Intl. Journal on Cyber Situational Awareness, Vol. 1, No. 1, 2016

Attack Simulation based Software Protection Assessment Method with Petri Net

Gaofeng Zhang, Paolo Falcarin, Elena Gómez-Martínez, Shareeful Islam
University of East London, London, UK

Christophe Tartary
Saarland University, Saarbrücken, Germany

Bjorn De Sutter
Ghent University, Ghent, Belgium

Jérôme d'Annoville
Gemalto, Meudon, France

ABSTRACT

Software protection is an essential aspect of information security to withstand malicious activities on software, and preserving valuable software assets. However, software developers still lack an effective methodology for the assessment of deployed protections, especially in the area of mobile applications. To solve these issues, we present a novel attack simulation based software protection assessment method to evaluate and compare different protection solutions. Our solution relies on Petri Nets to specify and visualize attack models of mobile applications. We developed a Monte Carlo based approach to simulate attacking processes and to deal with the uncertainty. Then, based on this simulation, a novel protection comparison model is proposed to compare different protection solutions. Finally, our attack simulation based software protection assessment method is presented.

We illustrate our method by means of a case study process to demonstrate that our approach can provide a suitable software protection assessment for developers and software companies.

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

1. 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, like the anti-piracy, binary analysis, and so on. To ensure security against malicious software attacks, many tools have been developed, such as data obfuscation, tamper-proofing, code splitting, software watermarking, among others (Falcarin et al., 2011).

In this regard, assessing the effectiveness of these protections is crucial before embedding them into real commercial software 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. For example, in the area of Android APP development, it is important to consider different attack risks on one single APP: some attackers try to steal the encryption key in the APP execution, others may focus on the illegal tampering to obtain improper utilisations, etc. 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 (Ceccato et al., 2014) focuses on the evaluation of individual protection methods and does not consider *PS*s with multiple protection methods. Another kind of software protection assessment (Basile et al., 2013) 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 complex *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 involved, such as the computing resources for attacking, the decisions or selections made by particular 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 (Murata et al., 1989; Wang et al., 2013), 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 scenario is a huge task, considering the myriad of possible combinations of various *PS*s (multiple protection methods with their parameters) and various software protection scenarios (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*. Besides, various protection scenarios will be discussed to implement *ASSPAM* in the real world.

This research has been carried out within the European *FP7* project *ASPIRE*, Advanced Software Protection: Integration, Research, and Exploitation (ASPIRE, 2016). 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* can store security experts' knowledge and experiences in the Knowledge Base, and generate *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 (Murata et al., 1989), our *PN* based attack models focus on attack steps (transitions) with related simulation information for assessment. 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 2 describes related work. Section 3 discusses preliminary concepts and background. In Section 4, our new method is proposed. In Section 5, we use a protection assessment instance to demonstrate that our proposed method can provide suitable protection assessments for mobile software protection. Section 6 concludes this paper and points out future work.

2. 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.

2.1 Software Analysis and Measurement

Software analysis and measurement are important areas in software engineering (Briand et al., 1996). For example, in the software process improvement area, measurement has been emphasised as a central function and activity (Cheng, 2012). In the field of object-oriented design, software metrics provide the suitable approach to measure the software development processes (Chidamber et al., 1994). From a software security viewpoint, Tonella et al. (2014) 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.

2.2 Attack Modelling

Currently, attack modelling is an important area of information security (Sgandurra et al., 2016). Attack Tree models and Attack Graphs are widely used for representing network attacks, virus attacks, and so on (Dewri et al., 2007; Sheyner et al., 2002). For example, the scalable modelling process is studied for attack graph generation with logic formalism in (Ou et al., 2006). 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 (Murata et al., 1989). These nets can model specific cyber-physical attacks on smart grid (Chen et al., 2011). In the formal way, security policies can be verified by Coloured Petri Nets (Huang et al., 2010). Taking software protection as objective, Wang et al.

(2013) focused on coloured *PN* based attack modelling. As discussed in Section 1, *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.

2.3 Monte Carlo Simulation

The Monte Carlo simulation (method) is a powerful tool for dealing with uncertainty and probability (Raychaudhuri et al., 2008). It is very useful for analysing and simulating complex systems and problems, due to its flexibility and error-quantifiable features (Dell'Amico et al., 2015; Zeng et al., 2009).

Hence, the Monte Carlo method (Dell'Amico et al., 2015) is a suitable technique to simulate complex systems in terms of multiple random variables. As discussed in Section 1, 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.

2.4 Software Protection Assessment

As previous discussed, software protection assessment is an essential part of software protection (Falcarin et al., 2011). Basile et al. (2013) 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 (Ceccato et al., 2008; Ceccato et al., 2014; Ceccato et al., 2015b).

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.

3. 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*.

3.1 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 (Murata et al., 1989; Wang et al., 2013), 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 our protection assessment.

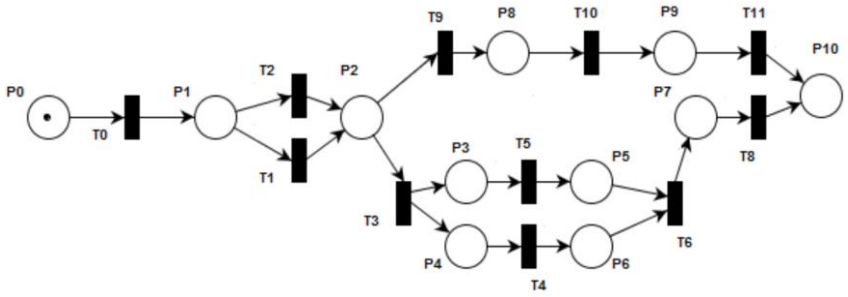

Figure 1. PN based attack model on a one-time password generator

As an example of a relevant use case, Figure 1 and Table 1 present relevant attack paths on a One-Time Password (*OTP*) generator (Falcarin et al., 2015) by means of *PSAM*. It is an important mobile software asset needed to be protected: 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.

3.2 Effort Consumption and Attacker Effort

In this subsection, we detail the Effort Consumption (*EC*) and the Attacker Effort (*AE*) as the important 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 1, 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*.

4. 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 3; 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.

4.1 Monte Carlo based Attack Simulation

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

4.1.1 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*.

4.1.2 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 3.3. The result of each *SAPS* is of type Boolean. Then, the Monte Carlo method executes the *SAPS* several 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 3.3, 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*.

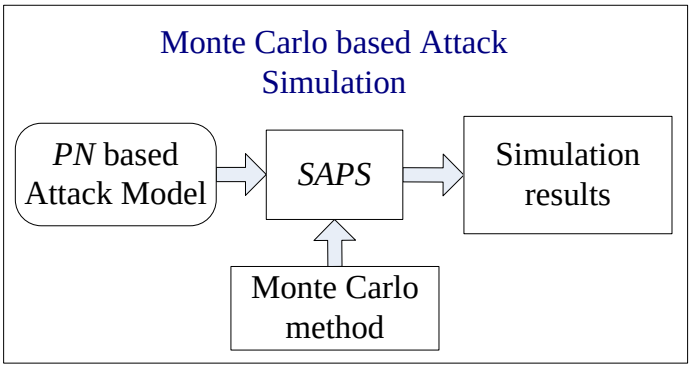

Figure 2. Monte Carlo based Attack Simulation (MCAS)

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

4.2 Attack Simulation based Protection Comparison Model

We now present our *ASPCM*. As indicated in Section 1, 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 4.1, 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 scenario 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*):

$$CC = p_2 - p_1 \tag{5}$$

Neutral Confidence (*NeuC*):

$$NeuC = 1 + 2 \times p_1 \times p_2 - p_1 - p_2 \tag{6}$$

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.

4.3 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*.

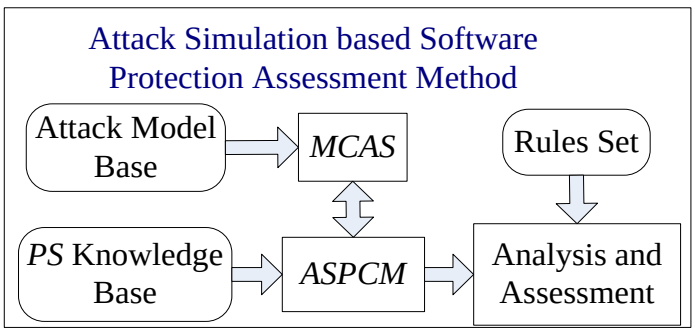

Figure 3. Attack Simulation based Software Protection Assessment Method (ASSPAM)

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 (ASPIRE, 2016), 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 scenarios, and we will deliver some examples in Subsection 5.3. 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*, firstly, users set the software protection scenarios (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. And 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.

Besides, to implement our protection assessment method in the real world, we need to consider the various scenarios in mobile software protection. For example, in one specific assessment scenario (*AS*), developers could foresee that one attack step (transition) requires a peculiar code analysis tool which is not available to the majority of potential software attackers. Hence, due to this scarcity and the low probability of this risk, developers could omit this attack step from the assessment consideration. In other words, it is a part of a whole *PN* based attack model to be considered. In some other *AS*s, different attackers could focus on one special piece of software to jeopardise software security, which means that multiple *PN* based attack models have to be considered together in the protection assessment. Hence, our assessment method has to support these different protection *AS*s.

Based on the previous discussions, we define the *AS* as equation (7):

$$AS = \{PSAM_i, ST_i, w_i\}, i \in [1, S] \tag{7}$$

In this *AS*, *PN* based attack models (*PSAMs*) have been considered and they are ordered by index i from 1 to S. ST_i is the Selected Transitions list in this scenario, which includes all transitions selected in this $PSAM_i$. It means that a part of this *PN* model will be included in this *AS*, as discussed before. w_i is the weight of this *PSAM* in the scenario, compared to other *PSAM*s.

Based on the discussions, we will implement our method on different *AS*s to demonstrate the effective and flexible mobile software protection assessment, in the next section.

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) and *AS*s.

5. 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*.

5.1 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).

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

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 3.3, 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.

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 include 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 3.3, we will obtain different results for *PS* comparison and protection assessment as described in the next subsections.

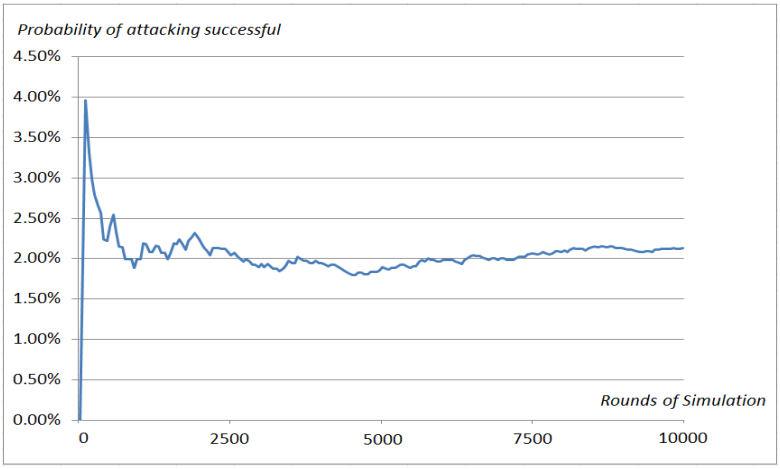

Figure 4. Probabilities of Successful Attack by MCAS

5.2 Implementation on *ASCPM*

In this subsection, we discuss a prototype implementation of *ASCPM* based on Subsection 5.1 to demonstrate *PS* comparison.

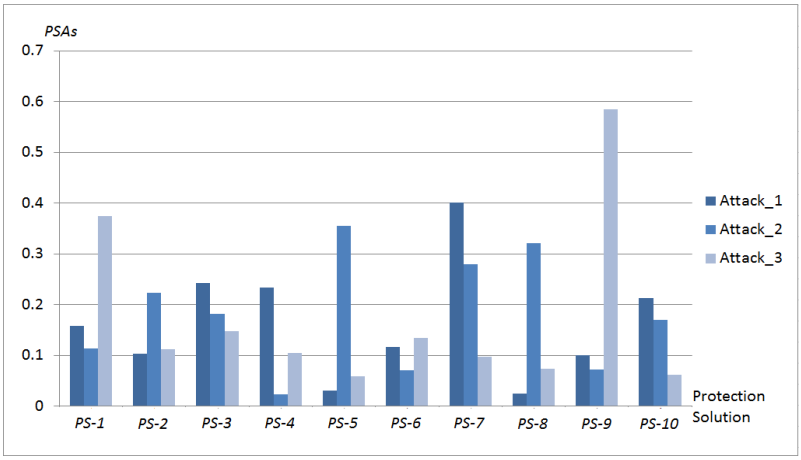

Figure 5. PSAs based on different attacks and PSs

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 (Sutter et al., 2015)), 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 (Ceccato et al. 2015a) 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 3.3.

Table 3. Ordered PSs 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 these 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.

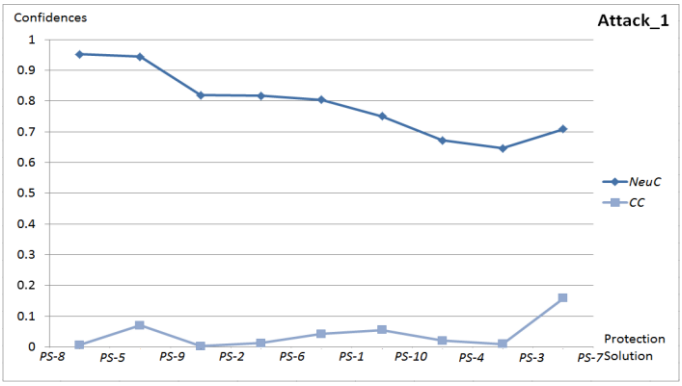

Figure 6. Confidences under Attack_1

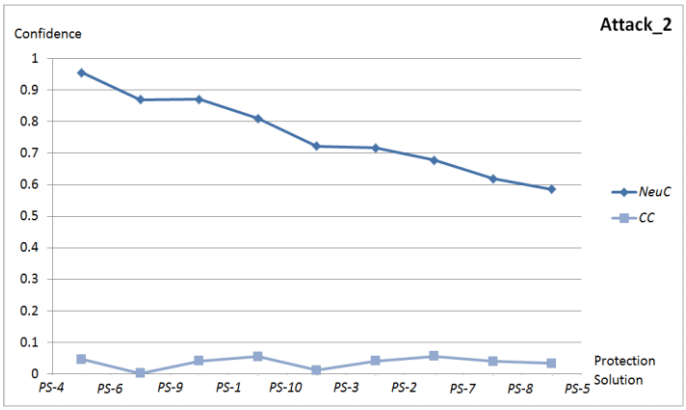

Figure 7. Confidences under Attack_2

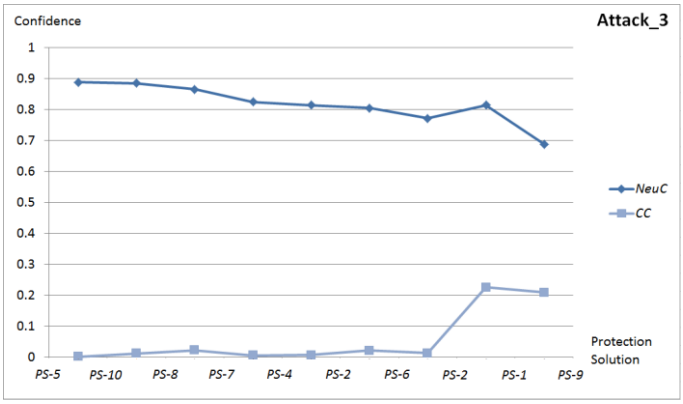

Figure 8. Confidences under Attack_3

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

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

5.3 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.

As introduced before in Section 4.3, supporting various *AS*s is an important aspect of the implementation of *ASSPAM*. In this section, we analyse three representative ASs: multiple *PSAM*s (*AS-1*), a part of one *PSAM (AS-2)*, and multiple *PSAM*s including a part of one *PSAM (AS-3)*.

5.3.1 *AS-1*: multiple *PSAM*s

As discussed before, in the *AS* (*AS-1*), 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, this *AS* 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.

AS-1 = {Attack_1, FULL transitions, 1.0}, {Attack_2, FULL transitions, 0.0}, {Attack_3, FULL transitions, 0.3}

Table 4. Ordered PSs List for Comparison in AS-1

Scenarios	***PS* lists ordered increasingly by weighted sums of *PSA*s**
AS-1: 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 scenario, 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.

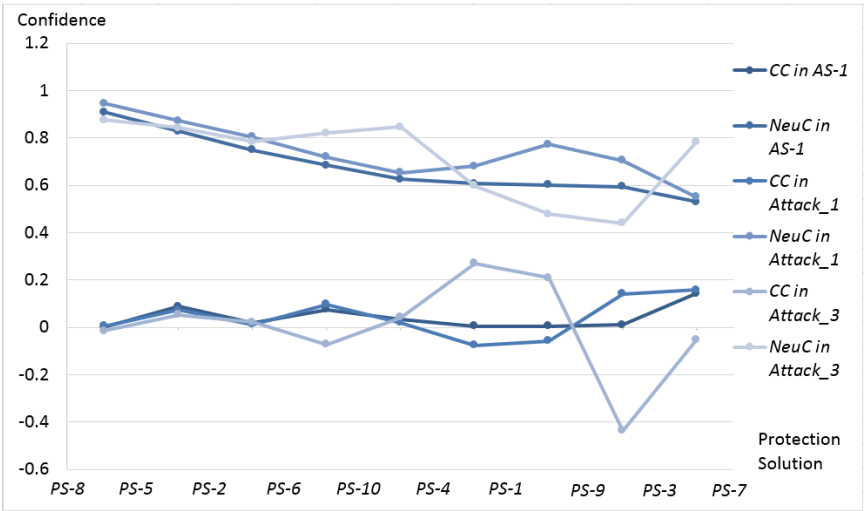

Figure 9. Comparisons in the specific scenario: AS-1

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 six lines represented *CC* and *NeuC* between all *PS*s in *AS-1*, 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 scenario. 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 more than 0.85, the two *PS*s are the same in the view of protection assessment", which means "not positive". And 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". 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 scenarios.

Table 5. Assessment Results depended on Rules in AS-1

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 AS, 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.

5.3.2 *AS-2*: a part of one *PSAM*

As introduced before, due to these changing real risks in mobile software protection, one kind of ASs is to remove some "unsuitable" transitions in the *PSAM* to execute the assessment process, which called a part of the *PSAM* in the assessment process.

In this regard, *AS-2* in this subsection focuses on Attack_1 (*OTP* attack) as introduced in Figure 1 and Table 1. Specially, in this *PSAM*, "T9: Code pruning for XOR localization" is a transition requiring that attackers have to be high-skilled on binary analysis and very familiar with the specific target code piece and AES encryption. Due to the scarcity of these requirements, it is reasonable to assume that this specific transition of the *PSAM* can be removed for specific protection assessments. Hence, we can use the "modified" *PSAM* in Figure 10 in this *AS*.

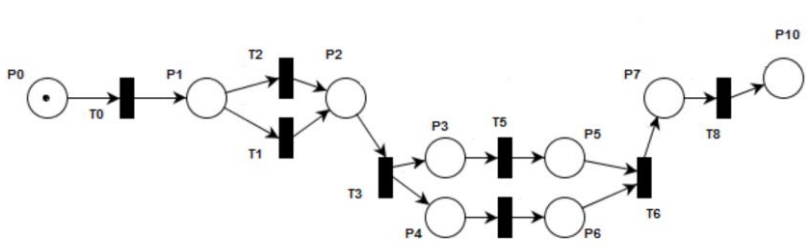

Figure 10. Modified OTP attack model

In Figure 10, it is the modified *OTP* attack. Compared to Figure 1, due to correlations among transitions, transitions T10 and T11 can be removed with T9 together, and the same for the related places: P8 and P9. So, *AS-2* is an *AS* included one *PSAM*: the modified Attack_1.

AS-2 = {Attack_1, (T0, T1, T2, T3, T4, T5, T6, T7, T10), 1.0}

Hence, based on the *MCAS* and *APSCM* functions of our assessment method, we can obtain the *PSA*s on this modified attack model in Figure 10.

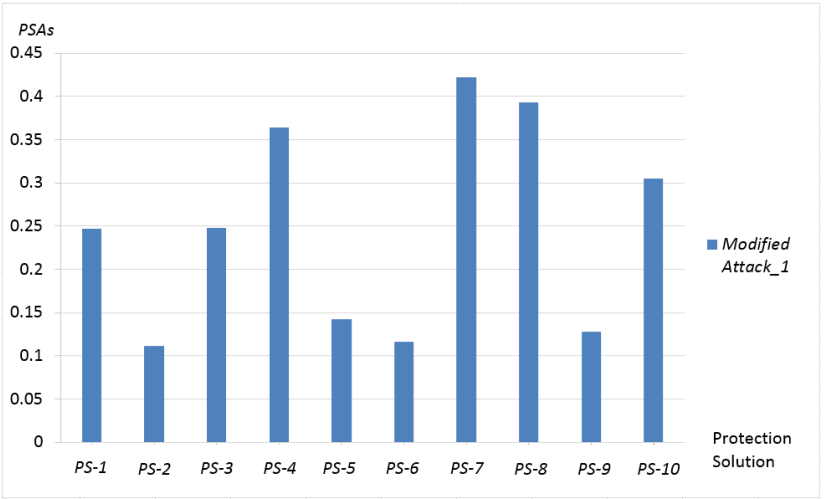

Figure 11. PSAs based on the modified Attack_1 and PSs

In Figure 11, all *PSA*s are listed based on the modified Attack_1 and *PS*s. Compared to the *PSA*s in Figure 5, we can find out that most of *PSA*s have different decrements. Because one attack path has been removed from the attackers' actions, which mean they have a lower degree of freedom to execute a successful attack process. So they have to face lower *PSA*s.

Based on the data in Figure 11, we can operate *ASPCM* with confidences, similar to the previous *AS*. In this part, we will discuss these comparison confidences for the modified Attack_1. Based on the *PS* list in Table 6, we will compare adjacent *PS*s pair by pair: *PS-2* and *PS-6*, *PS-6* and *PS-9*, *PS-9* and *PS-5*, *PS-5* and *PS-1*, *PS-1* and *PS-3*, *PS-3* and *PS-10*, *PS-10* and *PS-4*, *PS-4* and *PS-8*, *PS-8* and *PS-7*.

Table 6. Ordered PSs List for Comparison under the modified Attack_1

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

Hence, Figure 12 shows the comparisons when different *PS*s are operated under the modified Attack_1. The vertical coordinate is the value of confidences in [0, 1], and the horizontal coordinate is the *PS* list according to Table 6.

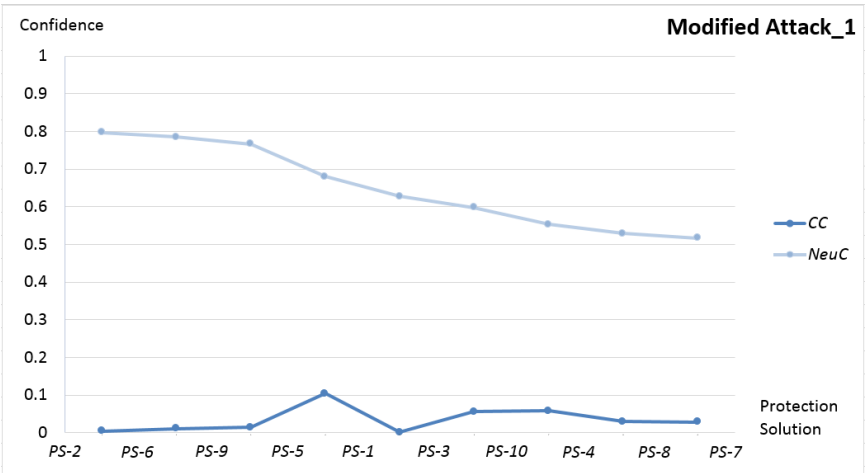

Figure 12. Confidences under modified Attack_1 (Comparison in AS-2)

Similar to Figure 6, Figure 7 and Figure 8, in Figure 12, there are two lines represented *CC* and *NeuC* between all *PS*s under the modified Attack_1. Just similar to the previous discussions, related *PS*s can be analysed in terms of comparisons. For instance, the assertion that *PS-2* is better than *PS-6*, may be not very "positive". And for *PS-5* and *PS-1*, its *CC* may be "adequate" to support the assertion: *PS-5* is better than *PS-1*, to be "positive". Besides, due to that this scenario only includes one attack model (modified Attack_1), this Figure 12 can also viewed as the comparison figure in this scenario (*AS_2*) as Figure 9 in the *AS-1*.

In this regard, we can use the same "Rule 1" and "Rule 2" in Table 5 to understand these "positive" and "adequate". Rule 1 is "If *NeuC* is more than 0.85, the two *PS*s are the same in the view of protection assessment". And Rule 2 is "If |*CC*| is smaller than 0.01, and *NeuC* is more than 0.7, the two *PS*s are the same". Based on these rules, we can obtain assessment results as Table 7.

In Table 7, under Rule 1, *PS-2* is the best *PS* as the assessment results, which is the same to the "No Rule". And under Rule 2, *PS-2* and *PS-6* are the two best *PS*s as the assessment results. Therefore, as discussed in the previous *AS*, customer-defined rules can provide flexible *PS*s as assessment results, compared to Table 7 Row 1.

Table 7. Assessment Results depended on Rules in AS-2

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

Briefly, in the specific software protection scenario: *AS-2*, similar to *AS-1*, our *ASSPAM* provides Figure 12 and Table 7 as the final protection assessment results for developers and software companies: Table 7 outlines flexible premier *PS*s as assessment results; and Figure 12 shows the details about these *PS*s, like confidences of *PS*s' comparisons.

5.3.3 *AS-3*: multiple *PSAM*s including a part of one *PSAM*

In this subsection, we will discuss the last scenario *AS-3*: multiple *PSAM*s including a part of one *PSAM*, for the implementation of our *ASSPAM*. Specifically, the specific scenario includes three *PSAM*s just like *AS-1*. And the only difference is that the Attack_1 is replaced by the modified Attack_1 introduced in Figure 10. Same to *AS-1*, the weight of modified Attack_1 is 1.0, the weight of Attack_2 is 0.0, and the weight of Attack_3 is 0.3.

AS-3 = {Attack_1, (T0, T1, T2, T3, T4, T5, T6, T7, T10), 1.0}, {Attack_2, FULL transitions, 0.0}, {Attack_3, FULL transitions, 0.3}

Table 8. Ordered PSs List for Comparison in AS-3

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

Hence, in this specific scenario, 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 8. Based on this, we can obtain comparison confidences in Figure 13.

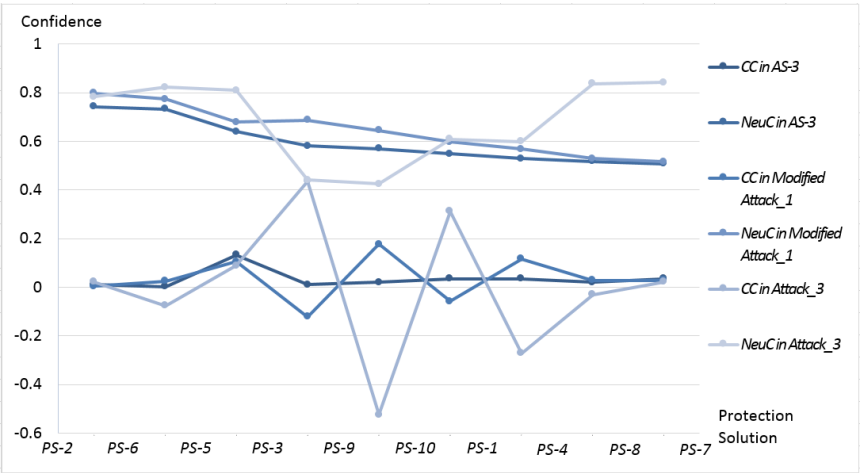

Figure 13. Comparisons in the specific scenario: AS-3

Similar to previous Figure 9, in Figure 13, the vertical coordinate is the value of confidences in [0, 1], and the horizontal coordinate is the *PS* list in Table 8. There are six lines represented *CC* and *NeuC* between all *PS*s in *AS-3*, modified Attack_1 and Attack_3. This figure illustrates an intuitive and detailed picture about all *PS*s' assessment in this specific scenario. For example, the assertion that *PS-2* is better than *PS-6*, may be not very "positive". And for *PS-5* and *PS-3*, its *CC* may be "adequate" to support the assertion: *PS-5* is better than *PS-3*, to be "positive".

Table 9. Assessment Results depended on Rules in AS-3

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

Hence, similar to the previous *AS*s, we can use the same "Rule 1" and "Rule 2" in Table 5 to understand these "positive" and "adequate". Rule 1 is "If *NeuC* is more than 0.85, the two *PS*s are the same in the view of protection assessment". And Rule 2 is "If |*CC*| is smaller than 0.01, and *NeuC* is more than 0.7, the two *PS*s are the same". Based on these rules, we can obtain assessment results as Table 9.

In Table 9, under Rule 1, *PS-2* is the best *PS* as the assessment results, which is the same to the "No Rule". And under Rule 2, *PS-2*, *PS-6* and *PS-5* are the three best *PS*s as the assessment results. Therefore, as discussed in the previous *AS*s, customer-defined rules can provide flexible *PS*s as assessment results, compared to Table 9 Row 1.

So far, similar to the previous *AS-1* and *AS-2*, in the specific software protection scenario: *AS-3*, our *ASSPAM* provides Figure 13 and Table 9 as the final protection assessment results for developers and software companies: Table 9 outlines flexible premier *PS*s as assessment results; and Figure 13 shows the details about these *PS*s, like confidences of *PS*s' comparisons.

In summary, for real mobile 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.

6. 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 mobile 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 described to assess mobile software protections via the *PS* comparing results of *ASPCM* and *MCAS*. We implemented *ASSPAM* by means of software protection assessment processes with various ASs to demonstrate that our method could provide suitable assessments for mobile 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, such as digital rights management.

7. ACKNOWLEDGEMENT

This research is supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 609734, project ASPIRE. 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 his research was supported by a Mid-Career Researchers award from the university.

8. REFERENCES

ASPIRE Project (Advanced Software Protection: Integration, Research and Exploitation). On-line at https://aspire-fp7.eu/, accessed on 21/09/2016.

Basile, C., & Ceccato, M. (2013, May). Towards a Unified Software Attack Model to Assess Software Protections. Paper presented at the 2013 IEEE 21st International Conference on Program Comprehension, San Francisco, USA.

Briand, L. C., Morasca, S., & Basili, V. R. (1996). Property-based Software Engineering Measurement. IEEE Transactions on Software Engineering, 22(1), 68-86.

Ceccato, M., Di Penta, M., Nagra, J., Falcarin, P., Ricca, F., & Torchiano, M. (2008, October). Towards Experimental Evaluation of Code Obfuscation Technique. Paper presented at the 4th ACM Workshop on Quality of Protection, Alexandria, Virginia, USA.

Ceccato, M., Di Penta, M., Falcarin, P., Ricca, F., Torchiano, M., & Tonella, P. (2014). A Family of Experiments to Assess the Effectiveness and Efficiency of Source Code Obfuscation Techniques. Empirical Software Engineering, 19(4), 1040-1074.

Ceccato, M. (2015). 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 21/09/2016.

Ceccato, M., Capiluppi, A., Falcarin, P., & Boldyreff, C. (2015). A Large Study on the Effect of Code Obfuscation on the Quality of Java Code. Empirical Software Engineering, 20(6), 1486-1524.

Chen, T. M., Sanchez-Aarnoutse, J. C., & Buford, J. (2011). Petri Net Modeling of Cyber-Physical Attacks on Smart Grid. IEEE Transactions on Smart Grid, 2(4), 741-749.

Cheng, C. K. (2012). Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review. IEEE Transactions on Software Engineering, 38(2). 398-424.

Chidamber,S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476 – 493.

De Sutter, B. (2015). Preliminary Complexity Metrics. https://aspire-fp7.eu/sites/default/files/D4.02-ASPIRE-Preliminary-Complexity-Metrics.pdf, accessed on 21/09/2016.

Dell'Amico, M., & Filippone, M. (2015, October). Monte Carlo Strength Evaluation: Fast and Reliable Password Checking. Paper presented at the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA.

Dewri, R., Poolsappasit, N., Ray, I., & Whitley, D. (2007, November). Optimal Security Hardening using Multi-Objective Optimization on Attack Tree Models of Networks. Paper presented at the 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA.

Falcarin, P., Collberg, C., Atallah, M., & Jakubowski, M. (2011). Guest Editors' Introduction: Software Protection. IEEE Software, 28(2), 24-27.

Falcarin. P. (2015). Preliminary ASPIRE Security Model. https://aspire-fp7.eu/sites/default/files/D4.01-Preliminary-ASPIRE-Security-Model.pdf, accessed on 18/07/2016.

Huang, H., & Kirchner, H. (2010). Formal Specification and Verification of Modular Security Policy Based on Colored Petri Nets. IEEE Transactions on Dependable and Secure Computing, 8(6), 852 – 865.

Raychaudhuri, S. (2008, December). Introduction to Monte Carlo Simulation. Paper presented at the 2008 Winter Simulation Conference, Miami, USA.

Murata, T. (1989, April). Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 77(4), 541-580.

Ou, X., Boyer, W. F., & McQueen, M. A. (2006, November). A Scalable Approach to Attack Graph Generation. Paper presented at the 13th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA.

Sgandurra, D., & Lupu, E. (2016). Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems. ACM Computer Surveys, 48(3), Article 46.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., & Wing, J. M. (2002, May). Automated Generation and Analysis of Attack Graphs. Paper presented at the 2002 IEEE Symposium on Security and Privacy, Oakland, California, USA.

Tonella, P., Ceccato, M., Sutter, B. D., & Coppens, B. (2014, November). A Measurable Framework to Quantify Software Protections. Paper presented at the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, Arizona, USA.

Wang, H., Fang, D., Dong, H., Lei, Y., Gong, X., & Gu, Y. (2013, November). Software Attack Modelling and Its Application. Paper presented at the 2014 IEEE International Conference on High Performance Computing and Communication, Zhangjiajie, China.

Zeng, Y., Cao, J., Hong, J., Zhang, S., & Xie, L. (2009, October). SecMCL: A Secure Monte Carlo Localization Algorithm for mobile sensor networks Paper presented at the IEEE 6th International Conference on Mobile Adhoc and Sensor Systems, Macau, China.

KEY TERMS

Keywords:

- ***Mobile Software Security*** - The area to take mobile software as the assets and make sure software to be executed in ethnic ways.
- ***Software Protection Assessment*** - The assessment on the level of software protection methods.
- ***Attack Simulation*** - Simulating the processes of attacks.
- ***Monte Carlo Method*** - The method to use random variables and sampling to deal with uncertainly computing tasks.
- ***Petri Net*** - A modelling language to describe and reason complicated processes with solid mathematical basis.

BIOGRAOHICAL NOTES

Gaofeng Zhang obtained his PhD degree in Information and Communication Technologies in 2013, from Swinburne University of Technology, Australia. Before that, he received his BEng and MEng degrees in Computer Science and Technology in 2005 and 2008, from Hefei University of Technology, China. His research interests include mobile

security, cloud security & privacy, security assessment, IoT software and security.

Paolo Falcarin received his Ph.D. in Software Engineering in 2004 and MEng in Computer Engineering in 2000, from Polytechnic of Turin University, Italy. He is currently a Reader in Computer Science, at the School of Architecture Computing and Engineering, University of East London.
His research interests include software protection and security, software and systems engineering.

Elena Gómez-Martínez received her M.Sc. and Ph.D. degrees in Computer Engineering from the University of Zaragoza, Spain, in 1999 and 2014, respectively. Her Ph.D. dissertation was related to the study of software performance assessment at architectural level based on software performance engineering principles with stochastic Petri nets. She worked at the R&D department of Ilunion (former Technosite) as researcher, developer, project coordinator of INREDIS project and Simplext project (was awarded by Vodafone Foundation) and technical coordinator of the ATIS4all European Thematic Network. Later, she worked as researcher at the Babel group and at the Center for Open Middleware, both in the Universidad Politécnica de Madrid, Spain. At the present, she is research assistant in the ASPIRE project at the University of East London, United Kingdom.

Shareeful Islam is currently working at school of ACE, University of East London, UK. He received the PhD from Technische Universität München, Germany. He received M.Sc. in Information Communication System Security from Royal Institute of Technology (KTH), Sweden and M.Sc. in CS and B.Sc. (Hon's) in APE from the University of Dhaka, Bangladesh. He is a Fellow of the British Higher Education Academy (HEA). He has published more than 60 referred papers in high quality journals and international conferences. He participated in EU, industry, KTP projects. His research interest and expertise is risk management, requirements engineering, security, privacy, and cloud computing.

Christophe Tartary obtained his Master in Cryptography, Coding Theory and Calculus from the University of Limoges (France) in 2001 and his PhD in Computer Science from Macquarie University (Sydney, Australia) in 2008. His research interests include cryptography, coding theory, number theory, complexity theory and information security.

Bjorn De Sutter is a professor at Ghent University in the Computer Systems Lab. He obtained his Msc. and Ph.D. degrees in Computer Science from Ghent University's Faculty of Engineering in 1997 and 2002. His research focuses on the use of compiler techniques to aid programmers with non-functional aspects of their software, such as performance, code size, reliability, and software protection.

Jerome d'Annoville received a degree in Computer Science in 1983 from Pierre et Marie Curie University in Paris, France. He is working in the Advanced R&D Department of Gemalto and is head of a team in charge of developing innovative components to be embedded in future products in the field of mobile devices security. He currently works on Trusted Computing and IoT.

Reference to this paper should be made as follows: Zhang, G., Falcarin, P., Gómez-Martínez, E., Islam, S., Tartary, C., De Sutter, B., & d'Annoville, J. (2016). Attack Simulation based Software Protection Assessment Method with Petri Net. *International Journal on Cyber Situational Awareness*, Vol. 1, No. 1, pp152-181