Attack Simulation based Software Protection Assessment Method

Gaofeng Zhang, Paolo Falcarin, Elena Gómez-Martínez, Shareeful Islam
University of East London, London, UK
{g.zhang, falcarin, e.gomez, s.islam}@uel.ac.uk

Christophe Tartary
Saarland University, Saarbrücken, Germany
christophe_tartary@icloud.com

Bjorn De Sutter
Ghent University, Ghent, Belgium
bjorn.desutter@ugent.be

Jérôme d'Annoville
Gemalto, Meudon, France
jerome.d-annoville@gemalto.com

***Abstract*—Software protection is an essential aspect of information security to withstand malicious activities on software, and preserving software assets. However, software developers still lacks a methodology for the assessment of the deployed protections. To solve these issues, we present a novel attack simulation based software protection assessment method to assess and compare various protection solutions. Our solution relies on Petri Nets to specify and visualize attack models, and we developed a Monte Carlo based approach to simulate attacking processes and to deal with uncertainty. Then, based on this simulation and estimation, a novel protection comparison model is proposed to compare different protection solutions. Lastly, our attack simulation based software protection assessment method is presented. We illustrate our method by means of a software protection assessment process to demonstrate that our approach can provide a suitable software protection assessment for developers and software companies.**

Keywords—Software Security; Software Protection Assessment; Attack Simulation; Monte Carlo Method; Petri Net

I. INTRODUCTION

Currently, software is an extremely important asset for customers to support and execute their businesses. Consequently, software protection has attracted much attention from developers and software companies in terms of software security. To ensure security against malicious software attacks, many tools have been developed, such as data obfuscation, tamper-proofing, code splitting, software watermarking, among others [13].

In this regard, assessing the effectiveness of these protections is crucial before embedding them into real commercial products. In particular, in practical use cases, like mobile computing, multiple protection methods could be utilised together as Protection Solutions (*PS*s) to thwart actual threats. Therefore, a software protection assessment method needs to be able to assess potential *PS*s with respect to various types of attacks. This is the context where this paper takes place.

Currently, one main type of software protection assessment [15] focuses on the assessment of individual protection methods and does not consider *PS*s with multiple protection methods. Another kind of software protection assessment [14] discussed general software measurement frameworks for protection, and did not involve *PS*s either. Hence, none of these two approaches is suitable for protection assessment in terms of complicated *PS*s to provide convincing results.

Besides, for real software attack processes, uncertainty is another challenge for these existing assessment methods. In uncertain software attacking processes, there are many random variables and factors at play, such as the computing resources for attacking, the decisions or selections made by specific attackers, and so on. Moreover, specific environments, like mobile computing, could jeopardise this uncertainty by the fragmentation of mobile *OS*. To capture this phenomenon, we could use a non-deterministic attack simulation based on the Monte Carlo method to describe the real uncertain software attacking processes. This idea will be the basic tool for our proposed method.

Petri Net (*PN*) based attack models are suitable objects to model software attacks [8, 10], and in this work we use them to support software protection assessment in terms of Monte Carlo based attack simulation.

In real software protection implementations, assessing every possible *PS*s in each specific software protection situation is a huge task, considering the myriad of possible combinations of various *PS*s (multiple protection methods with their parameters) and various software protection situations (multiple attacks with weights). Hence, the relations (comparing results) among *PS*s under protection assessments are particularly valuable for this. Our assessment method can use a comparison model to manage the comparisons among *PS*s on the basis of the attack simulation. As such, the protection comparison model is the central component of our assessment methodology.

To summarize our approach, our novel Attack Simulation based Software Protection Assessment Method (*ASSPAM*) uses a Monte Carlo based Attack Simulation (*MCAS*) to simulate specific software attack processes with implementing *PS*s, based on *PN* based attack models. Then, using the results obtained from the *MCAS*, our Attack Simulation based Protection Comparison Model (*ASPCM*) provides a numeric estimation of the *PS* and thus this can be compared in the *ASSPAM* to search for the best *PS.*

The main contributions of this paper includes: 1) a Monte Carlo based attack simulation for protection assessment; 2) a novel attack simulation based protection comparison model to compare *PS*s with numeric confidences; 3) a novel attack simulation based software protection assessment method to run the assessment.

This research has been carried out within the European *FP7* project *ASPIRE*, Advanced Software Protection: Integration, Research, and Exploitation [1]. Our method focuses on the assessment of *PS*s, and does not cover the generation and optimisation of these *PS*s. Some other components of *ASPIRE* store security experts' knowledge and experiences in the Knowledge Base, and generates *PS*s by means of reasoning technique. Besides, to generate and optimise *PS*s, there are some aspects, such as the cost of protections, the dependency among protection methods, and so on, which are out of the scope of this paper.

Furthermore, compared to traditional *PN*s [8], our *PN* based attack models focus on attack steps (transitions) with related simulation

information. Therefore, some features in traditional *PN*s are not involved in this paper, such as tokens and liveness. *PN*s with full characteristics will be utilised by other software protection assessment approaches in the *ASPIRE* project.

The paper is organized as follows. Section II describes related work. Section III discusses preliminary concepts and background. In Section IV, our new method is proposed. In Section V, we use a protection assessment instance to demonstrate that our proposed method can provide suitable protection assessments. Section VI concludes this paper and points out future work.

II. RELATED WORK

This section introduces existing research progress in the areas of software analysis and measurement, attack modelling, Monte Carlo simulation and software protection assessment.

A. Software Analysis and Measurement

Software analysis and measurement are important areas in software engineering [2]. For example, in the software process improvement area, measurement has been emphasised as a central function and activity [3]. From a software security viewpoint, Tonella *et al.* [4] presented a general framework to assess a software by various measurable features and metrics to withstand software attacks. Related software analysis and measurement mechanisms are valuable references for our software protection assessment.

B. Attack Modelling

Currently, attack modelling is an important area of information security. Attack Tree models and Attack Graphs are widely used for representing network attacks, virus attacks, and so on [5,6]. For example, the scalable modelling process is studied for attack graph generation with logic formalism in [7]. However, none of them can precisely describe preconditions, actions and external impacts in software attack processes properly.

In this regard, Petri Nets (*PN*s) were originally introduced as a modelling technique for concurrent systems [8].These nets can model specific cyber-physical attacks on smart grid [9]. Taking software protection as objective, Wang *et al.* [10] focused on coloured *PN* based attack modelling. As discussed in Section I, *PN* based attack models are suitable to describe preconditions, post-conditions and actions and, therefore, they will play a core role in our attack modelling and protection assessment.

C. Monte Carlo Simulation

The Monte Carlo simulation (method) is a powerful tool for dealing with uncertainty and probability [11]. It is very useful for analysing and simulating complex systems and problems, due to its flexibility and error-quantifiable features [12].

Hence, the Monte Carlo method is a suitable technique to simulate complex systems in terms of multiple random variables. As discussed in Section I, in this paper, we assess various *PS*s in real uncertain attacking processes to provide suitable *PS*s for developers and software companies. With the help of the Monte Carlo method, we can use attack simulation to support the *ASSPAM* when assess protections.

D. Software Protection Assessment

As previous discussed, software protection assessment is an essential part of software protection [13]. Basile *et al.* [14] described a unified high-level software attack model to assess software protections for developers. Experiments have been designed to assess the effectiveness and efficiency of related software protection techniques for code obfuscation [15] [18] [19].

Existing protection assessment methods are too specialised or too general to cope with uncertain software attack processes and *PS*s. That is why, to overcome this issue, our *ASSPAM* will be relying on *PN* based attack models and Monte Carlo based attack simulations.

III. PETRI NET BASED ATTACK MODEL

In this section, we discuss the *PN* based attack model for attack simulation, which is the basic supporting tool of our *ASSPAM*, especially for our *MCAS*.

A. Petri Net based Software Attack Modelling

PN based attack models are the essential rationale for our assessment method in this paper. Generally speaking, *PN* based attack models represent all possible attack paths and attack steps in software attacks, and support the attack simulations on these attacks via some extra information.

Based on existing work [8,10], we present our models to describe software attacks for protection assessment:

Definition 1 *PSAM*: A *PN* based Software Attack Model for simulation is a five-tuple, *PSAM = (P, T, A, EC, AE)*, where:

- *P* is a finite set of states, represented by circles. These states model sub-goals reached by an attacker after having executed a number of attack steps. $P=\{P_0,\ldots\ldots,P_n\}$.
- *T* is a finite set of transitions, represented by rectangles. These transitions model attack steps, i.e., specific actions undertaken by attackers to reach a sub-goal on a path to the final end goal of their attack. $T=\{T_0,\ldots\ldots,T_m\}$.
- $P \cup T \neq \emptyset$, $P \cap T = \emptyset$.
- $A \subseteq \{T \times P\} \cup \{P \times T\}$, is a multi-set of direct arcs, relating sub-goals and attack steps. $A=\{A_0,\ldots\ldots,A_l\}$.
- *EC* represents the Effort Consumption. It is a finite set of attacker's effort consumed at each transition in *T*, where $EC=\{ec_0,\ldots\ldots,ec_m\}$. It is utilised as the preconditions.
- *AE* represents the Attacker Effort. It is a finite set of attacker's effort at each state in *P*, where $AE=\{ae_0,\ldots\ldots,ae_n\}$. Attackers have the capability including resources and skills to execute attacks on protected or unprotected software. This "capability" is represented with *AE* and will be "consumed" in transitions of attack processes via *EC* in attack simulations.

EC and *AE* will be discussed further in the next subsection, which are the key issues to support the attack simulations for protection assessment.

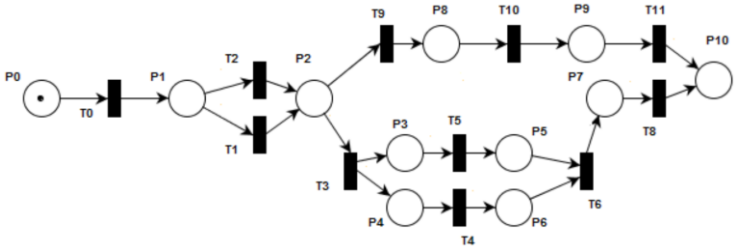

Figure 1. *PN* based attack model on a one-time password generator

As an example of a relevant use case, Figure 1 and Table 1 present relevant attack paths on a One-Time Password (*OTP*) generator [16] by means of *PSAM*: P0 is the starting state in which attackers start to attack the *OTP* software, and P10 is the final state which means a successful attacking. Specially, this success means that attackers obtain the seed of the *OTP* generator. P1, P2, P3, P4, P5, P6, P7, P8, and P9 are nine intermediate states in the attack, corresponding to different sub-goals being reached. T0, T1, T2, T3, T4, T5, T6, T8, T9, T10, and T11 are

eleven transitions, which describe various attack steps (actions) in attack processes, detailed in Table 1.

Table 1. Attack Table of *PN* based attack model on a one-time password generator

	Description/Objective	Input	Output
T0	Identify PIN section of the code	Original code	Piece of code containing PIN checking
T1	Bypass PIN check	Piece of code containing PIN checking	N/A
T2	Bypass PIN check	Piece of code containing PIN checking	PIN obtained
T3	Set-up for parallel run	N/A	N/A
T4	Unlock provisioning phase	Original code	Reusable provisioning phase (piece of code)
T5	Fake server setting	N/A	Server ready
T6	AES decryption code identification	Fake server (P5) + Reusable provisioning code (P6)	Piece of code containing the AES deciphering algorithm
T8	Seed recovery	AES decryption code + real server	Seed
T9	Code pruning for XOR localization	Original code	Code fragments (executed before OTP display)
T10	XOR chains identification	Code fragments (P8)	Sequence of XOR operations (piece of code)
T11	Seed recovery	Sequence of XOR operations	Seed

PSAM is the basic supporting tool in this paper by providing attack models with attack paths and steps, and the part with *AE* and *EC* will be introduced further in the next subsection to complete the model.

B. Effort Consumption and Attacker Effort

In this subsection, we detail the Effort Consumption (*EC*) and the Attacker Effort (*AE*) as the part of the *PSAM* to support attack simulation for protection assessment particularly.

In this paper, we use uniform distributions to describe Effort Consumption—*EC* and ec_i. For each ec_i, a Maximum boundary—Max_i and a Minimum boundary—Min_i decide this random variable by the uniform distribution in equation (1).

$$EC = \{ec_0, ..., ec_i, ..., ec_m\}, ec_i = fec(Min_i, Max_i), i \in [0, m] \quad (1)$$

In equation (1), *fec()* represents the sampling process of the uniform distribution with two boundaries: Min_i and Max_i. For example, T0 in the *OTP* attack model is to "Identify the PIN check portion of the code". Both Max_0 and Min_0 can be set in the attack modelling by users or security experts in industry, based on real attack data. After that, ec_0 is the random variable with the uniform distribution and two boundaries: Max_0 and Min_0. Both boundaries can be increased due to the fact that some protections have been applied: for example, when some software protection methods increase the code size or the flow complexity, this can make the T0 attack step more difficult, which will change the uniform distribution for ec_0 with Max_0 and Min_0. These methods could be specific *PS*s to change ec_0. The relations between methods and transitions are decided by users, as well as existing knowledge. In other words, these Min_i and Max_i (and *EC*) depend on various *PS*s.

Another concept of *PSAM* is *AE*, which represents the current effort of the attacker in the state of this attack process. *AE* can be described by equation (2). In this equation, ae_0 is the attacker effort before attack processes (in the initial place). In this paper, we set ae_0 as a random variable with a normal distribution.

$$AE = \{ae_0, ..., ae_i, ..., ae_n\} \quad (2)$$

As introduced in Section I, since the attacker is one key part of the simulation, we will use a normal distribution to represent real uncertain attacking processes for one attacker.

Using a *PSAM* is the basis of our method in this paper, and the attack simulation, protection comparison model and protection assessment method rely on this.

In the next section, we will introduce the main content in this paper—*ASSPAM*.

IV. ATTACK SIMULATION BASED SOFTWARE PROTECTION ASSESSMENT METHOD

Based on previous discussions, we will introduce our novel *ASSPAM* in three steps: firstly, a *MCAS* simulates attack processes with *PS*s, based on *PN* based attack models described in Section III; secondly, we will introduce our *ASPCM* to compare different *PS*s based on previous attack simulations; lastly, *ASSPAM* will be introduced based on *MCAS* and *ASPCM* to provide suitable *PS*s as the protection assessment results.

A. Monte Carlo based Attack Simulation

Monte Carlo based Attack Simulation (*MCAS*) includes two parts: the Single Attack Process Simulation (*SAPS*) and the Monte Carlo Method. They will be introduced in the following.

1) Single Attack Process Simulation

The main process of Single Attack Process Simulation (*SAPS*) works as follows: in one *PSAM* (as a Directed Acyclic Graph), one attacker will try to move from the starting state to the final state. If he/she successes, the result of this *SAPS* is TRUE; otherwise, it is FALSE. It can be viewed as a route searching process in the directed acyclic graph.

In *SAPS*, in each node, e.g. transition, we use the Passing Probability—*PP* to control the probability that the attacker completes this transition (attack step) and reach the next state.

Passing Probability (*PP*): A finite set for each transition in *T*, and $pp_i \in PP, i \in [0,m]$.

$$pp_i = \begin{cases} 0, ae_{CUR} < ec_i \\ \tanh(ae_{CUR}/ec_i - 1), ae_{CUR} \geq ec_i \end{cases}, i \in [0, m] \quad (3)$$

In equation (3), ec_i comes from equation (1), which is the effort consumption for each attack step. And ae_{CUR} comes from equation (2), which is the current attacker effort in one attack simulation process. If ae_{CUR} is smaller than ec_i, the probability is zero, which means that the current attacker effort is too low to complete this attack step. Otherwise, if ae_{CUR} is not smaller than ec_i, the passing probability is required to be monotonically increasing and in the range of [0, 1) when x is in $[0,+\infty)$. To match this, we use the hyperbolic tangent function: $\tanh(x) = (1 - e^{-2x})/(1 + e^{-2x})$.

Besides, after discussing the pre-conditions of transitions (*PP*), the actions of transitions will focus on the changing on *AE* and the current place.

$$ae_{NEW} = \begin{cases} ae_{CUR} - ec_i, on\_the\_probability\_of\_(pp_i) \\ ae_{CUR}, on\_the\_probability\_of\_(1 - pp_i) \end{cases} \quad (4)$$

In equation (4), for transition *Ti*, on the probability of pp_i, ae_{CUR} will be subtracted by ec_i, which means that the attacker passes this transition to arrive the next place after this transition. Otherwise (i.e., with probability 1- pp_i), the attacker needs go back to the previous state to find other paths to reach the final state, and ae_{CUR} is still the same.

Briefly, the *SAPS* is the basis to Monte Carlo based attack simulation for protection assessment by indicating attack processes on *PSAM*.

2) Monte Carlo based Attack Simulation

Based on this *SAPS model*, we use the Monte Carlo method to manage the *SAPS* and to provide a randomized simulator emulating the attack processes success. The *MCAS* is illustrated in Figure 2. The key component is our *SAPS*. To run the *SAPS*, we need to perform an initialisation phase in order to build the underlying *PN* based attack model with *EC* and *AE* as introduced in Subsection III.B. The result of each *SAPS* is of type Boolean. Then, the Monte Carlo method executes the *SAPS* multiple times. Finally, the simulation provides a probability of attack success (the ratio of *SAPS*s with TRUE in all *SAPS*s).

Besides, as introduced in Subsection III.B, specific *PS*s can decide specific *EC*s in *PSAM*. Hence, the result of one *MCAS* process is a probability of attack success on one *PN* based attack model and one *PS*.

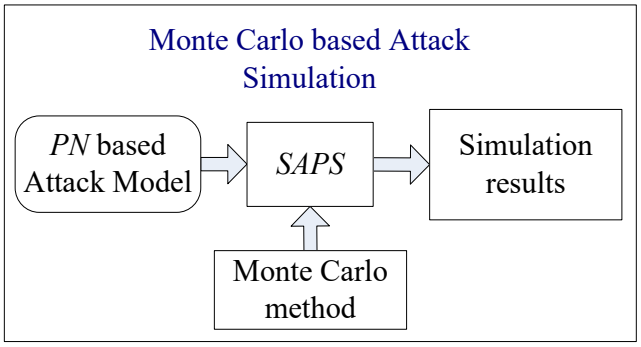

Figure 2. Monte Carlo based Attack Simulation (*MCAS*)

In brief, *MCAS* is the basic tool of *ASPCM* and *ASSPAM*.

B. Attack Simulation based Protection Comparison Model

We now present our *ASPCM*. As indicated in Section I, the main target of *ASPCM* is to compare *PS*s with numeric confidences by means of *MCAS*. To reach this aim, we introduce two such values as Compare Confidence and Neutral Confidence.

Based on Subsection IV.A, a probability of attack success is the result of one *MCAS* process with one *PN* based attack model and one *PS*. If we compare two Protection Solutions, for instance *PS-1* and *PS-2,* we can assume that there are two probabilities: p_1 and p_2 representing the results of *MCAS*s being executed based on *PS-1* and *PS-2* respectively.

To describe the confidence of the comparison, it is an intuitive way to use the difference of these two probabilities, like $p_2 - p_1$ under the assertion: *PS-1* is better than *PS-2*. Besides, to enhance the previous confidence, we consider the situation that these two *PS*s cannot be distinguished, which includes two kinds of events: an attacker can successfully break *PS-2* while he/she is able to break *PS-1*; and an attacker cannot break *PS-2* while he/she is unable to break *PS-1*. Therefore, the probability of these two events can be defined as $p_1 \times p_2 + (1 - p_1) \times (1 - p_2)$, which is equation (6). As such, our two confidences are expressed by equations (5) and (6).

For the assertion: *PS-1* is better than *PS-2*, with confidences including:

Compare Confidence (*CC*):

$$CC = p_2 - p_1 \quad (5)$$

Neutral Confidence (*NeuC*):

$$NeuC = 1 + 2 \times p_1 \times p_2 - p_1 - p_2 \quad (6)$$

Based on results of *MCAS*—the probabilities of successful attack, the *ASPCM* consider the comparisons based on assertions (*PS-1* is better than *PS-2*, or *PS-2* is better than *PS-1*.) with using the corresponding confidence values.

Therefore, based on *MCAS*, the *ASPCM* can generate assertions with numeric confidences as the comparison results of various *PS*s. These results can be utilised to generate the final protection assessment results of *ASSPAM*, which will be introduced in the next subsection.

C. Attack Simulation based Software Protection Assessment Method

In this subsection, we will introduce our novel Attack Simulation based Software Protection Assessment Method (*ASSPAM*), based on previous *MCAS* and *ASPCM*.

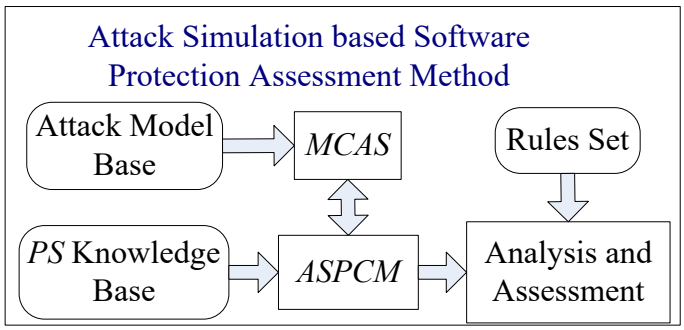

Figure 3. Attack Simulation based Software Protection Assessment Method (*ASSPAM*)

In Figure 3, we depict our *ASSPAM*. Component "*PS* Knowledge Base" provides all potential Protection Solutions (*PS*s) as specific and validated empirical accumulations of developers and software companies. Component "Attack Model Base" provides all *PN* based attack models required to be assessed. In *ASPIRE* project [1], these two bases are provided by other components, and will be out of the scope of this paper.

Component "*MCAS*" is the simulation part of the whole assessment method, and it receives *PN* based attack models from the "Attack Model Base". Component "*ASPCM*" receives *PS* candidates from the "*PS* Knowledge Base", forwards them to the "*MCAS*", and executes "*MCAS*" for comparing these *PS* candidates by simulation results. Component "Rules Set" provides some specific rules to aid comparing results and generate specific suitable *PS*s as final assessment results. These rules are specified by implementation situations, and we will deliver some examples in Subsection V.C. Component "Analysis and Assessment" analyses the results of "*ASPCM*"—comparison assertions with confidences and corresponding software attacks to assess *PS* candidates by "Rules Set" for developers and software companies.

In *ASSPAM*, the main process is that users need to set the software protection situations firstly (selecting attack models from the "Attack Model Base"), including which attacks need to be considered and the weights on them. Then, the *ASPCM* can be triggered to select potential *PS*s (from the "*PS* Knowledge Base") to be compared and assessed. These potential *PS*s can be executed by the *MCAS* for generating related probabilities of attacking successful. Based on these probabilities, the *ASPCM* can generate the comparison results between *PS*s with numeric confidences. In the last step, relying on these comparison outputs, users can use some specific rules (from the "Rules Set") to select some suitable *PS*s as the final assessment results of our *ASSPAM*. These results can be used to optimise *PS*s in the *ASPIRE* project [1].

In short, the *ASSPAM* executes as sub-routines the *ASCPM* and the *MCAS* to assess different *PS*s under the *PSAM* in order to obtain suitable assessment results in terms of software protection requirements (rules).

V. IMPLEMENTATION

In this section, we will illustrate our *ASSPAM* with *MCAS* and *ASPCM* by implementations and experiments on software protection assessment for developers and software companies. Generally speaking, the implementation of *ASSPAM* will be introduced in the order of *MCAS*, *ASPCM* and *ASSPAM*. Firstly, we use an example to illustrate the implementation on *MCAS*. Then, we use specific *PN* based attack models to compare various *PS*s via *ASPCM* in terms of numeric confidences. Lastly, we will analyse the results from *ASPCM* and generate the suitable *PS*s with rules sets as the final protection assessment results of *ASSPAM*.

A. Implementation of MCAS

In this subsection, we use a prototype implementation of *MCAS* on the *OTP* attack to demonstrate the process of attack simulation. We set the ae_0 as a normal distribution variable with mean 200 and variance 25.

Based on the *OTP* attack model shown in Figure 1, these 11 transitions can be classified into four categories: Category 1-locating code pieces (T0, T6, T8, T9, T10, and T11), Category 2-bypassing or tampering code pieces (T1, T4), Category 3-code injecting (T2, T5), and Category 4-NULL activities (T3).

We hosted a student attacking experiment on these attack activities in 2015/10/23-2015/10/29 at University of East London, involving postgraduates (5 persons), PhD candidates (3 persons), and Post-Docs (4 persons). And we can use the time records of this experiment to support the setting of *EC* in each transition in the *OTP* attack model of Figure 1. For example, for the attack activities in Category 2: bypassing or tampering code pieces, "attackers" in our experiments spent different times: the shortest one is 10 mins, and the longest one is 75 mins. So, we can use these "10" and "75" as the boundaries: Min_i and Max_i to related transitions: T1 and T4 as discussed in Subsection III.B, which be used to build the ec_i as a discrete uniform distribution. Similarly, for each other transition, ec_i can be built on the basis of these shortest and longest times.

Table 2. Time Ranges for Various Attack Activities

	Category 1	**Category 2**	**Category 3**	**Category 4**
Time Range (*mins*)	[3, 120]	[10, 75]	[50, 110]	[0, 0]
Transitions in *OTP*	T0, T6, T8, T9, T10, T11	T1, T4	T2, T5	T3

The results of these experiments are summarised in Table 2. As it can be observed, the time ranges of attack activities from participants can be used to configure these transitions' *EC* to demonstrate our method in this paper. In future work, we will execute this experiment in different groups of people, such as terms of ethic hacker experts, and collect more data to simulate real attack processes to match the real world.

Moreover, the "NULL" attack activities, like "T3" in the *OTP* attack model in Figure 1, are some attack steps which do not includes any solid attack actions, and are used to represent branching multiple attack paths. Hence, its time range is [0, 0], without any time consuming for attackers.

We therefore obtain the results for *MCAS* depicted in Figure 4: the horizontal axis represents the rounds of *SAPS*; and the vertical axis is the Probability of Successful Attack (*PSA*). As it can be observed, we can find out that by increasing the rounds of *SAPS*, the probability of successful attack becomes stable and is within the interval (2.05%, 2.22%). If we simulate the impact of different protection methods with corresponding different *ECs* as discussed in Subsection III.B, we will obtain different results for *PS* comparison and protection assessment as described in the next subsections.

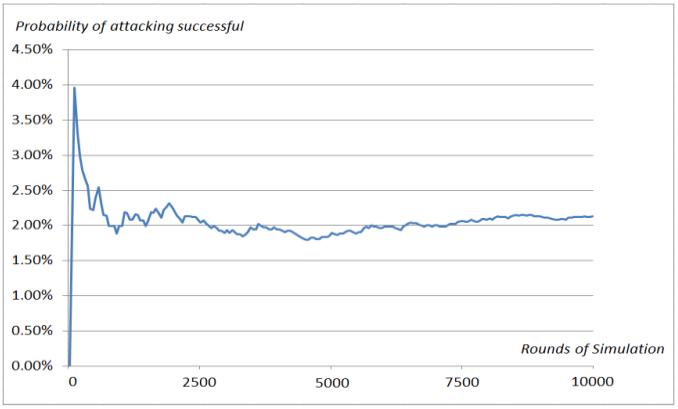

Figure 4. Probabilities of Successful Attack by *MCAS*

B. Implementation on ASCPM

In this subsection, we discuss a prototype implementation of *ASCPM* based on Subsection V.A to demonstrate *PS* comparison.

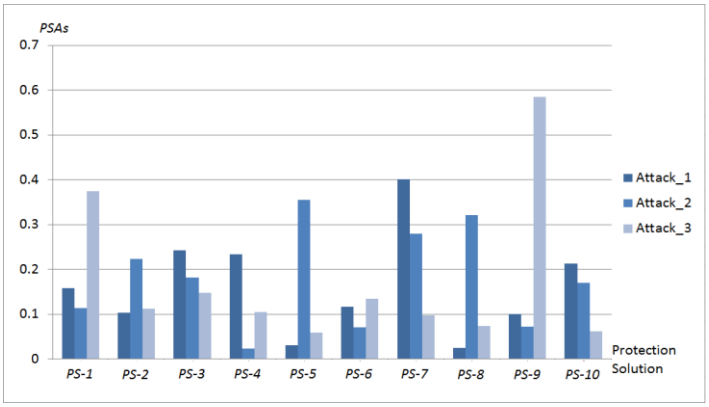

Figure 5. *PSA*s based on different attacks and *PS*s

Currently, our "Attack Model Base" includes three *PN* based attack models (one of them is the *OTP* attack introduced before, another two are attacks on White Box Cryptography and SoftVM [17]), and "*PS* Knowledge Base" currently includes ten *PS*s for protection assessment and software development. Specially, these *PS*s are randomly generated based on some existing protections now [18] and will be improved by real usable *PS*s. Hence, for all these attacks and *PS*s, we can execute *MCAS* repeatedly and generate the Probabilities of Successful Attack (*PSA*s) depicted in Figure 5.

In Figure 5, all *PSA*s are listed based on different attacks and *PS*s. It can be observed that, there are Attack_1 (the *OTP* attack), Attack_2 and Attack_3, and *PS*s from *PS-1* to *PS-10*. For each *PS*, there are corresponding *EC*s for each transition in *PN* based attack models, as discussed in Subsection III.B.

Table 3. Ordered *PS*s List for Comparison under Each Attack

Attack	***PS* lists ordered increasingly by *PSA*s**
Attack_1	*PS-8, PS-5, PS-9, PS-2, PS-6, PS-1, PS-10, PS-4, PS-3, PS-7*
Attack_2	*PS-4, PS-6, PS-9, PS-1, PS-10, PS-3, PS-2, PS-7, PS-8, PS-5*
Attack_3	*PS-5, PS-10, PS-8, PS-7, PS-4, PS-2, PS-6, PS-3, PS-1, PS-9*

Based on the data in Figure 5, we can operate *ASPCM* with confidences. In this part, we will discuss these confidences in different attacks. We can list all *PS*s under each attack increasingly by *PSA*s as Table 3 to compare. For Attack_1, we will compare adjacent *PS*s pair by pair: *PS-8* and *PS-5*, *PS-5* and *PS-9*, *PS-9* and *PS-2*, *PS-2* and *PS-6*, *PS-6* and *PS-1*, *PS-1* and *PS-10*, *PS-10* and *PS-4*, *PS-4* and *PS-3*, *PS-3* and *PS-7*.

Figure 6 shows the comparisons when they are operated under Attack_1. The vertical coordinate is the value of confidences in [0, 1], and the horizontal coordinate is the *PS* list according to Table 3 Row 1. There are two lines represented *CC* and *NeuC* between all *PS*s in *ASPCM*. For instance, for *PS-8* and *PS-5*, the assertion "*PS-8* is better than *PS-5"*, its *CC* is very low and *NeuC* is quite high. In other words, for the assertion that *PS-8* is better than *PS-5*, it is not a "positive" assertion. On the other hand, for *PS-3* and *PS-7*, its *CC* may be high "adequately" to support the assertion: *PS-3* is better than *PS-7*, to be "positive". These "positive" and "adequately" are decided by specific rules in "Rules Set", and will be implemented in the next subsection.

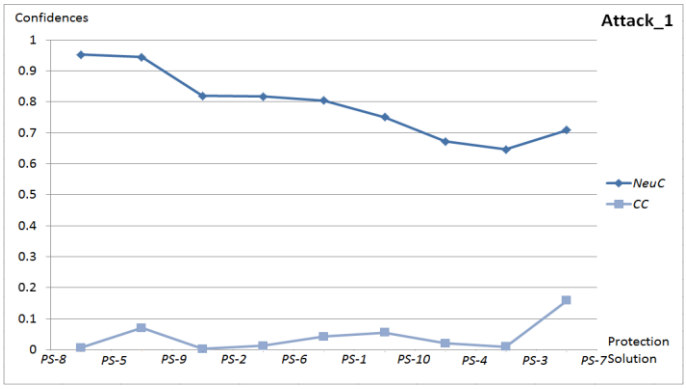

Figure 6. Confidences under Attack_1

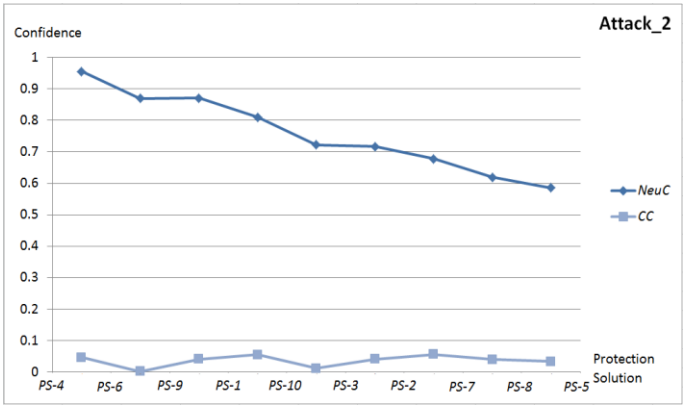

Figure 7. Confidences under Attack_2

Similarly, Figure 7 and Figure 8 are confidences under Attack_2 and Attack_3.

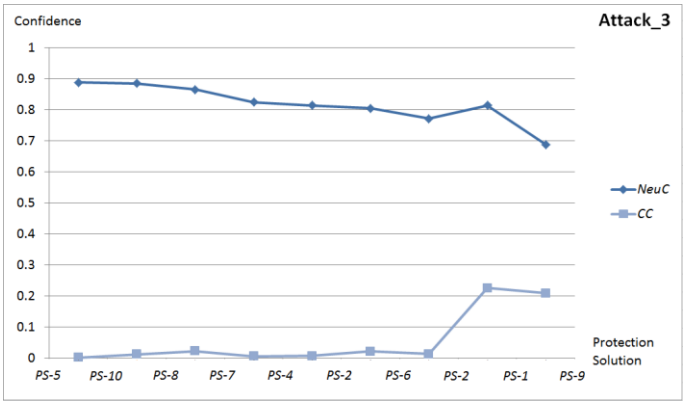

Figure 8. Confidences under Attack_3

In brief, we introduced the implementation of *ASPCM* for *PS* comparison in this subsection, based on *MCAS*.

C. *Implementation on ASSPAM*

In this subsection, we discuss *ASSPAM's* implementation, especially the components of "Analysis and Assessment" and "Rules Set" in Figure 3, based on previous subsections.

The implementation of *ASSPAM* needs to consider the multiple attack threats in real software developing and protecting processes. Specifically, all attacks need to be evaluated together by specific weights. In this regard, one real software protection situation includes that the weight of Attack_1 is 1.0 (this attack is the main concern), the weight of Attack_2 is 0.0 (Attack_2 will not be considered), and the weight of Attack_3 is 0.3 (Attack_3 will be considered, but not as important as Attack_1). Besides, the single attack threat can be viewed as a special case: only one attack's weight is 1.0, and other ones are 0.0.

Table 4. Ordered *PS*s List for Comparison under Situations

Situations	***PS* lists ordered increasingly by weighted sums of *PSA*s**
Attack_1(1.0) + Attack_2(0.0) + Attack_3(0.3)	*PS-8, PS-5, PS-2, PS-6, PS-10, PS-4, PS-1, PS-9, PS-3, PS-7*
……	……

Hence, in this specific situation, we can obtain an ordered *PS* list, increasingly ordered by the weighted sum of *PSA*s of each *PS* under different attacks with these weights as Table 4. The obtained confidences are depicted in Figure 9.

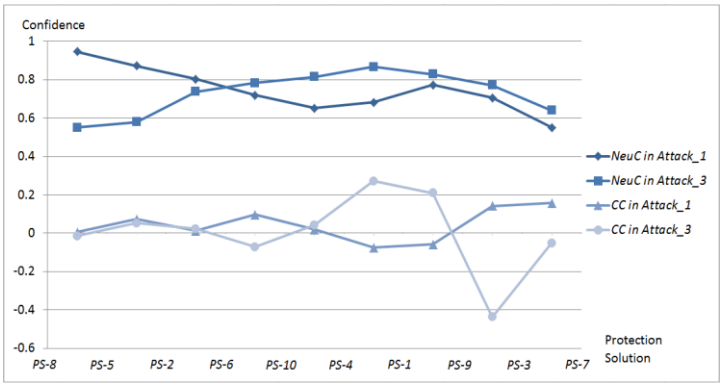

Figure 9. Comparisons in the specific situation

In Figure 9, the vertical coordinate is the value of confidences in [0, 1], and the horizontal coordinate is the *PS* list in Table 4 Row 1. There are four lines represented *CC* and *NeuC* between all *PS*s under Attack_1 and Attack_3 (which have non-zero weights). This figure illustrates an intuitive and detailed picture about all *PS*s' assessment in this specific situation. For instance, the assertion that *PS-8* is better than *PS-5*, may be not very "positive". And for *PS-2* and *PS-1*, its *CC* may be "adequate" to support the assertion: *PS-2* is better than *PS-1*, to be "positive".

In this regard, different developers and software companies have their own unique knowledge about these "positive" and "adequate", which are the specific "Rules Sets" for their own. For example, Rule 1 is "If *NeuC* is less than 0.85, the two *PS*s are different in the view of protection assessment", which means "positive". A different one: Rule 2 is "If |*CC*| is smaller than 0.01, and *NeuC* is more than 0.7, the two *PS*s are the same", which means "not positive". Based on these rules, we can obtain assessment results as Table 5.

In Table 5, under Rule 1, *PS-8*, *PS-5* and *PS-2* are the three best *PS*s as the assessment results. But under Rule 2, *PS-8* and *PS-5* are the two best *PS*s as the assessment results; *PS-6* and *PS-10* are the same in the list; the same to *PS-1* and *PS-9*. No rule means that only one *PS*: *PS-8* will be selected as the assessment result. Therefore, customer-defined rules can provide flexible *PS*s as assessment results, compared to Table 5 Row 1. This flexibility is also valuable in our *ASPIRE* project too. Hence, this flexibility on assessment results can provide alternatives for protection assessment in real software protection situations.

Table 5. Assessment Results depended on Rules

Rules	Assessment Results
No Rule	*PS-8 > PS-5 > PS-2 > PS-6 > PS-10 > PS-4 > PS-1 > PS-9 > PS-3 > PS-7*
Rule 1	*PS-8 = PS-5 = PS-2 > PS-6 > PS-10 > PS-4 > PS-1 > PS-9 > PS-3 > PS-7*
Rule 2	*PS-8 = PS-5 > PS-2 > PS-6 = PS-10 > PS-4 > PS-1 = PS-9 > PS-3 > PS-7*
……	……

So far, in the specific software protection situation, our *ASSPAM* provides Figure 9 and Table 5 as the final protection assessment results for developers and software companies: Table 5 outlines flexible premier *PS*s as assessment results; and Figure 9 shows the details about these *PS*s, like confidences of *PS*s' comparisons.

In summary, for real uncertain software attack processes, our Attack Simulation based Software Protection Assessment method (*ASSPAM*) with Monte Carlo based Attack Simulation (*MCAS*) and Attack Simulation based Protection comparison Model (*ASPCM*) can assess complicated Protection Solutions (*PS*s) effectively.

VI. CONCLUSIONS AND FUTURE WORK

Software protection is a critical aspect in software security. In this regard, to assess complicated Protection Solutions (*PS*s) on uncertain attack processes, we presented a novel attack simulation based protection assessment method called *ASSPAM*. In this method, Monte Carlo based Attack Simulation (*MCAS*) used *PN* based attack models to simulate attacking processes with different *PS*s. Based on this attack simulation, a novel Attack Simulation based Protection Comparison Model (*ASPCM*) was presented to generate comparisons among potential *PS*s. Finally, *ASSPAM* was presented to assess software protections via the *PS* comparing results of *ASPCM* and *MCAS*. We implemented *ASSPAM* by means of a software protection assessment process to demonstrate that our method could provide suitable assessments for software developers.

For future work, we plan to extend our approach by using software metrics to improve the assessment methodology and to search for the optimal protection solution in other case studies, like digital rights management.

ACKNOWLEDGEMENT

This research is supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 609734, project ASPIRE [1]. The work of Christophe Tartary is supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Project PROMISE (No. 16KIS0362K). Part of Christophe Tartary's research was done while he was still affiliated with the University of East London where is research was supported by a Mid-Career Researchers award from the university.

REFERENCES

[1] ASPIRE Project (Advanced Software Protection: Integration, Research and Exploitation). On-line at https://aspire-fp7.eu/, accessed on 21/03/2016.

[2] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based Software Engineering Measurement", IEEE Transactions on Software Engineering, vol. 22, issue 1, pp. 68-86, January 1996.

[3] C. K. Cheng, "Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review", IEEE Transactions on Software Engineering, vol.38, issue 2, pp. 398-424, March-April 2012.

[4] P. Tonella, M. Ceccato, B. De Sutter, and B. Coppens, " A Measurable Framework to Quantify Software Protections", in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1505-1507, November 3-7, 2014, Scottsdale, Arizona, USA.

[5] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, "Optimal Security Hardening using Multi-Objective Optimization on Attack Tree Models of Networks", in Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 204-213, October 29-November 02, 2007, Alexandria, Virginia, USA.

[6] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graphs", in Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 273-284, May 12-15, 2002, Oakland, California, USA.

[7] X. Ou, W. F. Boyer, and M. A. McQueen, "A Scalable Approach to Attack Graph Generation", in Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 336-345, October 30-November 03, 2006, Alexandria, Virginia, USA.

[8] T. Murata, "Petri Nets: Properties, Analysis and Applications", in Proceedings of the IEEE, vol.77, No.4, pp. 541-580, April, 1989.

[9] T. M. Chen, J. C. Sanchez-Aarnoutse, and J. Buford, "Petri Net Modeling of Cyber-Physical Attacks on Smart Grid", IEEE Transactions on Smart Grid, vol. 2, issue 4, pp. 741-749, December 2011.

[10] H. Wang, D. Fang, H. Dong, Y. Lei, X. Gong, and Y. Gu, "Software Attack Modelling and Its Application", in Proceeding of the 2014 IEEE International Conference on High Performance Computing and Communication, pp. 1152-1158, November 13-15, 2013, Zhangjiajie, China.

[11] S. Raychaudhuri, "Introduction to Monte Carlo Simulation", in Proceedings of 2008 Winter Simulation Conference, pp. 91-100, December 7-10, 2008, Miami, USA.

[12] M. Dell'Amico and M. Filippone, "Monte Carlo Strength Evaluation: Fast and Reliable Password Checking", in Proceedings of the 22nd ACM Conference on Computer and Communications Security, pp. 158-169, October 12-16, 2015, Denver, Colorado, USA.

[13] P. Falcarin, C. Collberg, M. Atallah, and M. Jakubowski, "Guest Editors' Introduction: Software Protection", IEEE Software, vol.28, no. 2, pp. 24-27, March/April 2011.

[14] C. Basile and M. Ceccato, "Towards a Unified Software Attack Model to Assess Software Protections", in Proceeding of 2013 IEEE 21st International Conference on Program Comprehension, pp. 219-222, May 20-21, 2013, San Francisco, USA.

[15] M. Ceccato, M. DiPenta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "A Family of Experiments to Assess the Effectiveness and Efficiency of

Source Code Obfuscation Techniques", Empirical Software Engineering, vol. 19, issue 4, pp. 1040-1074, August 2014.

[16] P. Falcarin, "Preliminary ASPIRE Security Model", https://aspire-fp7.eu/sites/default/files/D4.01-Preliminary-ASPIRE-Security-Model.pdf, accessed on 18/02/2016.

[17] B. D. Sutter, "Preliminary Complexity Metrics", https://aspire-fp7.eu/sites/default/files/D4.02-ASPIRE-Preliminary-Complexity-Metrics.pdf, accessed on 21/03/2016.

[18] M. Ceccato, "Early White-Box Cryptography and Data Obfuscation Report", https://aspire-fp7.eu/sites/default/files/D2.01-ASPIRE-Early-White-Box-Cryptography-and-Data-Obfuscation-Report-v1.01.pdf, accessed on 21/03/2016.

[18] M. Ceccato, A. Capiluppi, P. Falcarin, and C. Boldyreff, "A Large Study on the Effect of Code Obfuscation on the Quality of Java Code", Empirical Software Engineering, vol. 20, issue 6, pp. 1486-1524, December 2015.

[19] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, "Towards Experimental Evaluation of Code Obfuscation Techniques", in Proceedings of the 4th ACM Workshop on Quality of Protection, pp. 39-46, October 27-32, 2008, Alexandria, Virginia, USA.