Detecting and Displaying Novel Computer Attacks with Macroscope[1]

Robert K. Cunningham, *Senior Member, IEEE*, Richard P. Lippmann, *Senior Member, IEEE*, and Seth E. Webster, *Member, IEEE*

***Abstract*- Macroscope is a network-based intrusion detection system that uses Bottleneck Verification to detect user-to-superuser attacks. Bottleneck Verification (BV) detects novel computer attacks by looking for users performing high privilege operations without passing through legal "bottleneck" checkpoints that grant those privileges. Macroscope's BV implementation models many common Unix commands, and has extensions to detect intrusions that exploit trust relationships, as well as previously installed Trojan programs. Bottleneck Verification performs at a false alarm rate more than two orders of magnitude lower than a reference signature verification system, while simultaneously increasing the detection rate from roughly 20% to 80% of user-to-super-user attacks.**

***Index terms*—intrusion detection, security, bottleneck verification**

I. Introduction

The quantity of valuable information transferred via the Internet for commerce, banking, and other forms of communication has increased the potential damage that can be done by computer attacks, while worldwide connectivity allows attackers to initiate their attacks from safe locations. Although many have tried to prevent attacks by developing safe and secure software, complex interactions between the software for operating systems, network connectivity and applications make it difficult to prevent all of the weaknesses that are exploited by attackers. Intrusion detection systems are used to detect attacks that exploit these inevitable weaknesses and bugs. Most commercial and government systems detect previously known and studied attacks, while more complex intrusion detection systems detect never-before-seen, new attacks.

The many different approaches to intrusion detection systems are described elsewhere [4][19]. Figure 1 shows four major approaches to intrusion detection and the different characteristics of these approaches. The figure is divided vertically into approaches that detect new, unseen attacks, and those that only detect previously known attacks. Simpler approaches are on the left and approaches that are computationally more demanding with greater memory requirements are shown towards the right.

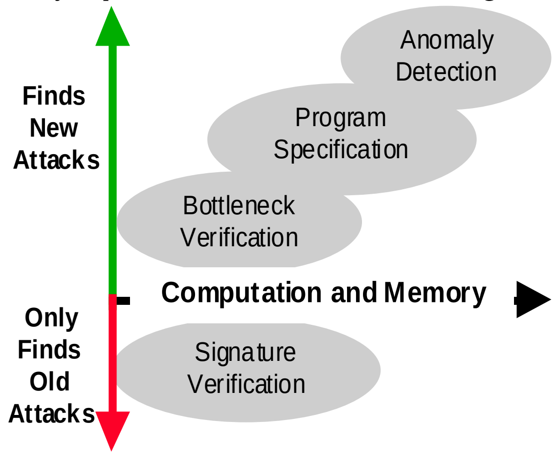

Fig. 1. Approaches to Intrusion Detection.

The most common approach to intrusion detection is "signature verification". Systems using this approach find only previously known attacks by looking for an invariant signature left by these attacks. Attack signatures can be found in audit records recorded on a victim machine, or in a stream of network packets captured by a "sniffer" that records packets traveling across the network. The Network Security Monitor (NSM) was an early signature-based intrusion detection system that searched for key-strings in network traffic captured using a sniffer. Early versions of the NSM [12][14] were the foundation for many government and commercial intrusion detection systems including NID [17] and NetRanger [5]. This type of system is popular because one sniffer can monitor traffic to many workstations without requiring any modification to those monitored workstations. In practice, these

[1] This work was sponsored by the Department of the Air Force under Air Force contract F19628-95-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

The authors design, develop and evaluate information assurance technology at MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420-9185.

systems can have high false alarm rates (e.g. 100's of false alarms per day on moderately loaded 10MB/s networks) because it is difficult to create signatures that successfully detect real attacks without creating false alarms for normal traffic. In addition, these signature-based systems must be updated frequently to detect newly discovered attacks, and these and other systems which rely on network sniffing can be defeated by user or network encryption which makes reconstruction of network sessions effectively impossible. Most current commercial systems, including NetRanger [5] and packages built upon Network Flight Recorder [25], include some form of signature verification. Recent research on systems which rely on signature verification include BRO [21] which uses network sniffer data and NSTAT [15] which uses audit information from one or more hosts.

Approaches shown in the upper half of Figure 1 can find novel attacks. This capability is essential to protect critical hosts because new attacks and attack variants are constantly being developed. Anomaly detection, shown in the upper right of Figure 1, is one of the most frequently suggested approaches to detect new attacks and attack variants. NIDES was one of the first statistical-based anomaly detection systems used to detect unusual user [13] and unusual program [2] behavior. The statistical component of NIDES forms a model of a user, system, or network activity during a training phase. After training, anomalies or departures from normal behavior are detected and are flagged as attacks. Anomaly detection is most useful if normal user or system behavior is repetitive and easily modeled and less useful when behaviors vary widely. When behavior is regular, this technique can discover attacks that rely on human interactions not observed on the computer system or network. Social engineering attacks, such as those in which an attacker tricks the victim into revealing passwords, can only be found using this method. Recent research on anomaly detection includes the development of EMERALD [22], which combines statistical anomaly detection from NIDES with signature verification, and the learning of audit record sequences [11]. Other research, motivated by the natural immune system, detects anomalous behavior of system programs by examining system calls and looking for unusual call sequences that didn't occur during normal training [9][10]. Finally, some researchers are beginning to use neural networks for anomaly detection [8][11][26] and to simultaneously model both normal behavior and known attack behavior.

Specification-based intrusion detection [16] is a second approach from the top half of Figure 1 that can be used to detect new attacks. It detects attacks that make improper use of system or application programs by using separately written security specifications that describe the normal intended behavior of programs. Host-based audit records are then monitored to detect behavior that violates the security specifications. Specification-based intrusion detection has the potential for providing very low false alarm rates when detecting a wide range of attacks including many forms of malicious code such as Trojan horses, viruses and attacks that take advantage of race conditions. Unfortunately, it has not become popular because security specifications must be written for all monitored programs. This is difficult because system and application programs are constantly updated, because all programs must be monitored for effective protection, and because many recent browser, mail, and word processing applications are extremely complex and are difficult to model. Specification-based intrusion detection is thus best applied to a small number of critical user or system programs that might be considered prime targets for exploitation.

The final approach to intrusion detection shown in Figure 1 is bottleneck verification. Bottleneck verification (BV) is designed to detect major security policy violations, without monitoring every system or application program. BV doesn't require specifications for all monitored programs as with specification-based approaches, nor signatures for attacks as with signature verification, nor a model of normal user behavior as with anomaly detection. It has very low computational and memory requirements. Bottleneck verification detects a user that transitions to a privileged state without going through the normal system bottlenecks used to permit this type of transition.

II. Bottleneck Verification

The bottleneck verification approach was motivated by careful examination of reconstructed telnet sessions, captured by network sniffers, of many actual attacks on government sites. It was found that the goal of many actual attacks on Unix systems and also of many exploits posted to web sites is to obtain an interactive shell running at the highest level of privilege. It was also noticed that, no matter which exploit was used to obtain such a privileged shell, evidence was present in the transcript that could be used to determine that a privileged shell had been created and that it had been created without following normal system procedures.

Figure 2 illustrates the BV approach to intrusion detection. This approach applies to well-designed operating systems where there are only a few legal "bottleneck" methods to transition from a lower privilege level (the lower, normal user states) to a higher privilege level (the upper, privileged user states) and where it is relatively easy to determine when a user is at a higher level. The key concept is to detect the use of legal bottleneck methods to transition to higher privilege levels and user activity indicative of a high privilege level. High privilege activity that arises in a session where a user did not pass through a bottleneck is indicative of an attack on the system. This approach can theoretically detect any novel attack that illegally transitions a user to a high privilege level, without prior knowledge of the attack mechanism. New attacks can thus be detected even when the attack mechanism is not understood.

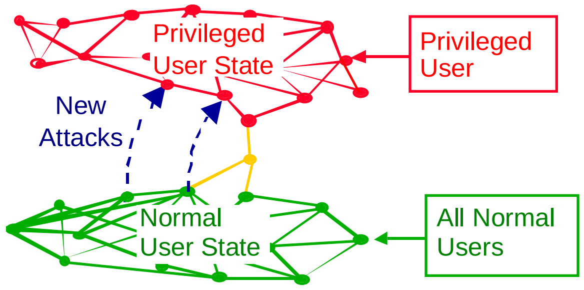

Fig. 2. System user state diagram (notional).

A recent addition to bottleneck verification, indicated on the right side of Figure 2, makes use of the observation that most security policies restrict the number of privileged users permitted to access a system. If a normal user appears to have super-user privilege, it is likely that a successful attack occurred prior to the current session.

Initial work on bottleneck verification has focused on detecting when users illegally obtain interactive privileged (root) shells on UNIX hosts by examining sniffing data or host audit data. The general approach, however, can be extended to other operating systems to determine whether users illegally access or modify data or illegally use specific application or system programs.

This approach was used to develop a successful (low false alarm rate, high detection rate) off-line implementation of a sniffer-based bottleneck verification intrusion detection system [18] and a real-time host-based version of the bottleneck verification intrusion detection system[7]. The following sections describe the design of Macroscope and the performance of the system on two different data sets.

III. MACROSCOPE SYSTEM DESIGN

Macroscope is a network-based intrusion detection system whose architecture is depicted in Figure 3. The name is chosen to stress the system's ability to perform critical forensic analysis for a user. Packet data is captured from a network by the NetTracker tool [26], which also records high-level packet statistics. Bottleneck Verification acts as a filter for telnet and rlogin sessions, recording in the database only those sessions that appear suspicious. A fixed set of queries and displays is provided by the RapIDisplay tool, which gives the user an easy way to view and report detected intrusions.

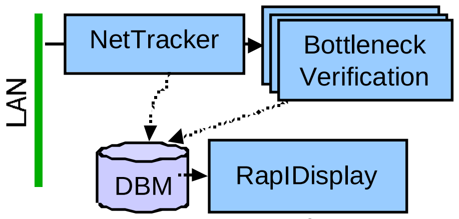

Fig. 3. System Components of Macroscope

A. *NetTracker*

NetTracker uses the libpcap libraries to acquire network data. Output is to database and analysis processes, such as the BV parser. For that parser, network traffic for an rlogin or telnet session is collected and assembled into two transcripts, one for each side of the duplex communication. While creating the transcripts, NetTracker strips out client and server control data for that protocol (e.g., telnet option negotiation), so that only actual data is included. Packet assembly is surprisingly difficult to do properly. There are many details to the TCP/IP protocol, and proper assembly requires a proper implementation of the TCP/IP stack. This is so difficult to do that many common operating systems do not have correct implementations of all aspects of the protocol. Attackers can perform insertion or evasion attacks that use implementation differences to make it difficult to reliably interpret network data [24]. To guard against some insertion attacks, NetTracker assumes that it can see both sides of a communication, and usually only records data when acknowledgements are available. Although this traffic pattern can be assured by careful architecture of a protected Intranet, many analysts report that pre-existing networks are often not configured this way. To address evasion attacks, we concentrate our analysis on the victim system's reply, which is more difficult to modify than the attacker's transmissions. Otherwise, we (incorrectly) assume that packet-assembly is performed as specified in RFC 793 [18].

NetTracker also records statistics for UDP and ICMP and reports these to other analysis algorithms not described here.

B. *Network-based Bottleneck Verification*

BV parses interactive telnet and rlogin sessions, recording in a database both attack transcripts and a parse-tree that helps RapIDisplay color-code and identify the exact line (or lines) on which an attack (or attacks) occurred.

The parser monitors transactions between the source and destination machines, detecting and identifying prompts, commands and their responses. Processes are marked to indicate privilege level, and the transitions are examined. For UNIX and Windows/NT systems, the only command that is authorized to effect the transition from regular user to super-user privileges is the su command. All other commands that effect such a change are an attack on the system. This style of context-dependent parsing reduces the amount of scanning required over previous keystring-matching detection systems that scan for a set of specific attacks, but the software required to perform this parsing is more complex.

There are many subtleties that make this parsing process difficult. First, the mapping from user input to system interpretation is many-to-one, and malicious users can exploit common tools to obscure their input. System output is more difficult to manipulate (at the application, transport, and network layers), so our parser examines the target host's replies. Usually (but not always) the process echoes commands back, to allow the user to verify that they were properly received and interpreted. The echoed commands, along with system and program output, appear in the reply transcript. Future versions of the system may use both streams to verify user actions and identify typing patterns, although since both streams are not always available, the ability to process just a single stream is often an advantage.

A greatly simplified state diagram for network-based Bottleneck Verification is depicted in Figure 4. When a new session starts, Bottleneck Verification enters the "init" state (top center of the diagram), where data structures are created and initialized. If the session starts normally, then Bottleneck Verification immediately begins looking for the login-related information and follows a successful login by scanning for post-login banner-like information (center). When a new session is created, the destination computer system rarely just prompts the user for input. Banners often occur before a login (to let visitors know about restrictions on using a system) and after a login (to alert users to important system information. This banner message is sometimes followed by mail messages or other news messages. The format for this information is not defined—in practice we see a range of short messages (e.g., "Routine Maintenance on Saturday—computers down from 8:00-10:00") to long messages where text has been turned into large banners. When these banners are examined on a line-by-line basis, each line has an odd assortment of characters and punctuation. These banners make it difficult to use simple models of English sentences to differentiate preset "messages of the day" from user-issued commands.

If the session started before Bottleneck Verification analysis, then BV immediately starts looking for a prompt, skipping the login and banner stages. Once a prompt is found, BV then backtracks to the first instance of that prompt and alternately examines commands (examples on the right side of the diagram), their outputs, and the next prompt. Some commands (e.g., su, telnet, rlogin) can start new processes, thereby restarting the init analysis. When an unexpected prompt is found or unknown command is executed, an analysis is performed to determine if an intrusion has occurred.

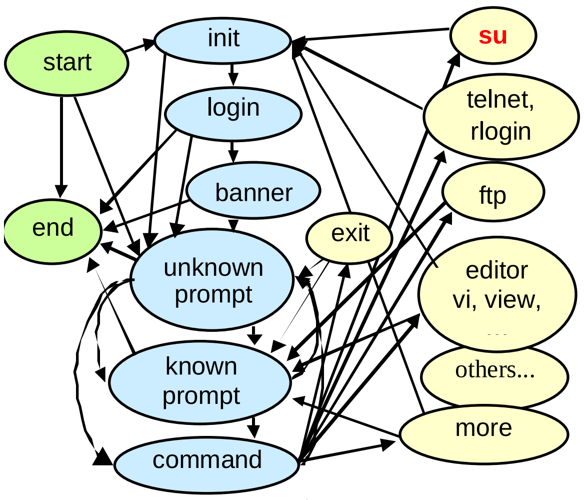

Fig. 4. State Diagram for Network-based BV

Amidst this stream of information, the first prompt itself is difficult to detect and identify. To detect prompts we use a number of heuristics: prompts usually end in one of ">:]%$#" followed by a space, they are often followed by a common UNIX command, and the line on which they appear usually takes some time to type. This is enough to rapidly parse most user sessions, but not experienced system administrators or clever hackers, so fallback prompt-identification techniques are also used. Among these are scans for common UNIX commands, combined with heuristics identifying and validating less strictly prompt-like preceding text. Finally, when the parser gets confused, the entire session is simply recorded for subsequent analysis by the analyst.

Even this is not enough to fully identify the command stream. Modern shells allow the user to set the prompt to be a portion of the path to the current directory. Changing directories (perhaps via cd, pushd, popd) can also change the prompt. Sometimes the new prompt is not easily guessed in advance, such as when one changes to a directory that is linked to another directory, or when one changes to a directory that the shell locates via the CDPATH environmental variable. When the prompt is lost, it is not enough to simply examine the next line of text, as type-ahead can blend multiple lines together.

Parsing the output of all varieties of all UNIX commands would be impossible. To reduce the amount of parsing code, we identify a variety of command classes. Some common commands, such as those that display the contents of a directory or a file, merely print information to the screen, then return the user to his previous prompt. Many commands are of this class, and all of these can be handled by a single, generic parsing routine. Other commands are more interactive, requiring additional feedback from the user. Commands that limit the amount of a file that is shown on a screen at one time (e.g., more, less) and simple editors (e.g., vi, vedit) are of this type. Many of these commands allow a user to escape to execute commands, including starting a new shell. These commands must be parsed, watching for the special escape commands to determine if a user just violated the security bottleneck. Each of these parsing routines has a common structure, but different escape commands. Another class of commands set default arguments to more common commands or wrap the common commands with additional functionality (e.g., crontab, vipw), allowing the command parser to consist of a small module that calls the parser of the core common commands. Still other commands are even more complicated, giving the user an entirely new environment (e.g., emacs). These commands are the most difficult to parse properly, because there are so many ways to interact with the system. The strategy here is the same as above: reduce the work by parsing the command and only the command escapes. By classifying most UNIX commands into one of the above types and having generic routines for each of the classes, the amount of commands that require specific parsing is reduced.

Although many users treat command shells as a way to enter a single command at a time, shells are in fact complex and powerful command interpreters that allow users to connect the output of one command to the input of another command, to execute multiple commands sequentially, or to establish an environment in which to execute future commands. Parsing commands requires understanding the most common multi-command line structures, and recording unrecognized commands for future human analysis.

The permission level at which the shell is running is determined by examining the prompt, the commands executed, and the output of a select few commands. Most shell prompts indicate the permission level of the user: in csh, for example, the default prompt for a regular user is “hostname %”, while the default prompt for a privileged user is “hostname #”. Other commands and command forms are restricted for use by only the super user. For example, the “mount disk/” command form can only be executed by the super-user. If this command executes, then it is clear that the user is the super-user. Some commands indicate who the user is: in Unix, “who am i” will respond with “root” if the user is the super-user. Once a hypothesis about the level that a shell is operating at is developed, the command history is examined to see if this level of permission was obtained legally. If it was not, a bottleneck violation has occurred.

C. RapIDisplay

The RapIDisplay intrusion detection analysis tool was designed after talking with intrusion detection analysts. It uses a network browser interface, with navigation via simple clicks on underlined elements. There are two primary screens with a common title bar to provide rapid access to documentation and report generation. At the overview screen, all attacks recently recorded by the system that the user has not examined are sorted based on the confidence that an attack occurred, the protocol displayed, and the time that the attack occurred. One line of data appears for each putative attack; the user can select each line to access forensic data.

After clicking on an attack line, a new screen is displayed that is separated into four regions. In the upper-leftmost is the name of the service, with a list of other recorded transcripts of this service. In the upper rightmost is a summary of the selected transcript, including information about the address of the attacker and victim machines, and the start and the duration of the session (up until the analyst looked at the transcript). In addition, a thermometer graph indicates confidence that an attack occurred. Along the left side of the screen are two other windows: a quick access window to allow the analyst to view both sides of a connection or to jump directly to the line of the putative attack, and a keystring window to support integration with keystring-based intrusion detection systems. In the lower right window is the transcript itself, color coded to indicate the prompts received and commands issued. Keystrings and attacks are highlighted to enhance visibility.

IV. System Performance

A. Accuracy on a Standard Corpus

In 1998 MIT Lincoln Laboratory created for DARPA the first large-scale realistic database that could be openly distributed and used to evaluate intrusion detection systems [6][20]. The full corpus was designed to evaluate both the false alarm rate and the detection rate of intrusion detection systems using many types of both known and new attacks embedded in a large amount of normal background traffic. Actual network traffic was generated to be similar to the type of traffic observed flowing between U.S. Air Force Bases and the publicly accessible Internet.

Traffic and attacks were generated using conventional Unix system and application programs by humans and automatic traffic generators on a network which simulated 1000’s of Unix hosts and 100’s of users using fewer than 20 actual machines. This network simulates the inside of an Air Force base connected through a Cisco router to outside machines on the Internet. The inside contains Linux, SunOS, and Solaris UNIX victim hosts and a gateway to 100's of simulated PC and Unix Workstation hosts. Most attacks were launched from outside workstations through the router against one or more of the inside victim workstations.

Attacks were divided into four categories with regard to their purpose. Since this algorithm defends against user-to-root attacks, we report our results on this class of attacks. The results presented in this section should be considered "unofficial," as some members of the intrusion detection team interacted with members of the evaluation corpus development team. Nevertheless, this data was processed only once following the formal evaluation procedures described in [20].

The results of an analysis of system accuracy are presented in Figure 5 for both BV and the naïve keystring IDS. For this corpus, BV detected 79% of the attacks at the rate of about one false alarm per day. By comparison, the baseline keystring system detected less than 5% of the attacks at this low false-alarm rate.

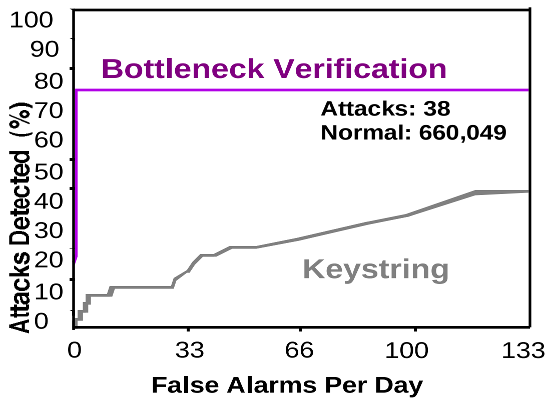

Fig. 5. Accuracy of Network-based Bottleneck Verification on 1998 DARPA IDS Data.

B. Accuracy on captured network data

The system has also been used for off-line analysis of approximately a hundred and forty thousand Internet sessions captured across three months at more than 100 different sites. Of these, Bottleneck Verification was used to scan the captured telnet and rlogin sessions from a wider variety of operating systems than were present in the standard corpus. These transcripts were formed using a different session reconstruction tool than NetTracker, and exhibited some character and word doubling. In addition, some transcripts did not capture a full session, starting or ending mid-way through a session. In these data, 67 illegal transitions to root were detected with fewer than one false alarm per site per day. Although it is difficult to compare results across differing data sets, a similar test was done using an earlier version of the NBBV engine [18]. The rewrite of the NBBV engine that allows for more careful processing of command-line applications (as described above) and handling of escape-to-shell commands reduced the false alarm rate and increased the detection rate.

V. USING NETWORK-BASED BOTTLENECK VERIFICATION AS PART OF A SECURITY SYSTEM

In addition to the many security actions that a system administrator should perform (e.g., patching the OS with security fixes, maintaining a firewall between the protected network and the Internet), there are a few configuration choices can be made to enhance the performance of Macroscope.

First, one can increase the accuracy and speed of the parser. For command prompts, a system administrator should either use the shell defaults, or make the prompts easier to locate. Most shells access a system-wide file to set the prompt: for sh, the file is /etc/profile, for csh the file is /etc/.login. Consider using these files to set the defaults to something clear like "`<user>@<host> <dir>%` " for a regular user, or "`<user>@<host> <dir>#` " for the super-user. Although an individual can change his own prompt, the overall work for the parser will be diminished. If an organization has access to the source code for the shells that it uses, then all shells with super-user privilege could be changed to have exactly the same prompt to clearly indicate when a super-user is accessing the system.

By further modifying the login configuration and BV, an even more secure environment can be achieved. If the same string is used to indicate the end of the login process, then BV can know for certain where the system login is complete and user interaction starts. A message that indicates the command shell is particularly useful, e.g. "`Welcome to $HOST using the $SHELL shell`".

VI. FUTURE WORK

Unfortunately, there continue to be many ways to elude the system. In this section a few evasion techniques are described.

Since Macroscope only monitors a few services, it is vulnerable to attacks that occur on other services. Modern computers exchange information via a wide variety of services (e.g., time, telnet) using different protocols (e.g., HTTP, TCP, UDP). Most services were designed to support a limited set of operations transmitted via a single protocol, although an attacker can cause some services to operate over different protocols. For example, in the 1998 DARPA Intrusion Detection Evaluation, an attack was developed using hypertext transfer protocol [2] that allowed arbitrary execution of shell commands on a remote machine. This capability is similar to what is commonly provided by the telnet service over Transmission Control Protocol (TCP) [5][14]. In order to set up this attack or other similar attacks that tunnel a service over an unusual protocol, the attacker must at least have access to the attacked system. To obtain this, one must either attack the system or be granted access to the system. If access is via a user-to-super-user attack, then Bottleneck Verification may find the precursor attack. However, once the attacker is granted access, then methods other than Bottleneck Verification must be used to find the attacker.

For the first version of Macroscope, it is assumed that shell-level access is obtained via the rlogin or telnet services operating over the TCP, but we recognize that this assumption could be incorrect, either because the attacker uses different services to access the system, or because the attacker never obtains a super-user shell.

VII. Conclusions

Macroscope rapidly and efficiently detects novel attacks at exceedingly low false alarm rates. It has been tested using the 1998 DARPA Intrusion Detection data for which it detected nearly 80% of the user-to-super-user attacks against a variety of UNIX systems, while only producing a single false alarm per day. It has also been used to analyze almost 140,000 previously recorded transcripts, with similar results. Future work will focus on extending Macroscope to find other abuses of privilege and to find more stealthy attacks.

VIII. References

[1] “Axent Intruder Alert User Manual,” Version 3.0, October 1998.

[2] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. "Safeguard final report: detecting unusual program behavior using the NIDES statistical component," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, December 1993.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0,” RFC 1945, 1996.

[4] M. Bishop,, S. Cheung, C. Wee. "The Threat from the Net", IEEE Spectrum, 1997, 38(8).

[5] Cisco Systems, Inc. "NetRanger Intrusion Detection System Technical Overview, ”http://www.cisco.com/warp/public/778/security/netranger/ntran_tc.htm, 1998.

[6] R. Cunningham, R. Lippmann, D. Fried, S. Garfinkel, I. Graf, K. Kendall, S. Webster, D. Wyschogrod, and M. Zissman, "Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation," in Proceedings of ID'99, Third Conference and Workshop on Intrusion Detection and Response, San Diego, CA: SANS Institute, 1999.

[7] R. Cunningham, R. Lippmann, D. Kassay, S. Webster, and M. Zissman, Host-based Bottleneck Verification Efficiently Detects Novel Computer Attacks,” MILCOM’99, November 1999.

[8] H. Debar, M. Becker, and D. Siboni, “A Neural Network Component for an Intrusion Detection System,” in Proc. of IEEE Computer Society Symposium on Research in Security and Privacy, 1992.

[9] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, "A sense of self for Unix processes," in Proceedings of 1996 IEEE Symposium on Computer Security and Privacy, 1996.

[10] S. Forrest, S. Hofmeyr, and A. Somayaji, "Computer Immunology," Communications of the ACM, 40(10), 88-96, 1997.

[11] A. K. Ghosh, A. Schwartzbard and M. Shatz, “Learning Program Behavior Profiles for Intrusion Detection”, in *Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, April 1999, http://www.rstcorp.com/~anup/.

[12] T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor", in IEEE Symposium on Research in Security and Privacy., 1990, pp. 296-304.

[13] H. Javitz, and A. Valdes, "The NIDES statistical component description and justification," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, March 1994.

[14] T. Heberlein, "Network Security Monitor (NSM) - Final Report”, U.C. Davis: Feb. 1995, http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf.

[15] R. Kemmerer. "NSTAT: A Model-based real-time network intrusion detection system," Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, http://www.cs.ucsb.edu/TRs/TRCS97-18.html.

[16] C. Ko, M. Ruschitzka, and K. Levitt, "Execution Monitoring of Security-Critical Programs in a Distributed System: A Specification-Based Approach," in Proc. IEEE Symposium on Security and Privacy, 1997, pp. 134-144, Oakland, CA: IEEE Computer Society Press.

[17] Lawrence Livermore National Laboratory (1998). "Network Intrusion Detector (NID) Overview," Computer Security Technology Center, http://ciac.llnl.gov/cstc/nid/intro.html.

[18] Lippmann, R.P., et al. “Using Bottleneck Verification to Find Novel New Attacks with a Low False Alarm Rate,” in Recent Advances in Intrusion Detection, 1998, Louvain-la-Neuve, Belgium.

[19] T. Lunt,, "Automated Audit Trail Analysis and Intrusion Detection: A Survey", in Proceedings 11th National Computer Security Conference., 1998, pp. 65-73.

[20] MIT Lincoln Laboratory, “Intrusion detection evaluations,” http://www.ll.mit.edu/IST/ideval/index.html.

[21] V. Paxon, "Bro: A System for Detecting Network Intruders in Real-Time," in Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998, http://www.aciri.org/vern/papers.html.

[22] P. Porras, and P. Neumann, "EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances," in Proceedings 20th National Information Systems Security Conference, Oct 7, 1997.

[23] J. Postel, “Transmission Control Protocol,” RFC 793, USC/Information Sciences Institute September 1981.

[24] T. Ptacek, and T. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection,” Secure Networks, Inc. January, 1998.

[25] M. Ranum, K. Landfield, M. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall. “Implementing A Generalized Tool For Network Monitoring”, Eleventh System Administrators Conference (LISA), 1997.

[26] J. Ryan, L. Meng-Jang, and R. Miikkulainen, "Intrusion Detection with Neural Nets," in Advances in Neural Information Processing Systems 10, Edited by M. Jordan, M. Kearns, and S. Solla, MIT Press: Cambridge, MA, 1998, pp. 943-94

[27] S. Webster, “The Development and Analysis of Intrusion Detection Algorithms,” Masters Thesis in Computer Science, Massachusetts Institute of Technology: Cambridge, MA, June 1998.