There have been two well-known models for host based intrusion detection. They are called Anomaly Intrusion Detection (AID) model and Misuse Intrusion Detection (MID) model. The former model analyzes user behavior and the statistics of a process in normal situation, and it checks whether the system is being used in a different manner. The latter model maintains database of known intrusion technique and detects intrusion by comparing a behavior against the database. An intrusion detection method based on an AID model can detect a new intrusion method, however it needs to update the data describing users behavior and the statistics in normal usage. We call these information profiles. There are several problems in AID to be addressed. The profiles are tend to be large. Detecting intrusion needs a large amount of system resource, like CPU time and memory and disk space. An MID model requires less amount of system resource to detect intrusion. However it cannot detect new, unknown intrusion methods. Our method solves these problems by recording system calls from daemon processes and setuid programs. We have further improved the method to eliminate false positive intrusion detections by adopting a DP matching scheme.

1. Introduction

In order to prevent unauthorized access to computers, current working computers and networks need some security reinforcement. However, it is in fact impossible to completely prevent from malicious intrusions. Thus, we need to detect intrusions and make the damage less serious. Intrusion Detection Systems\(^1,2\) are the systems to detect intrusions for the purpose described above. In this paper, we pay attention on processes running on each host, or computer. We collect system calls from a process. We build a profile of the process based on this information. With this profile, we can detect intrusions by using only a small amount of data. This method is also aimed to be applicable to remote attacks, which have been considered to be difficult.

An intruder will take control of a target system and the system can be a stepping stone to other attacks or intrusions. If we fail to find the intrusion in an early stage, damages caused by the intrusion can spread on other hosts through the network. This is why we need to detect intrusions as early as possible.

Intruders often show different usage behavior from normal system users. Anomaly Intrusion Detection model (AID)\(^1\) finds intrusions by detecting illegal usage of the system. In this model, a detecting method generates profiles from behavior of users and system by observing usage in normal situation. The detecting system detects an intrusion by calculating deviation of actual behavior of users and system usage from the profiles.

The Misuse Intrusion Detection model (MID)\(^1\) employs database of known intrusion methods and security holes. This model detects intrusion by looking for the behavior. Because specific intrusion methods are recorded in the database, events under observation in MID model is very specific. For example, existence of a file with setuid bit on, and invocation of a specific system call is put under surveillance.

The advantage of intrusion detection system based on AID is that an unknown intrusion method can be detected. At the same time, however, this system requires profile be generated in advance, which takes time. Also, calculating deviation can cause heavy load to the computer. Still more, generated profiles need to be updated periodically. Intrusion detection systems based on MID model require less system resource because a target to be observed is limited and specific. Unfortunately, unknown intrusions which are not described in the database cannot be detected by MID.

For practical use, both AID and MID methods needs to be implemented. In our study, we propose a new method based on AID model.

The rest of this paper is organized as follows. In Section 2, we describe how to generate profiles. In Section 3, we evaluate our intrusion detection method using profiles generated with...
the method described in Section 2, and we propose to improve our method further. Section 4 discusses other related works. Section 5 concludes this paper.

2. Profile Generation

In UNIX operating systems, a program calls for a number of system calls. There may be some specific patterns in the method, the order, and the frequency of invocation of system calls. In an unauthorized access, system calls might be called in a different manner from normal usage. Figure 1 shows an example of system calls in an rlogin process. We record system calls in normal usage, to compare with those of working system.

2.1 Process Profile

2.1.1 Overview of Our Method

We generate profiles for each process based on the contiguity and frequency of system calls. Generating profiles for all of the processes requires a large amount of time and system resources. This is the typical disadvantage of the intrusion detection system based on AID model, as we mentioned in the Introduction. We propose a new method which does not require large system resources.

The attacker’s ultimate goal is to gain the root privilege, regardless of their technique. Thus, we need to prepare profiles of daemon programs running as a privileged user and programs with setuid attributes.

2.1.2 Profile Preparation Environment

We provide a target host with Solaris 2.5.1 operating system. We also use eight hosts on the same network in Waseda University to make the normal profile. The eight hosts are all running UNIX operating system. Five of them use Solaris 7, the rest use FreeBSD 3.4-RELEASE (see Table 1).

2.1.3 Organization of Process Profile

We record the sequence of system calls in normal usage described in Section 2.1.1. It is called a process profile. The Process profile consists of three type of profiles.

2.1.3.1 Base Profile:

Base profile is a collection of system calls in normal situation. In this profile, type of a system call and its ranking of occurrence frequency in a host are recorded. Since the number shows the rank, a small value means a system call which frequently appears. A large value corresponds to a system call which rarely occurs. By this representation, base profile can efficiently express small differences of system calls occurring in a process. Table 2 shows an example of a base profile. Numerals in the table represent rank of occurrence frequency.

Table 1 Organization of hosts in our environment.

<table>
<thead>
<tr>
<th>Host</th>
<th>Type of operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host A,B,C,D,E</td>
<td>Solaris 7</td>
</tr>
<tr>
<td>Host F,G,H</td>
<td>FreeBSD 3.4-RELEASE</td>
</tr>
<tr>
<td>Target host I</td>
<td>Solaris 2.5.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host</th>
<th>Type of operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host A,B,C,D,E</td>
<td>Solaris 7</td>
</tr>
<tr>
<td>Host F,G,H</td>
<td>FreeBSD 3.4-RELEASE</td>
</tr>
<tr>
<td>Target host I</td>
<td>Solaris 2.5.1</td>
</tr>
</tbody>
</table>

Table 2 An example of a base profile.

<table>
<thead>
<tr>
<th>System call</th>
<th>Rank</th>
<th>System call</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>9</td>
<td>connect</td>
<td>66</td>
</tr>
<tr>
<td>creat</td>
<td>45</td>
<td>creat64</td>
<td>57</td>
</tr>
<tr>
<td>door_info</td>
<td>38</td>
<td>dup</td>
<td>26</td>
</tr>
<tr>
<td>execve</td>
<td>27</td>
<td>fcnt1</td>
<td>15</td>
</tr>
<tr>
<td>fstatsync</td>
<td>68</td>
<td>fork</td>
<td>34</td>
</tr>
<tr>
<td>fstat</td>
<td>13</td>
<td>fstat64</td>
<td>16</td>
</tr>
<tr>
<td>getdents</td>
<td>74</td>
<td>getdents64</td>
<td>17</td>
</tr>
<tr>
<td>getgid</td>
<td>32</td>
<td>getmsg</td>
<td>49</td>
</tr>
<tr>
<td>getpeername</td>
<td>68</td>
<td>getpgid</td>
<td>68</td>
</tr>
</tbody>
</table>

Footnotes:

1. Privileged user is called root in UNIX operating system.
2. Programs with setuid attributes are programs that enable non-privileged users run programs with privilege of root (privileged user).
We could collect enough sample only within 30 minutes recording of system calls. In Section 3, we will evaluate the base profile. We reduce the cost in making profiles, which is a weak point of AID model. Furthermore, base profile need no updates unless another OS is installed, which is an advantage over the existing profiling method in AID model. Our base profile simply maps system calls and numbers. Thus, a base profile generated in one host can be used in another host with the same operating system installed.

We use Base profile because we need to consider the likelihood of occurrence of frequently used system calls, such as read and write, in a sample. By assigning large number to a rarely used system call, distance of a profile and sample (these words are described in following sections) can be emphasized, when the system call was used by an intruder.

2.1.3.2 SUID Profile
SUID profile is a record of system calls. The recording is started when the suid program is executed with execute() system call. SUID profile is a sequence of system call occurrences from a program in normal situation. We can use base profile (Section 2.1.3.1) to make SUID profile. A system call is transformed into a number which is indicated in the base profile. Some processes accept user input on the fly\(^2\). SUID profile for these processes ends when the process waits for users' input. Other processes which terminate automatically are recorded until the system call _exit() is called. We call these profiles SUID profiles. SUID profiles can be shared among hosts with the same operating system. Figure 2 illustrates SUID profiles of ping program on host A and B. The curves looks similar, although there is some time lag between the two.

The example shows that SUID profiles may be shared among hosts. Sequence of system calls do not change so often, unless another operating system is installed. We will discuss it in more detail in Section 3.3. Our profiles do not have to be updated frequently, whereas the AID model requires frequent renewal.

2.1.3.3 Daemon Profile
Daemon profile records sequences of system calls in a daemon process running on a system. A daemon process waits for a call from external processes. Daemon profile treats idle state as delimiter of profiles. A profile starts recording when a client connected to a daemon. Again, the base profile is used to make the daemon profile. Daemon process can be recorded in a small amount of data. This method also simplifies the comparison of profiles.

3. Evaluation of the New Method
3.1 Evaluating the Base Profile
We use a base profile to map a system call to a number. A system call which is frequently used is mapped to a small number. A system call which is rarely used is mapped to a large number.

Frequently used system calls may be used many times within a loop in a program. If these system calls are translated to large numbers, resulting distance of the profile and the sample will be large. By assigning small numbers to frequently used system calls, we can make the distance small. This is because system calls in a profile and system calls in a sample are likely to have small numbers assigned.

We have investigated actual data. We used the trace data of wu-ftp. This trace data is publicized at University of New Mexico web site\(^4\). We tried three types of system call-to-number mappings. First method is the most straightforward way, which maps a system call name (e.g., fork) to a system call number of the operating system (e.g., fork = 2 in SunOS). Second method is to assign a smaller number to frequently used system call. Finally, we tried with a mapping which randomly maps a system call to a number.

- Map a system call to its system call number
- Map a system call to its frequency rank
- Map a system call to random number

We evaluated these three methods of map-

\(^2\) Password input in rlogin program is an example.
ping system calls to numbers. Among those methods, the mapping of system call to its frequency rank gave the best result (See Fig. 3–6). With the frequency rank method, normal status score and detect status score are large, compared to those of other two mapping methods (See Table 3).

### Table 3 Comparison of mapping methods.

<table>
<thead>
<tr>
<th></th>
<th>Normal process</th>
<th>Intruded process</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call number</td>
<td>301.73</td>
<td>1364.10</td>
</tr>
<tr>
<td>Frequency rank</td>
<td>38.07</td>
<td>368.85</td>
</tr>
<tr>
<td>Random number</td>
<td>469.43</td>
<td>1628.41</td>
</tr>
</tbody>
</table>

#### 3.2 Daemon Profiles and SUID Profiles

We call SUID profile and daemon profile created from processes in normal usage simply “profiles” in this section. We call data used to compare with the profile a “sample.” A sample is a sequence of system calls converted to sequence of integers with the base profile.

If we have a profile: \( A = a_1, a_2, \ldots, a_i, \ldots, a_J \), and a sample: \( B = b_1, b_2, \ldots, b_i, \ldots, b_J \), We calculate the difference between the profile and the sample. It is called a distance represented as \( D(A, B) \) in Formula 1. We use the term score which means the values of \( D(A, B) \).

This definition is similar to definitions used in speech recognition field.

\[
D(A, B) = \begin{cases} 
\sqrt{\sum_{k=1}^{I} (a_k - b_k)^2 + \sum_{k=I+1}^{J} (b_k)^2} & (I < J) \\
\sqrt{\sum_{k=1}^{I} (a_k - b_k)^2} & (I = J) \\
\sqrt{\sum_{k=1}^{J} (a_k - b_k)^2 + \sum_{k=J+1}^{I} (a_k)^2} & (I > J) 
\end{cases}
\] (1)

We classify normal state, caution state and detect state based on the distance, or score,
$D(A, B)$. There are about 80 to 90 kinds of system calls observed during profile preparation. We define the normal state as a state when distance is less than score 25. The figure of 25 is approximately equal to one quarter of 80 or 90. Similarly, we call it caution state when distance is less than score 50, i.e., a half of number of types of system calls. It is in a detect state when the distance is larger than score 50. Table 4 shows three states and the relationship to the distance $D(A, B)$.

The distance described above becomes large if the length of the profile and the sample is big. We may redefine the distance to reduce the value of the distance, by dividing $D$ by length $I$: $D' = D(A, B)/I$.

We adopt the simple definition, because we do not necessarily need to suppress the impact of the sample length. In fact, if the sample is very long, it may be a sign of actual intrusion.

The separating score of status (Table 4) are not subordinate to sample length or profile length. This is for simplicity. Precisely speaking, we need to adjust these values for each programs. To figure those values out is to run a program several times and calculating an exact value.

3.3 Testing in Normal Usage

3.3.1 Testing with ping

We evaluate the proposed method in a normal usage of a setuid program, that is a command of ping. Sequences of system calls in a profile and a sample are illustrated in Fig. 7. Both of the curves overlap completely, and you cannot see any difference. The distance (score) is 0.00, which means it is in normal state. This result is trivial because the system calls occurred in the same order both in the profile and the sample. We also found that the behavior of ping is the same among the target host and user shells in other computers.

3.3.2 Testing with rlogin

Next, we evaluate our method with a normal usage of a setuid program rlogin. We calculate the distance of the profile and a sample. The result is 74.94, which means detect state. This is a false result because the two curves are similar elsewhere. Figure 8 shows system calls of the profile and a sample. There is some time lag between the curve of profile and a sample. The sequence of system calls in this sample has some system calls which were not recorded in the profile. The reason is because the user used a different shell program for a sample. In Section 3.4, we will improve the calculation method to adjust the time lag caused by different user shells.

3.4 Apply DP Matching to Distance Calculation

We use DP Matching\(^5\),\(^6\) to overcome the time lag in calculating the distance. DP matching is a kind of pattern matching method, which finds best matching when the compared patterns have different number of elements with them. This method is mainly applied in speech recognition, where it is necessary to calculate best matching with a pattern and a sample by expanding time or contracting time nonlinearly. We apply this method to calculate best matching score of two patterns when they have a time lag. DP matching is explained in the Appendix in this paper.

3.4.1 Re-definition of the Distance

In order to apply DP matching to intrusion
detection, we use the technique of DP Matching as follows.
\[
\begin{align*}
A &= a_1, a_2, \ldots, a_i, \ldots, a_I \\
B &= b_1, b_2, \ldots, b_j, \ldots, b_J
\end{align*}
\]

(2)

A. We calculate \textit{optimal matching} (see Appendix) of system call sequences of a profile and a sample. We use the optimal matching to modify the profile \( A \) and the sample \( B \). Modified profile \( A' \) and the sample \( B' \) will be compared.

B. Profile \( A \) is made as described in Section 2.1.3. The profile has elements (Formula 2) represented by the integer in base profile.

C. The distance \( d(a_i, b_j) \) of characteristic vector \( a_i \) and \( b_j \) (see Appendix) is the difference of the two elements. Calculate optimal matching of a profile and a sample using DP matching, and represent modified elements in the order of occurrence.

\[
\begin{align*}
A' &= a'_1, a'_2, \ldots, a'_i, \ldots, a'_K \\
B' &= b'_1, b'_2, \ldots, b'_j, \ldots, b'_K
\end{align*}
\]

(3)

We redefine the distance between a profile and a sample based on the Formula 3. Hereafter, we calculate \textit{distance} modified with DP matching treatment as in Formula 4.

\[
D(A, B) = \sqrt{\sum_{k=1}^{K} (a'_k - b'_k)^2}
\]

(4)

3.5 Testing Normal Usage with DP Matching

3.5.1 Testing \texttt{rlogin} in Normal Usage (revised)

We evaluate normal usage of \texttt{rlogin} command with the distance calculation using DP matching. In Section 3.3.2, we failed to correctly evaluate \texttt{rlogin} program. The result shows a false alarm. Figure 9 shows the modified profile and a sample. Both of them are dealt with DP matching to adjust the time lag.

As a result, the distance between two patterns is score 14.04, which means it is in normal state. This result shows DP matching can eliminate surge of system calls caused by the difference of user shells.

3.5.2 Testing \texttt{inetd} Program in Normal Usage

Secondly, we evaluate the improved intrusion detection method with a daemon profile of \texttt{inetd} daemon. This profile is a record of system call sequence which begins when \texttt{inetd} accepts a connection request from a remote host, and ends when the connected process exits. Figure 10 illustrates an \texttt{inetd} daemon profile on host I, when a remote host F executes an \texttt{ftp} command and a \texttt{finger} command. In this figure, we handle profiles using the DP matching to obtain optimal matchings. In this example, distance in the \texttt{ftp} command is score 0.00, the distance in the \texttt{finger} command is score 3.00, where both of them are decided as normal states. If we calculate the distances without DP matching treatment, we get score 0.00 in the \texttt{ftp} command, which does not change, and score 68.40 in the \texttt{finger} command, which means this signals a false alarm. These examples show that the optimal matching treatment with the DP matching method is valid for the distance calculation in daemon profile.

3.6 Testing Unauthorized Access with DP Matching

This section tries to detect an unauthorized access to a host I, and see if our intrusion detection method can effectively detect the intrusion or not.

3.6.1 Testing Intrusion through \texttt{rlogin}

We try a buffer overflow attack to \texttt{rlogin}
command which has a bug, i.e. a security hole, in it. We deal with the profile and a sample (attack) with DP matching, then we calculate the optimal matching (Fig. 11). In this example, we could successfully detect the intrusion with distance of 247.68, that is an extraordinary high score.

In this testing, the profile and the sample behave similarly until around 110-th sequence, however the rest of the sequence behaves much differently. DP matching could not correct the lag time in the latter half. The distance is 270.54 when DP matching was not used, which is close to the result we get when DP matching is used. This example shows that DP matching does not adjust the difference among samples due to an intrusion. That is the reason why we can apply our method to detect intrusions.

### 3.6.2 Testing with Intrusion through eject

Next, we test with an eject command, which also contain a bug. This example shows a local attack using a buffer overflow. We treat the profile and sample (attack) with DP matching, then we calculate optimal matching (Fig. 12). Again, we can successfully detect the intrusion with the distance value of 208.09. The distance is 255.55 when we skipped the DP matching process. In this example, DP matching does not give us significant difference.

### 3.6.3 Testing with portscan

We also evaluated our method for portscan. A portscan means a user scans all the ports of a host. Although it is not an unauthorized access, it is oftenly followed by remote attacks. It is useful to detect a portscan to prevent intrusions. We treat the profile and sample (attack) with DP matching, then we calculate optimal matching (Fig. 13). The result shows that the distance is 200.68 and we can successfully detect the intrusion. The distance is 256.92 when we do not apply DP matching. DP matching gives a small difference in this example.

### 3.6.4 Testing with Intrusion through POP

$qpopper$\(^\text{a}\) is a well known implementation of POP\(^2\), which is an e-mail handling protocol. The $qpopper$ is executed via $inetd$ when $inetd$ receives a POP request. We remotely attack a hole in $qpopper$ program and evaluate our method with the collected sample. We treat the profile and sample (attack) with DP matching, then we calculate optimal matching (Fig. 14). The distance is score 0.00 and we fail to detect the intrusion.

The reasons of failure are described as follows:

- Profile for $qpopper$ is not prepared properly, because $qpopper$ is not a setuid program.
- Profile for $inetd$ is used for detection, because $qpopper$ is executed via $inetd$. We need some more refinement to detect remote attacks utilizing a bug in $qpopper$.

\(^{a}\) An abbreviation of Post Office Protocol
Wefailtoobservethebehaviorof qpopper for these reasons and thus cannot detect the intrusion. We prepare a profile by invoking qpopper process manually.

Then, we perform DP matching treatment on the profile and a sample (intrusion) (Fig. 15). The distance was 183.69, which correctly means that it is in detect state. The distance was 185.71 when we do not apply DP matching, and the result is almost the same as the score without DP matching.

3.6.5 Testing with LPR Trace Data from University of New Mexico

We also tried with lpr trace data. We extracted one process trace from normal data (http://www.cs.unm.edu/~immsec/data/LPR/UNM/normal/synth/syn.int.gz), and one process trace from intrusion trace data (http://www.cs.unm.edu/~immsec/data/LPR/UNM/anom/lprcp/exploit-unm.int.gz). The distance calculated without DP matching was 664.33. The distance calculated with DP matching was 138.30. This result shows DP matching is in fact useful.

4. Discussion

This paper proposes a new method for detecting intrusions. A similar approach was taken by Asaka\(^8\). Asaka’s method is based on the number of occurrences of system calls in UNIX operating system. It is based on the single parameter. Our method uses multiple parameters which comprises profiles, and it differentiates our method from Asaka’s method.

Warrender\(^9\) wrote a survey on intrusion detection method based on system calls. He counts four techniques: (1) analyzing a sequence of system calls, (2) counting the number of system calls, (3) using data mining, and (4) tracing the state by a finite state machines. Our method is classified into (1). The features of our method is adopting multiple parameters instead of a single counter.

A typical approach using (1) is illustrated by Forrest\(^10,11\). They analyses the patterns of system calls. Our method is a lightweight analysis because it uses three profiles instead of a mechanically cut patterns of system calls. Our methods can be processed efficiently.

There are other types of intrusion detection systems (IDSs). For example, there is an intrusion detection system which looks at syslog messages to find suspicious usage\(^12\). The IDS is based on a different information from system calls.

A cracker may intrude into a target system without being detected, if he/she can do it with inserting only few steps of system calls. This problem is common to most AID methods. Our new method is not an exception. If the difference between an intruders behavior and the profile is small, it is hard to detect the intrusion. There is no perfect solution to intrusion detection. It is necessary to combine several methods to detect intrusions in real working networks.

5. Conclusion

We propose a new intrusion detection method. It uses three process profiles which are generated from system call sequences. We have evaluated the new method through several examples: ftpd, ping, rlogin, eject, pop and lpr. We have successfully suppressed a false alarm by applying DP matching to calculate the optimal matching of a profile and a sample. We also show that the application of DP matching does not decrease the capability of intrusion detection. Thus, our new method can
effectively detect intrusions and misuses.

Process profiles are generated by analyzing system calls. Those process profiles can be classified into three types. Namely, base profile, SUID profile and daemon profile are generated. The base profile can be applied to other hosts and other shells, because frequently used system calls do not vary among hosts with the same operating system. The following are the advantages of our method over the existing Anomaly Intrusion Detection (AID) method:

- Quantity of profile data is small. The profiles of AID method tend to be large. In our new method, DP matching eliminates small differences among profiles. Thus, it requires smaller number of profiles to be prepared.
- Time required to prepare profile is less. As described above, our new method needs to prepare smaller number of profiles compared to AID without DP matching.
- Frequency to update the profile is low. Three types of profiles are used in our method. Base profile is one of the profiles, and it requires no update unless the Operating System is changed.
- System load is light. AID intrusion detection method compares a sample with many profiles. AID with DP matching needs to compare a sample with smaller number of profiles. Although DP matching is necessary for each comparison, DP matching can be processed with light system load.

We have achieved success in profile generation which does not require heavy system load. However, we have much room for improvement. First, we manually build a method for remote attack detection which is capable of detecting ports.

Acknowledgments We would like to appreciate participants of the “Internet Security” session in SAINT-2002 for useful comments.

The study was performed through Special Coordination Funds for promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology of the Japanese Government.

References

17) Inagaki, T.: Measurement of Network Performance in MBone, Master’s Thesis, Graduate
School of Science and Technology, Waseda University (1999).

Appendix

A.1 DP Matching method

If an input pattern A is a vector whose i-th element is \( a_i \) \( (i = 1, 2, \ldots, I) \). And an input pattern B is a vector \( b_j \) whose j-th element is \( b_j \) \( (j = 1, 2, \ldots, J) \). We represent characteristic vector of input patterns A and B.

\[
\begin{align*}
A &= a_1, a_2, \ldots, a_i, \ldots, a_I \\
B &= b_1, b_2, \ldots, b_j, \ldots, b_J 
\end{align*}
\]

(5)

Figure 16 shows \( a_1 \) and \( b_1 \), \( a_2 \), \( a_3 \), \( a_4 \) and \( b_2 \), \( a_5 \), \( a_6 \) and \( b_3 \) are corresponding. This line is called a path.

When a matching of pattern A and B is given by a path \( w(i(k), j(k)) \), \( k = 1, 2, \ldots \), the distance of two elements \( a_{i(k)}, b_{j(k)} \) is represented as \( d(a_{i(k)}, b_{j(k)}) \), the distance of two patterns \( D(A, B; w) \) will be given (Formula 7):

\[
D(A, B; w) = \sqrt{\sum_{k=1}^{w} d(a_{i(k)}, b_{j(k)})} 
\]

(7)

Finding the best matching means finding a path \( w \) which minimizes the value of \( D(A, B; w) \). In other words, Formula 8 gives the smallest distance \( D(A, B) \) when we take the best matching.

\[
D(A, B) = \min[D(A, B; w)] \quad (8)
\]

If we define \( g(i, j) \) as shown in Formula 9 and 10 and calculate \( g(i, j) \) iteratively, the distance between two patterns \( D(A, B) \) can be obtained by \( g(I, J) \), where \( I \) and \( J \) are the length of pattern \( A \) and pattern \( B \), respectively.

\[
g(i, j) = \min[g(i, j; w)] 
\]

(9)

\[
g(i, j; w) \text{ is the accumulation of paths from } (1, 1) \text{ to } (i, j) \text{ when } w((i(1), j(1)), \ldots, (i(k), j(k))) \text{ are given. If } g(i-1, j), g(i-1, j-1), g(i, j-1) \text{ are the minimum value of paths for the grids } (i-1, j), (i-1, j-1), (i, j-1), \text{ the minimum value } g(i, j) \text{ of the path to } (i, j) \text{ can be found as follows (Formula 10).}
\]

\[
g(i, j) = \min \begin{bmatrix} g(i-1, j) + d(i, j) \\
                        g(i-1, j-1) + 2d(i, j) \\
                        g(i, j-1) + d(i, j) \end{bmatrix} 
\]

(10)

Figure 17 illustrates Formula 10.

By adding initial value of \( g(1, 1) = d(a_1, b_1) \), Formula 10 can be calculated recursively. We can find the optimal matching path \( g(I, J) \) of two patterns.

Izuru Sato received a B.E. degree in 1997 and an M.E. degree in 1999 from Waseda University, Japan. He is currently a graduate student at Graduate School of Science and Engineering, Waseda University. His current research area is network security.
**Yoshinori Okazaki** received a B.E. degree in 1999 and an M.E. degree in 2001 from Waseda University, Japan. He works currently at Matsushita Electric Industrial Co., Ltd. He is engaged in development of Network Appliances.

**Shigeki Goto** is a professor at Department of Information and Computer Science, Waseda University, Japan. He received his B.S. and M.S. degrees in Mathematics from the University of Tokyo. Prior to becoming a professor at Waseda University, he has worked for NTT for many years. He earned a Ph.D. in Information Engineering from the University of Tokyo. He is a member of ACM and IEEE. Dr. Goto was a trustee of ISOC from 1994 to 1997. He is the chairman of the steering committee of IMnet. He is the vice president of JPNIC.