ELSEVIER

Computer Communications 25 (2002) 1356–1365

computer communications

www.elsevier.com/locate/comcom

Intrusion detection techniques and approaches

Theuns Verwoerd, Ray Hunt*

Department of Computer Science, University of Canterbury, Private Bag 4800, Christchurch, New Zealand

Received 13 November 2001; accepted 11 December 2001

Abstract

Recent security incidents and analysis have demonstrated that manual response to such attacks is no longer feasible. Intrusion detection systems (IDS) offer techniques for modelling and recognising normal and abusive system behaviour. Such methodologies include statistical models, immune system approaches, protocol verification, file and taint checking, neural networks, whitelisting, expression matching, state transition analysis, dedicated languages, genetic algorithms and burglar alarms. This paper describes these techniques including an IDS architectural outline and an analysis of IDS probe techniques finishing with a summary of associated technologies. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Intrusion detection; Probe technique; scanning; honeynet; Worm/virus attack

1. Introduction

Recent events have highlighted the need for fast reactionary capabilities in network security. According to an analysis from Ref. [7], the code red worm infected over 359,000 hosts within less than 14 h, as illustrated by Fig. 1. Later theoretical analysis place the time for a complete spread of a network worm as low as 15 min [35], or 30 s [14].

Current conventional security techniques seem incapable of dealing with these threats (leaving the task to security personnel). In many respects, this mirrors the epidemic spread of diseases and electronic viruses. Intrusion detection systems (IDS) have the potential to mitigate or prevent such attacks, if updated signatures or novel attack recognition and response capabilities are in place.

In this paper, we attempt to give a brief overview of the techniques behind current IDS, how they are structured, model acceptable and abusive behaviour, observe and respond to protected systems.

2. Intrusion detection systems

2.1. Conceptual outline

IDS have evolved from monolithic batch-oriented systems to distributed real-time networks of components (Fig. 2). In current systems, a number of common building functional blocks can be distinguished.

- Sensor, probe: These modules form the most primitive data-gathering components of an IDS. Implemented in a highly system-specific fashion, they track network traffic, log files or system behaviour, translating raw data into events useable by the IDS monitors.
- Monitor: Monitor components, the main processing segment of an IDS, receive events from sensors. These events are then correlated against the IDS behaviour models, potentially producing model updates and alerts. Alerts, events in themselves, indicate occurrences significant to the security of a system, and may be forwarded to higher-level monitors, or to resolver units.
- Resolver: Resolver components receive suspicion reports from monitors (in the form of one or more alerts), and determine the appropriate response: logging, changing the behaviour of lower-level components, reconfiguring other security mechanisms (e.g. adding firewall rules), and notifying operators.
- Controller: Facilitating component configuration and coordination, controller components are most significant in distributed IDS, where manually upgrading, configuring and starting a network-wide series of components would be infeasible. In addition, controller units offer a single point of administration and interrogation for an IDS, and may act in a supervisory capacity, restarting failed components [10].

* Corresponding author. Tel.: +64-3-3642347; fax: +64-3-3642569.
E-mail address: ray@cosc.canterbury.ac.nz (R. Hunt).

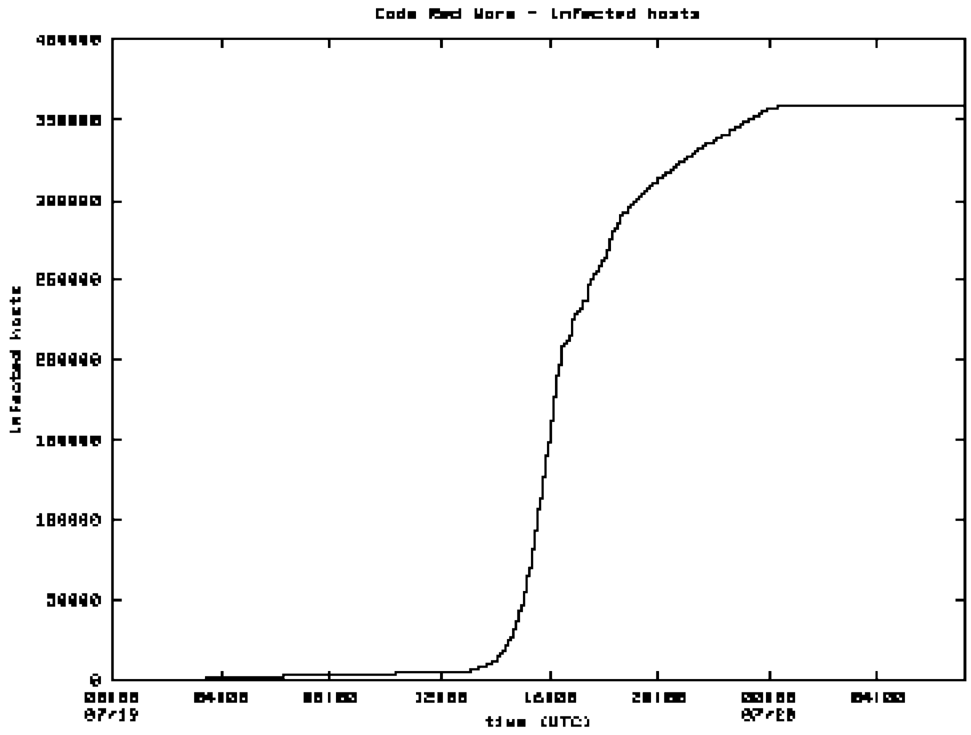

Fig. 1. Code red worm spread [7].

In many IDS architectures these divisions are often indistinct. In monolithic IDS, for example, all of these components may form a single unit, or may be divided across multiple processes and tools.

2.2. Anomaly vs misuse detection

At the heart of intrusion detection lies the ability to distinguish acceptable, normal system behaviour from that which is abnormal (possibly indicating unauthorised activities) or actively harmful. Two approaches to this problem can be distinguished, with IDS implementations using some combination of these:

- *Anomaly detection* attempts to model normal behaviour. Any events which violate this model are considered to be

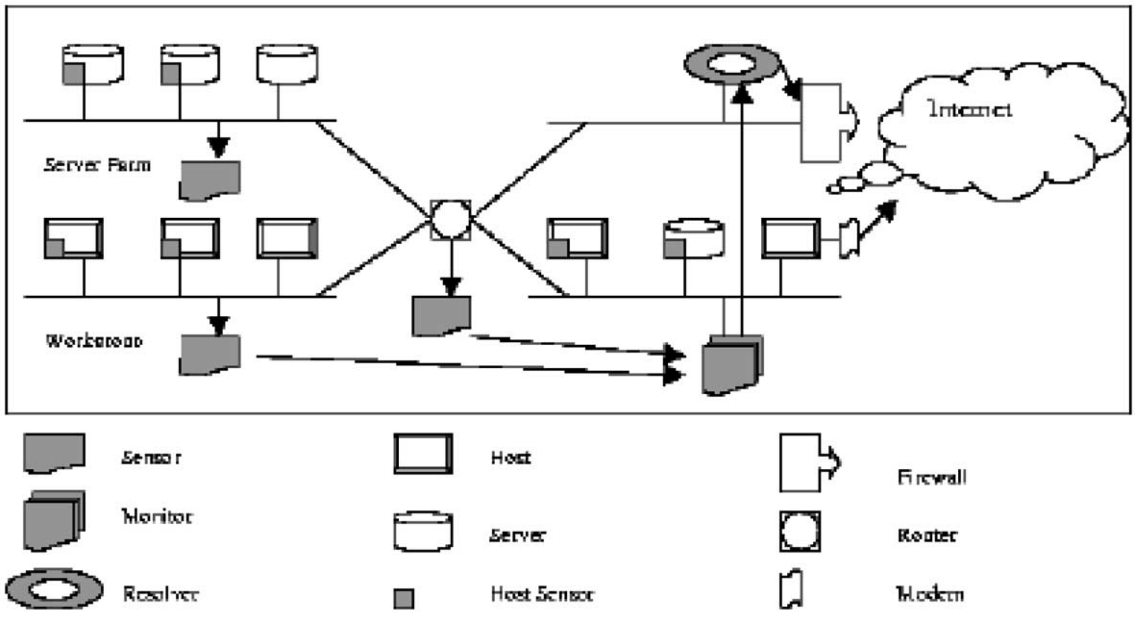

Fig. 2. Intrusion detection network structure.

suspicious. For example, a normally passive public web server attempting to open connections to a large number of addresses may be indicative of a worm infection.

- *Misuse detection* attempts to model abnormal behaviour, any occurrence of which clearly indicates system abuse. For example, an HTTP request referring to the *cmd.exe* file may indicate an attack.

Anomaly detection suffers from accuracy problems, as building an accurate model (avoiding false negatives) may not fully reflect the complex dynamic nature of computer systems (leading to false positives). This technique has had some success in detecting previously unknown attack techniques [21], a major shortcoming in misuse detection.

Misuse detection, by virtue of the simpler scope of the objects being modelled, can attain high levels of accuracy. The major difficulty with this approach, however, lies in creating compact models of attacks—models that cover all possible variants of an attack, while avoiding benign patterns. In addition, this approach is vulnerable to novel attacks (attacks dissimilar to all previously known examples), arguably the most dangerous kind.

Due to the complementary nature of these two approaches, many systems attempt to combine both of these techniques. The problem of false positives cause many commercial IDS offerings to focus on misuse detection, leaving anomaly detection to research systems.

2.3. Anomaly modelling techniques

2.3.1. Statistical models

In Denning's ground laying paper on IDS [11], she described a number of statistical characterisations of events and event counters. These, and more refined techniques, have been implemented in anomaly detection systems [24]:

- Threshold measures: Referred to as the operational model in Ref. [11], this scheme applies set or heuristic limits to event occurrences or event counts over an interval. A common example is logging and disabling user accounts after a set number of failed login attempts.
- Mean and standard deviation: By comparing event measures to a profile mean and standard deviation, a confidence interval for abnormality can be established. The profile values are fixed or based on weighted historical data.
- Multivarate model: Calculating the correlation between multiple event measures, relative to the profile expectations.
- Markov process model: This model regards event types to be state variables in a state transition matrix, where an event is considered anomalous if its probability, given the previous state and associated value in the state transition matrix, is too low.
- Clustering analysis: This non-parametric method relies on representing event streams in a vector representation, examples of which are then grouped into classes of behaviours using some clustering algorithm (e.g. *k*-nearest-neighbour). Clusters represent similar activities or user patterns, where normal and anomalous behaviour can be distinguished [3].

2.3.2. Immune system approach

Application implementations inherently provide a model of normal behaviour, in the form of application code paths. In the immune system approach, applications are modelled in terms of sequences of system calls for a variety of different conditions: normal behaviour, error conditions and attempted exploits. Comparing this model to observed event traces allows classification of normal or suspicious behaviour. For example, an anomalous *exec* system call in a web server process may be indicative of a buffer overflow attack.

This approach has demonstrated the ability to detect a number of typical attack techniques, but cannot detect attacks based on race conditions,[1] policy violations or masquerading [3].

2.3.3. Protocol verification

Many attack techniques rely on the use of unusual or malformed protocol fields, which are incorrectly handled by application systems. Protocol verification techniques rigorously check protocol fields and behaviour against established standards or heuristic expectations. Data that violates the relevant boundaries is tagged as suspicious.

This approach, used in a number of commercial systems, can detect many commonly used attacks, but suffers from the poor standards-compliance of many protocol implementations [20]. In addition, using this technique on proprietary or under-specified protocols may be difficult or lead to false positives.

2.3.4. File checking

Best known through its implementation in the tripwire system,[2] this technique uses cryptographic checksums of sensitive system data to detect changes, including unauthorised software installations, back-doors left by successful intrusion, and system corruption.

These techniques are extremely useful in system recovery and forensic examination. Detection is inherently after the fact, however, and can be bypassed if the cryptographic checksums are modified or the examination process is compromised [3] [34].

[1] A race condition attack corrupts the time-critical interaction between different processes or systems.

[2] This technique is also used in antivirus applications to detect modifications to executable files.

[3] The t0rn rootkit, for example, provides pristine versions of compromised system files to the file checking system, while executing modified binaries.

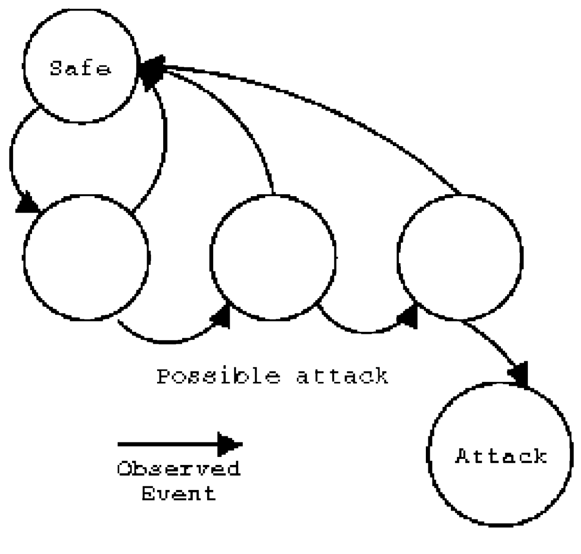

Fig. 3. Schematic structure of a state machine.

2.3.5. Taint checking

An application-centric approach to anomaly detection lies in creating risk-aware applications. One example of this is in the perl programming language, commonly used to implement HTTP CGI[4] applications. In this system, all user-provided input is considered 'tainted'; any attempt to use such tainted data in a sensitive context (such as an exec system call) fails. Untainting a variable must be done explicitly, by extracting expected content with a regular expression avoiding the risk of embedded shell commands, or otherwise unexpected content, from being used [26].

2.3.6. Neural nets

A neural net is essentially a network of computational units that jointly implement complex mapping functions. Initially, the network is trained with normal system behaviour traces. Observed event streams are then fed into the network, which classifies these streams as normal (observations match training data), or anomalous. The system may also undergo continuous training using this observed data, allowing the network to learn changes in system behaviour.

Since this approach does not rely on preconceptions about the behaviour of the monitored system, it avoids the need to preselect suitable features and thresholds. Similarly, the learning ability of the network allows compensation for behaviour drift, though this may allow undetected violations actions to be included in the model. A greater difficulty with this technique lies in the fact that only an outcome may be observed; the reason for a mismatch between the model and observed behaviour is not evident [22].

2.3.7. Whitelisting

Whitelisting, a simple technique described in Ref. [28], involves passing a raw event stream (for example, a system log) through a number of filters, each corresponding to known benign patterns. Any remainder after all known events have been filtered out are novel or suspicious.

Primarily a data reduction technique, this makes manual review of event streams feasible. Filters are manually incrementally refined, reducing the number of false positives, though sufficiently specific filters may be hard to generate. In addition, recognition of attacks typified by large numbers of otherwise normal events (e.g. login failure) is difficult.

2.4. Misuse detection modelling techniques

2.4.1. Expression matching

The simplest form of misuse detection, expression matching, searches an event stream (log entries, network traffic, or the like) for occurrences of specific patterns.[5] A simple example would be '^GET [^$]*/etc/passwd$'; this checks for something that looks like an HTTP request for the Unix password file.

As can be seen from this example, the resulting expressions can quickly become hard to read, and representing context is difficult. Signatures can be very simple to construct, however, especially when combined with protocol-aware field decomposition [20].

2.4.2. State transition analysis

State transition analysis models attacks as a network of states and transitions (matching events). Every observed event is applied to finite state machine instances (each representing an attack scenario), possibly causing transitions. Any machine that reaches its final (acceptance) state indicates an attack. (Fig. 3)

This approach allows complex intrusion scenarios to be modelled in a simple way, and is capable of detecting slow or distributed attacks, but may have difficulty in expressing elaborate scenarios [33]. Other state machine representations (e.g. coloured petri nets) offer similar advantages [19].

2.4.3. Dedicated languages

A number of IDS implementations represent intrusion signatures using specialised languages, including those in Refs. [5,23,30,31]. While the generality of such languages vary (ranging from compiled filtering expressions in Ref. [30] to complete programming languages [23]), all offer great flexibility in matching attack scenarios. A signature takes the form of a specialised program, with raw events as input. Any input triggering a filtering program, or input that matches internal alert conditions, is recognised as an attack.

To illustrate, consider the examples (taken from Ref. [29]) in Figs. 4 and 5. Both represent filters that match HTTP probes for certain CGI scripts. In the first case, this is done by matching any connection with TCP destination

[4] Hypertext transfer protocol common gateway interface.

[5] This is similar to the signature matching used in some antivirus applications.

```
alert tcp any any -> any 80 (msg:"CGI-nph-tst-cgi"; content:"cgi-bin/nph-test-cgi?"; flags: PA;)
alert tcp any any -> any 80 (msg:"CGI-test-cgi"; content:"cgi-bin/test-cgi?"; flags: PA;)
alert tcp any any -> any 80 (msg:"CGI-perl.exe"; content:"cgi-bin/perl.exe?"; flags: PA;)
alert tcp any any -> any 80 (msg:"CGI-phf"; content:"cgi-bin/phf?"; flags: PA;)
```

Fig. 4. Snort CGI probe filter.

port 80, and a specific string in the HTTP segment body. In the second case, the filter checks that the connection destination is one of the nominated destination servers, creates an HTTP data segment by concatenating the current and previous TCP segments, and searches each line for occurrences from the signature set. If any matches are found, a log entry is created with the current time, connection and request details.

As can be seen, creating signatures may require significant understanding of the protocols and attacks involved, and some programming ability. In addition, attacks that use variations on the signature strings may bypass this

```
badweb_schema = library_schema:new( 1, ["time", "int",
                                "ip", "ip", "str"], scope());
# list of web servers to watch. List IP address of servers or a netmask
# that matches all. use 0.0.0.0:0.0.0.0 to match any server
da_web_servers = [ 0.0.0.0:0.0.0.0 ] ;
query_list = [ "/cgi-bin/nph-test-cgi?",
            "/cgi-bin/test-cgi?",
            "/cgi-bin/perl.exe?",
            "/cgi-bin/phf?"
            ] ;
filter bweb tcp ( client, dport: 80 )
{
    if (! ( tcp.connDst inside da_web_servers) )
        return;
    declare $blob inside tcp.connSym;
    if ($blob == null)
        $blob = tcp.blob;
    else
        $blob = cat ( $blob, tcp.blob );
    while (1 == 1) {
      $x = index( $blob, "\n" );
      if ($x < 0)        # break loop if no complete line yet
        break;
      $t=substr($blob,$x-1,1);     # look for cr at end of line
      if ($t == '\r')
        $t=substr($blob,0,$x-1);   # tear off line
      else
        $t=substr($blob,0,$x);
      $counter=0;
      foreach $y inside (query_list) {
        $z = index( $blob, $y );
        if ( $z >= 0) {
          $counter=1;
          # save the time, the connection hash, the client,
          # the server, and the command to a histogram
          record system.time, tcp.connHash, tcp.connSrc, tcp.connDst,
                                    $t to badweb_hist;
        }
      }
      if ($counter)
        break;
    }
    # keep us from getting flooded if there is no newline in the data
    if (strlen($blob) > 4096)
        $blob = "";
    # save the blob for next pass
    $blob = substr($blob, $x + 1);
}
badweb_hist = recorder ("bin/histogram packages/test/badweb.cfg",
    "badweb_schema" );
```

Fig. 5. NFR N-Code CGI probe filter.

type of filter. A sufficiently expressive filtering language, however, can duplicate the effect of any other representation.

2.4.4. Genetic algorithms

The genetic algorithm as an alternative tool for security audit trail analysis (GASSATA) system [15] uses a genetic algorithm to search for the combination of known attacks (expressed as a binary vector, each element indicating the presence of a particular attack) that best matches the observed event stream. A hypothesis vector is evaluated based on the risk associated with the attacks involved, and a quadratic penalty function for mismatched details. In each cycle, the current set of best hypotheses are mutated and retested, so that the probability of false positives and negatives approach zero.

This technique, like the neural net approach, offers good performance but does not identify the reason for an attack match. In addition, expressing some forms of behaviour, and expressing simultaneous or combined attacks is not possible in this system.

2.4.5. Burglar alarms

A technique proposed by Markus Ranum [6] to reduce the risk of false positives and allow identification of novel attacks, focusing on identifying events that should never occur. Applying dedicated monitors to search for instances of such policy violations effectively places traps for would-be attackers. An example would be monitoring for any attempt to connect outward from an HTTP server (where this is contrary to the site policy), indicating a user error, or an attacker attempting to use the server as a relay point.

For an appropriate selection of trigger events, false positives are negligible. Similarly, since probes are set to monitor specific policy violations, this approach is applicable to known, permuted and novel attacks equally. Selecting an appropriate set of triggers, however, requires in-depth system and site knowledge.

3. Probe techniques

3.1. Host log monitoring

The earliest forms of IDS were batch-oriented systems, periodically searching accumulated system, audit and application logs for signs of suspicious activity [2]. Many modern systems continue to use host logs as a source of raw events.

Host logs, comprised of the combination of audit, system and application logs, offer an easily-accessible and non-intrusive source of information on the behaviour of a system. In addition, logs generated by high-level entities can often summarise many lower-level events (such as a single HTTP application log entry covering many system calls) in a context-aware fashion.

A number of details complicate the use of such logs, however. Foremost among these is the questionable validity of log entries on a victim host, especially those generated after the point of compromise. Second, recreating the context of interleaved event chains can be difficult, especially where multiple distributed processes interact. Finally, the quality of information held in logs is frequently low; entries omit critical information, while including large quantities of meaningless detail.

3.2. Promiscuous network monitoring

The primary source of abuse on many systems is due to the network connectivity of that system. Many IDS, therefore, focus on this as a source of events, monitoring all traffic between hosts. This ensures that the IDS can observe all communication between a network attacker and the victim system, resolving many of the problems associated with log monitoring.

- No processing impact on monitored hosts.
- The ability to observe network-level events.
- Monitoring an entire segment at once.
- Isolating the monitoring station from the influence of host compromise.

As shown in Ref. [27], however, this approach has limitations. In particular, ensuring that data streams are reconstructed identically on monitored hosts and inside the monitor is difficult; different software platforms present mutually exclusive results when faced with malformed data streams. Similarly, the use of cryptographic protocols negate the ability of these IDS probes to observe application data, effectively obscuring attacks [36]. Finally, as the complexity and capacity of networks increase, the performance requirements on probes can become prohibitive.

3.3. Host network monitoring

An approach used in personal firewalls and some IDS probe designs lies in combining network monitoring with host-based probes. By observing data at all levels of the host's network protocol stack, the ambiguities of platform-specific traffic handling and the problems associated with cryptographic protocols can be resolved. The data and event streams observed by the probe are those observed by the system itself.

This approach offers advantages and disadvantages similar to both the alternatives listed earlier. It resolves many of the problems associated with promiscuous network monitoring, while maintaining the ability to observe the entire communication between victim and attacker. Like all host-based approaches, however, this approach implies a performance impact on every monitored system, requires additional support to correlate events on multiple hosts, and is subject to subversion when the host is compromised [4].

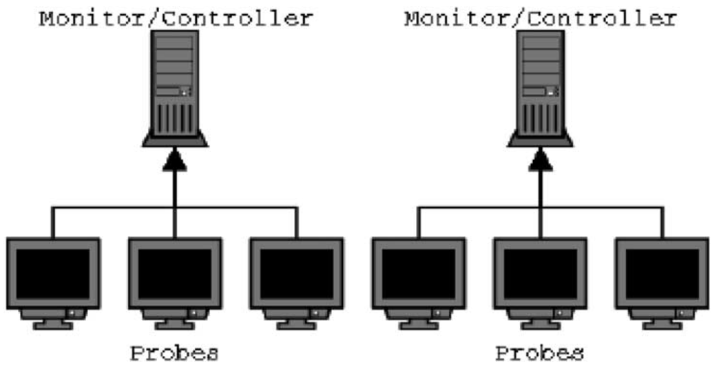

Fig. 6. Monolithlic IDS architecture.

3.4. Target-based IDS

Another attempt to resolve the ambiguities inherent in protecting multiple platforms lies in combining a network knowledge with traffic reconstruction. These target-based IDS typically use scanning techniques to form an image of what systems exist in the protected network, including such details as host operating system, active services, and possible vulnerabilities. Using this knowledge, a probe can reconstruct network traffic in the same fashion as would be the case on the receiver system, preventing attackers from injecting or obscuring attacks.

In addition, this approach allows an IDS to automatically differentiate attacks that are a threat to the targeted system, from those that target vulnerabilities not present, thus refining generated alerts (for example, IIS-based attacks on Apache HTTP servers might be ignored). Whether attacks that cannot succeed should be reported is something of a contentious issue, offering a trade-off between lower (and more applicable) security alerts being generated, vs the possibility of recognising novel attacks when combined with known sequences. In addition, the need to maintain an accurate map of the protected network, including valid points of vulnerability may reduce the ability of this class of system to recognise novel attacks [20].

4. Architecture

4.1. Monolithic systems

The simplest model of an IDS is a single application, containing probe, monitor, resolver and controller all in one. More advanced monolithic systems use a number of independent probe, monitor and resolver components, each implementing specific techniques. In common between all such systems, however, is the fact that these focus on a specific host or system with no correlation of actions that cross system boundaries (Fig. 6).

Such systems are conceptually simple, and relatively easy to implement. Their major weakness lies in the ability for an attack to be implemented using a sequence of individually innocuous steps. The alerts generated by such systems may

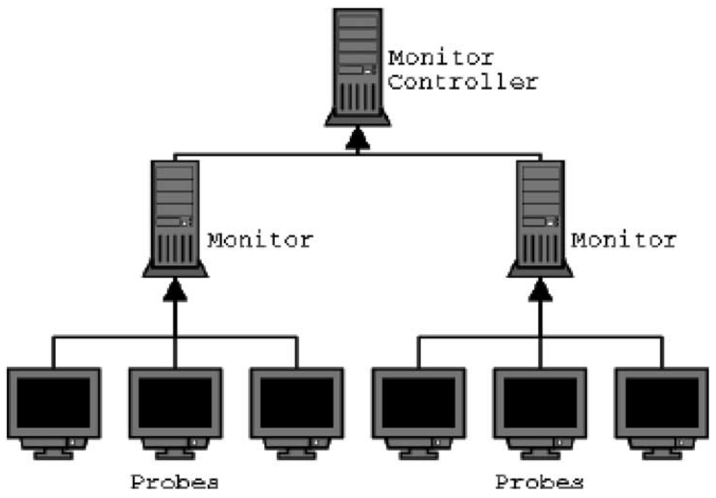

Fig. 7. Hierarchal IDS architecture.

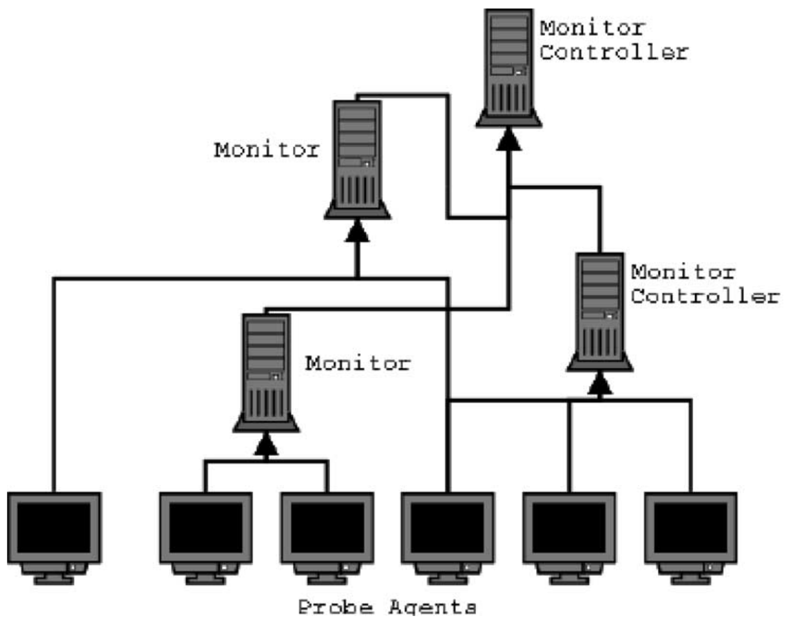

Fig. 8. Agent-based IDS architecture.

in fact be aggregated centrally but this architecture offers no synergy between IDS instances.

4.2. Hierarchic systems

If one considers the alerts generated by an IDS instance to be events in themselves, suitable for feeding into a higher-level IDS structure, an intrusion detection hierarchy results. At the root of the hierarchy lies a resolver unit and controller. Below this lies one or more monitor components, with subsidiary probes distributed across the protected systems. Effectively, the whole hierarchy forms a macro-scale IDS (Fig. 7).

The use of a centralised controller unit allows information from different subsystems to be correlated, potentially identifying transitive or distributed attacks. For example, a simple address range probe, while difficult to detect using a network of monolithic host IDS instances, can be trivial to observe when correlating connections using a hierarchic structure [12].

4.3. Agent-based

A more recent model of IDS architecture divides the system into distinct functional units: probes, monitors, resolver and controller units. These may be distributed across multiple systems, with each component receiving input from a series of subsidiaries, and reporting to one or more higher-level components. Probes report to monitors, which may report to resolver units or higher-level monitors, and so forth (Fig. 8).

This architecture, implemented in systems such as in Refs. [1,13], allows great flexibility in the placement and application of individual components. In addition, this architecture offers greater survivability in the face of overload or attack, high extensibility, and multiple levels of reporting throughout the structure.

4.4. Distributed (GrIDS)

All the IDS architectural models described earlier consider attacks in terms of events on individual systems. A recent development, typified by Ref. [17] system, lies in regarding the whole system as a unit. Attacks are modelled as interconnection patterns between systems, with each link representing network activity. The graphs that form can be viewed at different scales, ranging from small systems to the interconnection between large and complex systems (where subnetworks are collapsed into points).

This novel approach promises high scalability and the potential to recognise widely distributed attack patterns (such as worm behaviour).

5. Associated technologies

5.1. Personal firewalls

With the advent of broadband access technologies such as cable and xDSL, persistent connection to the internet for home users has become a reality. With that reality came a whole new world of hosts to break into, as clearly evidenced by reports such as the attack on grc.com [16].

One response to this pattern has been the emergence of personal firewalls: software that effectively implements a small firewall and IDS on each network host [37]. A feature found in some of these systems is restricting outbound connections to only those specifically authorised; a response to trojan and other software that open outbound network links [32].

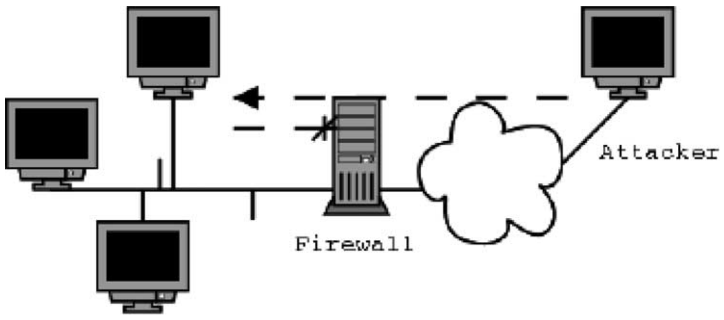

Fig. 9. Structure of a honeynet.

5.2. Honeynets

The concept of a honeytrap is simple: a dedicated sacrificial system placed in an attractive or ubiquitous position on a network and designed to receive attacks as shown in Fig. 9. Ref. [8] contains a good example of such a system, manually implemented.

Originally, honeytraps consisted of heavily monitored, real systems or virtual systems implemented by software. A recent innovation was the use of a so-called honeynet: an entire network of systems, in its entirety sacrificial. Separating this network from the outside world is a firewall, configured to allow unrestricted incoming access, but limit outgoing access. In this manner, an attacker is prevented from using the machines in the honeynet as a relay point for attacking other systems.

Any traffic to or from the honeynet is fully logged. Experimentation with such systems have provided fascinating details, ranging from the development of passive fingerprinting techniques,[6] to insights into the social interaction of hackers [18].

5.3. Responses

The ideal IDS would be capable of recognising and neutralising attacks, prevent further attack, and hardening the vulnerable system to prevent reoccurrence. In current IDS implementations, a number of reactive capabilities can be recognised.

- *Attack tracing* where the system attempts to passively or indirectly gather information to aid in identifying the source of attack (using techniques such as DNS lookups, passive fingerprinting, etc.).
- *Shunning* where the IDS reconfigures another system (such as a firewall or router) to block out the attacker, or uses TCP reset frames to tear down any connection attempts.
- *Extended information gathering*, increasing the level of information stored about events surrounding the attack, for future forensic analysis.

The use of these responses are currently quite limited, due to the problems inherent in false positives, belated attack recognition, and the risk of an automated response constituting an attack (either against an innocent third party or the protected system). Contentious issues also arise here, regarding the level of response acceptable, and the contradictory relationship between mitigating the effects of an attack and gathering evidence.

6. Conclusion

The field of intrusion detection has been and will continue to develop rapidly. A number of techniques and solutions found in current systems are outlined in this paper. As evidenced by recent events, however, network security has some way to go before any network can be considered safe.

Intrusion detection has the potential to alleviate many of the problems facing current network security. However, a number of issues remain to be resolved.

- Scaling to large, fast and complex systems. Many of the IDS currently in use are essentially monolithic; in order to respond effectively to large-scale attacks, a more distributed architecture is necessary.
- Rapid distribution of new attack signatures. Misuse detection systems are completely reliant on having models of new attacks; in order to mitigate attacks such as the code red worm, distribution of these models must outpace the epidemic-like spread of a worm.
- Ubiquitous acceptance. Many current IDS are complex to configure and run, restricting their use.
- Strong reactive capabilities. It is clear from the analysis referenced earlier that relying on human intervention to handle attacks is infeasible. Most current IDS implementations have limited reactionary capabilities; an IDS needs to be capable of preventing, not just reporting an attack.

There has been some attempts to create a common list of attacks for comparison between IDS implementations [9]. One possible solution to the problem of signature

[6] Passive fingerprinting extends the techniques present in Ref. [25] to allow identification of remote systems, using implementation characteristics observed in network traffic.

dissemination may lie in the creation of a vendor-independent attack representation; one that can be converted to IDS-specific representations or model (as described in Section 3.4) automatically.

So far, research into IDS has focused on architectural and representation issues. Current systems have proven capable of handling known attacks but the next incident may not be as benign.

References

[1] CERIAS Autonomous Agents for Intrusion Detection Group, COAST Autonomous Agents for Intrusion Detection, Project homepage, http://www.cs.purdue.edu/coast/projects/autonomous-agents.html, 7 September, 1999.

[2] J.P. Anderson, Computer security threat monitoring and surveillance, Technical Report, James P. Anderson Co., Fort Washington, PA, April, 1980.

[3] R.G. Bace, Intrusion detection, ISBN 1-57870-185-6, 2001.

[4] A. Plato, Network ICE BlackICE Defender User's Guide version 1.0, 1999, http://networkice.com/support/Docs/BlackICEDefUG.pdf.

[5] V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time USENIX Security Symposium, January 1998, ftp://ftp.ee.lbl.gov/papers/bro-usenix98-revised.ps.Z.

[6] M.J. Ranum, Intrusion detection: challenges and myths, 1998, http://www.nfr.net/forum/publications/id-myths.html.

[7] D. Moore, The spread of the code-red worm (Crv2), September, 2001, http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml.

[8] B. Cheswick, An evening with Berferd in which a cracker is lured, endured, and studied, Proceedings of Winter USENIX Conference, January 1992, ftp://ftp.research.bell-labs.com/dist/ches/berferd.ps.

[9] Common vulnerabilities and exposures, http://www.cve.mitre.org.

[10] H. Debar, M. Dacier, A. Wespi, Towards a taxonomy of intrusion-detection systems, Computer Networks 31 (1999) 805–822 http ref depends on user id.

[11] D.E. Denning, An intrusion-detection model, IEEE Transactions on Software Engineering SE-13 (2) (1987) 222–232.

[12] S.R. Snapp, et al., DIDS (distributed intrusion detection system)—motivation, architecture, and an early prototype, Proceedings of the 14th National Computer Security Conference, Washington, DC, October, 1991, pp. 167–176, http://seclab.cs.ucdavis.edu/papers/DIDS.ncsc91.pdf.

[13] P.G. Neumann, P.A. Porras, Experience with EMERALD to date first USENIX workshop on intrusion detection and network monitoring, April, 1999, http://www.sdl.sri.com/emerald/det99.ps.gz.

[14] S. Staniford, G. Grim, R. Jonkman, Flash worms: thirty seconds to infect the internet, Silicon Defense, August, 2001, http://www.silicondefense.com/flash/.

[15] Ludovic Me, GASSATA, a genetic algorithm as an alternative tool for security audit trails analysis, RAID '98, SUPELEC, France, September 1998, http://www.zurich.ibm.com/%7Edac/Prog_RAID98/Full_Papers/gassata_paper.pdf.

[16] S. Gibson, The strange tale of the denial of service attacks against GRC.COM, Gibson Research Corporation, May 2001, http://grc.com/dos/grcdos.htm.

[17] S. Cheung, et al., The design of GrIDS: a graph-based intrusion detection system. U.C. Davis Computer Science Department Technical Report CSE-99-2, 1999, http://seclab.cs.ucdavis.edu/arpa/grids/grids.pdf.

[18] L. Spitzner, Know your enemy: Honeynets, Honeynet Project, April 2001, http://project.honeynet.org/papers/honeynet/.

[19] S. Kumar, Classification and detection of computer intrusions, PhD Thesis, Purdue University, 1995, http://coast.cs.purdue.edu/pub/COAST/papers/sandeep-kumar/kumar-intdet-phddiss.ps.Z.

[20] Focus-IDS mailing list, 2001, http://www.securityfocus.com/focus/ids/list/focus_idsfaq.html.

[21] J. Hochberg, K. Jackson, C. Stallings, J.F. McClary, D. DuBois, J. Ford, NADIR: an automated system for detecting network intrusions and misuse, Computers and Security 12 (3) (1993) 235–248.

[22] H. Debar, M. Becker, D. Siboni, A neural network component for an intrusion detection system, Proceedings of IEEE Symposium on Research in Computer Security and Privacy, 1992.

[23] M.J. Ranum, K. Landfield, M. Stolarchuk, M. Sienkiewicz, A. Lambeth, E. Wall, Implementing a generalized tool for network monitoring, LISA '97, http://www.nfr.net/forum/publications/LISA-97.htm.

[24] H.S. Javitz, A. Valdes, The NIDES statistical component: description and justification, March 1993, SRI International, http://www.sdl.sri.com/nides/reports/statreport.ps.gz.

[25] Fyodor, Remote OS detection via TCP/IP Stack FingerPrinting, 18 October, 1998, http://www.insecure.org/nmap/nmap-fingerprinting-article.txt.

[26] perlsec—Perl security, http://www.perldoc.com/perl5.6/pod/perlsec.html.

[27] T.H. Ptacek, T.N. Newsham, Insertion, evasion, and denial of service: eluding network intrusion detection, Secure Networks Inc., January 1998, http://www.securityfocus.com/data/library/ids.ps.

[28] M.J. Ranum, Artificial ignorance: how-to guide, Firewall Wizards Mailing List, September 1997, http://lists.insecure.org/firewall-wizards/1997/Sep/0096.html.

[29] M. Roesch, Snort—lightweight intrusion detection for networks, Proceedings of LISA 99, 1999, http://www.snort.org/docs/lisa-paper.txt.

[30] SANS Institute, Building a network monitoring and analysis capability, July 1998, http://www.nswc.navy.mil/ISSEC/CID/step.htm.

[31] Snort—the open source network intrusion detection system, http://www.snort.org/.

[32] S. Gibson, OptOut—tell unwelcome spyware to pack its bags!, Gibson Research Corporation, http://grc.com/optout.htm.

[33] P. Porras, STAT—a state transition analysis tool for intrusion detection, Technical Report TRCS93-25, Computer Science Department, University of California at Santa Barbara, November, 1993.

[34] G.H. Kim, E.H. Spafford, Experiences with tripwire: using integrity checkers for intrusion detection, Systems Administration, Networking and Security Conference III, USENIX, 1994, ftp://coast.cs.purdue.edu/pub/COAST/Tripwire/Tripwire-SANS.ps.Z.

[35] N.C. Weaver, Warhol worms: the potential for very fast internet plagues, August 2001, http://www.cs.berkeley.edu/~nweaver/warhol.html.

[36] P. Chowdhry, Attacked and hacked!, PC Week Labs, 11 October 1999, http://www.zdnet.com/pcweek/stories/news/0,4153,2350743,00.html.

[37] ZoneAlarm 2.6, ZoneLabs, http://www.zonelabs.com/products/za/index.html.