Collaboration Projects

Anomaly Detection System for Mobile Terminals

To deal with the threat of viruses that target mobile terminals, an ADS was proposed and evaluated on an embedded board simulating a mobile terminal. This research was conducted jointly with the Kato Laboratory (Professor Kazuhiko Kato) of the Graduate School of Systems and Information Engineering, University of Tsukuba.

Research Laboratories

Yuka Ikebe
Takehiro Nakayama
Masaji Katagiri

1. Introduction

The first virus to infect mobile terminals was discovered in June 2004. Since then, the number of mobile-terminal viruses has been increasing and there are currently several hundred types. Because platforms[*1] on which viruses were discovered are limited, these viruses do not present a problem in Japan at the moment. However, they may present a threat in the future and the need for mounting effective countermeasures is growing. The current mainstream antivirus technology is a virus scan[*2]. Given a newly created virus, it is defined as "unknown" at the time of its completion and as "known" once a security vendor becomes aware of it. Accordingly, a virus scan can detect a known virus with good efficiency but has more difficulties in detecting an unknown virus. During the time interval from unknown to known and to the creation and distribution of a pattern file that records the new virus pattern, the number of people harmed by an unknown virus can increase continually. With damage by viruses on the increase, it becomes increasingly important to shorten this time interval. In response to this need, we first focused our attention on shortening the time involved in changing from unknown to known considering the great effect that this would have on easing the threat of viruses.

In general, security vendors obtain information on unknown viruses by using honeypots[*3] or surveying Web sites. These methods need time to collect the information. It is thought that the time taken for getting hold of such information can be shortened by obtaining information on anomalous behavior occurring in a mobile terminal. Agreeing with this idea, we decided to adopt anomaly detection technology for detecting the occurrence of anomalous behavior. Here, overhead and accuracy can be used as indexes of Anomaly Detection System (ADS) performance, but since these two characteristics have a trade-off relationship, a technology that can provide good balance between the two appropriate for the application and usage environment needs to be found. In our research, our goal is to develop an ADS that can run on a mobile terminal, and we therefore established the constraint that the system must be able to run in an environment with minimal computational resources while also being accurate. In addition, the accuracy index can be classified into false positive and false negative, which also have a trade-off relationship. In light of the above, we decided to target an ADS that priori-

*1 **Platform**: OS or operating environment for running applications
*2 **Virus scan**: An antivirus technique that records virus features as patterns and checks for matches
*3 **Honeypot**: An intentionally prepared easy-to-attack location for discovering attacks on a system

tizes the reduction of the false negative rate while allowing a false positive rate at a level that can be handled by security vendors that receive detection notices. Many anomaly detection technologies have already been proposed, but there is none that can satisfy all of the above conditions. We therefore proposed an ADS that takes these conditions into account. We also implemented this system on an embedded board having the same level of performance as a mobile terminal and using this board, we measured accuracy and overhead and demonstrated the possibility of running an ADS on a mobile terminal. This research was conducted jointly with the Kato Laboratory of the Graduate School of Systems and Information Engineering, University of Tsukuba.

2. ADS

Anomaly detection technology judges an event that diverges from a previously defined normal event to be an anomaly. In this regard, we consider that a virus performs malicious operations according to the attacker's design and that an application infected with a virus demonstrates behavior different from the norm. The proposed system therefore adopts a technique that uses application behavior as an event. With this technique, those applications for which prevention of virus infection is desired are designated as monitored applications. Each application is then modeled and monitored.

2.1 Modeling

In modeling, the system acquires the behavior of a monitored application and creates a model that expresses normal behavior of that application. Behavior can be acquired by either of two methods: learning actual behavior or analyzing source code. The former leaves open the possibility of false positives since insufficient learning can result in labeling certain behavior as anomalous despite the fact that it is perfectly normal. The latter, on the other hand, may result in false negatives since incorrectly judged behavior that can occur or cannot occur depending on operating conditions can result in the labeling of anomalous behavior as normal. Given that one requirement of the proposed system is to give priority to the reduction of false negatives, the former method is adopted here.

False positives as described above when modeling by learning needs to be investigated. The basic principle behind anomaly detection technology is that only events included in the model are considered to be normal while all other events are considered to be anomalies. However, if such be the case, non-exhaustive learning at the time of modeling will give rise to false positives. Denoting all events that can happen as X, normal events as Y, and the set of normal events included in the model as M, ideally $Y=M$. In the case of insufficient learning, however, we have $Y \supset M$. During monitoring, the occurrence of M behavior and $X \cap \overline{Y}$ behavior will be correctly judged to be normal and anomalous, respectively, but $Y \cap \overline{M}$ behavior will be judged anomalous despite it being normal. In a simple application, all normal events can be easily learned so that $Y=M$ and false positives do not occur. In contrast, an application like an editor or browser is complicated in input and other operations, and learning all normal events can be extremely difficult with the result that $Y \supset M$. Here, the problem of false positives cannot be ignored. The proposed system attempts to reduce the false positive rate caused by the above situation.

In general, a particular application is used for a specific purpose, which means that behavior within the same application tends to be similar. Thus, even normal behavior that has not yet been learned ($Y \cap \overline{M}$) has a high possibility of having features similar to normal behavior obtained through learning (M). On the other hand, anomalous behavior associated with virus-based attacks ($X \cap \overline{Y}$) differs from the inherent purpose of the application in that it carries out the intentions of the attacker such as stealing data, crashing the system, etc., which means that it has features different from the behavior obtained through learning. Based on the above, we adopted an approach that statistically models normal behavior obtained through learning and quantifies the extent to which inspected behavior diverges from the model with the aim of detecting anomalies based on that value.

There is generally a high possibility of a system call[*4] occurring when a virus performs operations according to the intent of the attacker. This is because OS functions like file operations must be used to affect considerable damage in an attack, and in most cases, a system call is used for this purpose. For this reason, there have been many proposals for techniques that monitor applications for system calls as behavior to be watched for. However, an attack method that cannot be detected by this system-call technique has been discovered [1]. Thus, in the proposed system, we used both system calls and return addresses[*5] as behavior to watch for and attempted to reduce the false negative rate.

Specifically, we used a system call and return addresses stored on a call stack[*6] at the time of a system call as behavior to be examined and defined this as I_t (where t is any integer).

$$I_t = \{Sys, m_a, m_{a\text{-}1}, \cdots, m_1, m_0\} \qquad (1)$$

Here, Sys denotes system call number, m_x a return address, and x the position of the return address on the call stack. Setting the uppermost item on the stack to 0, this value is incremented by 1 for each item moving downward on the stack with an indicating the lowermost item.

In model creation, all return addresses included in obtained behavior are classified and grouped by system call number Sys and position x. In other words, this model is configured so that the return addresses stored at certain positions on the call stack at the time of a certain system call can be understood. Once grouping is completed for all behavior, a score can be computed by equation (2) for each group for use during monitoring.

$$Score = \frac{1}{N \times 2^x} \qquad (2)$$

Here, N is the number of return addresses included in each group. A modeling example is shown in **Figure 1**.

2.2 Monitoring

The monitoring process obtains behavior of the monitored application, performs an inspection by comparing that behavior with the model, and calculates an anomaly score that represents the extent of anomalousness. A monitoring example is shown in **Figure 2**. In the inspection, the model checks whether a return address obtained as behavior is included in a proper group (a group with the same system call and position as that of the model). If not included, an error occurs and the score of the group in question is added to the anomaly score.

In this method, a score is added to the total anomaly score for all return addresses not fitting the model, where the formula used to compute each score is designed to express the degree of divergence from the model. The score is large for small N and small x meaning that divergence is large. A small N means that the number of elements in the group is small and that return addresses other than those occurring during learning have a low possibility of appearing. Thus, the occurrence of an error for a group with small N is thought to indicate large divergence from the model. A small x, moreover, means a position near the top of the call stack. The closer a return address is to

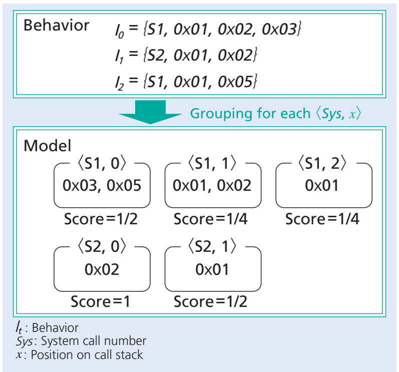

Figure 1 Modeling example

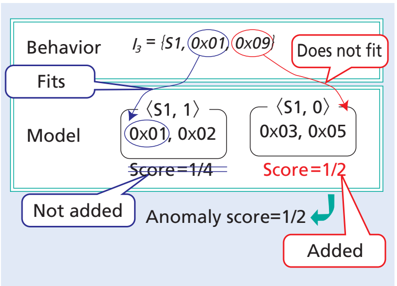

Figure 2 Monitoring example

*4 **System call**: A mechanism for calling an OS function from an application

*5 **Return address**: The address that a function returns to; the address of the instruction that must be executed next on completion of a certain function

*6 **Call stack**: A memory area that records information on functions to be called during execution of an application the top of the stack, the stronger is the correlation with the system call. As a result, the occurrence of an error for a group with small x likewise indicates large divergence from the model. Scoring in this way enables the system to judge behavior as normal or anomalous even if that behavior has not actually been learned.

3. Evaluation

We first implemented the proposed ADS on the evaluation board (Armadillo®[*7] -500) shown in **Photo 1**. The CPU incorporates ARM™ architecture[*8] widely used in mobile terminals and Linux 2.6 as the OS. Next, we ran actual applications on the implemented system and evaluated overhead and accuracy.

3.1 Overhead Performance Evaluation

For the evaluation, we used four types of applications for which the beginning and end of execution could be clearly measured. These were a2ps[*9], scp[*10], tnftp[*11], and vilistextum[*12]. Overhead was calculated by equation (3) using the time to execute the application with monitoring (T_{ads}) and the time to execute the application itself (T_{app}).

$$Overhead\,[\%] = \frac{T_{ads} - T_{app}}{T_{app}} \times 100 \quad (3)$$

The value for T_{app} fluctuates even for the same application so that overhead changes as well. Thus, in this experiment, we selected input data size as the factor behind this fluctuation in T_{app} and carried out measurements using files of 1 kB, 10 kB, 100 kB, 1 MB, and 10 MB in size. Measurement results are shown in **Table 1**.

More than 60% of measurement results in this experiment were 15% or less. The reason why the other overhead values were so large is thought to be that a small T_{app} makes the fixed time required for monitoring relatively large. In actuality, T_{app} was 0.32 s or less for all overhead in Table 1 greater than 15%. In this experiment, overhead was small for large T_{app} and large for small T_{app}. For the latter, though overhead is large, execution time is short and any feeling of discomfort felt by the user is thought to be small. Based on the above results, we consider the overhead generated by this system to be acceptable in actual use.

3.2 Accuracy Performance Evaluation

In this evaluation, we used four applications having vulnerabilities, namely, a2ps, emacs[*13], greed[*14], and vilistextum. Furthermore, to generate anomalous behavior, we created pseudo-viruses that exploit these vulnerabilities. Here, a model was prepared for each application using behavior obtained with one type of input. With these models, we evaluated accuracy by comparing anomaly score when monitoring normal behavior and that when monitoring anomalous behavior induced by the above pseudo-viruses. Experimental results are shown in **Table 2**. These results show that anomalous behavior results in scores that are much larger than those of normal behavior. An existing anomaly detection technique called VtPath [2] makes use of system calls and return addresses the same as the proposed sys-

Photo 1 Embedded board

Table 1 Overhead measurement results (%)

Application / Input data size	a2ps	scp	tnftp	vilistextum
1 kB	55	8	69	Could not measure
10 kB	38	11	76	Could not measure
100 kB	13	12	81	26
1 MB	7	3	61	≒0
10 MB	3	2	15	≒0

🟪 15% or less

*7 **Armadillo®**: Embedded devices manufactured by Atmark Techno, Inc.; a registered trademark of the same

*8 **ARM™ architecture**: CPU architecture for embedded devices featuring low power consumption. ARM™ is a trademark of ARM Ltd.

*9 **a2ps**: An open source application that converts a document file to an image file in PostScript format

*10 **scp**: An open source application that safely copies files between hosts by encryption

*11 **tnftp**: An open source application that exchanges files and directories with a server

*12 **vilistextum**: An open source application that converts an HTML file to a text file

*13 **emacs**: An open source application having sophisticated text-editor functions

*14 **greed**: An open source application having a function for downloading files

Table 2 Output anomaly score

Monitored behavior \ Application	a2ps	emacs	greed	vilistextum
Normal	0.37	0.53	0.017	9.69
Anomalous	132	526	763	454

tem, but its basic anomaly detection principle is to treat as normal only learned events as described earlier. As a result, false positives occur for all of the above applications for the amount of learning performed in this experiment. For the proposed technique, however, the experiment demonstrates that normal and anomalous behavior can be judged by an anomaly score and that false positives and false negatives can be prevented.

4. Conclusion

We proposed an ADS for detecting anomalous behavior for mobile terminals and demonstrated the usefulness of the proposed system by implementing the system in an environment equivalent to a mobile terminal.

In future research, we plan to perform more detailed evaluations by using more practical applications and pseudo-viruses having more advanced attack techniques. We will continue our studies on the feasibility of equipping mobile terminals with this system.

REFERENCES

[1] D. Wagner and P. Soto: "Mimicry attacks on host-based intrusion detection systems" in Proc. ACM Conference on Computer and Communications Security, pp. 255-264, 2002.

[2] H.H.Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong: "Anomaly detection using call stack information" in Proc. 2003 IEEE Symposium on Security and Privacy, pp. 62-75, 2003.