IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 13, 2017, accepted March 2, 2018, date of publication March 6, 2018, date of current version March 16, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2812764

A Method for Construction of Software Protection Technology Application Sequence Based on Petri Net With Inhibitor Arcs

QING SU[1], FAN HE[1], NAIQI WU[2], (Senior Member, IEEE), AND ZHIYI LIN[1]
[1]School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China
[2]Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China

Corresponding authors: Qing Su (sqsavage@qq.com) and Zhiyi Lin (lzy291@gdut.edu.cn)

This work was supported in part by the Natural Science Foundation of China under Grant 61273118, in part by the Guangdong Science and Technology Program of China under Grant 2017A040405050, and in part by the Guangzhou High-Tech Developmental Plan under Grant 201604016041.

ABSTRACT In the field of software protection, when there is a dependence between the various software protection technologies, the application order of these technologies must be arranged in a correct way in order to maximize the protection effect. When applying these technologies in a random way as the traditional methods do, an unexpected consequence may be produced, such as weakening the software protection effect and causing the protected software malfunction. To solve this problem, in this paper, a Petri net model is developed to describe the dependence behavior of applying multiple protection technologies. Then, algorithm is proposed to generate the reachable marking graph for the obtained Petri model. In considering different user requirements, based on the reachable marking graph, a method is presented to obtain a user-required and correct sequence of applying multiple protection technologies. The correctness of the obtained sequence is verified by a finite state automaton model. Experimental results show that the proposed method outperforms the traditional ones.

INDEX TERMS Code obfuscation, Petri nets, software protection, software protection technology.

I. INTRODUCTION

Software plays a vitally important role and is one of the core components of industrial control systems. Due to more and more malicious behaviors against software products, software security has become an important aspect for the implementation of industry 4.0 [37]. There are three primary threats aiming at software security named as reverse engineering, piracy, and tampering [8]. They threaten software in a dangerous ''white-box attack environment'' [34]. Currently, the major software protection technologies for countering these threats include code obfuscation [3], software watermark [18], and tamper resistance [1]. Code obfuscation is mainly used to defend against the reverse analysis of a program. This technology essentially converts the original program into a new one such that the semantics of the code is preserved while the converted program is more difficult to understand by an attacker. By software watermark, identification information (i.e., watermark) is embedded into a software product. When the software is run, if it is necessary, the embedded identification information can be extracted for software copyright attribution and integrity verification. Tamper resistance prevents the illegal modification of the normal operations of the program via the hardware-and-software-based measures. It has been proved that a single software protection technique is unable to ensure the absolute security [4]. Also, software watermark and tamper resistance can only be used as a means for reducing the loss when a software is compromised. Therefore, to improve the security of a software, the above three categories of technologies are usually applied simultaneously in a practical software protection system.

However, there may exist dependencies among these protection technologies, i.e., the use of one technique may affect the effect of the other techniques, or the effect of multiple techniques is sequence dependent. For example, the role of loop fusion and loop split is opposite in code obfuscation. Therefore, it is necessary to analyze the dependencies among these technologies such that one can determine a reasonable order of them and ensure the effect of code protection and the equivalence of original program when applying these

technologies; otherwise, the protective effect may be offset or even though the software is damaged by the reaction.

With multiple software protection technologies being applied in a same code, for the first time, Collberg *et al.* [9] present a sequence such that when these technologies are applied, it should be done according to the given sequence. However, they do not point out why it should be done in such a sequence and do not present a method to determine a proper sequence. Wrorblewski [38] use hard-coded software protection technology for protecting x86 machine code. Lacey and Moor [20] show that two code transform techniques cannot be simultaneously applied to the same piece of code and present an algorithm that selects the technology to be applied according to the characteristics of the code being protected. Zhang *et al.* [49] propose a software protection method for multi-watermark embedding, which can embed a number of sub-watermarking into the program that can be confused by certain strategies. To a certain degree, they notice the dependencies of software protection technologies. However, these methods are not applicable when the number of software protection technologies is large and the interdependence is complicated. For the case of complex dependencies, Gaurav and Pieprzyk [12] summarize three types of dependencies among the protection technologies by using weighted finite state automata (WFSA) [25]. The statements accepted by WFSA are the practicable ones. The weakness of this method is that it suffers from the state explosion problem.

In practice, since the diversity of the multiple software protection technologies and the complexity of their dependencies, it is impossible to manually determine the order in which the various technologies are applied to meet the user-specified criteria. Therefore, an automated and low-cost method that can be used to determine the sequence of applying multiple technologies for software protection is necessary. This motivates us to conduct this study.

The objective of this paper is to search for a method that can automatically determine the sequence of the technologies applied for protecting a software product. Due to that Petri nets are powerful for modeling concurrency and conflict control, they are widely used for modeling, analysis, control, and scheduling [2], [15], [39]–[45], [48], we use Petri nets to model the dependencies of the software protection technologies. It is a kind of Petri net with inhibitor arcs. With the built models, methods are proposed to determine the appropriate sequence of technologies applied for protecting a software product. In this paper, we make the following contributions:

- Petri net model with inhibitor arcs is developed to describe the behavior of dependencies of currently applying multiple software protection technologies.
- An algorithm is presented to generate a reachable marking graph for the obtained Petri net model.
- Based on the reachable marking graph, a method is proposed to obtain a user-required and correct sequence for applying multiple software protection technologies.

II. PETRI NET WITH INHIBITOR ARCS

With rigorous mathematical theory, Petri nets are powerful in modeling, analysis, and control for discrete event systems and hybrid systems [2], [6], [15], [39]–[45], [48]. Petri nets are also applied to the field of system security and software protection. Jasiul *et al.* [17] establish a mechanism for tracking malware behavior by using Colored Petri Net. Dunaev and Lengyel [10] use Petri net to confuse control-flow chart in a multithreaded environment. Wang *et al.* [35] use Petri net to create a model for evaluating code obfuscation from the attacker's perspective. Zhang *et al.* [47] use Petri net to simulate the attack process for detecting weaknesses in code protection schemes for mobile applications.

A sequence of technologies applied for protecting software is essentially a method of combing a number of software protection technologies for practical application requirements. With rigorous mathematical reasoning and graphically intuitive representation, Petri nets are a good choice for studying dependence issue of software protection technologies.

Petri nets with inhibitor arcs are an expansion prototype of the original Petri nets, which has zero-test capability [28], and its simulation capability is equivalent to Turing machine [23]. By taking these advantages, this paper proposes a method to model the dependence of the software protection technologies by using Petri nets with inhibitor arc. Based on the developed models, the problem of determining the right sequence of applied technologies is solved according to the user-specified requirement.

A Petri net with inhibitor arcs is formed by adding some inhibitor arcs that connecting a place and a transition to an ordinary Petri net. In an ordinary Petri net, whether a transition is enabled to fire depends on whether all the places in its pre-set have a token. An inhibitor arc is a kind of arc from a place s to a transition t, where s is one of the places in the pre-set of t If there is at least one token in s at marking M, i.e. $M(s) \geq 1$, even if all other spaces in the pre-set of t have tokens, t is disabled and cannot fire. When the inhibitor arc is ineffective, i.e. $M(s) = 0, t$ is enabled to fire. Once t fired, the inhibitor arc no longer affects the changes of tokens any more.

Definition 1: Petri net with inhibitor arcs is a quintuple [46]:

$$\Sigma = (S, T; F, I, M) \tag{1}$$

- $N = (S, T; F)$ is the basic petri net;
  - S is the place set;
  - T is the transition set;
  - $F = (S \times T) \cup (T \times S)$ is the arc set;
- $I \subset S \times T$ is the inhibitor arc sets. Where $I \cap F = \emptyset$, i.e.

$$\forall s \in S \wedge \forall t \in T : (s, t) \in F \rightarrow (s, t) \notin I \tag{2}$$

- M is the initial marking of the net.
- All the transitions in Σ have the following firing rules:
  - For $t \in T$, if

$$a) \forall s \in S : (s, t) \in F \rightarrow M(s) \geq 1 \tag{3}$$

$$b) \forall s \in S : (s, t) \in I \rightarrow M(s) = 0 \tag{4}$$

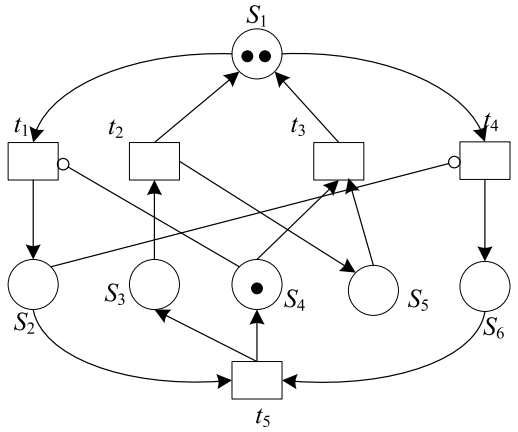


FIGURE 1. An example of Petri net with inhibitor arcs.

Then t is allowed to fires at M, represented as $M[t>$.

- If $M[t>$, then$t$ fires at M. A new markingM' is reached from M once t fires. For $\forall s \in S$:

$$M(s) = \begin{cases} M(s) - 1, & \text{if } (s,t) \in F \wedge (t,s) \notin F \\ M(s) + 1, & \text{if } (t,s) \in F \wedge (s,t) \notin F \\ M(s), & \text{otherwise} \end{cases} \quad (5)$$

When a Petri net with inhibitor arcs is shown graphically, an inhibitor arc looks like "—○", which is a normal directed edge replacing the arrow " ▶" with a small circle " ◯ ". For any place s and any transition t, the arc from$s$ to t and the inhibitor arc from s to t cannot exist at the same time.

Fig. 1 shows an example of Petri net with inhibitor arcs, marked as N_1. There are two inhibitor arcs in N_1, i.e. $I = \{(s_2, t_4), (s_4, t_1)\}$. The initial marking is $M_0 = [2, 0, 0, 1, 0, 0]^T$. At the initial marking M_0, for t_1 and t_4:

$$\forall s \in \bullet t_i : M(s) \geq 1, \quad i \in \{1, 4\} \quad (6)$$

According to Definition 1, (s_4, t_1) is an inhibitor arc that controls whether t_1 can fire or not. Since $M_0(s_4) = 1$, (s_4, t_1) acts so that t_1 is not allowed to fire at M_0. On the other side, although (s_2, t_4) is an inhibitor arc which controls t_4, $M_0(s_2) = 0$ negates the effects of (s_2, t_4) so that t_2 is allowed to fire at M_0. Once t_2 fires, a new marking M_1 is obtained with $M_1 = [1, 1, 0, 0, 0, 0]^T$.

III. MODELING DEPENDENCIES OF SOFTWARE PROTECTION TECHNOLOGIES BY PETRI NETS

For software protection, in the literature, the existing technologies include all kinds of anti-privacy technologies, tamper-proof technologies, and code obfuscation technologies. With the applications of these technologies, we can sum up four types of dependencies of software protection technologies. They are pre-requirements, post-requirements, pre-prohibitions, and post-prohibitions [25]. In the follows, we introduce all these four types of dependencies and develop Petri net models to describe them.

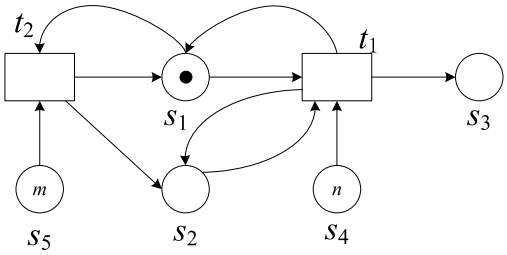


FIGURE 2. The Petri net model for pre-requirements dependency.

A. REQUIREMENT DEPENDENCY

There exists a kind of requirement dependency among some software protection technologies. That is to say that if protection technology A relies on protection technology B, then A and B should be applied in accordance with a certain order; otherwise, the protected software by applying A only would has obviously weaknesses and can be cracked easily.

1) PRE-REQUIREMENT DEPENDENCY

If a protection technology B must be applied at least once before A is applied, then we say that A pre-requires B, which is denoted as $A \xleftarrow{r} B$. For example, when inserting false branch into a program that is to be protected [5], it is necessary to apply the opaque predicate technology before enhancing the confusion potency of false branch [29].

A Petri net model is built to describe the behavior of pre-requirement dependency as shown in Fig. 2 with the initial mark $M_0 = [1, 0, 0, n, m]^T$, where

- s_1 : the place giving the initial status of the process for controlling the dependency;
- t_1 : represents that technology A is applied when it fires;
- t_2 : represents that technology B is applied when it fires;
- s_4 : a control place that controls A such that A can apply n times at most;
- s_5 : a control place that controls B such that B can be applied m times at most.

At the initial status of this Petri net, $M(s_2) = 0$. The tokens in s_2 are generated by firing t_2. Hence, once t_2 fires (which means that B is applied), we have

$$M'(s_2) = M(s_2) + 1 \geq 1 \quad (7)$$

When $M(s_2) \geq 1$, t_1 is allowed to fire (that means A is allowed to apply). Once t_1 fires, a token is returned to s_2, i.e.

$$M'(s_2) = M(s_2) + 1 \quad (8)$$

Since s_2 cannot be emptied once there is a token in it, the firing of t_1 is no longer dependent on t_2. That is to say, B must be applied at least once before A can be applied. In this way, the pre-requirement is well modeled.

2) POST-REQUIREMENT DEPENDENCY

If protection technology B must be applied at least once after technology A is applied one or multiple times, then we say that A post-requires B, which is denoted by $A \xrightarrow{r} B$.

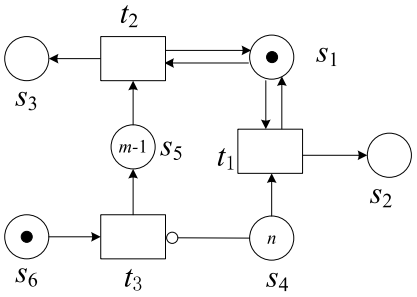


FIGURE 3. The Petri net model for post-requirements dependency.

For example, after applying the serial number protection technology [11], the encryption technology [16] must be applied for preventing dynamic debugging attack of crawling serial number in memory [12].

The Petri net model for describing the post-requirement dependency is shown in Fig. 3 with the initial marking $M_0 = [1, 0, 0, n, m-1, 1]^T$, where

- s_1 : the place giving the initial status of the process for controlling the dependency;
- t_1 : represents that technology A is applied when it fires;
- t_2 : represents that technology B is applied when it fires;
- s_4 : a control place that controls A such that A can be applied n times at most;
- s_5 : a control place that controls B such that B must be applied once at least and can be applied $m-1$ times at most.

When there are tokens in s_4, i.e. $M(s_4) \geq 1$, the inhibitor arc (s_4, t_3) disables the firing of t_3. Only when t_1 fires n times (which means that A is applied repeatedly n times), is then $M(s_4)$ emptied so that the inhibitor arc (s_4, t_3) is invalidated. At this time, t_2 is enabled and can fire (that means B is allowed to be applied), and a token is added into $M(s_5)$, i.e.:

$$M'(s_5) = M(s_5) + 1 = m - 1 + 1 = m \tag{9}$$

At M', $M'(s_5) = m$ means that s_5 controls t_2 such that it can fire m times at most consecutively.

B. PROHIBITION DEPENDENCY

There exists a prohibition dependency between certain protection technologies. For example, array merging and array splitting are mutually-exclusive in terms of implementation. When there is a prohibition dependency between protection technologies A and B, A and B cannot be applied to the same software to be protected in violation of the dependency. Otherwise, the protection effect of the software being protected would be diminished. Moreover, in the worst case, the functions of the protected software are not equivalent to the original one any more, or even the protected software is corrupted.

1) PRE-PROHIBITION DEPENDENCY

If the application of protection technology B is prohibited before A is applied, then we say that A pre-prohibits B, which is denoted by $A \overset{p}{\longleftarrow} B$. For example, alias analysis technology that is used to locate the convertible parts of a program is often called by other software protection technologies [21]. However, any technology that may disturb alias analysis, such as an opaque predicate technology [32], should be prohibited before the alias analysis technology is applied.

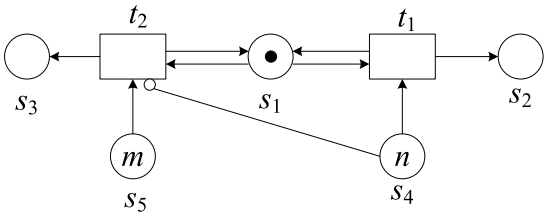


FIGURE 4. The Petri net model for pre-prohibition dependency.

To describe the behavior for the pre-prohibition dependency, we develop the Petri net model as shown in Fig. 4 with its initial marking $M_0 = [1, 0, 0, n, m]^T$, where

- s_1 : the place giving the initial status of the process for controlling the dependency;
- t_1 : represents that technology A is applied when it fires;
- t_2 : represents that technology B is applied when it fires;
- s_4 : a control place that controls A such that A can be applied n times at most;
- s_5 : a control place that controls B such that B can be applied $m-1$ times at most.

When there are tokens in s_4, i.e. $M(s_4) \geq 1$, the inhibitor arc (s_4, t_2) disables t_2. Only when t_1 fires n times consecutively (which means that A is applied n times without a firing of t_2), is then $M(s_4)$ equal to zero such that the inhibitor arc (s_4, t_2) is invalidated. At this time, t_2 is enabled (which means that B is firable). S_5 controls the firing of t_2 such that it can fire consecutively at most m times. In this way, the pre-prohibition dependency is ensured as required, i.e., the behavior is well modeled.

2) POST-PROHIBITION DEPENDENCY

In the application of software protection technologies, it may requires that if the technology B is prohibited after A is applied, then we say A post-prohibits B, which is denoted by $A \overset{p}{\longrightarrow} B$. For example, if a watermark has been embedded into a program execution path [22], any path-diversity technology [19] should not be applied.

A Petri net model for post-prohibition dependency is developed as shown in Fig. 5 with the initial mark $M_0 = [1, 0, 0, n, m]^T$, where

- s_1 : the place giving the initial status of the process for controlling the dependency;
- t_1 : represents that technology A is applied when it fires;
- t_2 : represents that technology B is applied when it fires;
- s_4 : a control place that controls A such that A can be applied n times at most;
- s_5 : a control place that controls B such that B can be applied m times at most.

For this model, at the initial status of Petri net, $M(s_2) = 0$. At this marking, oncet_1 fires (which means that A is applied),

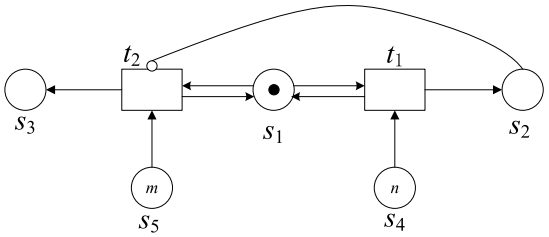


FIGURE 5. The Petri net model for post-prohibition dependency.

a token is added to s_2, i.e.

$$M'(s_2) = M(s_2) + 1 \geq 1 \tag{10}$$

S_4 controls t_1 such that it can fire n times at most. When there are tokens in s_2, i.e., $M(s_2) \geq 1$, the inhibitor arc (s_2, t_2) disables t_2. At this time, even though $M(s_5) = m \geq 1$, t_2 is not allowed to fire, which means that B cannot be applied after A is applied. Thus, the behavior of post-prohibition dependency is accurately described.

Up to now, we have developed Petri net models for the dependency of two types of technologies. With these models, we discuss Petri net model for the dependency of more than two types of technologies.

C. COMBINATION DEPENDENCY

When three or more protection technologies are dependent on each other, the behavior for applying these technologies is much more complicated and such a kind of dependency is called a combination dependency. It is necessary to model such a complicated dependency. To do so, we combine the Petri net models for the dependency of two types of technologies to model the behavior of a combination dependency. Notice that, in the above four Petri net models, a token in t's input place s is consumed when t fires. However, at the same time, a token is returned to s immediately after the firing of transition t (which represents the application of a software protection technology). This structure makes the model scalable.

1) MODELING METHOD

Note that, in the above four Petri net models for a dependency for two types of technologies, there is a place that gives the initial state of the model, i.e., initially only this place has tokens. The general idea for describing the combination dependency for more than two types of technologies is to combine the Petri net models for the dependencies for two technologies such that these models share the place that gives the initial state. In this way, we can integrate them as one model. To do so, Algorithm 1 given below is presented for this purpose.

Algorithm 1 Modeling the Combination Dependency

Step 1. For each pair of software protection technologies, define and model the dependencies by using the models for the four different dependencies. Assume that we obtain n such Petri net models. These models are named as $N_1, N_2, \ldots, N_n$ such that a set of Petri net models $N_1, N_2, \ldots, N_n$ is obtained.

Step 2. Combine $N_1, N_2, \ldots, N_n$ into one Petri net model named as N, i.e.,

$$N = \bigcup_{i=1}^{n} N_i \tag{11}$$

According to the following sub-steps:

Step 2.1 Choose $N_k \in \{N_1, N_2, \ldots, N_n\}$ randomly as the first Petri net model to be integrated to N.

Step 2.2 Let place s_1 be the place that presents the initial status of the process for controlling the dependencies in applying the technologies. Then, Σ is formed such that any $N_i \in \{N_1, N_2, \ldots, N_n, 1 \leq i \leq n$, shares s_1 as the place that gives the initial state at M_0 that is the initial marking of N.

Step 2.3 Reduce Σ and re-number all its places.

Step 3. Draw the reachable marking graph of N, denoted by $D(N)$. There are two sub-steps:

Step 3.1 Assume that the current marking of N is M. Then, fire every transition t at M and a new marking M' is obtained. This is done by starting from the initial marking M_0 at the beginning.

- If M' is equal to an existing marking M'' in $D(N)$, then draw the arc from M to M''.
- Or else, a new node is created to denote M', and draw an arc from M to M'. After that, makeM'as the current marking.

Step 3.2 Repeat Step 3.1 until marking $M$is reached such that there is no transition t that can fire at M or no any new marking can be generated. By doing so, the process of drawing $D(N)$ is completed.

Step 4. Locate the marking M that satisfies the user-specified requirement. Then, find a simple path Γ from the initial marking M_0 to M. All transitions in Γ form a transition sequence T. Sequence T consists of two parts, one is a set of control transitions and the other is a set of non-control transitions that present the applied software protection technologies (each non-control transition represents one type of technologies). A transition sequence T' can be obtained by removing all the control transitions from T. As each transition in T' represents a software protection technology, therefore T' is exactly a satisfactory software protection technology application sequence.

2) AN EXAMPLE OF COMBINATION DEPENDENCY MODELING

Assume that there are dependences among technologies A, B, and C with the dependencies for two types of technologies being A $\xleftarrow{r}$ B and $B \xrightarrow{p}$ C. Then, by using Algorithm 1, we first obtain the Petri net model N as shown in Fig. 6 with the initial mark $M_0 = [1, 0, 0, n, m, 0, k]^T$, where:

- s_1 : the place that gives the initial status of the process for controlling the dependency;

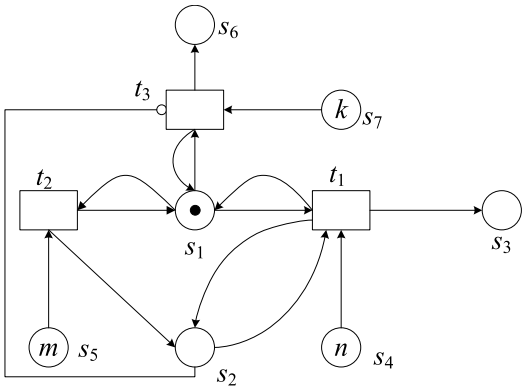


FIGURE 6. Petri net model of a combination dependency for Example 1.

- t_1 : represents that technology A is applied when it fires;
- t_2 : represents that technology B is applied when it fires;
- t_3 : represents that technology C is applied when it fires;
- s_4 : a control place that controls A such that A can be applied n times at most;
- s_5 : a control place that controls B such that B can be applied m times at most;
- s_7 : a control place that controls C such that C can be applied k times at most.

In Fig. 6, the subnets that model the dependencies of $A \xleftarrow{r} B$ and $B \xrightarrow{p} C$ share place s_1 that gives the initial tokens at initial marking M_0. Place s_2 controls whether Transitions t_1 and t_3 can fire or not. Transition t_1 is enabled when t_2 fires such that $M(s_2) \geq 0$, but t_3 is disabled due to the inhibitor arc (s_2, t_3).

IV. DETERMINATION OF SOFTWARE PROTECTION TECHNOLOGY APPLICATION SEQUENCE

Different users have different requirements for software protection technology applications. In general, the most common requirements are:

- Some users want to apply software protection technology as many times as possible;
- Some users want to apply software protection technology as many types as possible;
- Some users want to specify a technique that applied as much as possible.

Thus, we need to determine the application sequence of multiple technologies for different requirements. We discuss this issue next.

A. THE SOLUTION METHOD

In the follow, with the Petri net model for the dependencies of technologies and its reachable marking graph, we present a method to determine the application sequence of technologies for the above requirements by using an example.

Assume that seven kinds of technologies are to be applied and they are denoted by A, B, C, D, E, F, and G, respectively. For these technologies, we have the following dependency relations:

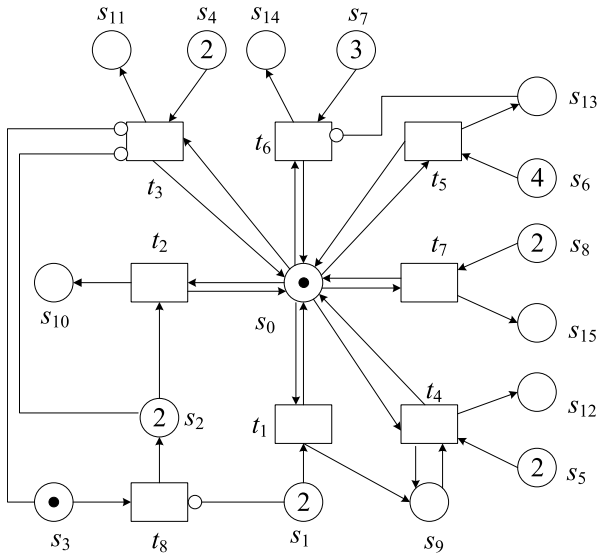


FIGURE 7. Petri net Model for the combination dependency among $A \sim G$.

- $A \xrightarrow{r} B$;
- $B \xleftarrow{p} C$;
- $D \xleftarrow{r} A$;
- $E \xrightarrow{p} F$;
- G can be applied independently of others.

Assume that the maximal applying times for these technologies are 2, 3, 2, 2, 4, 3, and 2, respectively. Obviously, these dependencies form a kind of a combination dependency.

By using Algorithm 1, we obtain the Petri net model denoted by Σ for this combination dependency and it is showed in Fig. 7, where

- s_1 : the place that gives the initial status of the system;
- $t_1 \sim t_7$: transitions representing technology A, B, C, D, E, F, and G are applied when they fire, respectively;
- s_1 : the control place that controls A such that A can be applied two times at most;
- **The subnet formed by** s_2, t_8, and s_3: for controlling B such that B can be applied two times at most;
- $s_4 \sim s_8$: control places that control C, D, E, F, and G such that they can be applied two, two, four, three, and two times at most, respectively.

There are two following spaces in each t: one is the initial space s_0; another is used to count the firing times of t, which is counting space. For example, we will find that s_9 in Fig. 7 is the counting space of t_1, a token will be added to s_9 while t_1 fires.

By Algorithm 1, reachable marking graph $D(N)$ of Σ is obtained and there are totally 2339 markings. Due to that $D(N)$ is very large, a part of it is shown Fig. 8 for illustration.

The reachable marking set [50], $R(M_0)$, collects all the marks of $D(N)$. All these marks is generated in the execution of Petri net N.

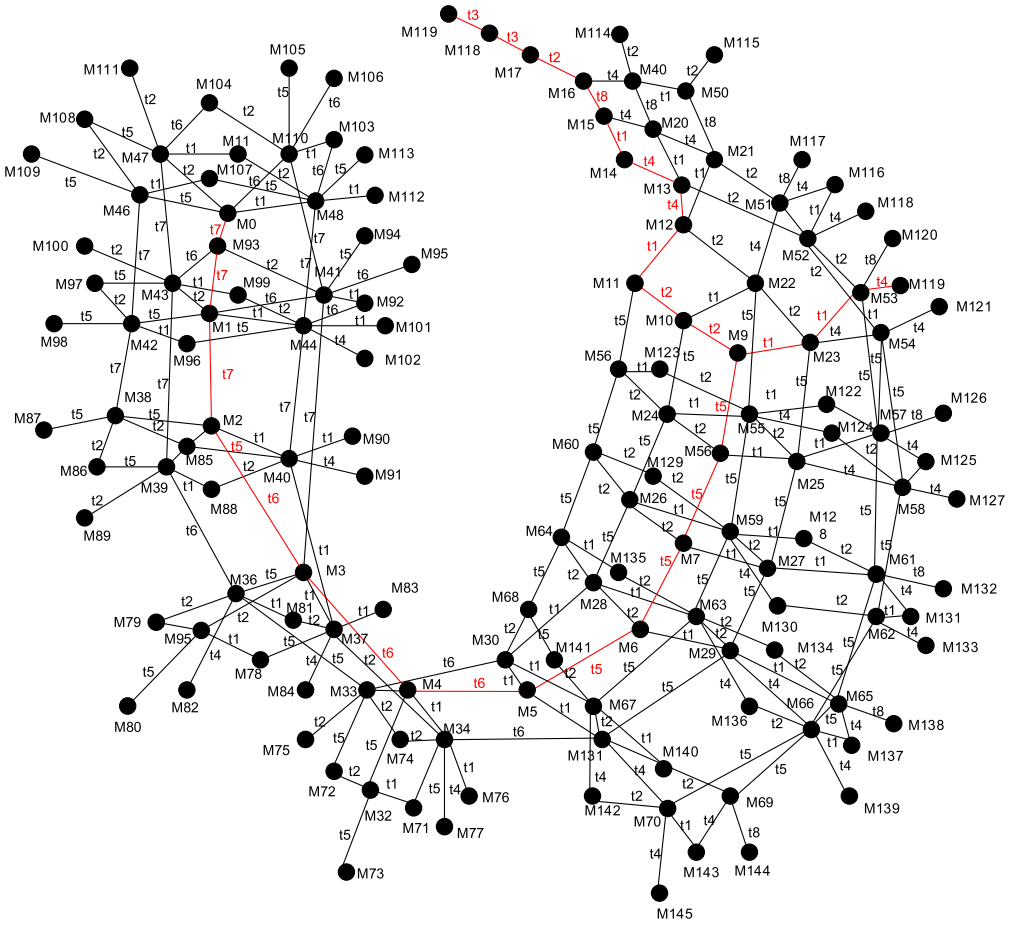


FIGURE 8. Part of the reachable marking graph of N.

1) MAXIMIZING THE TOTAL APPLIED TIMES OF TECHNOLOGIES

Maximizing the total applied times of technologies requires that the total number of times for firing Transitions $t_1 \sim t_7$ is maximized. The firing of any transition t results in that a token is added to the output places of t. For example, Place s_9 is an output place of t_1, a token goes into s_9 when t_1 fires. As can be seen in Fig. 7, the spaces $s_9 \sim s_{15}$ are the counting spaces of $t_1 \sim t_7$ respectively. Let $F(M, t_i)$ denote the number of t_i's firing times via a transition firing sequence that makes the system evolve from M_0 to $M \in R(M_0)$. Then, if we find a transition firing sequence and marking $M \in R(M_0)$ such that $\sum_{i=9}^{15} F(M, t_i)$ is maximized, then the obtained transition firing sequence must be the required sequence. Note that, by Algorithm 1, during process for generating the reachable marking graph, we can count $\sum_{i=9}^{15} F(M, t_i)$ for every sequence. Thus, we can find the required $M \in R(M_0)$ and the sequence. Then, by Step 4 of Algorithm 1, we can find the corresponding non-control transition sequence T'. For the example, as shown in Fig. 8, Marking M_{19} is such a marking and it is given as

$$M_{19} = [1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 2, 2, 4, 3, 2]^T. \quad (12)$$

and the corresponding non-control transition sequence is T_i and it is given as

$$T_1 = (t_7t_7t_6t_6t_6t_5t_5t_5t_5t_2t_2t_1t_4t_4t_1t_8t_2t_3t_3). \quad (13)$$

With M_{19} and T_1 being found, we have

$$\sum_{i=9}^{15} F(M_{19}, t_i) = 2 + 3 + 2 + 2 + 4 + 3 + 2 = 18. \quad (14)$$

Thus, the software protection technology application sequence S_1 corresponding to the transition sequence T_1 is

$$S_1 = (GGFFFEEEEBBADDABCC). \quad (15)$$

2) MAXIMIZING THE VARIETY OF APPLIED TECHNOLOGIES

Maximizing the variety of applied technologies requires that the number of transitions in $t_1 \sim t_7$ that fire is maximized. For such an objective, for any marking $M \in R(M_0)$, define $f(M)$ as

$$f(M) = \sum_{i=9}^{15} g(M(i)), \quad (16)$$

where

$$g(x) = \begin{cases} 1 & x > 0, 9 \le i \le 15 \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

Then, we need to maximize $f(M)$.

To maximize $f(M)$ is to find a marking $M \in R(M_0)$ such that $f(M) \geq f(M')$ for any marking $M' \in R(M_0)$ with $M \neq M'$. We can record $f(M)$ when we generate the reachable marking graph such that M is found. With M being found, the transition sequence from the initial mark M_0 to M is the required one. For the example, M_{18} is such a marking as shown in Fig. 8 and M_{18} is given as

$$M_{18} = [1, 0, 0, 0, 1, 1, 3, 2, 1, 2, 3, 1, 1, 1, 1, 1]^T. \quad (18)$$

Then, according to Step 4 of Algorithm 1, we can obtain the non-control transition sequence from initial mark M_0 to M_{18} and it is denoted by T_2 and it is given as

$$T_2 = (t_7t_7t_6t_6t_6t_5t_5t_5t_5t_2t_2t_1t_4t_4t_1t_8t_2t_3). \quad (19)$$

Note that T_2 contains all transitions $t_1 \sim t_7$, implying that it is the optimal one. The software protection technology application sequence S_2 corresponding to the transition sequence T_2 can be easily obtained and it is given as

$$S_2 = (GGFFFEEEEEBBADDABC). \quad (20)$$

3) MAXIMIZING THE APPLIED TIMES OF A SPECIFIC TECHNOLOGY

Maximizing the applied times of a specific technology requires that that the number of times for a transition t_i representing Technology i the specific technology, is maximized. For this objective, if we find a marking $M \in R(M_0)$ and a transition firing sequence such that for any $M' \in R(M_0)$ we have $F(M, t_i) \geq F(M', t_i)$, then the obtained transition firing sequence that makes the system evolve from M_0 to M is the required sequence.

For the example, assume that t_5 corresponds to the specific technology appointed by the user. Then, by Algorithm 1, we find the required marking M_{119} as shown in Fig. 8 and it is given as

$$M_{119} = [1, 0, 2, 1, 2, 1, 0, 0, 0, 2, 0, 0, 1, 4, 3, 2]^T. \quad (21)$$

The required transition firing sequence is T_3 and it is given as

$$T_3 = (t_7t_7t_6t_6t_6t_5t_5t_5t_5t_1t_1t_4). \quad (22)$$

With the obtained M_{119} and T_3, the software protection technology application sequence S_3 corresponding to transition sequence T_3 is

$$S_3 = (GGFFFEEEEEAD). \quad (23)$$

B. VERIFICATION BY FSA

In this section, to show the correctness of the results obtained by the proposed method, we verify the obtained results by using a finite state automaton (FSA) model [32]. To do so, we first model the dependencies of the software protection technologies as regular expressions [24]. Then, we build the FSA model base on these regular expressions. With the built model, if an input sequence can make the model evolve from the initial state to the acceptable state, then this sequence is the required software protection technology application sequence.

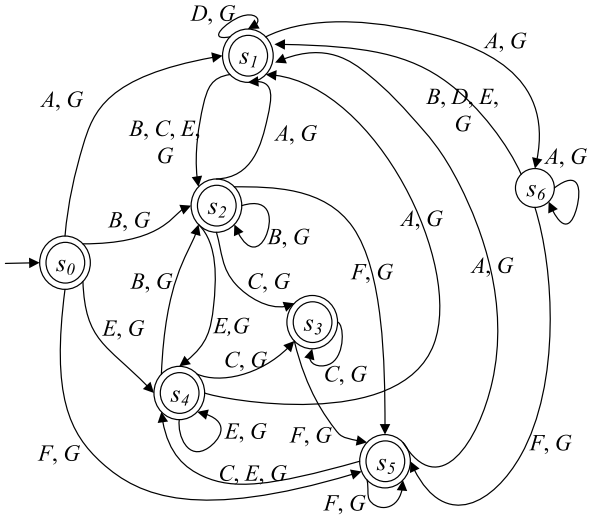


FIGURE 9. FSA for dependencies $\{A \xrightarrow{r} BB \xleftarrow{p} CD \xleftarrow{r} AE \xrightarrow{p} F$, and $G\}$.

TABLE 1. Reached Stat sequences of S_1, S_2 and S_3.

Input sequence	Reached stat sequence
S_1	$s_0s_5s_5s_5s_4s_4s_4s_4s_2s_2s_1s_1s_1s_6s_1s_2s_3$
S_2	$s_0s_5s_5s_5s_4s_4s_4s_4s_2s_2s_1s_1s_1s_6s_1s_2$
S_3	$s_0s_5s_5s_5s_4s_4s_4s_4s_1s_1$

The regular expressions that present the dependencies among $A \sim G$ are as follows:

- $(B|(A^+B))^*$;
- B^*A^*;
- $D(A|D)^*$;
- F^*E^*;
- G^*.

With these expressions, the FSA model for the example is built and is shown in Fig. 9. In building the automaton model, since the software protection technology G it has no dependency on any other software protection technology, it should be attached in every arcs in the FSA model to ensure it can be applied in any case.

Now, with the built FSA model, we take S_1, S_2, and S_3 that are given in Equations (15), (20), and (23), respectively, as input sequences of the FSA. For each of these inputs, we examine the evolution of the FSA model and the state sequence obtained by each input is obtained as shown in Table 1. From Table 1, it can be seen that, for every input, the system finally evolves to an acceptable state.

It follows from the above verification results that S_1, S_2, and S_3 are all legal software protection technology application sequences with the FSA model, i.e., the proposed method is correct and effective.

TABLE 2. The seven software protection technologies.

Mark	Name	Brief introduction
A	Local integers merging	Merge two 32-bit integers into one 64-bit integer.
B	Watermark embedding to initialization	Embed watermark into a construction method of class and save as an integer variable.
C	Opaque branch insertion	Randomly insert opaque branches into a method.
D	Branch inverter	Exchange *if* clause and *else* clause of the *if-else* statement.
E	Collberg-Thomborson watermarking	Encodes the message of a watermark into a graph which is generated in program execution, then embed the graph into the program being protected.
F	Function name overload	Modify the name of methods.
G	Dead code embedding	Randomly insert dead code.

V. EXPERIMENTAL RESULTS

Normally, the most commonly traditional methods for generating software protection technology application sequence are those that obtain a sequence in a random way. To compare the proposed method with the traditional methods, in this section, we use experiments to test the performance of proposed method and the traditional methods. To do so, in the experiments, we generate two types of sequences for the applied technologies. One is obtained by randomly generating the application sequence denoted by S_4 to represent the traditional methods. The other type of sequence denoted by S_5 is obtained by the proposed method.

Both types of sequences are applied to the same testcase base on *Sandmark* [32]. With different sequences are applied, we test the software complexity, i.e., the complexity to be attacked. In the experiments, watermark is one of the protection technologies to be applied. Thus, beside the software complexity, we also test the effectiveness of watermark to examine the effect of other technologies on watermark. By doing so, it is meaningful to test the performance of the proposed method.

A. THE TEST CASE

Take a Java program *DES* as a test case. It is a common program for data encryption and decryption. To protect this program, seven technologies are to be applied as listed in table 2. We then carry out experiments on *DES* to test the performance for randomly generated application sequence S_4 and S_5 that is obtained by the proposed method.

For the seven technologies, between two technologies, we have the following dependencies:

- $A \xleftarrow{p} B$;
- $C \xleftarrow{r} D$;
- $E \xrightarrow{r} F$;
- G is no dependency to others.

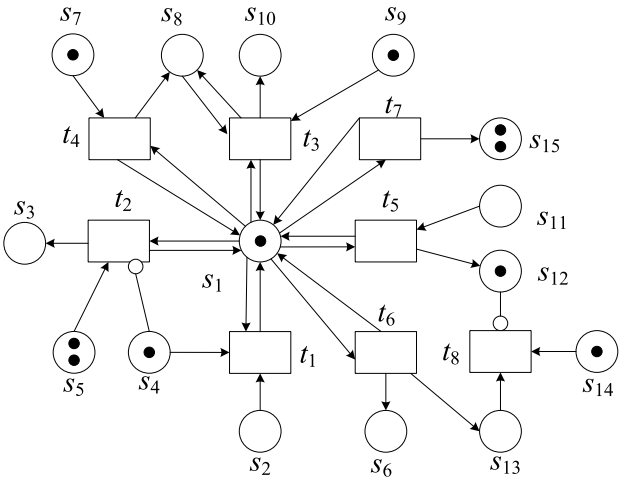


FIGURE 10. Petri net Model for the dependencies among $A \sim G$.

According the traditional method, we randomly generate an application sequence S_4 of these technologies as

$$S_4 = (FBACDEBGFG). \tag{24}$$

Observe S_4, we can find that the dependences $A \xleftarrow{p} B$ and $C \xleftarrow{r} D$ listed above are violated. We will see the performance for this sequence.

To determine the application sequence of the technologies by the proposed method, we assume that the maximal applied times of these seven technologies are one, two, one, one, one, two, and two, respectively. Then, based on the Petri net models developed in Section III for dependencies $A \xleftarrow{p} B$, $C \xleftarrow{r} D$, and $E \xrightarrow{r} F$, we obtain the Petri net model Σ for the combination dependency of the seven technologies as shown in Fig. 10, where the firings of Transitions $t_1 \sim t_7$ correspond to the application of Technologies $A \sim G$, respectively.

Then, by using Algorithm 1, the reachable marking graph of Σ is generated as shown in Fig.11.

Without loss of generality, in the experiments, we assume that, in applying these protection technologies, the objective is to maximize the total number of times of applying these technologies. By Algorithm 1, when the reachable marking graph is generated, we find marking M_9 and a non-control transition firing sequence from M_0 to M_9 such that the objective is maximized. As shown in Fig. 10, M_9 is

$$M_9 = [1, 1, 2, 0, 0, 2, 0, 1, 0, 1, 1, 0, 0, 0, 0]^T. \tag{25}$$

With M_9 and the non-control transition firing sequence, the total number of times of applying these technologies is calculated as

$$\sum F(M_9, t_i) = 1 + 2 + 2 + 1 + 1 + 1 = 8. \tag{26}$$

The software protection technology application sequence S_5 corresponding to the non-control transition firing sequence is

$$S_5 = (GADCBEFBGF). \tag{27}$$

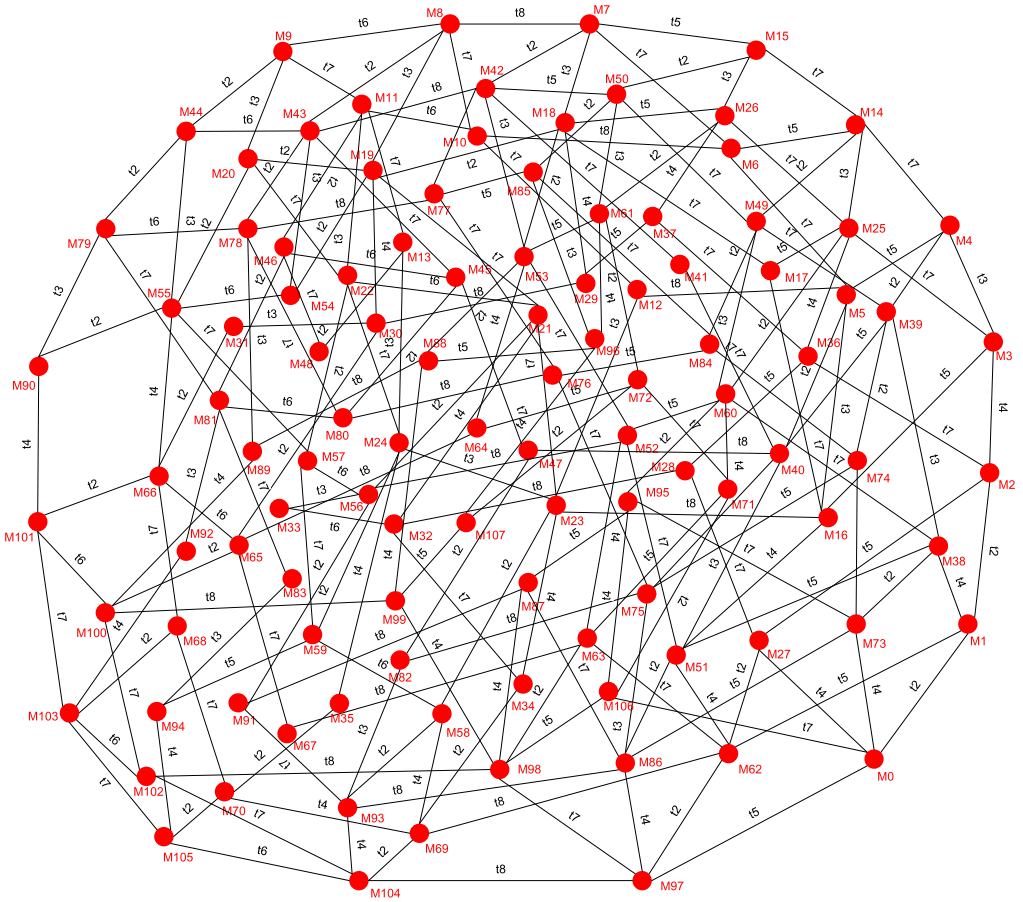


FIGURE 11. The reachable marking graph of Σ.

For the dependences of { $A \overset{p}{\longleftarrow} B$, $C \overset{r}{\longleftarrow} DE \overset{r}{\longrightarrow} F$, and G}, an FSA model is built as shown in Fig. 12. The initial state of this FSA model is s_0.

Now we take S_5 as the input sequence of this FSA model. The output sequence, $s_0s_2s_2s_1s_1s_4s_5s_3s_4s_5s_3$, is evaluated with this FSA model. From the output sequence, we find that the last state, s_3, is an acceptable state, so S_5 is verified as a legal software protection technology application sequence of this FSA model.

B. EXPRIMENTAL PROCEDURES

The experiments are carried out on platform *Sandmark* [7]. *Sandmark* is a famous platform built for studying code obfuscation, software watermarking and tamper proofing. To facility such studies, software protection technologies $A \sim G$ are integrated in it.

In the experiments, we use the test program *DES* as the input of *Sandmark* and the original program without any protection technology being applied is denoted as P. Then, *Sandmark* calls the software protection technologies sequentially according to the sequence specified by S_4 and S_5, respectively. The resulting programs with protection sequences S_4 and S_5 being applied are denoted as P_1 and P_2, respectively.

As examples, Fig. 13 illustrates the *Sandmark's* interface for the situation of applying the software protection technology G (Overload Names), while Fig. 14 illustrates the situation of applying the software protection technology E (Static Watermark).

C. EXPRIMENTAL RESULTS

1) SOFTWARE COMPLEXITY

As an important index for applying protection technology, we consider the software complexity to be compromised. We use *Sandmark* to calculate the software complexity indices of P, P_1 and P_2 and the results are shown in Table 3, where commonly recognized indices are used.

As shown in Table 3, for indices Halstead Measure [27], Harrison Measure [13], Munson Measure [26], and McCabe Cyclomatic [33] compared with P, the complexity of both P_1 and P_2 are increased. However, obviously the increase rate of the indices for P_2 is greater than that of P_1. The reason is that there is a dependency requirement: $C \overset{r}{\longleftarrow} D$, which requires that D should be applied before C. However, this dependency is violated in S_4. Note that$D$ stands for branch inverter and it is an important pre-process before C. Thus, the violation of dependency $C \overset{r}{\longleftarrow} D$ in S_4 results such a result.

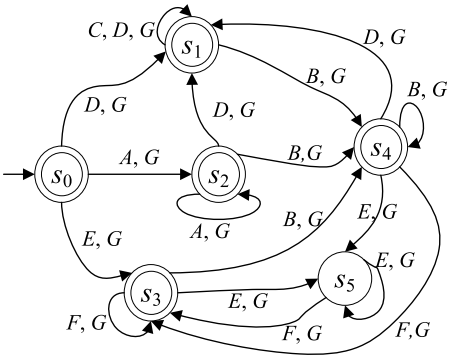


FIGURE 12. FSA for dependencies $\{A \xleftarrow{p} B, C \xleftarrow{r} D, E \xrightarrow{p} F \textit{andG}\}$.

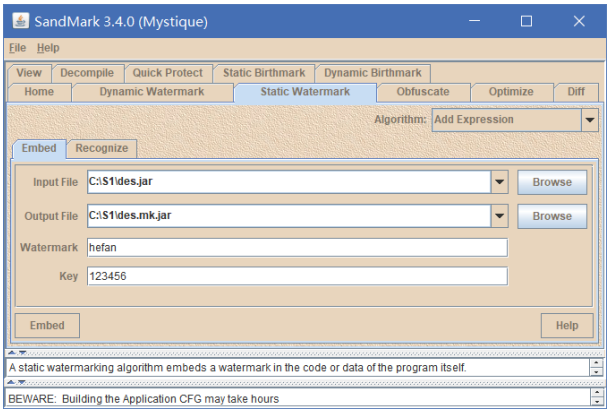


FIGURE 13. Code obfuscation panel of Sandmark (Overload Names).

TABLE 3. The complexity indices of P, P_1 and P_2.

Software complexity	P	P_1	P_2
Halstead Measure	38316	273362	366700
McCabe Cyclomatic	30	58	75
Harrison Measure	1450	1984	2292
Munson Measure	253	449	557
Kafura Measure	15129	26569	26569
CK Measure	54	54	54

The index Kafura measure [36] measures the fan-in/out complexity. All the seven technologies have no impact on this index. Thus, for Kafura Measure, both P_1 and P_2 have the same complexity.

The CK measure [30] is used to evaluate the complexity of object-oriented features. In this experiment, since the object-oriented features of *DES* remain intact when these seven technologies are applied. Hence, for the CK measure, the complexity of both P_1 and P_2 is equal to that of P.

Figs.15 and 16 show the control-flow graph of P_1 and P_2 drawing by *Sandmark*, denoted by G_1 and G_2, respectively. The detail of control-flow graph can be revealed by using the zoom out function in *Sandmark*. According to the statistics, the number of basic block is 171 and the number of branches is 46 in P_1, while they are 236 and 62 in P_2, respectively.

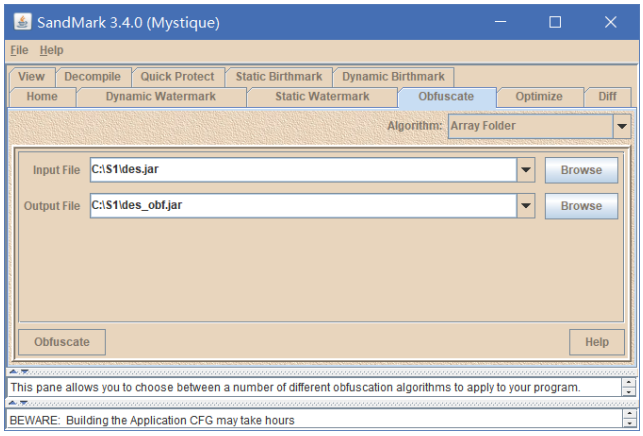


FIGURE 14. Watermarking panel of Sandmark (Static Watermark).

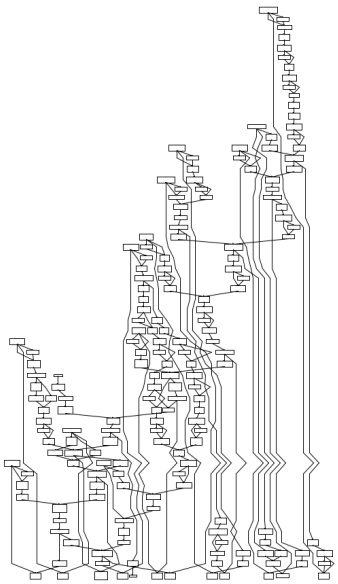


FIGURE 15. The control-flow graph of P_1.

Since G_2 is far more complex than G_1, we can conclude that the protection effect of applying these technologies according to sequence S_5 is much better than that according to sequence S_4.

2) WATERMARK EXTRACTION

Among the seven technologies in this experiment, B and E are both watermarking technologies, which are applied twice and once, respectively in both sequences S_4 and S_5. Thus, three watermarks should be extracted out in both P_1 and P_2 if they work. By experiment, we present the number of watermarks that are extracted out in P_1 and P_2, as shown in Table 4.

It follows from Table 4 that, all three watermarks are successfully extracted out in P_2, while only two are extracted out in P_1 with the first try being failed. The reason is that the pre-prohibition dependency $A \xleftarrow{p} B$ is violated in S_4. Dependency $A \xleftarrow{p} B$ means that B is prohibited to be applied

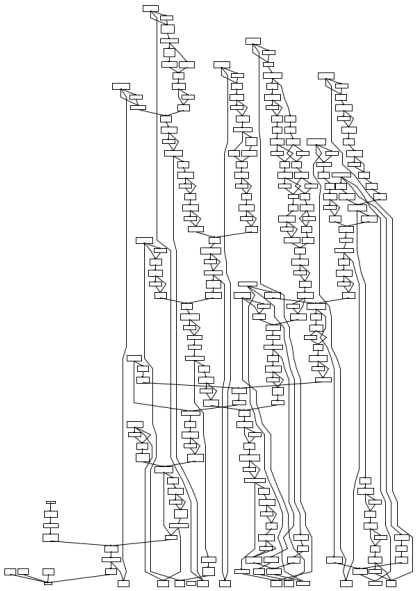


FIGURE 16. The control-flow graph of P_2.

TABLE 4. The results of watermarks extracted from P_1 and P_2.

Watermark technology	Watermark extraction from P_1	Watermark extraction from P_2
B	×	√
E	√	√
B	√	√

before A, which also means that once B is applied, A should not be applied thereafter. By analyzing P_1, it can be found that the watermark embedded by applying B earlier is then corrupted when A is applied after B.

VI. CONCLUSION

Nowadays, software security is a vitally important issue and various technologies are applied concurrently to protect a software product. It is known that, to make the application of multiple technologies to a software product, these technologies should be applied according to a certain sequence. However, there is no effective method to determine such a sequence. To solve this problem, in this paper, we use Petri nets to model the dependency of two technologies and develop Petri net models for four basic software protection technology dependencies. With these models, we then develop a Petri net model for the combination dependency for multiple technology application. Then, based on the Petri net model, we propose a method to generate the reachable marking graph such that a required and correct sequence of applying multiple protection technologies can be obtained. The feasibility of the proposed method has been verified by experiments.

In the future, we will study how to improve the performance of software protection as much as possible. Practice shows that the simple superposition of software protection technologies cannot achieve the expected results. Although, by this work, some achievements have been made, the obtained results need to be further refined and optimized.

REFERENCES

[1] B. Anckaert, M. H. Jakubowski, and R. Venkatesar, "Virtualization for diversified tamper resistance," U.S. Patent 8 584 109 B2, Nov. 12, 2013.

[2] L. Bai, N. Wu, Z. Li, and M. Zhou, "Optimal one-wafer cyclic scheduling and buffer space configuration for single-arm multicluster tools with linear topology," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 10, pp. 1456–1467, Oct. 2016.

[3] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner, "Code obfuscation against symbolic execution attacks," in *Proc. ACM Conf. Comput. Secur. Appl. (CSA)*, Los Angeles, CA, USA, 2016, pp. 189–200.

[4] B. Barak *et al.*, "On the (im)possibility of obfuscating programs," *J. ACM*, vol. 59, no. 2, 2001, Art. no. 6.

[5] N. Chaussade, F. Begon, M. Teyssier, R. Teyssier, and J. F. O. Jaubert, "Checkpointing long latency instruction as fake branch in branch prediction mechanism," U.S. Patent 8 578 139 B2, Nov. 5, 2013.

[6] H. Chen, N. Wu, and M. Zhou, "A novel method for deadlock prevention of AMS by using resource-oriented Petri nets," *Inf. Sci.*, vol. 363, pp. 178–189, Oct. 2016.

[7] C. Collberg. (2012). *A Tool for the Study of Software Protection Algorithms*. [Online]. Available: http://sandmark.cs.arizona.edu

[8] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation—Tools for software protection," *IEEE Softw. Eng.*, vol. 28, no. 8, pp. 735–746, Aug. 2015.

[9] C. Collberg, C. D. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. #148, 1997.

[10] D. Dunaev and L. Lengyel, "Method of software obfuscation using Petri nets," in *Proc. 24th Central Eur. Conf. Inf. Intell. Syst. (CECIIS)*, 2013, pp. 242–296.

[11] A. Edelsten, F. Fomichev, J. Huang, and T. Lottes, "Code protection using online authentication and encrypted code execution," U.S. Patent 9 177 121 B2, Nov. 3, 2015.

[12] G. Gupta and J. Pieprzyk, "Software watermarking resilient to debugging attacks," *J. Multimed.*, vol. 2, no. 2, pp. 10–16, 2007.

[13] W. A. Harrison and K. I. Magel, "A complexity measure based on nesting level," *ACM SIGPLAN Notices*, vol. 16, no. 3, pp. 63–74, 1981.

[14] K. Heffner and C. Collberg, "The obfuscation executive," in *Proc. Inf. Secur. Conf. (ISC)*, 2004, pp. 428–440.

[15] Y. Hou, N. Wu, M. Zhou, and Z. Li, "Pareto-optimization for scheduling of crude oil operations in refinery via genetic algorithm," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 517–530, Mar. 2017.

[16] S. Ilić and S. Đukić, "Protection of Android applications from decompilation using class encryption and native code," in *Proc. IEEE Zooming Innov. Consum. Electron. Int. Conf. (ZICEIC)*, Jun. 2016, pp. 10–11.

[17] B. Jasiul, M. Szpyrka, and J. Śliwa, "Malware behavior modeling with colored Petri nets," in *Proc. Int. Conf. Comput. Inf. Syst. Ind. Manage. (CISIM)*, 2014, pp. 667–679.

[18] D. D. Karale, A. A. Tapase, and B. S. Chaudhari, "Software protection against piracy and reverse engineering using software watermarking technique," *Int. J. Emerg. Trends Sci. Technol.*, vol. 1, no. 7, pp. 1205–1210, 2014.

[19] A. Kulkarni, "Software protection through code obfuscation," M.S. thesis, Dept. Comput. Eng. Inf. Technol., College Eng., Pune, India, 2012.

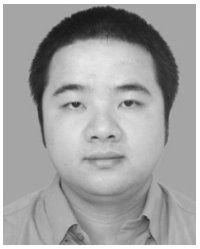
[20] D. Lacey and O. de Moor, "Detecting disabling interference between program transformations," Comput. Lab., Oxford Univ., Oxford, U.K., Tech. Rep., 2001.

[21] S.-H. Lin, "Alias analysis in LLVM," M.S. thesis, Dept. Comput. Sci., Lehigh Univ., Bethlehem, PA, USA, 2015.

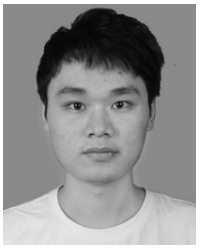
[22] H. Lin, X. Zhang, M. Yong, and B. Wang, "Branch obfuscation using binary code side effects," in *Proc. Int. Conf. Comput. Netw. Commun. Eng. (ICCNCE)*, Beijing, China, 2013, pp. 152–157.

[23] M. Llorens and J. Oliver, "Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets," *IEEE Trans. Comput.*, vol. 53, no. 9, pp. 1147–1158, Sep. 2004.

[24] M. Becchi and P. Crowley, "Efficient regular expression evaluation: Theory to practice," in *Proc. 4th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, Nov. 2008, pp. 50–59.
[25] M. Mohri and M.-J. Nederhof, "Systems and methods for generating weighted finite-state automata representing grammars," U.S. Patent 0 120 480 A1, Jun. 26, 2013.
[26] J. C. Munson and T. M. Kohshgoftaar, "Measurement of data structure complexity," *J. Syst. Softw.*, vol. 20, no. 3, pp. 217–225, 1993.
[27] A. E. Okeyinka and O. M. Bamigbola, "Numerical study of depth of recursion in complexity measurement using Halstead measure," *Int. J. Appl. Sci. Technol.*, vol. 2, no. 6, pp. 106–111, 2012.
[28] K. Reinhardt, "Reachability in Petri nets with inhibitor arcs," *Electron. Notes Theor. Comput. Sci.*, vol. 223, pp. 239–264, Dec. 2008.
[29] R. Senthilkumar and A. Thangavelu, "Code security using control flow obfuscation with opaque predicate," *Int. J. Appl. Eng. Res.*, vol. 10, no. 2, pp. 3239–3250, 2015.
[30] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.
[31] Q. Su, Z.-Y. Wang, W.-M. Wu, J.-L. Li, and Z.-W. Huang, "Technique of source code obfuscation based on data flow and control flow tansformations," in *Proc. 7th Int. Conf. Comput. Sci. Edu. (ICCSE)*, 2012, pp. 1093–1097.
[32] Y.-S. Han and D. Wood, "Obtaining shorter regular expressions from finite-state automata," *Theor. Comput. Sci.*, vol. 370, nos. 1–3, pp. 110–120, 2007.
[33] J. J. Vinju and M. W. Godfrey, "What does control flow really look like? Eyeballing the cyclomatic complexity metric," in *Proc. IEEE Int. Working Conf. Source Code Anal. Manipulation*, Sep. 2012, pp. 154–163.
[34] W. H. Jun, F. D. Yi, W. Ni, and G. Y. Xiang, "Effectiveness evaluation of software protection based on MPN," in *Proc. 2nd Int. Conf. Adv. Comput. Sci. Eng. (ACSE)*, 2013, pp. 83–86.
[35] H. Wang, D. Fang, N. Wang, Z. Tang, F. Chen, and Y. Gu, "Method to evaluate software protection based on attack modeling," in *Proc. IEEE 10th Int. Conf. High-Perform. Comput. Commun., Int. Conf. Embedded Ubiquitous Comput. (ICHPCC/ICEUC)*, Nov. 2014, pp. 837–844.
[36] Y.-Y. Wang, Q.-S. Li, P. Chen, and C.-D. Ren, "Dynamic fan-in and fan-out metrics for program comprehension," *J. Shanghai Univ.*, vol. 11, no. 5, pp. 474–479, Oct. 2007.
[37] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, "Towards smart factory for industry 4.0: A self-organized multi-agent system with big data based feedback and coordination," *Comput. Netw.*, vol. 101, pp. 158–168, Jun. 2016.
[38] G. Wroblewski, "General method of program code obfuscation," Ph.D. dissertation, Inst. Eng. Cybern., Wrocław Univ. Technol., Wrocław, Poland, 2002.
[39] N. Wu, F. Chu, C. Chu, and M. Zhou, "Petri net modeling and cycle-time analysis of dual-arm cluster tools with wafer revisiting," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 1, pp. 196–207, Jan. 2013.
[40] N. Wu, Z. Li, and T. Qu, "Energy efficiency optimization in scheduling crude oil operations of refinery based on linear programming," *J. Cleaner Prod.*, vol. 166, pp. 49–57, Nov. 2017.
[41] N. Q. Wu and M. C. Zhou, "Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 1, pp. 203–209, Jan. 2012.
[42] N. Q. Wu and M. C. Zhou, "Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 2, pp. 446–454, Apr. 2012.
[43] N. Wu, M. Zhu, L. Bai, and Z. Li, "Short-term scheduling of crude oil operations in refinery with high-fusion-point oil and two transportation pipelines," *Enterprise Inf. Syst.*, vol. 10, no. 6, pp. 581–610, May 2016.
[44] N. Wu, M. Zhou, and Z. Li, "Short-term scheduling of crude-oil operations: Enhancement of crude-oil operations scheduling using a Petri net-based control-theoretic approach," *IEEE Robot. Autom. Mag.*, vol. 22, no. 2, pp. 64–76, Jun. 2015.
[45] F. Yang, N. Wu, Y. Qiao, M. Zhou, and Z. Li, "Scheduling of single-arm cluster tools for an atomic layer deposition process with residency time constraints," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 502–516, Mar. 2017.
[46] J. Ye, Z. Li, and X. Chen, "An algorithm for the minimum initial marking problem of a structurally live Petri net with inhibitor arcs," *IEEJ Trans. Elect. Electron. Eng.*, vol. 11, no. 5, pp. 586–592, 2016.
[47] G. Zhang *et al.*, "Attack simulation based software protection assessment method," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (CSPDS)*, 2016, vol. 1. no. 1, pp. 1–8.
[48] S. Zhang, N. Wu, Z. Li, T. Qu, and C. Li, "Petri net-based approach to short-term scheduling of crude oil operations with less tank requirement," *Inf. Sci.*, vol. 417, pp. 247–261, Nov. 2017.
[49] S. Zhang, G. Zhu, and Y. Wang, "A strategy of software protection based on multi-watermarking embedding," in *Proc. 2nd Int. Conf. Intell. Syst. Appl. Mater. (ISAM)*, 2013, pp. 444–447.
[50] J. Q. Zhang, L. N. Ni, and C. J. Jiang, "An algorithm to construct concurrent reachability graph of Petri nets," (in Chinese) *J. Donghua Univ.*, vol. 21, no. 3, pp. 180–184, 2004. [Online]. Available: http://www.airitilibrary.com/Publication/alDetailedMesh?docid=16725220-200406-21-3-180-184-a



QING SU received the B.S. and M.S. degrees from the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China. He is currently an Associate Professor with the School of Computer Science and Technology, Guangdong University of Technology. His research interests include software security and protection, discrete event systems, and Petri net theory and applications.



FAN HE received the B.S. degree in computer science and technology from the Jingchu University of Technology in 2014. He is currently pursuing the master's degree in computer science and technology with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China. His research interests include software security and protection.

NAIQI WU (M'04–SM'05) received the B.S. degree in electrical engineering from the Anhui University of Technology, Huainan, China, in 1982, and the M.S. and Ph.D. degrees in systems engineering from Xi'an Jiaotong University, Xi'an, China, in 1985 and 1988, respectively. From 1988 to 1995, he was with the Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, China, and from 1995 to 1998, he was with Shantou University, Shantou, China. He moved to the Guangdong University of Technology, Guangzhou, China, in 1998. He joined the Macau University of Science and Technology, Macau, China, in 2013, where he is currently a Professor with the Institute of Systems Engineering. His research interests include production planning and scheduling, manufacturing system modeling and control, discrete event systems, Petri net theory and applications, intelligent transportation systems, and energy systems. He has authored or co-authored one book, five book chapters, and over 130 peer-reviewed journal papers. He was an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C, the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, and the Editor-in-Chief of the *Industrial Engineering Journal*. He is an Associate Editor of *Information Sciences* and the IEEE/CAA JOURNAL OF AUTOMATICA SINICA.

ZHIYI LIN received the M.S. degree in computer science and technology from the Wuhan University of Technology, Wuhan, China, in 2006, and the Ph.D. degree in computer software and theory from the State Key Laboratory of Software Engineering, Wuhan University, Wuhan, in 2009. He is currently a Teacher with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China. His research interests include evolutionary computation, pattern recognition, and software security and protection.