Short Paper: bufSTAT - a tool for early detection and classification of buffer overflow attacks *

Svetlana Radosavac, Karl Seamon and John S. Baras
Department of Electrical and Computer Engineering and the Institute for Systems research
University of Maryland, College Park 20742
Email:{svetlana,kks,baras}@isr.umd.edu

Abstract

Buffer overflows constitute by far the most frequently encountered class of attacks against computer systems. In this paper we introduce a tool, termed bufSTAT that achieves a low probability of false alarm and issues early attack warnings. BufSTAT relies on Finite State Machines (FSM) for attack modeling and can detect every stage of an ongoing attack and can thus prevent its execution by issuing early warning in a progressive manner. It can also detect sophisticated multi-stage attacks that are executed over long periods of time. A significant attribute of our approach is that it is amenable to detecting unknown attacks as well after appropriate modification of bufSTAT.

1 Introduction

With the increased use of networked computers for critical systems, network security is attracting increasing attention and computer network intrusions have become a significant threat in recent years. In this work we explore and compare two techniques for detection of buffer overflow attacks: Hidden Markov Models (HMMs) and Finite State Machines (FSMs). The approach applied in [3, 4] uses sequences of characteristic system calls for training, detection and classification of HMMs. The approach applied in [5] uses a State Transition Analysis Tool (STAT) that profiles attacks by comparing them against attack models described by deterministic FSMs. In this work we use the FSM models of buffer overflow attacks presented in [3, 4] for extending the STAT tool to enable *early attack detection and classification*. We develop the new tool, bufSTAT, apply it for detection and classification of buffer overflow attacks and compare its performance with the approach presented in [3, 4]. Since time response (i.e. time to detection) is the most important factor in attack detection, we can prevent delays by issuing early warnings as the attack's critical events happen.

[1]Research supported in part by the U.S. Army Research Office under CIP URI grant No DAAD19-01-1-0494

2 Buffer overflow attacks representation

A buffer overflow is the result of passing more data into a buffer than it can handle. As the stack contains not only the variables of vulnerable functions but also the return addresses for the points from which the functions were called, overwriting enough data potentially gives the exploit full control over the process. To achieve execution of exploit code, in most cases it is sufficient to write a script that spawns a shell. When we write or obtain the script code we know it must be part of the string which we are going to use to overflow the buffer. Also, the return address must point back into the buffer. After that the attacker is able to execute all other commands from the spawned shell.

Our goal is to recognize a malicious user who executes a specified sequence of actions in an attempt to break into the computer system. When HMMs are used, a database of sequences of malicious actions is needed. Those action sequences are converted into a set of HMMs, where each HMM is trained to fit attack sequences generated by a specific attack. Using the observed sequence the IDS performs matching to the HMMs, and computes the likelihoods that the observed action sequence could have been generated by each HMM. The HMM that matches the observed action sequence with the highest probability classifies the attack. Due to the fact that each HMM needs to be trained on numerous sequences of events, which requires significant computing power, this method cannot be used for online attack detection. The second downside of this approach is inability to recognize unknown, altered attacks that spread over long sequences of time.

In [5] the authors describe computer penetrations as attack scenarios using the FSM approach. This approach mitigates the disadvantages of signature-based approaches by

Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'0
0-7695-2369-2/05 $20.00 © 2005 IEEE

abstracting from the details of the modeled attacks. It represents only those steps in an intrusion that are critical for the effectiveness of the attack. Since the details of a particular attack are not used, it is possible to detect previously unknown variations of an attack or attacks that exploit similar mechanisms. This approach results in shorter processing times and higher efficiency of attack detection.

3 bufSTAT design

BufSTAT is an implementation of the state machines described in [3, 4] using a modification of the STAT tool suite [5]. In this section we illustrate the development of the bufSTAT tool by using the ffbconfig attack. bufSTAT also includes scenarios for attacks against eject, fdformat and ps attacks.

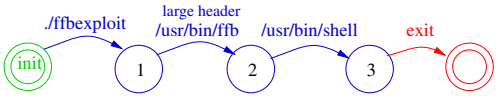

Figure 1. FSM of ffbconfig attack.

The ffbconfig attack is specific to the domain of a Solaris system. Thus, extension and event provider modules are required to provide STAT with the information needed to provide sufficient facilities for the attack scenario to use. These modules already exist in the form of USTAT [2], which provides types, helper functions, and events for use in a Solaris environment, and the ability to produce these events from a Solaris Basic Security Module (BSM) log file.

Creating the attack scenario in bufSTAT involves defining states and transitions. Each transition occurs when a user executes a program and the circumstances surrounding the execution match a certain set of parameters. The USTAT extension provides the scenario with both the path and the execution arguments of the executed program and functions to compare against them. The FSM diagram of the ffbconfig attack is presented in Fig. 1.

Testing of the above scenario was performed against the 1998 MIT data set [1]. Analyzing the data to compare it against the attack scenario's results was nontrivial. BSM is a binary format, so it is not simple to search through. To solve this problem, a simple attack scenario that records every execution event that takes place was written. Using that scenario, it was possible to search for the sequence of events that the ffbconfig attack scenario was observing.

Initial testing was performed on only a few days of data. Various problems arose immediately, such as the executed shell not matching those specified in the search and the path to the ffbconfig executable being in different locations on different systems. To solve these problems we added additional paths and shells to the appropriate conditional statements in the transitions and the resulting FSM is presented in Fig. 2.

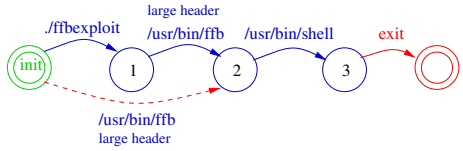

Figure 2. Final FSM of ffbconfig attack.

4 Comparison of bufSTAT and USTAT

USTAT, the STAT extension that bufSTAT is built upon, comes with two buffer overflow attack scenarios developed by the University of California, Santa Barbara. We now outline the important differences between these two scenarios and those contained in bufSTAT. Because of their different advantages and disadvantages, the buffer overflow scenarios from USTAT and bufSTAT are useful as complementary tools. USTAT scenarios can detect a wide range of buffer overflow attacks, and bufSTAT can detect and classify known attacks before the system is compromised.

The two USTAT scenarios serve two different purposes, but detect the same attacks in most situations. One checks for execution with large headers that contain non-ascii characters (this scenario has only one state), while the other looks for users gaining a root shell from a program that runs as root (see Fig. 3). Both scenarios are generic—they are not tailored to specific attacks, and thus should catch most buffer overflow attacks, even those that are not currently known.

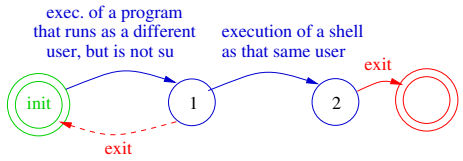

Figure 3. USTAT scenario: checks for users that gain root shell.

While bufSTAT cannot detect unknown attacks (though it is easy to extend to add new known attacks), it is very good at detecting the attacks it does know. Most importantly, it detects an additional stage in many instances of the attacks, the execution of an exploit script. Since STAT allows the user to link response modules to any state in a scenario, bufSTAT can in many cases launch a response before the attack has occurred. This response module could log a warning, or even disconnect and block the malicious user (whose ID is identified by the attack scenario) before any damage has been done.

	Data proc.	Training	Detection	Class.
HMM	1h	30min	<3min	<1min
bufSTAT	<2min	N/A	<1min	

Table 1. Performance of bufSTAT and HMM

5 Performance evaluation and comparison

In this section we evaluate the performance of the bufSTAT tool and compare the efficiency of the FSM and HMM approaches. We measured the bufSTAT data processing times for each data set. We can see that the processing times of any week of data for buf STAT are below 16s. However, the system overhead is as long as the bufSTAT processing times. For example, average bufSTAT data processing time for Week 3 of the MIT LL data set was 8.43s and average overhead for that week was 9.72s. Most of the system overhead comes from hard drive access time.

A comparison of the HMM approach and bufSTAT is presented in Table 1. We notice that data processing time is considerably longer for the HMM approach due to the fact that data has to be processed 3 times: for training, for determining whether the sequence is normal or anomalous and for classifying the anomalous sequences. Observing the obtained results, it is clear that the obvious advantage of bufSTAT is that training is not needed. Therefore, the resulting time difference among the above mentioned methods is significant: around 2h of total processing time for the HMM approach versus a couple of minutes for the bufSTAT.

When comparisons between detection success rates for HMM and bufSTAT approaches were performed, the latter technique outperformed the HMM approach by a small margin (100% for ffbconfig, eject and fdformat vs. approximately 99.7% for the HMM approach).The high detection rate is due to the fact that in [3, 4] we used a tool conceptually similar to a simplified version of the USTAT detection tool and avoided training on normal or anomalous sequences. Due to the very similar attack signatures of all buffer overflow attacks, the misclassification rate among different buffer overflow attacks for the HMM approach is relatively high. On the other hand, the similar structure of buffer overflow attacks represents an advantage in constructing the classification code in bufSTAT. In addition, early attack detection is enabled by the very nature of FSM models, while with HMM models the user has to wait until all sequences of data are processed to be able to obtain a result. Obviously, the performance of bufSTAT is far superior to the performance of the HMM tool. It should be noted that the data processing phase in column 1 of Table 1 refers to processing of one week of data from the MIT Lincoln Labs data sets. Because it can process logs at a far faster rate than in the previous approach, bufSTAT can easily be used for online detection and classification of buffer overflow attacks. The below average performance of the HMM method for online attack detection is due to the length of data processing and training phases. The data processing phase of bufSTAT is approximately 60 times faster than that of the HMM approach. An additional advantage of the bufSTAT approach is that no training is needed. Further, similar structure of different buffer overflow attacks represents a significant disadvantage in the HMM approach, resulting in a very low classification rate. At the same time, this represents an advantage in the bufSTAT approach.

6 Conclusion

In [3, 4] we have shown that HMMs can be used for detection of buffer overflow attacks with very low false alarm rates. However, due to the nature of machine learning algorithms, this method exhibits a very low classification rate when the attacks have similar signatures. More importantly, the method fails to detect multiple stage, rare and new attacks. In order to increase classification rate, more training data needs to be obtained. Another downside of the HMM approach is that it cannot be used for online detection of attacks due to long data processing periods. HMM training is expensive (complexity is O(TS^2), where T is the length of the trace in number of system calls and S is the number of states). bufSTAT, on the other hand, does not require any training and exhibits extremely high detection and classification rate. Due to the fact that it searches for execution of *specific events*, it can detect every stage of an ongoing attack, providing early warnings as the events happen and preventing the execution of the attack. Furthermore, it can detect sophisticated multi-stage attacks that are executed over long periods of time. Overall, bufSTAT is a useful extension to USTAT due to the capability of issuing early attack warnings (and thus preventing attacks from happening), quick attack detection and classification and low misclassification rates.

References

[1] The MIT Lincoln Labs evaluation data set, 1998.

[2] K. Ilgun. USTAT: A Real-time Intrusion Detection System for UNIX. In *Proceedings of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, May 1993.

[3] S. Radosavac. Detection and Classification of Network Intrusions Using Hidden Markov Models. MS Thesis, University of Maryland College Park, 2003.

[4] S. Radosavac and J. S. Baras. Detection and Classification of Network Intrusions Using Hidden Markov Models. In *Proceedings of the 37th Conference on Information Sciences and Systems (CISS)*, Baltimore, Maryland, March 2003.

[5] G. Vigna, S. T. Eckmann, and R. A. Kemmerer. The STAT tool suite. In *Proceedings of DISCEX 2000*, pages 46–55, Hilton Head, South Carolina, January. IEEE Press.