Characterizing Dynamics of Information Leakage in Security-Sensitive Software Process

Yuichiro Kanzaki Hiroshi Igaki Masahide Nakamura Akito Monden
Ken-ichi Matsumoto

Graduate School of Information Science, Nara Institute of Science and Technology,
8916-5, Takayama, Ikoma, Nara, Japan 630-0192
Email: {yuichi-k, hiro-iga, masa-n, akito-m, matumoto}@is.naist.jp

Abstract

Minimizing information leakage is a crucial problem in DRM software development processes, where security information (e.g., device keys and S-BOX of CPRM systems) must be rigorously managed.

This paper presents a method to evaluate the risk of information leakage in a software process for security-sensitive applications. A software process is modeled as a series of sub-processes, each of which produces new work products from input products. Since a process is conducted usually by multiple developers, knowledge of work products is shared among the developers. Through the collaboration, a developer may tell others the knowledge of products that are not related to the process. We capture the transfer of such irrelevant product knowledge as the information leakage in a software process.

In this paper, we first formulate the problem of information leakage by introducing a formal software process model. Then, we propose a method to derive the probability that each developer d knows each work product p at a given process of software development. The probability reflects the possibility that someone leaked the knowledge of p to d, unless it is equal to 1.0. We also conduct a quantitative case study to demonstrate how the information leakage varies depending on the assignment of developers.

1 Introduction

The development of a complex and large-scale software system requires the collaborative effort of many people with an elaborate *software process* (Jacobson, Booch & Rumbaugh 1999). A (whole) software process is composed of partially-ordered *sub-processes* (simply called *processes*) such as design, coding and testing, as well as *work products* (simply *products*) which can be the input or output of each process.

Typically, multiple developers participate in a common process. Given input products, the developers collaborate with each other, and produce output products. Through the collaboration, they usually share their knowledge of the products in order to achieve the process efficiently. Thus, when multiple developers participate in a process, certain *knowledge transfer* may occur in the process. From the viewpoint of the efficiency, the knowledge transfer should be improved, since it helps developers to acquire a similar understanding of the process (Keller, Tabeling, Apfelbacher, Grone, Knopfel, Kugel & Schmidt 2002).

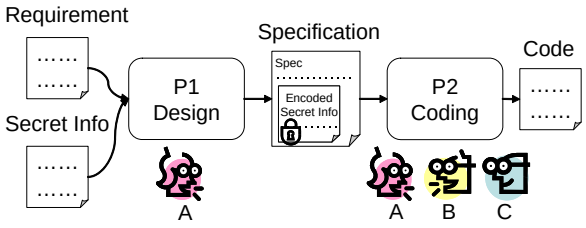


Figure 1: Security-Sensitive Software Process

However, in the development of security-sensitive software such as DRM applications (Chow, Eisen, Johnson & van Oorschot 2002)(Liong & Dixit 2004), the transfer of the product knowledge is not always encouraged. For more comprehension, we introduce a simple example.

Figure 1 shows a software process consisting of two processes P1 and P2. P1 is a design process conducted by developer A, where A produces a product *Specification* from given two products *Requirement* and *SecretInfo*. Here we assume that *SecretInfo* is confidential information (e.g., device keys or S-BOX of CPRM systems (4C-Entity 2001)) only A is authorized to see, and that it must appear in *Specification* in a certain encoded form. P2 is a coding process in which A, B and C participate. The three developers collaborate with each other and produce *Code* from *Specification*. Since A knows *Specification*, A may explain it to B and C in the collaboration. This knowledge transfer is reasonable, since *Specification* is necessary for B and C to perform P2 together. Even if A does not give the explanation, B and C must know *Specification*. On the other hand, both *Requirement* and *SecretInfo* are *irrelevant* to P2, since they are not directly connected to P2. However, what happens if A tells B or C the knowledge about *SecretInfo* in P2? Then, B or C gets to know *SecretInfo*, which ruins the security scheme.

From the above observation, we consider that in a security-sensitive software process, the knowledge transfer with any such irrelevant products should be warned as *information leakage*. Note that the risk of the information leakage varies, depending on the structure of the software process and the assignment of developers to each process. For example, in Figure 1 if A is not assigned to P2, no leakage occurs.

The goal of this paper is to propose a framework to evaluate the risk of the information leakage quantitatively, for a given software process. To achieve this, we first formulate the problem of information leakage by introducing a formal software process model. The model is based on the conventional *process-centered software engineering environment* (Feiler &

Humphrey 1993)(Garg & Jazayeri 1995). Our contribution is to formulate *product knowledge* of each developer on top of the model, focusing on the process structure and the developer assignment.

Next, assuming that the knowledge of the irrelevant products can be transferred (i.e., leaked), we present a method to compute a probability that each developer knows each product. The probability reflects the risk that someone leaked the product to the developer. We derive the probability by deriving a recurrence formula from the given software process model.

Finally, we conduct a case study to demonstrate how the information leakage varies depending on the assignment of developers. The case study shows quantitatively that too much collaboration among developers significantly increases the risk of information leakage.

The rest of this paper is organized as follows: In Section 2, we give definitions of software process model. Section 3 describes the proposed method for characterizing dynamics of information leakage. In Section 4, we conduct a case study. We review the related work in Section 5. Finally, Section 6 concludes the paper and presents directions of future work.

2 Preliminaries

2.1 Software Process Model

We start with a definition of a software process model. The model is based on the conventional *process-centered software engineering environment* (Feiler & Humphrey 1993) (Garg & Jazayeri 1995), where the software development is modeled by partially-ordered activities (*processes*) operating with given or intermediate *working products*. In addition to the conventional model, our model involves the *assignment of developers* to specify explicitly who participates in each process.

Definition 1 (Software Process Model) A software process model is defined by $P = (U, WP, PC, I, O, AS)$, where:

- U is a set of all *developers* participating in the development.
- WP is a set of all *work products*.
- PC is a set of all *processes*.
- I is an *input* function $PC \rightarrow 2^{WP}$ that maps each process $p \in PC$ onto a set $IP(\subseteq WP)$ of *input products* of p.
- O is an *output* function $PC \rightarrow 2^{WP}$ that maps each process $p \in PC$ onto a set $OP(\subseteq WP)$ of *output products* of p.
- AS is an *developer assignment function* $PC \rightarrow 2^{U}$ that maps each process $p \in PC$ onto a set of developers participating in the process p.

Figure 2(a) shows an example of the software process model, which simplifies an implementation stage of a security-sensitive application. The model contains five developers, eight work products, and six processes. The scenario assumed in the model is briefly explained as follows:

Example Scenario: The implementation stage produces an object code from a given design specification. In the stage, the design specification is revised by a review process. Then, by applying a security analysis to the reviewed specification, any security-sensitive information is isolated from the specification. The rest of the specification is called ModuleSpec. From the security information, authorized developers design a specification, called S-ModuleSpec for an independent security module in which the raw security information is encoded. A main module and the security module are coded respectively from ModuleSpec and S-ModuleSpec. Finally, these two modules are integrated as the object code.

In Figure 2(a), let us take the process *Review*. This models the review of the design specification. It takes *DesignSpec* as an input product, and outputs a reviewed specification (*Rev-Spec*). In this example, only developer A is responsible to conduct the process. Next, consider the process *SecAnalysis*. This takes *Rev-Spec* as an input, and outputs *ModuleSpec* and *SecurityInfo*. Two developers A and B participate the process. By the similar discussion, we can see the process model achieves the example scenario.

By definition, each process has a set of input products and a set of output products. This allows us to draw a given process model by a *Petri net* (Marsan, Balbo, Conte, Donatelli & Franceschinis 1995), by associating WP with places, PC with transitions, connecting a place and a transition with an arc according to I and O. Figure 2(b) shows a schematic representation of the example process with Petri net. Also, we associate a set of developers with each corresponding transition based on AS, depicted in the left of the transition. Note that the use of Petri net is just for better comprehension of the overview of the process structure, but is not essential to our methodology.

2.2 Order among Processes

Suppose that $P = (U, WP, PC, I, O, AS)$ is given. For $p \in PC$, $w \in I(p)$ and $w' \in O(p)$, we use a triple (w, p, w') to represent a *product dependency* of process p, where a work product w' is produced by p from w. The product dependencies implicitly specify a *partial order* between processes, since a process needs input products that have been generated by other processes.

Definition 2 (Order of Processes) For processes p and p', we say that p is *executed before* p' (denoted by $p < p'$) iff there exists a sequence (w_0, p, w_1) (w_1, p_1, w_2) ... (w_{n-1}, p', w_n) of product dependencies. For processes q and q', if any $<$ is not defined between q and q', we say that q and q' are *independent*.

Let us consider the previous example. As depicted in Figure 2(b), we can see the order among the six processes, i.e., *Review* $<$ *SecAnalysis* $<$ *Coding1* $<$ *Integrate*, and *Review* $<$ *SecAnalysis* $<$ *S-Design* $<$ *Coding2* $<$ *Integrate*. Note that the order is *partial* at this moment. Indeed, no order between *Coding1* and *S-Design* (or *Coding2*) is defined, thus they are independent. The independent processes can be executed in any order, even concurrently.

2.3 Assumption on Software Process Model

In this paper, we put the following two assumptions for a given process model $P = (U, WP, PC, I, O, AS)$.

Assumption A1: There exists no sequence (w_0, p_0, w_1) (w_1, p_1, w_2) ... (w_{n-1}, p_n, w_n) of product dependencies such that $w_0 = w_n$.

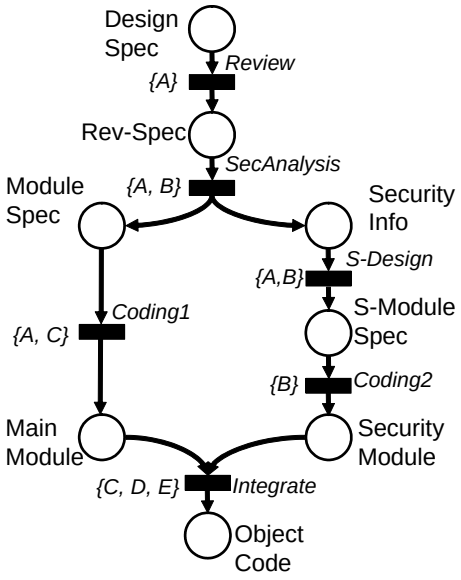
Assumption A2: For any pair of independent processes p and p', if $AS(p) \cap AS(p') \neq \phi$, then an order between p and p' must be given.

U = { A, B, C, D, E}
PD = { DesignSpec, Rev-Spec, SecurityInfo,
ModuleSpec, S-ModuleSpec, MainModule,
SecurityModule, ObjectCode }
PC = {Review, SecAnalysis, S-Design, Coding1,
Coding2, Integrate }
I(Review)={DesignSpec}
I(SecAnalysis)={Rev-Spec}
I(S-Design)={SecurityInfo}
I(Coding1)={ModuleSpec}
I(Coding2)={S-ModuleSpec}
I(Integrate)={MainModule, SecurityModule}

O(Review)={Rev-Spec}
O(SecAnalysis)={ModuleSpec,SecurityInfo}
O(S-Design) = {S-ModuleSpec}
O(Coding1)={MainModule}
O(Coding2)={SecurityModule}
O(Integrate)={ObjectCode}

AS(Review)={A}
AS(SecAnalysis)={A, B}
AS(S-Design)= {A, B}
AS(Coding1)={A, C}
AS(Coding2)={B}
AS(Integrate)={C, D, E}

(a) Software Process Model



(b) Petri-Net Representation

Figure 2: Process Model Example

Assumption A1 states that the product dependencies never form a loop. This is quite reasonable for general software processes. Indeed, it is unrealistic that a work product newly obtained is used as the input of the processes that have been completed previously. By this assumption, we have a consistent partial order among processes for a given sequence of product dependencies.

Assumption A2 says that independent processes p and p' must be ordered in case that the same developer is assigned to both p and p'. This is based on the observation that a developer cannot engage in more than one processes simultaneously. Let us consider the process model in Figure 2. In this example, processes *Coding1* and *S-Design* are independent. However, they cannot be executed simultaneously, since the same developer A is assigned to both processes (i.e., $AS(S-Design) \cap AS(Coding1) = \{A\}$). Hence, we need to give an order between them, for instance, *S-Design* $<$ *Coding1*, so that A conducts *S-Design* first.

By these assumptions, if we fix a developer u, then the processes in which u participates are totally-ordered.

Proposition 1 Let $P = (U, WP, PC, I, O, AS)$ be a given process model with Assumptions A1 and A2. For a developer $u \in U$, let $PC_u = \{p | p \in PC \wedge u \in AS(p)\}$ be a set of all processes to which u is assigned. Then, PC_u is *totally-ordered*.

Consider the process model in Figure 2 with *S-Design* $<$ *Coding1*. Then, the processes to be conducted by each user are ordered as follows:

PC_A : *Review* $<$ *SecAnalysis* $<$ *S-Design* $<$ *Coding1*
PC_B : *SecAnalysis* $<$ *S-Design* $<$ *Coding2*
PC_C : *Coding1* $<$ *Integrate*
PC_D : *Integrate*
PC_E : *Integrate*

Since PC_u are totally-ordered, any process in PC_u has at most one *immediate predecessor*.

Definition 3 (Predecessor of Process) Let $p_{u_1}, p_{u_2}, ..., p_{u_k}$ be all processes in PC_u such that $p_{u_1} < p_{u_2} < ... < p_{u_k}$. For $p_{u_i} \in PC_u$, we call $p_{u_{i-1}}$ *immediate predecessor* of p_{u_i} with respect to u, which is denoted by $pred_u(p_{u_i})$. Also, we define $pred_u(p_{u_1})$ to be ϵ (empty).

In the above example, we have $pred_A(Coding1) =$ *S-design*, which means that A participates in *S-design* immediately before $Coding1$. Also, we have $pred_C(Coding1) = \epsilon$ meaning that $Coding1$ is the first process that C engages in.

3 Characterizing Dynamics of Information Leakage

3.1 Product Knowledge of Developers

To perform a process p, developers engaging in p must know all the input products of p. Based on the input products, they develop the output products. Hence, when finishing p, they should be acquainted with the output products as well. Thus, when a process is performed, the developers acquire knowledge about the related (i.e., input/output) products. For each developer, the knowledge is accumulated in the sequence of completed processes. This dynamics depends on the given process model, specifically, I, O and AS.

For example, consider the example in Figure 2. Developer A participates process *Review*. Hence, when *Review* is finished, A must know products *DesignSpec* and *Rev-Spec*. Similarly, the completion of *SecAnalysis* provides the knowledge of *Rev-Spec*, *ModuleSpec* and *SecurityInfo* for both A and B. Thus, when A completes *SecAnalysis*, A knows four products; *DesignSpec, Rev-Spec, ModuleSpec, SecurityInfo*.

Definition 4 (Product Knowledge) Let $P = (U, WP, PC, I, O, AS)$ be a given software process model. For $u \in U$ and $p \in PC$, we define a set of working products $Know(u, p)$ ($\subseteq WP$) s.t.

$$Know(u, p) = \bigcup_{u \in AS(p') \wedge p' \leq p} (I(p') \cup O(p'))$$

$Know(u, p)$ is called *product knowledge* of developer u at the completion of process p.

We use the term "knowledge" in some abstract sense, which can be refined in terms of, for instance, the

essential idea or mechanism, the product's document itself, or the access method to the product.

Let us compute $Know(B, Coding2)$ with Figure 2. Before *Coding2*, B has participated in *SecAnalysis* and *S-Design*. Hence, accumulating the input/output products of these three processes, we have $Know(B, Coding2) = \{$ *Rev-Spec, SecurityInfo, ModuleSpec, S-ModuleSpec, SecurityModule* $\}$.

For convenience, we represent $Know(u, p)$ with a *binary vector*. Let $w_1, w_2, ..., w_n$ be all work products in WP. Then, we denote $Know(u, p) = [wp_1, wp_2, ..., wp_n]$, where $wp_i = 1$ iff $w_i \in Know(u, p)$, otherwise $wp_i = 0$. Then, the product knowledge of all users at the completion of the last process (i.e., *Integrate*) can be represented in Table 1.

Table 1: $Know(u, Integrate)$ $(u \in \{A, B, C, D, E\})$

u	DSpc	RSpc	SInfo	MSpc	SSpc	MMo	SMo	OCod
A	1	1	1	1	1	1	0	0
B	0	1	1	1	1	0	1	0
C	0	0	0	1	0	1	1	1
D	0	0	0	0	0	1	1	1
E	0	0	0	0	0	1	1	1

3.2 Leakage of Product Knowledge

Now suppose a situation that; a developer may *tell* his/her product knowledge to other developers sharing the same process.

As an example, consider *Coding1* in Figure 2. This process is shared by A and C. Assuming an order *S-Design* $<$ *Coding1*, the product knowledge of A and C at *Coding1* are computed as follows:

		DSpc	RSpc	Sinfo	MSpc	SSpc	MMo	SMo	OCod	
Know(A,Coding1) =	[	1	1	1	1	1	1	0	0	]
Know(C,Coding1) =	[	0	0	0	1	0	1	0	0	]

Coding1 is the first process that C participates in. Hence, at this moment, C is supposed to know only *ModuleSpec* and *MainModule*. C does not need to know all the rest of products. On the other hand, A has more product knowledge than C, because A has previously participated in three other processes.

Assume now that during *Coding1*, A tells C the product knowledge that C does not know, say *SecurityInfo*, with some probability. Then, C becomes to know *SecurityInfo* although C has never directly touched it before. Once C knows *SecurityInfo*, the knowledge would be *propagated* to D and E, since C shares the subsequent process, *Integrate*, with D and E. As a result, the isolation of security information would be in vain.

Thus, when multiple developers work in the same process, the product knowledge can be spread from the developer who knows the product to ones who do not know. We regard this as *information leakage in the software process*, which is specifically defined as follows.

Definition 5 (Leakage) For developers $u, u' \in D$, a work product $w \in WP$ and a process $p \in PC$, we say that u *may leak* w to u' at p iff $\{u, u'\} \subseteq AS(p)$ and $w \in Know(u, p)$ and $w \notin Know(u', p)$.

The above definition of leakage might be *broad* a bit. Indeed, it covers a case that a security product w is known to an unauthorized developer u'. On the other hand, someone may say that it is not leakage if w is not a security-sensitive product, or if u and u' work for the same company. However, for simplicity and generality of the model, we keep this broad definition. More detailed criteria of the leakage should be tuned depending on the target software process.

3.3 Stochastic Product Knowledge

Now, let us take the leakage of product knowledge into account in our model. Specifically, we introduce the following assumption for a given process model $P = (U, WP, PC, I, O, AS)$:

Assumption A3: For $u, u' \in U$ and $w \in WP$, let $leak(u, w, u')$ be a probability that u leaks w to u'. We assume that $leak(u, w, u')$ is given for any u, u' and w.

Then, in a process p, a developer u may happen to know a product w such that $w \notin Know(u, p)$, since someone could leak w to u with a certain probability. This motivates us to deal with the product knowledge in a *stochastic* manner.

Let us consider a probability that a developer u knows a work product w at the completion of process p, which we denote $Pkn(u, p, w)$. When u knows w at the completion of p, two cases can be considered.

Case C1: $w \in Know(u, p)$, or

Case C2: $w \notin Know(u, p)$ and some developers leak (or leaked) w to u.

Case C1 means that w is already count in u's product knowledge. For this case, we have $Pkn(u, p, w) = 1.0$. Case C2 can be further divided into two sub-cases.

Case C2a: u has already known w before p, or

Case C2b: $[u \in AS(p)]$ and [u did not know w before p] and [in p some developers sharing p with u leak w to u].

The probability that Case C2a holds is

$$P(C2a) = Pkn(u, pred_u(p), w)$$

which means that u knew w in the predecessor process. Next, the probability for Case C2b can be formulated by

$$P(C2b) = C(u, p) * (1 - Pkn(u, pred_u(p), w)) * P_{leak}$$

where $C : U \times PC \rightarrow \{0, 1\}$ such that $C(u, p) = 1$ iff $u \in AS(p)$, otherwise $C(u, p) = 0$, and P_{leak} is a probability that some developers sharing p leak w to u.

Next, we formulate P_{leak}. Let $u_1, u_2, ..., u_j$ be developers who share p with u (i.e., $\{u_1, u_2, ..., u_j\} = AS(p) - \{u\}$). In order for u_i to leak w in p, two conditions are required; (1) u_i needs to have known w before p, and (2) u_i leaks w to u. Therefore, the probability that u_i leaks w to u in p is

$$Pkn(u_i, pred_{u_i}(p), w) * leak(u_i, w, u)$$

Now, u knows w iff at least one of $u_1, u_2, ..., u_j$ leaks w to u in p, which is the complement of "none of $u_1, u_2, ..., u_j$ leaks w to u in p". Hence,

$$P_{leak} = 1 - \prod_{u_i \in AS(p) - \{u\}} \{1 - Pkn(u_i, pred_{u_i}(p), w) * leak(u_i, w, u)\}$$

Combining all together, we finally derive $Pkn(u, p, w)$, which is a probability that u knows w at the completion of p, in Figure 3.

Note that $Pkn(u, p, w)$ is specified as a recurrence formula with respect to the process p. According to Assumptions A1 and A2, the set of processes that u participates in is totally ordered. Hence, $pred_u(p)$ is uniquely obtained. Also, by Assumption A3, $leak(u_i, w, u)$ is given. Therefore, the value of $Pkn(u, p, w)$ can be calculated deterministically.

$Pkn(u, p, w)$ is now defined as *stochastic product knowledge*.

$$Pkn(u,p,w) = \begin{cases} 1.0 \quad (\cdots \textbf{if} \quad w \in Know(u,p)) \\ \\ \quad Pkn(u, pred_u(p), w) \\ + \ C(u,p) \\ * \ (1 - Pkn(u, pred_u(p), w)) \\ * \ [1 - \prod_{u_i \in AS(p)-\{u\}} \{1 - Pkn(u_i, pred_{u_i}(p), w) * leak(u_i, w, u)\}] \\ \quad (\cdots \textbf{if} \quad w \notin Know(u,p)) \end{cases}$$

Figure 3: Probability that a developer u knows a work product w at the completion of a process p

Definition 6 (Stochastic Product Knowledge) Let $P = (U, WP, PC, I, O, AS)$ be a given software process model with Assumptions A1, A2 and A3. Let $w_1, w_2, ..., w_n$ be all work products in WP. For $u \in U$, $p \in PC$, we define a vector $PKnow(u,p)$ s.t.

$$PKnow(u,p) = [Pkn(u,p,w_1), Pkn(u,p,w_2), ..., Pkn(u,p,w_n)]$$

$PKnow(u,p)$ is called *stochastic product knowledge* of u at the completion of p.

Consider the example in Figure 2 with *S-Design* < *Coding1*. For just simplicity, let us assume a fixed probability $leak(u,w,u') = 0.01$ for all $u, u' \in U$ and $w \in WP$. Then, the stochastic product knowledge of all users at the completion of the last process (i.e., *Integrate*) can be obtained as shown in Table 2.

Table 2: $PKnow(u, Integrate)$ ($u \in \{A,B,C,D,E\}$)

u	DSpc	RSpc	Sinfo	MSpc	SSpc	MMo	SMo	OCod
A	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0
B	0.0199	1.0	1.0	1.0	1.0	0.0	1.0	0.0
C	0.01	0.01	0.01	1.0	0.01	1.0	1.0	1.0
D	0.0001	0.0001	0.0001	0.01	0.0001	1.0	1.0	1.0
E	0.0001	0.0001	0.0001	0.01	0.0001	1.0	1.0	1.0

4 Case Study

In this section, we conduct a case study to demonstrate how the information leakage varies depending on the given software process model. For the experiment, we have implemented a software tool which automatically derives the stochastic product knowledge for a given process model.

In this case study, we further investigate the software process model shown in Section 2.1 (see Figure 2). We change AS on some processes, and investigate the stochastic product knowledge.

Let us recall the scenario of the process model. In the scenario, a work product *SecurityInfo* is assumed to be confidential. We also suppose that only developers A and B are authorized to access *SecurityInfo*. When *S-Design* is completed, A and B are the only developers that know *SecurityInfo*.

Our interest here is to evaluate the *risk* that *SecurityInfo* is leaked to unauthorized developers C, D or E, with varying the developers assignment AS in the subsequent three processes *Coding1*, *Coding2* and *Integrate*.

First, we define the following parameters for convenience:

- $U_{aut} = \{A, B\}$: authorized developers.
- $U_{uaut} = \{C, D, E\}$: unauthorized developers.
- $PC_{tgt} = \{Coding1, Coding2, Integrate\}$: target processes where the developers assignment is varied.

The risk of the leakage heavily depends on how the authorized developers (A, B) collaborates with the unauthorized ones (C, D, E) in the target processes (PC_{tgt}). To characterize this, we define the following parameter for a process p.

$$Col(p) = |U_{aut} \cap AS(p)| * |U_{uaut} \cap AS(p)|$$

Also, we define

$$Col = \sum_{p \in PC_{tgt}} Col(p)$$

$Col(p)$ represents the number of combinations of authorized and unauthorized developers in a process p. This intuitively characterizes the *degree of collaboration* where an authorized developer *interacts* with an unauthorized one in p. For example, if $AS(p) = \{A, B, C, D\}$ with $U_{aut} = \{A, B\}$, $U_{uaut} = \{C, D\}$, then $Col(p) = 4$, which implies that there are 4 patterns where an authorized A or B interacts with an unauthorized C or D. Col is the total number of such interactions in the target processes. Hence, the greater the value of Col, the more the authorized developers can collaborate with the unauthorized ones.

For a fixed developers assignment $as = [AS(Coding1), AS(Coding2), AS(Integrate)]$, the risk that *SecurityInfo* is leaked to unauthorized developers is formulated by:

$$Risk_{as} = \sum_{u \in U_{uaut}} Pkn(u, integrate, SecurityInfo)$$

Using the developed tool, we have computed $Risk_{as}$ for all possible assignments $as \in 2^U \times 2^U \times 2^U$. Also in the computation, we assume a fixed probability $leak(u,w,u') = 0.01$ for all $u, u' \in U$ and $w \in WP$.

Figure 4 depicts a scattered plot, where the the horizontal axis represents Col, while the vertical axis plots $Risk_{as}$. Table 3 shows the average value of $Risk_{as}$ with respect to Col.

As seen in the result, the risk that *SecurityInfo* is leaked grows as Col increases. This implies that more collaboration among authorized and unauthorized developers causes the higher risk of information leakage. In this case study, each probability that a developer u leaks a product w to another u' is not very large (i.e., $leak(u,w,u') = 0.01 = 1\%$). However, if the developers share many processes, the total probability of leakage becomes significantly large. For $Col = 18$ where all of five developers are assigned to every target process, the risk becomes as much as 18%.

Thus, the proposed method can be used as a simple but powerful means to evaluate the software process *quantitatively* from the viewpoint of the information leakage.

5 Related Work

As far as we know, there is no research study on a software process involving the leakage of product

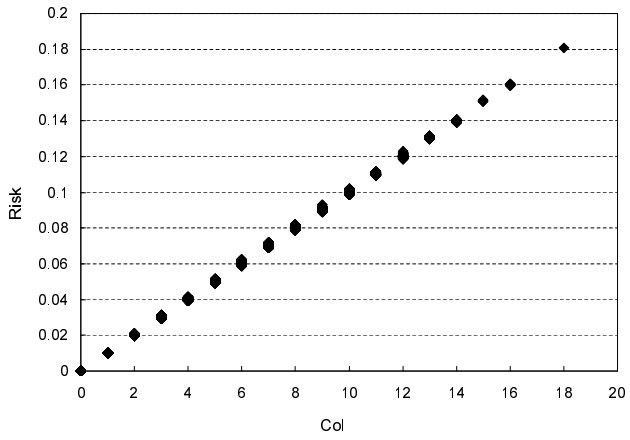


Figure 4: Computation Result of *Risk*

Table 3: Average of *Risk* w.r.t. *Col*

Col	Risk
0	0.000000000
1	0.010040352
2	0.020082816
3	0.030094976
4	0.040112381
5	0.050125019
6	0.060119186
7	0.070156853
8	0.080101327
9	0.090217868
10	0.100071167
11	0.110321914
12	0.120011146
13	0.130566804
14	0.139962521
15	0.151329923
16	0.160040993
17	—
18	0.180650505

knowledge from a person to another. Chou, et al. (Chou, Liu & Wu 2004) presented a model for access control named *WfACL*, which aims to prevent information leakage within work flows that may execute among competing organizations. They address issues related to management of dynamic role change and access control. However, the model includes no concrete method to *evaluate* the risk of leakage quantitatively.

A numerical approach to compute the information leakage might be to use extensively *Generalized Stochastic Petri Net (GSPN)* (Marsan et al. 1995). We first examined this approach. To do this, however, both the structure of process and the dynamics of leakage must be modeled in one GSPN. This complicates the net structure, and the state space becomes so large that the GSPN solver cannot compute the probability in reasonable time. Therefore, we decided to treat the process description and the leakage computation separately.

In addition, much research has been focused on different kinds of access control methods, such as *role-based access control* (Ferraiolo & Kuhn 1992)(Sandhu, Coyne, Feinstein & Youman 1996), *task-based access control* (Thomas & Sandhu 1997). The goal of access control is to ensure that only authorized people are given access to certain resources (i.e., products in our problem). However, the aim of the proposed is not to control the access authority, but to evaluate the risk of leakage as unexpected knowledge transfer among developers.

6 Conclusion

In this paper, we have presented a method to evaluate the risk of information leakage in software development process. We formulated the leakage as an unexpected transfer of product knowledge among developers sharing the same process. We then proposed a method to derive the probability that each developer knows each work product at any process of software development. We also conducted a case study. The result quantitatively shows that more collaboration among authorized and unauthorized developers causes the higher risk of information leakage.

Finally, we summarize our future work. In this paper, only a small case study is conducted. However, to show the practical effectiveness, we need further evaluation with more practical processes. The proposed method is simple and generic, therefore, it should not be limited to the security-sensitive software process. We expect that it is well feasible for other workflow-based applications, such as *medical work flows* (Quaglini, Mossa, Fassino, Stefanelli, Cavallini & Micieli 1999) where private information must be protected. Investigation of the emerging application domain is also an interesting issue.

References

4C-Entity (2001), *Policy statement on use of content protection for recordable media, (CPRM) in certain applications*. (Available online August 2001).
URL: *http://www.4centity.com/*

Chou, S.-C., Liu, A.-F. & Wu, C.-J. (2004), 'Preventing information leakage within workflows that execute among competing organizations', *The Journal of Systems and Software* . (Available online 4 February 2004).
URL: *http://www.elsevier.com/locate/jss*

Chow, S., Eisen, P., Johnson, H. & van Oorschot, P. (2002), A white-box DES implementation for

DRM applications, *in* 'Proc. 2nd ACM Workshop on Digital Rights Management', pp. 1–15.

Feiler, P. H. & Humphrey, W. S. (1993), Software process development and enactment: Concepts and definitions, *in* 'Proc. 2nd International Conference on Software Process', pp. 28–40.

Ferraiolo, D. & Kuhn, R. (1992), Role-based access controls, *in* '15th NIST-NCSC National Computer Security Conference', pp. 554–563.

Garg, P. K. & Jazayeri, M. (1995), *Process-Centered Software Engineering Environments*, IEEE Computer Society Press.

Jacobson, I., Booch, G. & Rumbaugh, J. (1999), *The unified software development process*, Addison-Wesley Longman Publishing Co., Inc.

Keller, F., Tabeling, P., Apfelbacher, R., Grone, B., Knopfel, A., Kugel, R. & Schmidt, O. (2002), Improving knowledge transfer at the architectural level: Concepts and notations, *in* 'Proc. The 2002 International Conference on Software Engineering Research and Practice'.

Liong, Y.-L. & Dixit, S. (2004), 'Digital rights management for the mobile internet', *Wireless Personal Communications* **29**(1-2), 109–119.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S. & Franceschinis, G. (1995), *Modelling with Generalized Stochastic Petri Nets*, John Wiley.

Quaglini, S., Mossa, C., Fassino, C., Stefanelli, M., Cavallini, A. & Micieli, G. (1999), *Guidelines-Based Workflow Systems*, Vol. 1620/1999 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 65–75.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L. & Youman, C. E. (1996), 'Role-based access control models', *IEEE Computer* **29**(2), 38–47.

Thomas, R. K. & Sandhu, R. S. (1997), Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented autorization management, *in* 'Proc. the IFIP Workshop on Database Security', pp. 166–181.