C. Kruegel, W. Robertson, and G. Vigna

Using Alert Verification to Identify Successful Intrusion Attempts

Christopher Kruegel is an Assistant Professor with the Automation Systems Group at the Technical University Vienna. Before that, he was working as a research post-doc for the Reliable Software Group at the University of California, Santa Barbara. He received his PhD with honors in computer science from the Technical University Vienna while working as a research assistant for the Distributed Systems Group. His research interests include most aspects of computer security, with an emphasis on network security, intrusion detection and vulnerability analysis.

William Robertson is a PhD student with the Reliable Software Group at UC Santa Barbara. His research interests include static analysis, vulnerability analysis, reverse engineering, and system hardening.

Giovanni Vigna is an Associate Professor in the Department of Computer Science at the University of California in Santa Barbara. His current research interests include intrusion detection, security of mobile code systems, vulnerability analysis, and wireless systems. In particular, he worked on developing frameworks for the modular development of both misuse-based and anomaly-based intrusion detection systems. Giovanni Vigna received his M.S. with honors and Ph.D. from Politecnico di Milano, Italy, in 1994 and 1998, respectively.

ABSTRACT

Intrusion detection systems monitor protected networks and attempt to identify evidence of malicious activity. When an attack is detected, an alert is produced, and, possibly, a countermeasure is executed. A perfect intrusion detection system would be able to identify all the attacks without raising any false alarms. In addition, a countermeasure would be executed only when an attack is actually successful. Unfortunately false alarms are commonplace in intrusion detection systems, and perfectly benign events are interpreted as malicious. In addition, non-relevant alerts are also common. These are alerts associated with attacks that were not successful. Such alerts should be tagged appropriately so that their priority can be lowered.

The process of identifying alerts associated with successful attacks is called alert verification. This paper describes the different issues involved in alert verification and presents a tool that performs real-time verification of attacks detected by an intrusion detection system. The experimental evaluation of the tool shows that verification can dramatically reduce both false and non-relevant alerts.

1 INTRODUCTION

The goal of intrusion detection systems (IDSs) is to identify evidence of malicious activity. This is done by analyzing one or more streams of input events, using different techniques.

Events can be of different nature and level of granularity. For example, they may represent network packets, operating system calls, audit records produced by the operating system auditing facilities, or log messages produced by applications.

One way to characterize IDSs is according to the type of events they analyze. Network-based IDSs are systems that analyze network traffic, application-based IDSs are systems that operate on the logs produced by specific applications, and host-based IDSs analyze events produced by the operating system.

Event streams are used by intrusion detection systems in two different ways, according to two different paradigms: anomaly detection and misuse detection. In anomaly detection systems [JV94, KRL97, GWC98, WFP99], historical data about a system's activity and specifications of the intended behavior of users and applications are used to build a profile of the "normal" operation of the system. Then, the intrusion detection system tries to identify patterns of activity that deviate from the defined profile. Misuse detection systems take a complementary approach [NFR01, Pa98, Ro99, NP99, VVK03, IS04]. Misuse detection systems are equipped with a number of attack descriptions (or "signatures") that are matched against the stream of audit data looking for evidence that the modeled attack is occurring. Misuse and anomaly detection both have advantages and disadvantages. Misuse detection systems can perform focused analysis of the audit data and they usually produce a limited amount of false positives (which are erroneous detection), but they can detect only those attacks that have been modeled. Anomaly detection systems have the advantage of being able to detect previously unknown attacks. This advantage is paid for in terms of the large number of false positives and the difficulty of training a system with respect to a very dynamic environment.

Independent of the type of event stream analyzed or the techniques used to identify malicious activity, every IDS produces as output intrusion alerts. An intrusion alert is a report that describes what malicious activity was identified and, often, additional information such as who are the attacker(s) and the victim(s). The information contained in an alert may change depending on the type of intrusion detection technique adopted and the type of event stream analyzed. For example, if an attack has been detected by analyzing the audit information produced by an operating system, it is likely that the alert will contain the process identifier (PID) of the attacking process or the identifier of a file that was involved in the attack. As another example, if the attack was detected by analyzing network traffic, the alert may contain the ports and IP addresses used on both the attacker's and the victim's side. The particular technique used to identify intrusions also influences the information that is included in an alert. For example, while a misuse-based system might provide a precise characterization of an attack (e.g., "Nachia Worm Scan"), anomaly detection systems are usually less precise and they normally provide an assessment of how unusual certain activity is (e.g., "This sequence of system calls has never been used before by this application").

The intrusion detection community soon recognized the problem of providing a common alert format and an Internet Engineering Task Force (IETF) working group, named the Intrusion Detection Working Group (IDWG) was created. The Intrusion Detection Working Group defined data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to management systems which may need to interact with them. The main outcome was the Intrusion Detection Message Exchange Format [CD03] data model (IDMEF). IDMEF provides a standard representation for intrusion alerts. This representation defines the syntax of an alert and specifies the semantics of certain attributes.

However, the IDMEF effort is mostly concerned with syntactic rules. It is possible, and common, that sensor implementers choose different names for the same attack[1], provide incomplete information for certain fields, or choose to include additional fields to store relevant data. As a result, similar information can be labeled differently or can be included in different fields. Because there is no specification for the content of most fields, they can be filled with meaningless strings (most commonly, "unknown" is used). To remedy this shortcoming, the intrusion detection community would benefit greatly from a shared alert model that extends the current IDMEF work with semantic information and a common attack-naming scheme.

In a perfect world, intrusion detection systems would be able to correctly detect and identify all the attacks whose evidence appears in the analyzed event stream. Moreover, IDSs would be able to do so without making any mistakes, that is, without producing any false positives, and, in addition, the alerts produced would be in a format that allows for interoperation between different intrusion detection systems.

Unfortunately, intrusion detection systems have been increasingly brought to task for failing to meet the expectations that researchers and vendors were raising. Promises that IDSs would be capable of reliably identifying malicious activity never turned into reality. While virus scanners and firewalls have visible benefits and remain virtually unnoticed during normal operation, intrusion detection systems are known for producing a large number of alerts that are either not related to malicious activity or not representative of a successful attack (non-relevant positives). Although tuning and proper configuration may eliminate the most obvious spurious alerts, the problem of the vast imbalance between actual and false or non-relevant alerts remains.

One problem is the fact that intrusion detection systems are often run without any (or very limited) information of the network resources that they protect. Marty Roesch, the developer of Snort [Sno, Ro99], routinely brings up this point in his whitepaper [RNA] and posts to security mailing lists [Sec] and calls for an IDS that possesses knowledge of the network components it defends. The classic example that Marty uses is the scenario of a Code Red attack that targets a Linux web server. It is a valid attack that is seen on the network, however, the alert that an IDS raises is of no use because the service is not vulnerable (as Code Red can only exploit vulnerabilities in Microsoft's IIS web server). To mitigate this problem, Roesch introduces a concept called RNA, real-time network awareness [RNA]. RNA is based on passive network monitoring to establish an overview of the hosts and services that are being protected. This overview contains enough contextual information to distinguish between Linux and Windows servers, thus enabling a "network-aware" IDS to discard a Code Red attack against a Linux machine.

The problem is that the concept of network-awareness is not broad enough to completely capture the complexity that is at the core of excessive amounts of false alarms. When a sensor outputs an alert, there are three possibilities.

1. The sensor has correctly identified a successful attack. This alert is most likely relevant (i.e., a true positive).
2. The sensor has correctly identified an attack, but the attack failed to meet its objectives (i.e., non-relevant positive).
3. The sensor incorrectly identified an event as an attack. The alert represents incorrect information (i.e., a false positive).

Most people/sites are only interested in type-1 alerts, that is, correct detections. Although some sites might be interested in failed attack attempts (type-2), the corresponding alert should be differentiated from a successful instance. The key idea of alert verification is to distinguish between successful and failed intrusion attempts (both false and non-relevant positives). While contextual information can be helpful to perform this distinction (as we have seen in the example with the Code Red worm above), it is not always sufficient. Consider a Code Red worm attacking a patched Microsoft IIS server. In this case, it is not enough to know which operating system the host is using, but it is also required to know which application is running and which patches have been applied.

Alert verification is a term that we use for all mechanisms that can help to determine whether an attack was successful or not. This information is passed to the intrusion detection system to help differentiate between type-1 alerts and type-2/type-3 alerts. When the success of an attack is a priori impossible (e.g., no vulnerable service is running) or the verification process determines that the attack has not been successful (e.g., because incorrect buffer overflow offsets were used), the IDS can react accordingly and suppress the alert or reduce its priority.

The next section classifies different mechanisms to implement alert verification. In Section 3, we present our implementation,

[1] There is no mandatory naming convention, and currently a few different competing schemes such CVE [CVE] or Bugtraq IDs [Bug] exist.

which is based on Nessus [Nes] and Snort [Sno]. With this configuration, we demonstrate how Snort, an open-source network intrusion detection system, was modified to utilize information provided by Nessus, a popular vulnerability scanner, to significantly improve Snort's detection accuracy. Section 4 gives more details on our experience with the deployed tool. Section 5 discusses related work and potential areas where the presented system could be applied. Section 6 concludes and outlines future work.

2 ALERT VERIFICATION

Alert verification is defined as the process of verifying the success of attacks. That is, given an attack (and a corresponding alert raised by an intrusion detection system), it is the task of the alert verification process to determine whether this attack has succeeded or not.

There are different techniques that can be used to perform this verification. One possibility is to compare the configuration of the victim machine (e.g., operating system, running services, service version) to the requirements for a successful attack. When the victim is not vulnerable to a particular attack (because the configuration does not satisfy the attack requirements), then the alert can be tagged as failed. For example, a certain exploit might require that the victim is running a vulnerable version of a Microsoft IIS server. When the victim's configuration shows that it is running an Apache server on Linux, the exploit cannot possibly succeed.

Another possibility is to model the expected "outcome" of attacks. The "outcome" describes the visible and verifiable traces that a certain attack leaves at a host or on the network (e.g., a temporary file or an outgoing network connection). When an alert has to be verified, the system can check for these traces.

An important distinction between different alert verification mechanisms is whether they are active or passive. Active verification mechanisms are defined as mechanisms that gather configuration data or forensic traces after an alert occurs. Passive mechanisms, on the other hand, gather configuration data once (or at regular, scheduled intervals) and have data available before the attack occurs. Both active and passive techniques can be used to check attack requirements against victim configurations. To check for traces that might be left after an attack, only active mechanisms can be employed. Note that the distinction between active and passive mechanisms is solely based on the point in time when the configuration or forensic data is collected. Passive mechanisms collect data about a protected network before an alert is received, while active mechanisms perform verification in real-time, as a reaction to a received alert.

The most important requirement for the alert verification process is accuracy. An accurate verification process will significantly reduce the number of both false negatives (i.e., alerts that are marked as non-relevant, when in fact they are) and false positives (i.e., alerts that are marked as relevant, although they are not). There are different factors that influence accuracy. One factor is the quality of the data that is gathered. Another factor is its timeliness. Both factors are critical; it is not sufficient to have high quality data that is out-of-date, but it is also unsatisfactory when incorrect data is collected, even though the data is collected frequently.

Another requirement is to keep the cost of the verification process low, where cost is measured along two axes. One axis reflects the cost of deploying and maintaining the alert verification system. The other axis reflects the costs of impact of the verification process on the normal operation of the network. This cost includes whether it necessary to shut down regular network operations to perform alert verification, or whether the alert verification process has adverse effects on the running services.

In the following, we describe the different ways to verify the success of attacks in more detail, and highlight the corresponding advantages and disadvantages. Note that the following description presents each approach separately. However, it is possible to combine techniques to compensate for drawbacks of individual techniques and to combine their advantages.

2.1 Passive Verification

As mentioned above, passive verification mechanisms depend on a priori gathered information about the hosts, the network topology, and the installed services. A description of the network installation is required and can be, for example, specified in a formal model such as M2D2 [MMDD02] or using hypergraphs [Vi03].

Given an alert, it is possible to verify whether the target of the attack exists and whether a (potentially vulnerable) service is running. For remote attacks, it is also possible to check whether malicious packets can possibly reach the target, given the network topology and the firewall rule configuration. Also it is possible to verify whether the target host reassembles the packets as expected by the intruder (e.g., using the tool by Shankar and Paxson [SP03]). The real-time network awareness approach advocated by Marty Roesch [RNA] would also fall into this class.

One advantage of passive mechanisms is that they do not interfere with the normal operation of the network. In addition, passive mechanisms do not require additional tests that delay the notification of administrators or the start of active countermeasures. On the other hand, passive mechanisms have the disadvantage that they may rely on outdated data. New services might have been installed or the firewall rules might have been changed without updating the knowledge base. This can lead to attacks that are tagged as non-relevant, even though a vulnerable target actually exists. Another disadvantage is the limitation of the type of information that can be gathered in advance. When the signature of an attack is matched against a packet sent to a vulnerable target, the attack could still fail for a number of other reasons (e.g., incorrect offset for a buffer overflow exploit). To increase the confidence in verification results, it is often required to actively check audit data recorded at the victim machine or other type of data that may provide conclusive evidence about the effectiveness of an attack.

2.2 Active Verification

Active alert verification mechanisms do not rely on a priori gathered information. Instead, the verification process actively initiates the information gathering process when an alert is received. This information-gathering process can check the current configuration of the victim host (see Section 2.2.1), or scan for attack traces (see Section 2.2.2 and Section 2.2.3).

2.2.1 Active Verification with Remote Access

Mechanisms in this group require that a network connection be established to the victim machine. One active verification mechanism with remote access is based on the use of vulnerability scanners. A vulnerability scanner is a program specifically designed to search a given target (piece of software, computer, network, etc.) for weaknesses. The scanner systematically engages the target in an attempt to assess where the target is vulnerable to certain known attacks. When an attack has been detected, a scanner can be used to check for the vulnerability that this attack attempts to exploit. Note that a vulnerability scanner could also be used in a passive setup. In this case, the full range of scans would be run in advance (or at regular intervals).

A network connection permits scanning of the attack target and allows one to assess whether a target service is still responding or whether it has become unresponsive. It also enables the alert verification system to check whether unknown ports accept connections, which could represent evidence that a back door is installed. In this case, however, care must be taken to prevent false positives that stem from dynamically allocated ports. To this end, one could use blacklists of well-known backdoor ports, white-lists that specify port ranges for well-known applications (e.g., X servers), or service fingerprinting (such as the one recently added to nmap [Fy]) to detect legitimate applications. Also, the active verification system can keep a list of applications that were found running during the last scan and raise an alert when this list changes.

Active alert verification has the advantage that the information gathered to verify an attack's outcome is current. This allows one to assess the status of the target host and the attacked service in a more reliable way when compared to passive verification techniques. In particular, it is possible to recognize changes at the victim host that might serve as an indication of an attack.

Although the information is current, however, it might not be completely accurate. One has to consider that a vulnerability scanner can also have false positives and false negatives. When an alert is verified, if the vulnerability scanner determines that the service is vulnerable when in fact it is not, the alert is simply reported by the IDS. In this case, the alert is a false positive (because the service is not vulnerable) and the verification mechanism has failed. However, the security of the system is not affected, and without verification, the alert would have been reported as well. A more significant problem is false negatives. In this case, a valid alert is suppressed because the vulnerability scanner determines that the target is not vulnerable when, in fact, it is. Although such a scenario is very undesirable, it is not very likely to occur frequently. The reason is that a vulnerability scanner actually launches a basic instance of the attack. When this attack fails, it is very improbable that a more sophisticated instance succeeds.

Another drawback is the fact that active actions are visible on the network and it is possible that scanning has an adverse effect on the hosts of the protected network. For example, port scanning consumes network bandwidth and resources at the scanned host. To minimize the impact on an operational network, results can be cached for some time. This is especially important when an intruder runs scripts that repeat the same attack with different parameters. Note, however, that caching involves a trade-off between resource usage and accuracy. When results are cached for too long, the advantage of active verification is reduced. As scans are only initiated on a per-alert base, it is not necessary to run all tests that a vulnerability scanner includes, but at most a single one for each alert (minus those for which cached results are available).

In addition, tests run by a vulnerability scanner might crash a service. More precisely, a vulnerability scanner can perform tests in a non-intrusive or in an intrusive manner. When running non-intrusive tests, no vulnerability is actually exploited, but inferred from the type and version of a running service (e.g., by analyzing the service's banner information). When running an intrusive test, the vulnerability is actually exploited. While this approach delivers more accurate results, it might results in the crash or disruption of the service being tested. Sometimes, the crash of a service process can be tolerated, for example, when the service is implemented using multiple threads (such as Apache's thread pool). In this case, the failure of a single thread does not have a negative impact, because other threads are still available to serve further requests. In addition, the failed thread is automatically restarted after a short period of time. On the other hand, when the crash of a service process interrupts the whole service, then the corresponding test should be excluded altogether from the active verification process. This also helps to prevent attacks that exploit the verification process itself. More specifically, an attacker may attempt to trigger an alert to have the alert verification system check the validity of the attack and, as a consequence, crash the service. The problem of selecting the appropriate tests is a result of the trade-off between the goal of getting accurate results and the goal of having minimal impact on the operational network. While intrusive tests are more reliable in obtaining proper results, the risk of affecting services in a negative way is greater.

Note that the alert verification mechanism should only be used to check alerts raised by packets that can possibly reach their destination. That is, the intrusion detection system (together with the alert verification system) should be located behind a firewall. This makes sure that only relevant packets are scanned for attacks by the IDS and are later checked by the verification mechanisms. Otherwise, an attacker could potentially bypass the firewall and launch attacks by means of the alert verification system itself.

The scope of remote scans is also limited, in that the identification of some evidence associated with an attack might require local access to the victim machine. In addition, one has to make sure that the alerts generated in response to the activity of the vulnerability scanner are excluded from the correlation process.

2.2.2 Active Verification with Authenticated Access

Mechanisms in this group gather evidence about the result of an attack using authenticated access to the victim host. The difference with respect to the previous group of techniques is the fact that the alert verification system presents authentication credentials to the target host.

Active verification with authenticated access can be implemented by creating dedicated user accounts with appropriate privilege settings at the target machines. The alert verification system can then remotely log in and execute scripts or system commands. This allows one to monitor the integrity of system files (e.g., the password file or system-specific binaries) or check for well-known files that are created by attacks (e.g., executable files left by worms). In addition, programs that retrieve interesting forensic data such as open network connections, open files, or running processes can also be invoked.

The advantage of mechanisms in this group is the access to high-quality data gathered directly from a target machine. One downside is the need to configure each machine for authenticated remote access. This might be cumbersome in large network installations or when hosts with several different operating systems are used. On the other hand, in large networks, such accounts may already exist for maintenance purposes and can be also leveraged for gathering forensic evidence. Another problem with this approach is that the information provided by general user-space tools might not be as complete and accurate as it is possible with specialized tools. For example, kernel-level tools may provide much more detailed information with respect to other monitoring tools such as netstat, lsof, or ps.

2.2.3 Active Verification with Dedicated Sensor Support

Mechanisms in this group require, in addition to authenticated access, special auditing support installed at the target machines. This auditing support can be provided by operating system extensions or special purpose tools, such as host-based intrusion detection systems. The difference between using standard tools and relying on dedicated sensors is that standard tools are common in most distributions. In addition, dedicated sensors often need complex configuration.

Dedicated sensors can be used to monitor system calls issued by user applications. This allows one to check for the spawning of suspicious processes (e.g., shell invocations) or for accesses to critical files (e.g., the inetd.conf file). In addition, auditing facilities can keep a record of malicious activity, while standard monitoring tools provide only a snapshot of the system. Therefore, monitoring mechanisms provide the verification system with access to events that are only visible for a short period of time, which could be missed by a snapshot.

The advantage of dedicated sensor support is the ability to provide the most detailed and accurate audit records. The drawback is the effort required to install and configure these sensors, and the fact that certain sensors are not available for all platforms.

2.2.4 General Issues in Active Verification

One issue that affects all active verification mechanisms is the problem that information is gathered directly from the victim machine. It can be argued that an attacker can tamper with the compromised system to eliminate suspicious traces or, at least, hide her activity from the auditing system. This is particularly true when the information is gathered remotely (e.g., using a vulnerability scanner).

There are different approaches to addressing this problem. One possibility is to operate in a best-effort mode and attempt to scan the potential victim host as fast as possible after the alert is received. This, of course, offers a small window of vulnerability that can be exploited by the attacker. A more secure option is to delay packets that have raised an alert until the verification mechanism has finished. This makes sure that the victim host has not been compromised by this attack, but it requires an in-line intrusion detection system.

Another option can be used when data is directly gathered on the victim machines via scripts or dedicated sensors. Here, audit tools should be run at least with privileges that require administrative (i.e., root) access to be turned off. By doing this, the integrity of the sensor is preserved even if the intruder obtains access or manages to crash a service. In this case, the sensors operate in a best-effort mode and deliver accurate results as long as possible. Also, simply disabling auditing is a suspicious action by itself. A more secure option is the use of a more restrictive access control system such as LIDS [LID] or Security-Enhanced Linux [LS01]. These systems can prevent the administrator from interfering with the audit facility, so that physical access to the machine is required to change or disable security settings.

3 IMPLEMENTATION

After the general discussion about various alert verification mechanisms in the previous section, the remainder of the paper presents the implementation and the evaluation of our verification tool. The tool implements active verification mechanisms with remote access and active verification mechanisms with authenticated access. The system is realized as an extension to Snort [Sno, Ro99] and can be downloaded at [SAV].

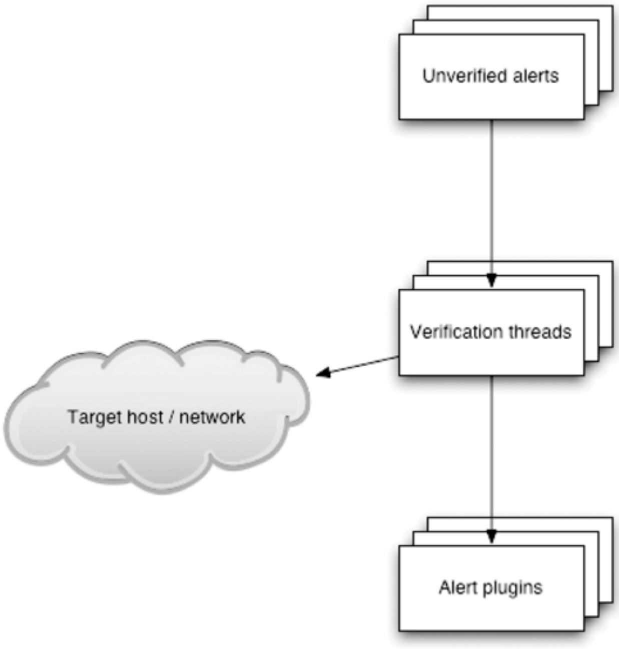

Fig. 1 Verification System Architecture

The alert verification tool consists of an addition to Snort's alert-processing pipeline. The alerts produced by Snort are queued for processing by a pool of verification threads. This design allows Snort to continue processing events while alert verification takes place in the background. An overview of the architecture of the current implementation is depicted in Fig. 1. In addition, the Snort rule language was extended to include new keywords to perform forensic checks at the target hosts.

Because our verification system is implemented as a part of the Snort sensor, the same process performs both the intrusion detection analysis and the verification process. However, this is not a requirement of active verification, and it would also be possible to have a separate verification system that receives alerts from multiple sensors. In this case, the alert verification tool could be integrated into a centralized alert collection framework.

Note that, because we implemented the verification process as a module of the Snort sensor, if multiple Snort sensors are used then multiple verification processes will be executed for attacks that are detected by more than one sensor. We anticipated that this would not be a problem, because the performance impact of the verification tool is low.

3.1 Active Verification with Remote Access

The component that performs active verification with remote access relies on NASL [Ar02] scripts written for the Nessus [Nes] vulnerability scanner. More precisely, the component is implemented as a patch to Snort, which integrates the Nessus vulnerability scanner into Snort's core to perform verification of alerts. Nessus was chosen as a verification mechanism because of the high quality of its vulnerability checks, its minimal impact on production networks, and the ease with which it could be integrated into Snort.

For each alert that is processed by a verification thread, the corresponding Common Vulnerabilities and Exposures identifier [CVE] is extracted and used as an index into Nessus' collection of NASL scripts. NASL is the scripting language designed for the Nessus security scanner. Its aim is to allow one to easily and quickly write plug-ins to test for security holes.

When an appropriate NASL script is found, the script is executed by an embedded NASL interpreter against the victim host or network identified by the alert. The vulnerable status of the target is extracted from the NASL interpreter's output and is used to flag the detected attack as either successful or unsuccessful. The alert is then queued for output by subsequent alert plug-ins that have been enabled in Snort. The result of each verification is also cached for a period of time in order to reduce load on the network. When no appropriate NASL script is found, the alert is flagged as undetermined.

3.2 Active Verification with Authenticated Access

The component that is responsible for active verification with authenticated access performs checks for the "known" outcome (i.e., evidence) of an attack at the target host. To this end, Snort rules can be augmented with simple rule extensions that specify forensic evidence to be looked for on the victim host or network. In the current version, the following extensions have been implemented.

- It can be checked whether a certain file exists (or does not exist) in the victim host's file system. For example, this extension can be used to verify the files that are often left by worms in well-known places.
- It can be checked whether a process with a certain name is running (or not running) on the target machine. For example, this can be used to detect the crash of a particular network service or the existence of a suspicious process.
- The content of a file can be checked for the occurrence of a certain pattern (defined as a regular expression). This can, for example, be used to assess whether a certain entry is present in a log file.

Consider the following example of a Snort rule with extensions:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443
(content:"TERM=xterm"; flow:to_server,established; nocase;
host_file_exists:/tmp/.uubugtraq;)
```

This rule augments the standard Snort rule for the Linux Slapper worm with a check for the existence of the worm executable (/tmp/.uubugtraq) on the target host, using the keyword host_file_exists. If the specified file exists, the attack was successful and the corresponding alert is tagged appropriately.

Whenever a Snort rule extended with forensic specifications triggers, the verification thread consults the Snort configuration file to check whether the target of the attack has been setup for authenticated access. If no authenticated access has been prepared, then the alert is tagged as undetermined. Otherwise, the verification system logs into the target machine and performs the necessary checks. When all host-based checks are successful (i.e., evidence of the attack is found) then the alert is tagged as successful, otherwise it is tagged as unsuccessful. In the current system, we have implemented two modules that can establish remote access to Unix and Windows machines via secure shell and run the appropriate commands there. However, it is straightforward to add modules that can run the necessary remote tests via different mechanisms (e.g., Windows Terminal Services).

3.3 Alert Post-processing

The verification subsystem marks each alert as successful, unsuccessful, or undetermined. Post-processing systems (e.g., alert correlation engines or system administrator scripts) can then utilize this additional information when performing their analysis upon the alert stream generated by verification-enabled Snort sensors.

4 EVALUATION

The current implementation of our alert verification tool has been evaluated on an experimental test bed with regards to its effectiveness in reducing Snort's false alarm rate. Three machines were present on this test bed:

1. An attacker machine,
2. a target machine (running RedHat Linux 7.0), and
3. a machine with a Snort sensor extended with the alert verification tool.

A variety of known vulnerabilities were introduced on the target machine, and corresponding signatures to detect attacks using these vulnerabilities were enabled on the sensor machine.

In the first experiment, the alert verification process was run in "active alert verification with remote access" mode. That is, we did not configure the target machine such that checks directly at the host could be performed. The reason was that we wanted to obtain a baseline for the verification effectiveness in a conservative setup where only remote access is available.

A wide range of attacks was then run against the target by the attacker. The attacks were run twice, once with alert verification enabled and once with alert verification disabled, to compare the number of false positives produced by Snort. Attack traffic was generated from a mix of Nessus runs and publicly available exploits. The results are shown in Table 1.

Table 1 Alert Verification Effectiveness

	Alerts	True Positives	False Positives
Stand-alone	6,659	24	99.64%
Verification enabled	24	24	00.00%

As one can see, with Snort running in stand-alone mode, 6659 attacks against the target machine were reported. However, because either no vulnerable services were actually present on the target or the targeted services were not vulnerable, most of these attacks could not have been successful and can thus be considered non-relevant. Only 24 of the alerts produced by Snort were true positives, and we arrive at a false positive rate (or, to be more precise, a non-relevant positive rate) of 99.64%. With alert verification enabled, however, alerts that relate to attacks that attempted to exploit missing or non-vulnerable services were tagged as such. By doing this, the false alarm rate for Snort with alert verification enabled dropped to 0% and only the 24 actual attacks were reported. Manual inspection of the alert stream was used to verify that no false positives or non-relevant alerts were produced.

It is important to note in interpreting these results that Snort and Nessus are open to generating both true and false positives and negatives. Thus, the following scenarios are possible:

1. *true positive/true positive*
   In this scenario, the attack is correctly detected, and the service is correctly reported as vulnerable.
2. *true positive/false positive*
   Here, the attack is correctly detected, and the service is incorrectly reported as vulnerable.
3. *true positive/true negative*
   Under this scenario, the attack is correctly detected, and the service is correctly detected as non-vulnerable.
4. *true positive/false negative*
   In this scenario, the attack is correctly detected, but the service is incorrectly determined to be non-vulnerable.
5. *false positive/true positive*
   With this scenario, benign traffic is misreported as an attack, and the service is reported as vulnerable to the reported attack.
6. *false positive/false positive*
   In this case, benign traffic is misreported as an attack, and the service is incorrectly reported as vulnerable to the reported attack.
7. *false positive/true negative*
   Under this scenario, benign traffic is misreported as an attack, and the service is correctly determined to be non-vulnerable.
8. *false positive/false negative*
   In this scenario, benign traffic is misreported as an attack, and the service is incorrectly determined to be non-vulnerable.
9. *false negative/true positive*
   In this case, an attack is not detected by the IDS, but the service would have been reported as vulnerable by the vulnerability scanner.
10. *false negative/false negative*
    Here, an attack goes undetected by the IDS, and the service would have been misreported as non-vulnerable by the vulnerability scanner.

Clearly, from the above list one can see that the ideal scenarios are 1 and 3, where alert verification either correctly reinforces confidence in IDS alerts or suppresses incorrect alerts, respectively. Scenarios 2, 5, and 6 correspond to a false positive from an IDS that is associated with incorrect alert verification. Therefore, the addition of alert verification does not degrade the effectiveness of the IDS. Scenarios 9 and 10 correspond to a successfully evaded IDS without alert verification; since alert verification triggers on IDS alerts, the technique cannot help in these scenarios. In scenario 7, alert verification is successful in suppressing a false positive that would be otherwise reported by an IDS. In scenario 8, although it is unfortunate that both components fail, the end result is that no successful attack occurs, and, furthermore, a false positive from the IDS is suppressed. Therefore, it is not a cause for concern outside of the inaccuracy of the IDS and vulnerability scanner. Thus, the only scenario in which alert verification may degrade the effectiveness of a stand-alone IDS is 4. However, because of the relative ease of writing correct vulnerability assessment checks when compared to writing intrusion detection signatures, the probability of this scenario occurring in the real world is not high. Additionally, in our evaluation manual inspection was used to verify that no scenarios resulting in false negatives were present.

For the second experiment, the verification tool was run in active mode “with authenticated access”. To this end, the target host was configured such that the verification system could log into the machine and check for attack traces locally. Because the system already removed all non-relevant alerts that were generated in the first experiment using only remote checks, re-running the same attacks with the new configuration could not yield any improvement. Thus, we decided to evaluate the potential of host checks for alert verification using three Linux worms (Ramen, Lion, and Slapper). For the experiment, we extended the Snort rules for these worms with the following host-based checks:

1. Slapper Worm: check for the existence of the file /tmp/.uubugtraq (copy of worm body)
2. Ramen Worm: check for open port 27374 (used to serve new worm copies)
3. Lion Worm: check for open port 60008 (root shell port)

Then, both vulnerable and patched versions of the programs used by the worms to spread were installed at the target machine (Apache SSL for Slapper, LPRng for Ramen, named for Lion). All three worms were launched from the attack machine, once against a set of vulnerable services and once against a set of patched services. As expected, when Snort was run without verification, every worm attack resulted in an alert. When using the verification system, however, only successful worm attacks lead to the existence of corresponding attack traces at the victim and as a result only relevant alerts were generated.

To gather real-world attack traffic and assess the amount of alerts that the system is capable of identifying as non-relevant in a more realistic scenario, a third experiment was conducted. For this experiment, we deployed two honeypots. One of the honeypot machines was running a standard RedHat 7.2 Linux installation; the other one was running an unpatched version of Microsoft Windows 2000 Server. Both hosts had a considerable amount of services with known vulnerabilities. The network link to both honeypots was monitored by Snort-2.0.2, using its complete set of 2625 rules.

During a period of 14 days, Snort reported 164,415 raw alerts referring to attacks against the RedHat Linux machine and

79,198 raw alerts referring to attacks against the Windows machine. Among these raw alerts, we noticed a large amount of attacks related to the Slammer and Nachia worms. Also, a large amount of scan activity against ports commonly used by web proxy and socks proxy servers was registered. We believe that spammers that use the proxies as mail relays perform these scans. Given the raw alerts, the alert verification process was capable of tagging 161,166 attacks against the Linux host (98.3%) and 78,785 attacks against the Windows host (99.4%) as unsuccessful. This tagging was manually verified, and we concluded that all attacks that have been tagged as unsuccessful actually failed (the manual checks could be performed reliably because most attacks targeted non-existent services). Although a default installation of Snort was used, the numbers clearly indicate that real-world attack traffic produces many false or non-relevant positives that can be suppressed using alert verification.

The results shown above demonstrate that alert verification improves the false positive rate of network intrusion detection implementations. However, the current alert verification implementation for Snort suffers from several limitations. A first problem is that the granularity of CVE IDs, which is somewhat needed by the choice of Nessus as the verification component, reduces the effectiveness of the tool as a whole. This stems from the lack of other additional information, such as host architecture, revision of the vulnerable program, etc., which could result in the vulnerability testing script reporting the service as non-vulnerable when in fact it is. It is also worth noting that this limitation generalizes to the fact that, barring implementation flaws, active alert verification is only as good as the available verification scripts, just as the quality of a signature-based IDS depends on the quality of its signatures.

Another issue is that the classification scheme of vulnerable, not vulnerable, or undetermined may not be expressive enough to capture information that is relevant to network security officers, as members of the focus-ids mailing list [Sec] have pointed out.

5 RELATED WORK

Several vendors and researchers [Gu02, De03, RNA] have proposed to include vulnerability analysis data when processing IDS alerts. The idea is to utilize previously gathered information to reduce the noise of the alert stream produced by intrusion detection sensors and disambiguate their results. These methods are all different realizations of passive alert verification techniques as described in Section 2. In this paper, on the other hand, an active alert verification mechanism is proposed. We query the potential victim in response to the sign of an attack to get the current configuration of the victim that either supports or refutes the hypothesis that a successful intrusion has occurred.

An important, related analysis process that also takes as input the alerts produced by intrusion detection systems is alert correlation. Its main task is the aggregation of alerts to provide a high-level view (i.e., the “big picture”) of malicious activity on the network. A major problem for correlation systems is false positives, which can degrade the quality of their results significantly. It is evident that correlating alerts that refer to failed attacks can result in the detection of whole attack scenarios that are actually non-existent.

Previous work [CM02, NCR02] states that alert correlation can be used both to reduce the total number of alerts and to reduce the number of false alerts. The latter, namely the reduction of false alerts, is directly related to our goal. However, the correlation systems mentioned above assume that real attacks trigger more than a single alert. As a result, the systems can focus on alert clusters and discard all alerts that have not been correlated. Unfortunately, this assumption has not been substantiated by experimental data or supported by a rigorous discussion. We claim, therefore, that the reduction of false alerts is an important prerequisite to achieve good correlation results instead of an outcome of the correlation process itself. Also, a recent paper on alert correlation [NX03] mentions that “false alerts generated by IDSs have a negative impact”. This supports our assumption that alert verification can act as a preprocessing step for correlation systems, cleaning the input stream from spurious alerts and thus improving the result of the correlation analysis.

6 CONCLUSIONS AND FUTURE WORK

We propose alert verification as a process that is launched in response to an alert raised by an intrusion detection system to check whether the corresponding attack has succeeded or not. When the attack has not succeeded, the alert can be either suppressed or its priority reduced. This provides an effective mean to lower the number of false alarms that an administrator has to deal with. It also improves the results of alert correlation systems by cleaning their input data from spurious attacks.

We have developed an active verification system based on Snort and Nessus. As the current implementation stands, it is a useful tool for reducing the false-alarm rate of Snort. There is, however, always room for improvement, and in this spirit we have planned some future directions for further development of our alert verification tool. One issue to be addressed is the coarse granularity of CVE IDs, which we plan to tackle by extending Nessus. Another planned area of development is the integration of an a priori knowledge base along with passive information gathering techniques to supplement the active verification techniques.

7 ACKNOWLEDGMENT

We would like to thank Roland Bueschkes for his numerous comments and the thorough review that helped to improve the quality of this paper. This research was supported by the U.S. Army Research Office under agreement DAAD19-01-1-0484 and by the U.S. National Science Foundation under grants CCR-0209065 and CCR-0238492. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

REFERENCES

[Ar02] Arboi, M.: The Nessus Attack Scripting Language Reference Guide. http://www.nessus.org/doc/nasl2_reference.pdf.

[Bug] Vulnerabilities by Bugtraq ID. http://www.securityfocus.com/bid/bugtraqid/.

[CD03] Curry, D.; Debar, H.: Intrusion Detection Message Exchange Format: Extensible Markup Language (XML) Document Type Definition. draft-ietf-idwg-idmef-xml-10.txt. January 2003.

[CM02] Cuppens, F.; Miege, A.: Alert Correlation in a Cooperative Intrusion Detection Framework. In: Proceedings of the IEEE Symposium on Security and Privacy. Oakland, CA. May 2002.

[CVE] Common Vulnerabilities and Exposures. http://www.cve.mitre.org/.

[De03] Desai, N.: IDS Correlation of VA Data and IDS Alerts. http://www.securityfocus.com/infocus/1708. June 2003.

[Fy] Fyodor: Nmap: The Network Mapper. http://www.insecure.org/nmap/.

[Gu02] Gula, R.: Correlating IDS Alerts with Vulnerability Information. Technical Report. Tenable Network Security. December 2002.

[GWC98] Ghosh, A.; Wanken, J.; Charron, F.: Detecting Anomalous and Unknown Intrusions Against Programs. In: Proceedings of the Annual Computer Security Application Conference. pages 259-267. Scottsdale, AZ. December 1998.

[IS04] ISS: Realsecure 10/100. http://www.iss.net/.

[JV94] Javitz, H.S.; Valdes, A.: The NIDES Statistical Component Description and Justification. Technical Report. SRI International. Menlo Park, CA. March 1994.

[KRL97] Ko, C.; Ruschitzka, M.; Levitt, K.: Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach. In: Proceedings of the IEEE Symposium on Security and Privacy. pages 175-187. May 1997.

[LID] Linux Intrusion Detection System. http://www.lids.org/.

[LS01] Loscocco, P.; Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System. In: Freenix Track of Usenix Annual Technical Conference. 2001.

[MMDD02] Morin, B.; Me, L.; Debar, H.; Ducasse, M.: M2D2: A Formal Data Model for IDS Alert Correlation. In: Proceedings of the International Symposium on the Recent Advances in Intrusion Detection. pages 115-137. Zurich, Switzerland. October 2002.

[NCR02] Ning, P.; Cui, Y.; Reeves, D.: Constructing Attack Scenarios through Correlation of Intrusion Alerts. In: Proceedings of the ACM Conference on Computer and Communications Security. pages 245-254. Washington, D.C. November 2002.

[Nes] Nessus Vulnerability Scanner. http://www.nessus.org/.

[NFR01] NFR Security: Overview of NFR Network Intrusion Detection System. February 2001.

[NP99] Neumann, P.; Porras, P.: Experience with EMERALD to Date. In: First USENIX Workshop on Intrusion Detection and Network Monitoring. pages 73-80. Santa Clara, CA. April 1999.

[NX03] Ning, P.; Xu, D.: Learning Attack Strategies from Intrusion Alerts. In: Proceedings of the ACM Conference on Computer and Communications. Washington, DC. October 2003.

[Pa98] Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. In: Proceedings of the 7th USENIX Security Symposium. San Antonio, TX. January 1998.

[RNA] RNA – Real-time Network Awareness. http://www.sourcefire.com/technology/whitepapers.html.

[Ro99] Roesch, M.: Snort – Lightweight Intrusion Detection for Networks. In: Proceedings of the USENIX LISA Conference. November 1999.

[SAV] Snort Alert Verification. http://www.cs.ucsb.edu/~wkr/projects/ids_alert_verification/.

[Sec] SecurityFocus Mailing Lists Archive. http://www.securityfocus.com/archive.

[Sno] Snort – The Open Source Network Intrusion Detection System. http://www.snort.org.

[SP03] Shankar, U.; Paxson, V.: Active Mapping: Resisting NIDS Evasion Without Altering Traffic. In: Proceedings of the IEEE Symposium on Security and Privacy. 2003.

[UJP03] Undercoffer, J.; Joshi, A.; Pinkston, J.: Modeling Computer Attacks: An Ontology for Intrusion Detection. In: 6th International Symposium on Recent Advances in Intrusion Detection. 2003.

[Vi03] Vigna, G.: A Topological Characterization of TCP/IP Security. In: Proceedings of the 12th International Symposium of Formal Methods Europe. LNCS 2805. Springer-Verlag. September 2003.

[VVK03] Vigna, G.; Valeur, F.; Kemmerer, R.A.: Designing and Implementing a Family of Intrusion Detection Systems. In: Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering. Helsinki, Finland. September 2003.

[WFP99] Warrender, C.; Forrest, S.; Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: IEEE Symposium on Security and Privacy. pages 133-145. 1999.