Adaptive Monitoring to Detect Intrusions in Critical Servers

João Antunes Nuno Neves
LASIGE, Departamento de Informática,
Faculdade de Ciências da Universidade de Lisboa, Portugal
{jantunes,nuno}@di.fc.ul.pt

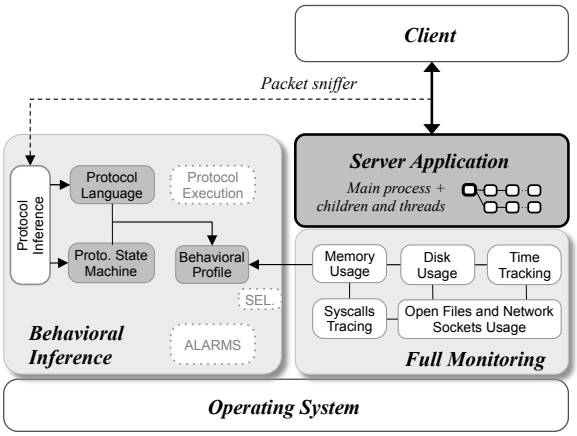
***Abstract*—An intrusion in a critical server can affect the security of an entire infrastructure that relies on it, including clients and other services. Hence, there is a constant concern in deploying and maintaining the correct execution of these servers. This paper presents an approach for continuous monitoring of a server execution in an adaptive way, where fundamental tasks are thoroughly monitored, and compared against a previously inferred behavioral profile.**

I. Introduction

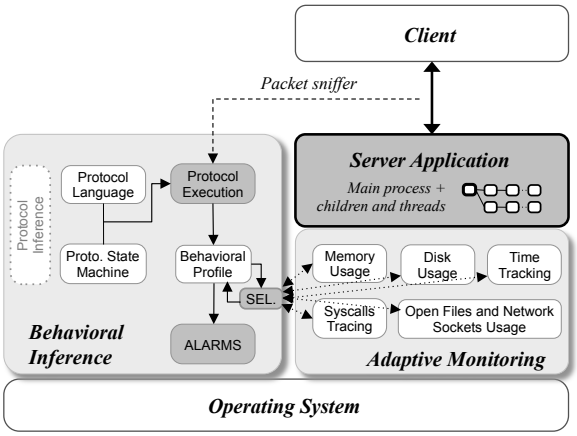
Nowadays, there is a significant reliance on several network servers for the execution of certain services, ranging from distributed name resolution to vital control operations in critical infrastructures. While providing these services, the servers are subject to a large number of threats and attacks, either carried out from the outside or internally, due to their necessary exposure.

However, despite the many efforts, vulnerabilities and exploits continue to surface everyday, compromising all kinds of servers. Often, when newer forms of attack are being used, intrusions are only detected when the server behavior is conspicuously incorrect or malicious, e.g., with a crash or due to client complains about abnormal actions. Still, a *post-mortem* analysis on the server can usually backtrace the failure to the point where its behavior started to deviate from the normal execution. Therefore, through a detailed monitoring and analysis, it should be possible to detect the intrusion right from the first steps of the attack. This has been tried by some intrusion detection systems (IDS) in the past, but these solutions are limited by a superficial model of the servers' behavior, such as logging the number, frequency and type of network connections and correlate that information with the users and other resources in the system [1], [2].

The paper presents an approach that automatically provides an accurate and detailed model of the normal execution of a server, which is suitable to be used as a frame of reference to identify malicious behavior in production systems. The solution resorts to reverse engineering techniques to infer a model of the specific protocol being implemented by the server, which includes the expected formats of the messages (both requests and responses) and the relations among them (e.g., a USERNAME request should come before the PASSWORD). This model is augmented with comprehensive monitoring data, capturing for instance the resources and code that is executed while the server processes requests of a certain type. The result is a detailed *behavior profile* of the server that allows the detection of an incorrect execution when an intrusion occurs.



(a) Inferring the behavior profile phase.



(b) Operational phase.

Figure 1: Adaptive monitoring approach.

II. Methodology

Our solution consists in learning a behavioral profile of an application server (*learning phase*) and in verifying if the server is operating in accordance with it (*operational phase*). The behavioral profile is an accurate model of the correct behavior of the server that captures how it follows

the protocol and operates internally. The general architecture of our tool is presented in Figure 1.

A. Learning Phase — Inferring the behavioral profile

During the learning phase, the Protocol Inference component listens to the traffic coming to and from the server (see Figure 1a). Based on the collected messages, it uses a protocol reverse engineering technique [3] to infer a specification of the protocol implemented in the server, which is capable of understanding the format of the messages (the Protocol Language) and their relations (the Protocol State Machine).

In parallel, a monitor component collects various kinds of data about the server execution. For example, the maximum amount of allocated memory *during* and *after* the handling of a request and the list of accessed files. The observed sequence of system calls and signals provides a pattern of the execution of the server for a given task. If a server executes a different set of instructions from the expected, such as when its flow of control is being hijacked, this will be revealed by a different set of system calls [4], [5].

The Behavior Profile component constructs a model that combines the inferred specification of the implemented protocol with the local monitoring data. We use a Mealy Finite State Machine (FSM) for the model because it can represent the interactions between the two protocol entities—it defines both a transition function (i.e., give the next server state for a given state and a client request) and an output function (i.e., the server response for a given state and request). The FSM is augmented with monitoring data that is requested from the monitor component at specific points of the server execution, in particular after replying to the clients. This monitoring data, therefore, depicts the server's progress with regard to the last monitoring request and provides a snapshot of the server's current internal state. Hence, besides the request and response, we also associate to each transition of the FSM the respective monitoring information (M_i). M_i corresponds to a tuple ($source_1$; ...; $source_n$), where each $source_i$ captures one of the dimensions of the server internal execution.

B. Operational Phase — Using the behavioral profile

In the operational phase, the tool uses the previously learnt behavior profile to evaluate the server's real-time execution. However, monitoring is an intensive task that may introduce a non-negligible overhead and affect the server's performance. Therefore, we resort to an *adaptive monitoring* approach, where only a subset of the protocol space is thoroughly monitored—the monitor gathers and collects internal execution information only at certain previously selected FSM transitions. The amount of monitoring data and which monitoring agents are used can also be adapted, thus providing a greater flexibility and control over the monitoring process.

If a client-server interaction correctly follows the protocol execution, the tool checks which monitoring data was selected for that particular transition (*SEL.* component) and requests the respective monitoring agents to collect internal data. The data is then sent back and compared against the behavior profile. Any deviant behavior is identified as a potential intrusion and an alarm is raised. For instance, the behavioral profile might define that for a particular protocol transition, the server executes 12 system calls, allocates 23 memory pages, and writes around 100k bytes to disk. An alarm is triggered if the server's memory usage is above that maximum threshold while executing that protocol transition (e.g., a potential memory leak). Or if an unusual number system calls and disk usage is observed, indicating that a different set of instructions is being executed (e.g., SQL injection attack). This approach automatically correlates different monitoring sources with the protocol execution, potentially improving the precision of the detection.

III. Conclusions and Future Work

Critical servers play such an important role in an organization that they deserve special protection measures. This work presents an approach for the detection of intrusions that is based on a behavior profile, which captures the implemented server protocol together with internal monitoring data. During the operational phase, the behavior profile is compared against the observed execution to discover deviations that could indicate that an attack / intrusion has occurred. To minimize the impact on the overall performance, only a subset of the functionality is thoroughly monitored.

Currently, we are in the process of implementing the tool, and at this moment we have completed most of the components needed in the learning phase.

Acknowledgments

This work was partially supported by EC through project FP7-257475 (MASSIF) and by FCT through the Multi-annual and CMU-Portugal Programmes and the project PTDC/EIA-EIA/100894/2008 (DIVERSE).

References

[1] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[2] M. Roesch, "Snort – lightweight intrusion detection for networks," in *Proc. of the USENIX Conf. on System Administration*, 1999, pp. 229–238.

[3] J. Antunes, N. Neves, and P. Verissimo, "ReverX: Reverse engineering of protocols," in *Proc. of the Working Conf. on Reverse Engineering*, Oct. 2011.

[4] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *IEEE Security and Privacy*, 1999, pp. 133–145.

[5] D. Leon, A. Podgurski, and W. Dickinson, "Visualizing similarity between program executions," in *Proc. of the Int. Symp. on Software Reliability Eng.*, 2005, pp. 311–321.