Reduction of Mimicry Attacks on Host -Based Intrusion Detection System using Argument Abstraction

M.Lakshmi Deepthi, Dr.N.Kumarathan

Department of Information Technology, Sri Venkateswara College of Engineering, Sri Perambadur, Chennai, India

***Abstract*-** Host-based Intrusion Detection System (IDS) are based on comparing the system call trace of a process against a set of k-grams. A simple extension to self-based IDS incorporates system call arguments and process privileges either to reduce or make Mimicry attacks more difficult. To avoid increasing the false positives supplied, specifications has to be used to abstract the system call arguments and process credentials. The specification takes into account what objects in the system that can be sensitive to potential attacks.

I. INTRODUCTION

Computers systems are being continually changed by vendors, system administrators, and users. Programs are added and removed, and configurations are changed. Formal verification of a statically defined system is time-consuming and hard to do correctly; formal verification of a dynamic system is impractical. We can incrementally improve security through the use of tools such as Intrusion Detection Systems (IDS). The IDS approach to security is based on the assumption that a system will not be secure, but that violations of security policy (intrusions) can be detected by monitoring and analyzing system behaviour.

There are many different levels on which an IDS can monitor system behaviour. It is critical to profile normal behavior at a level that is both robust to variations in normal, and perturbed by intrusions. Privileged processes are running programs that perform services (such as send or receive mail), which require access to system objects that are inaccessible to the ordinary user. To enable these processes to perform their jobs, they are given privileges over and above those of an ordinary user.

Privileged processes need superuser status to access system resources, but granting them such status gives them more permission than necessary to perform their specific tasks. Consequently they have permission to access emphall system resources, not just those that are relevant to their operation. Privileged processes are trusted to access only relevant system resources, but in cases where there is some programming error in the code that the privileged process is running, or if the privileged process is incorrectly configured, an ordinary user may be able to gain super user privileges by the use of system calls.

Corruption of the services can allow an intruder to access the system remotely. Monitoring privileged processes also offers some advantages over monitoring user behaviour, which has been the most method to date. An Intrusion Detection System (IDS) continuously monitors some dynamic behavioural characteristics of a computer system to determine if an intrusion has occurred.

A. Host-based intrusion detection system

Host-based Intrusion Detection System (IDS) identify attempts to exploit program vulnerabilities, frequently by monitoring the program's execution. Intrusion detection aims at raising an alarm any time the security of host system gets compromised. There are much different architecture for IDS. IDS can be centralized (i.e. processing is performed on a single machine) or distributed across many machines. Almost all IDS are centralized; the autonomous agents approach is one of the few proposed IDS that is truly distributed. Furthermore, an IDS can be host-based or network-based; the former type monitors activity on a single computer, whereas the latter type monitors activity over a network. Hofmeyr et al.[15] proposed a biologically-inspired host-based IDS which detects anomalies on a running process.

Though highly successful, Intrusion Detection Systems are all susceptible of mimicry attacks. IDS and their refinements[2-4] called as self-based IDS, compare the un-parameterized system-call trace of a process against the process normal profile stored as a set of k-grams, i.e. short sequences of system calls with length k. IDS can be classified as black, gray or white-box detectors.

While self-based IDS seem quite reasonable and have been shown to be executive in detecting intrusions, they can be susceptible to evasion or mimicry attacks which disguise an attack so that it appears "normal" to the IDS.

II. RELATED WORK

Mimicry attacks on self-based IDS were introduced in [6,9]. Wagner and Soto[6] use finite state automata (FSA) as a framework for studying and evaluating mimicry attacks. They show that a mimicry attack is possible because additional system calls which behave like no-ops can be inserted into the original attack trace so that the resulting trace is accepted by the automaton of the IDS model. They demonstrate how a mimicry attack can be crafted from the autowux WU-FTPD exploit.

Independently, Tan et al. [9] show attack construction on self-based IDS as a process of moving an attack sequence into the IDS detection's blind region through successive attack modification. The focus in these works was to demonstrate the feasibility of mimicry attacks, but not on a detailed look at automatic attacks.Recently, Gao et al. [5] performed a study of black-box self-based IDS and also several gray-box IDS. They investigated mimicry attacks with window sizes up to length 6

and showed the existence of mimicry attacks across the methods and window sizes studied. They demonstrated that various forms of IDS are susceptible to attacks but did not go into details of attack generation. Here given an automatic attack construction algorithm for self-based IDS and similar IDS models, and show empirically that it is computationally easy to generate attacks on self-based IDS for larger window sizes ranging from k = 5 to 11. There are a number of other gray-box enhancements using run-time information which aim to increase the IDS' robustness.

Sekar et al. [14] proposed a FSA model built from both system calls and program counter information. Feng et al. [7] also make use of the call stack to extract return addresses. These enhancements have been evaluated in [9] where it is shown that attacks still can be constructed. The idea of analyzing arguments of operations for detecting behavior deviance appears in a number of works. For example, [13] shows how the use of enriched command-line data can enhance the detection of masqueraders.

Kruegel et al. [3] make use of statistical analysis of system call arguments which can be used to evaluate features of the arguments such as: string length, string character distribution, structural inference and token finder. White-box techniques which incorporate some form of program analysis can complement gray-box techniques. Giffen et al. [7] present a white-box IDS which makes use of static analysis to counter mimicry attacks. They provide some partial results which show how static analysis can make it more difficult for an attacker to manipulate the process and generate a mimicry attack.

Finally, sandboxing techniques also make use of system call argument checking. The systrace system [12] uses system call policies to specify that certain system calls with specific arguments can be allowed or denied. This can be thought of as being a self-based IDS with a window size of one.

III. MIMICRY ATTACK CONSTRUCTION

Trace: A trace is a sequence of system calls invoked by a program in its execution.

Consider self-based IDS model where the alphabet for traces are the system call numbers. To distinguish traces generated by a "normal" program execution versus one where the program has been attacked in some fashion.

Subtraces: Which are simply substrings of a trace.

Consider subsequences, which differ from subtraces as they are a subset of letters from the trace arranged in the original relative trace order, i.e. need not be contiguous in the trace. The objective of a self-based IDS is to examine subtraces and determine whether they are normal or not. Let us call a basic attack subtrace, one which is detected by the IDS. A mimicry attack disguises a basic attack subtrace into a stealthy attack subtrace which the IDS classifies as being normal.

A. Pseudo Subtraces

A weakness of a self-based IDS which makes use of a normal profile represented as a set of k-grams is that it can accept subtraces which actually do not occur in the normal trace(s). For example, consider the following two subtraces of a normal trace: (. . . . ,m_{i-4}, m_{i-3} ,m_{i-2}, m_{i-1}, A, B, C,D ,E, ... ,B, C, D, E, F, n_{i+1}, n_{i+2}, n_{i+3}, n , ...)

Suppose the window size is 5, and assume that the subtrace (A,B,C,D,E,F) never occurs in the normal trace. This subtrace, however, will be accepted as normal by a self-based IDS since the two 5-grams derived are present in the normal profile .Call such a subtrace, a pseudo subtrace for window size k, since it is not supported by the actual normal trace, yet passes the IDS detections all its k-grams are present in the normal profile.

A pseudo subtrace can be constructed by finding a common substring of length k – 1 + l with l > = 0 in two separate subtraces of length (m > k + l) and (n > k + l) respectively, and then joining them to form a new subtrace of length m + n – k – l + 1, considering l = 0. Then concatenate a pseudo subtrace with a normal subtrace or another pseudo subtrace to create a longer pseudo subtrace. A stealthy attack version of a basic attack is simply a pseudo subtrace in whichthe basic attack subtrace is its subsequence.

The term pseudo subtrace is used specifically so as to refer the resulting overall subtrace which is obtained by joining two separate subtraces. The resulting subtrace contains a foreign sequence of length k + 1 + l as a substring. When l = 0 in the joining operation, the foreign sequence is a minimal foreign sequence. In the previous example, (A,B,C,D,E,F) is a minimal foreign sequence for k = 5. The above process constructs a pseudo subtrace for a mimicry attack where minimal foreign sequences of length k + 1 may exist along that subtrace, each combining two unconnected subtraces of normal traces together. Here, it is to emphasize that the core components of mimicry attacks depend on the notions of subtraces and subsequences.

B. The Overlapping Graph Representation

Given a normal trace, represent a profile using what is called an overlapping graph. Consider the normal trace of a program N of length n, (N_1, N_2, N_3,…., N_n) where N_i is the letter representing a system call. Let K be the set of all k-gram subtraces derived from N according to the profile generation rule of a self-based IDS. Given two strings p and q, the function overlap (p, q) gives the maximal length of a suffix of p that matches a prefix of q. The overlapping graph G is defined as a directed graph (V,E) where the vertices V are the k-grams in K and the edges E connect two vertices p and q whenever overlap(p, q) = k – 1. Augment the trace N by adding a suffix consisting of the k – 1 occurrences of sentinel symbol, denoted by '$', signifying the end of the trace. This adds some additional k-grams and simplifies the algorithm. Figure 1 illustrates the overlapping graph constructed from a normal trace N :(A,B,C,D,E,F,G, A,B,E, F,H) with a sliding window of length 3-grams corresponding to (F,H, $) and (H, $, $) which are in G.

There are two kinds of edges in G: direct edges and pseudo edges. The direct edges are those edges which result from normal subtraces. Pseudo edges are those which are not created by two consecutive substrings of length k- 1 in the trace. Thus, pseudo edges can be used to generate certain pseudo subtraces since it is not in a normal subtrace. In Figure 1, the direct edges are drawn with a single arrow, while the pseudo edges are drawn with a double arrow.

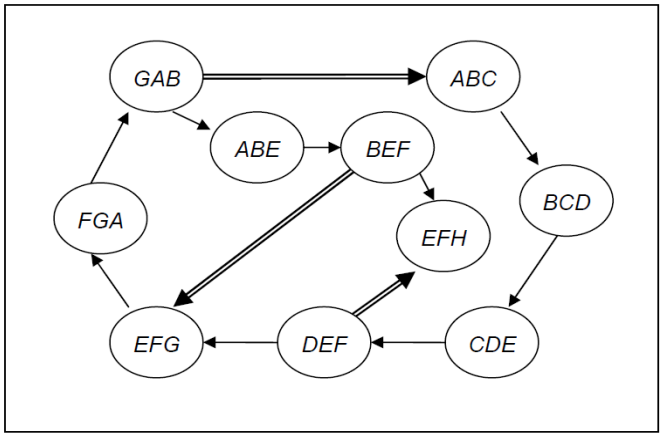

Figure 1. Overlapping graph G for N : (A,B,C,D,E, F, G, A,B,E, F,H) with k = 3

The graph G can also be viewed as a finite state automata model for recognizing normal traces. A slightly different graph representation is described where the k-gram database are the state transitions. Their representation however does not distinguish between what in the overlapping graph corresponds to direct and pseudo edges. Since our concern is to address the limitations of self-based IDS, the overlapping graph allows a natural variant where we can evaluate the difference between allowing pseudo edges and removing them.

C. Constructing of Mimicry attack

Rather than working with the FSA, it is more convenient to directly use the overlapping graph for constructing mimicry attacks. Given an overlapping graph G and a basic attack sequence A : (A_1, A_2, A_3 ,. . . A_l) is detectable by the IDS, to construct automatically the shortest stealthy attack subtrace L : (L_1, L_2, L_3 ,. . ..L_m) where m – l which contains A_1, A_2, A_3, ...A_l as a subsequence and where the other system calls in {L – A} behave as no-ops with respect to A. Transforming a basic attack subtrace A into the shortest stealthy subtrace L is equivalent to: finding the shortest path P on the overlapping graph G which monotonically visits nodes whose k-gram label begins with the symbol A_i for all $1 \leq I \leq l$.

Let augment G with an additional sub-graph, the occurrence subgraph. The nodes in the occurrence subgraph, which we will call W, are individual letters for each occurrence of the letter from its k-grams in G. For each node w_i in W, add an outgoing edge to all nodes in G where the first letter in its k-gram label is the same as the letter for w_i. The set of new edges from W to V be, the Occ set. The resulting graph G_0 is simply (V +W, E + Occ), which is called the extended overlapping graph.

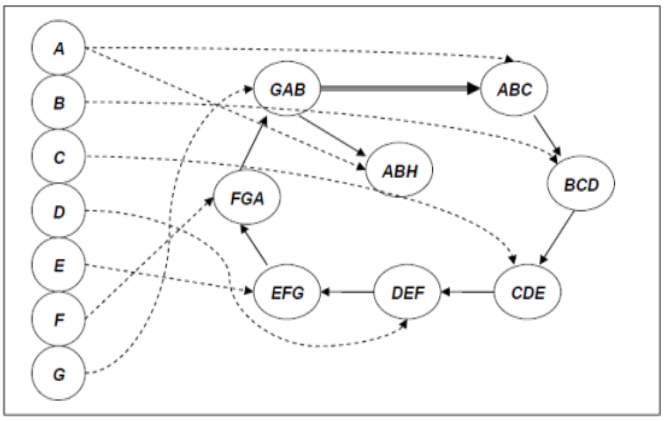

Figure 2, Extended overlapping graph for graph in Figure 1.

Illustration of the mimicry attack construction with the following example. Suppose that we want to construct a stealthy subtrace from a basic attack subtrace A : (G,C,D) using the extended overlapping graph G_0 in Figure 2 for the graph in Figure 1. Note that the subtrace (G,C,D) is detected as it is not a 3-gram of the normal trace. Inspecting graph G_0, the stealthy path: GAB-ABC-BCD-CDE-DEF. Thus, the stealthy attack subtrace is the sequence of (G, A, B, C, D), with A and B added as no-ops. This example uses the pseudo edge (GAB, ABC).

Attack construction performs a search to find the shortest mimicry attack. Each node in the search tree corresponds to one letter in the original attack string, A_i. The branches from A_i are the choices of constructing a subtrace starting from the potential k-grams for A_{i+1} pointed to by Occ. The process continues until we reach A_l which is the last node in the attack. In order to make the search more efficient, let us employ a branch-and-bound strategy to prune the constructed attacks which exceed the best solution found so far. The implementation uses the Dijikstra all-pair shortest path algorithm both to test connectivity between two nodes in G and also to assist in pruning for branch-and-bound search. A sketch of the algorithm is as follows.

Attack Construction Algorithm

Input:
- Sliding window length k
- A normal trace N : N_1, N_2, N_3, . . . , N_n
- Basic attack subtrace A : A_1, A_2, A_3, . . . , A_l

Output:
- Shortest stealthy subtrace L : L_1, L_2, L_3, . . . , L_m
- Or failure, if no solution trace can be found.

1. Perform Dijkstra all-pairs shortest path algorithm
   for all the nodes V .
   Between two adjacent nodes, set distance: =1.
   If two nodes are not connected then distance: = ∞.
2. Set Min_distance: = ∞ and Min_path: = < >.
   Create a special node v_0 where V_i .distance(v_0; v_i) := 0.
3. Perform branch-and-bound search on the search tree,
   for all i:=1 to l choose v_i from $\{v_i \mid (A_i; v_i) \in Occ\}$:
   - If distance(v_{i-1}, v_i) = ∞ then backtrack.

- Add distance(υ_{i-1}, υ_i) to current cost.
- If current_cost ≥ Min_distance then backtrack.
- If complete solution is found then
  If current cost < Min distance then
  Min_distance := current_cost;
  Min_path := current_path.

4. Once the search tree is fully explored:
   If Mi_ distance =∞ then return failure;
   Else return L : $L_1, L_2, L_3, \ldots, L_m$.

In order to use this algorithm in a buffer overflow setting, it needs to be modified to take into account the border k-gram. This is further discussed.

IV. IDS ENHANCEMENT

Consider a simple gray-box enhancement to an IDS which can either prevent or make mimicry attacks more difficult. To simplify the discussion and evaluation, apply the enhancement to the baseline self-based IDS which use system call numbers in the k-grams

A. *Arguments abstraction and privileges*

The credentials which determine the current privileges of a process are its effective user-id (euid) and effective group-id (egid). The euid (egid) is either the actual real uid (gid) of the user, or it has been changed by invoking a setuid (setgid) executable. So, euid and egid are simply a subset of all the user and group-id values defned in a system.

It is proposed to enhance k-gram to include not only the system number but also (abstracted) information about the euid, egid and system call arguments.It is common for attacks to try and exploit programs executing in a privileged mode. The idea is that such attacks can be detected if the corresponding system call subtraces are unprivileged in the normal trace(s). A program which conforms to a good setuid programming practice generally drops privileges as soon as possible.

Rather than using the actual values, we can abstract the euid, egid and system call arguments into categories based on a configuration specification. This is mainly to reduce the false positive rate which can be higher since the space of values is much greater. The abstraction technique also provides flexibility for us to group arguments and privileges together in terms of their importance/sensitivity level. Formally, we can represent the privilege and argument categorization in the operating system model with the following mapping functions:

Function EuidCat : $U \rightarrow U'$, where: U = the set of euid and $U' \subset N$ (the set of natural numbers).
Function EgidCat : $G \rightarrow G'$, where: G = the set of egid and $G' \subset N$.

For each s∈ S with S= the set of system call numbers, function ArgCats :
$A_{s,1}$ X $A_{s,2}$ X ……X $A_{s,max\_arg} \rightarrow C_s$,where $A_{s,i}$ for $i \in [1. . max\_arg]$ = the set of possible entries for i-th argument of the system call s, and Cs ⊂ N.

In the basic self-based IDS, the alphabet was over the system call numbers S, while in the extension, the alphabet is now a tuple U' X G' X S' X C where $C = \cup_{s \in S} C_s$

B. *Privilege abstraction*

The euid and egid section are meant to provide the actual value mapping for EuidCat:
$U \rightarrow U'$ and EgidCat : $G \rightarrow G'$. The example specification uses the following syntax for euid and egid:

$< u'_i >$: $<u_{i1}>$,$< u_{i2} >$, ……., $<u_{in}>$;
$< g'_i >$: $<g_{i1}>$;$< g_{i2} >$; ……., $<g_{im}>$;

Where $u'_i \in U'$, $u_{ij} \in U$, $g'_{ij} \in G'$, $g_{ij} \in G$

To ensure that EuidCat (EgidCat) is a total mapping, a special entry "*" is employed to indicate other euids (egids) so that the mapping satisfies the requirement for a function. As euid=0 and egid=0 signify important privileges in Unix, each of them has a distinguished mapping.

C. *Argument abstraction*

The specification is a straightforward one. It maps the system call together with its corresponding arguments (defined in an argument specific fashion, i.e. understands pathnames for open) into a number (its category). One point to note that while it is possible to have more complex abstractions, it is sufficient to only use a single abstract value to represent multiple arguments.

Some considerations in creating a definition:

The approach we have used is to focus the specification to a subset of system calls $S' \subset S$ which should be checked in order to prevent attacks aimed at gaining full control of the system.

Consider S' to be the system calls in Threat-Level 1 Category namely: open, chmod, fchmod, chown, fchown, lchown, rename, link,unlink, symlink, mount, mknod, init module and execve. Other system calls in $S - S'$ which have not been defined in the specification are mapped to a unique default value. Likewise Threat-Level 2 and Threat -Level 3 can be created. Given a system call $s' \in S'$, a simple approach for the choice of abstraction is to ensure that any critical operations on security-sensitive objects are mapped to a value different from a normal one. It is convenient, when specifying the abstractions and categories to make use of sequential matching from the start to the end of the definition. In this fashion, more specific mappings can be made first and the most general ones last. Pathnames require special treatment and we use a special notation,

p =<pathname>.

D. *Disallowing Transitions.*

It is also useful to specify the transitions that can lead to "bad states". The idea is to identify those singleton system calls with the corresponding privileges which can be sufficient to compromise the system's security. An example would be the operation of chown() on /etc/passwd with root privileges. Thus, the usual way of measuring anomaly signal by means of LFC

function. This can also be used as an enhancement to access control to actually deny such a system call invocation in a program.Our category specification defines bad transitions as:

s' c $[u', g']$*

where c is the abstracted value for the arguments of system call s', u' and g' are the abstracted privileges for user and group. Let D_0 be the set of bad transitions. This specification may be too strict and needs to be adjusted with respect to the normal traces. When the normal profile is extracted from the normal trace dataset, collect the set D_N, those transitions from D_0 which match against normal traces. The final adjusted negative transitions are

$D = D_0 - D_N$. The IDS detection then concludes that any system call in an execution trace matching an illegal transition d ϵ D constitutes an intrusion. In addition, we may also prevent the operation itself.

V. EXPERIMENTAL RESULTS

The construction of the shortest stealthy attacks on the two variations of self-based IDS and our improved IDS is presented with three IDS variants given in Table 1. Improved IDS is experimented against various mimicry attack strategies and investigate its false-positive rate. In the experiments,a sample generic configuration with several files which are security critical in the Unix/Linux environment: user and group related files (/etc/passwd,/etc/shadow, /etc/group), kernel memory device (/proc/kmem), and system configuration files (/etc/hosts.equiv) will be used. For simplicity most of the system configuration files in/etc (such as: /etc/inetd.conf, /etc/hosts,/etc/cron/*) and devices files in /dev are omitted. Included entries for various directories commonly found in the Unix/Linux file system hierarchy conforming to the File system Hierarchy Standard (http: //www.pathname.com/fhs/pub/fhs-2.3.html). While one can use a more detailed specification, this is already sufficient to show an increase in IDS robustness.

Remarks on the exploits used. Objective is to investigate the practicality of automated attack construction, that experiment with real programs using existing real exploits. Here,just consider two attack scenarios: (i)buffer-overflow scenario: where one can replace the shellcode of a buffer-overflow exploit with a code sequence executing a stealthy attack trace; and (ii) direct attack: which might be the result of replacing the program by a trojan which then executes a stealthy trace to fool the IDS.

The following remarks apply to our experiments:

- The three exploits make use of execve() system call to spawn a root shell. However, execve() is not present in the normal trace. Therefore, we can use an alternative strategy to write an entry to the file "/etc/shadow". This actually corresponds to Attack-strategy A2 from the list of strategies shown in Table 3. This particular attack strategy is chosen for detailed comparison here as it has been used for mimicry attacks in self-based IDS.

- In the buffer-overflow case, there is another constraint that the stealthy attack trace must be introduced at the "attack-introduction point" or "point of seizure". Hence, we need to manually determine this point and make note of the k system calls before the attack point, which we can call as border k-gram. Given this, we need to ensure that the concatenation of border k-gram and the stealthy attack trace still passes the IDS. Thus, we need to slightly modify the search algorithm as follows: (i) the border k-gram must be included as an additional input which will then define the associated border-node in V ;and (ii) the first-level nodes in V are explored during the search only if they are connected to the border node (and with *path _length* > k - 1).

Traceroot2 (Traceroute Exploit) The exploit attacks LBNL Traceroute v1.4a5which is included in the Linux Redhat 6.2 distribution. The original attack sequence is: setuid(0),setgid(0), execve("/bin/sh"). This is changed into: open(),write(), close(), exit().

The result of the attack construction on normal traces generated from three Traceroute's sessions(with a total of 2,789 system calls) for sliding-window sizes from k=5 to k=11 is given in Table 2.

A. *Improved IDS.*

Resistance against Various Attacks Having shown that the improved IDS can better withstand mimicry attacks, now evaluate the IDS against a number of different attack strategies.First, list some important files from the security viewpoint, namely F1:/etc/passwd, F2:/etc/shadow, F3: /etc/group, F4: /proc/kmem and F5: hosts. equiv. Next, Table 3 contains a list of number of common attack strategies in the Unix/Linux environment on those files above when the system calls are executed with superuser euid/egid privilege. While the list is not comprehensive, it suffices to demonstrate improvements in the resistance level of the IDS.Here, let chose the Traceroute program for this experiment. The experiment was done on normal traces described earlier (2,789 system calls) with a sliding-window size set to 5.Found that all the attack strategies listed in Table 3 fail on the tested normal traces even in the direct-attack search scenario. For most of the strategies (A6 - A61), the attacks fail because the needed attack system calls do not appear in the normal traces. In attacks A1 - A5, given the category specification, the attack searches fail because the normal traces do not contain the particular categories.

B. *False-Positive rate.*

Here are some preliminary results comparing the new IDS in terms of its false-positive rate to the baseline self-based IDS. Let chose a progras: traceroute in Redhat Linux 7.3. For this program, there are 10 trace sessions and then randomly chose one to be tested against the other 9. The results are shown in Table 4 below. Here we can simply measure the number of foreign k-grams. As can be seen, the enhancement does not increase the false positives.

VI. CONCLUSION

This paper presents an efficient algorithm for automated mimicry attack construction on self-based IDS. This is useful for evaluating the robustness of the IDS to attacks. Proposed an extension to self-based IDS using privilege and argument abstraction. This extension is both simple to use and also makes the IDS more robust. An important advantage of our IDS

extension is its simplicity. the simplicity means that it is easy to integrate into various IDS and also can be easily combined with other gray-box techniques to get a significantly more secure IDS.

ACKNOWLEDGMENT

The authors wish to thank Prof. E.G. Govindan and Dr.G.Sumathi, Dept of Information Technology, Anna University, Chennai for their valuable suggestions and cooperation throughout this research work.

REFERENCES

[1] A. Somayaji and S. Forrest(2000). "Automated response using system-call delays". In Proceedings of the "9th USENIX Security Symposium" pp.13-15.

[2] *A. Somayaji(2008). "Operating system stability and security through process homeostasis". Ph.D. Thesis, University of New Mexico.*

[3] *C. Kruegel, D. Mutz, F. Valeur, and G. Vigna(2009). "On the detection of anomalous system call arguments". In Proceedings of the European Symposium on Research in Computer Security (ESORICS).*

[4] *C. Warrender, S. Forrest, and B. Pearlmutter(2001). "Detecting intrusions using system calls: alternative data models", In Proceedings of the IEEE Symposium on Security and Privacy, pp:110-114.*

[5] *D. Gao, M.K. Reiter, and D. Song(2008). "On gray-Box program tracking for anomaly detection." In Proceedings of the 15th USENIX Security Symposium, pp1:22-128.*

[6] *D.Wagner and P. Soto(2002). "Mimicry attacks on host-based intrusion detection systems".In Proceedings of the 9th ACM Conference on Computer and Communications Security, vol.5, pp10:3-124. .*

[7] *H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong(2006)."Anomaly detection using call stack information". In Proceedings of the 2006 IEEE Symposium on Security and Privacy.*

[8] *J.Giffin, S.Jha, and B. Miller(2004)."Efficient context-sensitive intrusion detection".In Proceedings of the 11th Network and Distributed System Security Symposium, pp:14-16.*

[9] *K. M. C. Tan, K. S. Killourhy, and R. A. Maxion(2007). "An anomaly-based intrusion detection system using common exploits". In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection.*

[10] *K. M. C. Tan and R. A. Maxion(2003). "Determining the Operational Limits of an Anomaly-Based Intrusion Detector". IEEE Journal on Selected Areas in Communications, Special Issue on Design and Analysis Techniques for Security Assurance,21(1),pp:5-9.*

[11] *Mohan Rajagopalan, Matti A. Hiltunen, Trevor Jim, and Richard D. Schlichting, Fellow(2006) "System Call Monitoring Using Authenticated System Calls" IEEE Transactions on dependable andsecure computing,vol3,no 3, pp.17-19.*

[12] *N. Provos(2005). "Improving host security with system call policies". In Proceedings of the 12th USENIX Security Symposium.*

[13] *R. Maxion(2009). "Masquerade detection using enriched command lines". In proceedings of the International Conference on Dependable Systems & Networks vol 2,pp.155-172.*

[14] *R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni(2001). "A fast automaton-basedmethod for detecting anomalous program behaviors". In Proceedings of the IEEE Symposium on Security and Privacy.*

[15] *S. Hofmeyr, S. Forrest, and A. Somayaji(1998)." Intrusion detection using sequences of system calls". Journal of Computer Security vol6:pp.151-180.*

AUTHORS

First Author – M.Lakshmi Deepthi, Department of Information Technology, Sri Venkateswara College of Engineering, Sri Perambadur, Chennai, India

Second Author – Dr.N.Kumarathan Department of Information Technology, Sri Venkateswara College of Engineering, Sri Perambadur, Chennai, India