, analysis and applications," Proceedings of the IEEE, vol. 77, pp. 541-580, 1989.

[14] Y. X. Gu, "Software Security Patterns: White-Box Attack Analysis and Software Protection Technology," in 1st International ACM Summer School on Information Security and Protection Theme on Software Security and Protection, Beijing, China, 2010.

[15] E. Eilam, Reversing: secrets of reverse engineering. New Jersey: Wiley, 2011.

[16] H. J. Chun, "Research and Implementation of a Code Obfuscation Algorithm Based on Control Flow Flattening," Master, Northwest University, Xi'an, 2010.

[17] S. Chow, Y. Gu, H. Johnson, and V. Zakharov, "An approach to the obfuscation of control-flow of sequential computer programs," Information Security, vol. 2200, pp. 144-155, 2001.

[18] PEiD. Available: http://www.peid.info. (2013)

[19] IDA. Available: https://www.hex-rays.com/products/ida/index.shtml. (2013)