Synapse: Auto-correlation and Dynamic Attack Redirection in an Immunologically-inspired IDS

David Duncombe **George Mohay** **Andrew Clark**

Information Security Institute
Queensland University of Technology
GPO Box 2434, Brisbane 4001, Queensland, Australia
Email: d.duncombe@student.qut.edu.au, {g.mohay|a.clark}@qut.edu.au

Abstract

Intrusion detection systems (IDS) perform an important role in the provision of network security, providing real- time notification of attacks in progress. One promising category of IDS attempts to incorporate into its design properties found in the natural immune system. Although previous attempts to apply immunology to intrusion detection have considered the issue of accuracy, more work still needs to be done. We present an immunologically-inspired intrusion detection model in which the false positive rate is moderated through a process of event correlation between multiple sensors. In addition, the model offers a novel response mechanism. Previous research has flirted with a variety of response mechanisms, including those that are capable of tearing down connections, killing processes and dynamically updating firewall rules. Although such mechanisms may prevent or at least mitigate an attack before its full impact is achieved, they work against the collection of information for investigatory or evidence purposes. To overcome this limitation, a response strategy is proposed in which the attack is dynamically redirected to an isolated host deployed as a honeypot. In this way, it becomes possible to mitigate the effects of the attack while at the same time study the attack itself.

Keywords: intrusion detection, alert correlation, immunological.

1 Introduction

Intrusion detection systems (IDS) have an important role to play in the provision of network security, providing real-time notification of attacks in progress (McHugh, 2001; Verwoerd and Hunt, 2002). With increasing complexity of software systems, there has been increasing emphasis on combining and correlating heterogeneous IDS in order to provide better coverage and accuracy (Haines, 2003).

One promising category of IDS attempts to incorporate into its design properties found in the natural immune system. The natural immune system has evolved to protect organisms from pathogens such as bacteria, viruses, and parasites. Although there are significant differences between living organisms and computers, the similarities have the potential to help provide robust, distributed protection of computer systems. Traditionally, the immune system has been considered to solve the problem of discriminating between self and dangerous nonself. Self describes the body, and nonself describes material that is foreign to the body. There are two main difficulties for the natural immune system in its attempt to discriminate between self and nonself. Firstly, self and nonself in living organisms are comprised of the same basic material: protein. Secondly, there are limited resources available to the immune system. Similar difficulties are also faced by computer systems. The sense of self as applied to computer systems is very dynamic. Legitimate changes to self are routinely made: for example, new users and new programs can be introduced, and users can change their work habits. In addition, previously unseen patterns of legitimate activity often arise. This is referred to as perpetual novelty (Forrest, Hofmeyr, Somayaji, & Longstaff, 1996; Somayaji & Forrest, 2000).

Although previous attempts to apply immunology to intrusion detection have considered the issue of accuracy, more work still needs to be done. We present an immunologically-inspired intrusion detection model in which the false positive rate is moderated through a process of event correlation between multiple sensors. In addition, the model offers a novel response mechanism in which the attack is redirected to a honeypot using techniques similar to those currently employed by session hijacking tools. This response mechanism enables the attacker's activities to be monitored from a relatively safe environment.

The remainder of the paper is organized as follows: Section 2 addresses related work in computer immunology. Section 3 addresses the system design of Synapse, while Section 4 describes its current implementation. Section 5 then provides an evaluation of the system using a well known attack tool to demonstrate its success. Finally Section 6 presents our conclusions, itemising current limitations and future work.

2 Related Work

To provide comprehensive intrusion detection, a computer system's sense of self needs to be represented in many ways. This reflects the natural immune system's ability to represent self in many ways for example, the humoral response is targeted at intracellular infections while the cell-mediated response is targeted at extracellular material (Forrest, Hofmeyr, & Somayaji, 1997). An early example of applying immunology to intrusion detection looked at host-based anomaly detection. The definition of self was based on observing system call sequences. Of course, any non-trivial program would likely produce very large sets of system call sequences for a given execution. However, it was found that the local ordering of system calls was consistent, and that short sequences of system calls provided a compact signature for normal be-

haviour. The method used by this approach involved two phases: a training phase and a testing phase. During the training phase, a database of normal behaviour was constructed by monitoring programs under normal operation, and recording the associated sequences of system calls. During the testing phase, a comparison was made between the sequence of system calls found in both the running program and the normal database. All system calls not found in the normal database were regarded as mismatches. Anomalies were flagged when a sufficient number of local mismatches were found (Forrest, Hofmeyr, Somayaji, & Longstaff, 1996; Forrest, Hofmeyr, & Somayaji, 1997; Hofmeyr, 1999). The immunological concepts implemented in this research were implicit policy specification (the acceptable behaviour of the application was inherently defined by its normal operation), anomaly detection (the system could detect unknown attacks), and diversity (different normal databases were constructed for the same program, depending on the particular environment it was running under) (Hofmeyr, 1999). Although successful in detecting anomalous behaviour, the research on system calls failed to incorporate many important properties of the immune system.

A second example of applying immunology to intrusion detection borrows more intimately from the immune system. It explores a model of distributed detection called negative detection. In this model, activity patterns are represented by strings of fixed length. Negative detection learns the set of self strings during a training phase, so that anomalous (nonself) data can be distinguished during the test phase. Negative detection systems are comprised of many negative detectors, each represented by a string. Detectors are referred to as negative because they are generated to match nonself strings. In the training phase, detectors are randomly-generated and compared to all self strings. A detector that matches any self strings will be deleted and replaced by another randomly-generated detector. The system is left with a set of detectors that match nonself.

Previous attempts at applying immunology to intrusion detection have considered the issue of false positives to varying degrees. One answer to the problem of false positives was provided by Somayaji and Forrest (2000). Their research applied an automated response mechanism to system call tracing, such that unusual behaviour caused system calls to be delayed exponentially. Unlike previous implementations providing an automated response, where false positives could trigger a disproportionate all-or- nothing response, their implementation meant that occasional false positives had only a negligible impact. Another answer to the problem of false positives was provided by Hofmeyr (1999). Three mechanisms were presented to overcome a high false positive rate. Firstly, activation thresholds were defined to ensure that each detector had to match a certain number of times before an anomaly was signalled. Secondly, because activation thresholds required that a single detector match repeatedly, attacks launched from different sources may not be detected by a single detector. Sensitization was defined to enable detection of a burst of connections from multiple different locations. Finally, costimulation was used to provide a second signal of attack confirmation by a human operator whenever a detector was activated. An improvement on Hofmeyr's (1999) approach was suggested by Qiao and Weixin (2002) and their work is discussed further below.

An artificial immune system model called ARTIS has been developed by Hofmeyr and Forrest (2000) that provides a general framework for a distributed adaptive system which can potentially be applied to many domains. ARTIS incorporates several properties from the natural immune system, including distributed detection, diversity, robustness, implicit policy specification, flexibility, scalability, signature-based detection and anomaly detection.

The abstract model presented by ARTIS can be applied to network intrusion detection by firstly considering an appropriate characteristic to be modelled. Hofmeyr (1999) defined a concrete implementation of the model called Lisys, in which connection information and in particular TCP SYN packet header information is used as an appropriate characteristic. These connections are represented using "datapath triples", which consist of a source IP address, a destination IP address, and a service port. Self is then considered to be all legitimate "datapath triples" connecting either to an internal or external computer, and everything else is nonself. The base representation used by datapath triples is a binary string of length 49. Each detection node runs a local detection system, which consists of a set of detectors. These detectors are randomly generated at each node and are represented as a binary string of length 49. During the tolerization or training period, as each detector is generated, it is compared and tolerized against a single training set using the negative selection algorithm, whereby detectors that match the training set die, and only those detectors that fail to match the training set will survive. The training set consists of all new connections on the network, and is presented to the detectors by way of a broadcast mechanism which uses the Tcpdump line format to represent the connection. Each node that receives a Tcpdump line in this fashion proceeds to extract the datapath triple parameters and compress them into the base representation to allow for easy comparison with the randomly generated detectors. Note that this comparison makes use of approximate string matching using the r-contiguous bits rule. Accordingly, a datapath triple may 'match' a detector even though the two 49- length bit strings are not identical. LANs are assumed to be broadcast, and so the same traffic is seen by all local detection systems. Once the detection system is running, the same traffic is seen by all local detection systems.

Detectors are essentially intended to match nonself or anomalous connections and once training has been accomplished, if the training set was complete, then legitimate network connections will not match any of the generated detectors. However, it is unlikely that the training set will exhibit all self strings, and so it is possible that some strings will survive tolerization only to match self. To overcome this, a costimulation mechanism is adopted whereby a human operator is used to provide a second signal of confirmation. When a detector becomes activated by a broadcast Tcpdump line, a human operator is notified, who is then given a period of time in which to decide if the string really is nonself. If the operator decides that the string represents a connection that is indeed nonself, then a costimulation signal is sent to the detector, promoting it to a memory detector sensitive to any similar datapaths seen in the future.

The lifecycle of a detector can be summarised as follows. A detector is comprised of a randomly generated bit string that remains immature for the length of the tolerization period. If any match occurs during this period, the detector dies and is replaced by a new randomly generated detector. If the detector survives tolerization, it becomes a mature, nave detector with a life expectancy of 1/pdeath time-steps. If the detector accrues enough matches to exceed the activation threshold, it is activated. If the activated detector does not receive a costimulation signal within the costimulation delay period, it dies. If it does receive a costimulation signal, it enters a competition

with other local detectors to become a memory detector. Once a detector becomes a memory detector, it achieves immortality, and will only require a single match for activation (Hofmeyr & Forrest, 2000).

Note that a 'memory detector' is a detector which flags a matching Tcpdump line as anomalous automatically without the need for any costimulation signal for confirmation. The reason is that a memory detector has previously been accepted as a pathogen indicator, having been promoted from the status of a simple (mature or nave) detector to the status of a memory detector precisely because on that previous occasion costimulation had occurred and thus confirmed for evermore the malignant nature of the connection represented by that datapath triple. Note also that memory detectors have a threshold of one, whereas simple detectors will typically have higher thresholds as a guard against false alarms.

3 System Design of Synapse

One of the mechanisms used by Lisys to achieve a low false positive rate is to employ a costimulation signal, whereby a human operator indicates if the connection is truly anomalous. This mechanism has two major shortcomings. The first is that it is not autonomous and accordingly, the accuracy and in particular the success in limiting the frequency of false positives is dependent on the experience and skill of the operator. The second, which follows from the first, is that it is not scalable. What is needed is a system that can detect intrusions and execute an appropriate response independently and automatically. The system design we have adopted for the Synapse system addresses both of these requirements that are needed in order to provide such a system.

The natural immune system acts autonomously in detecting and eliminating dangerous pathogens from the body. There is no outside control or interference in this process. Ideally, the model of distributed detection should mirror this approach, so that the system can detect intrusions independently. For Lisys to faithfully represent the immune analogy, it must incorporate some type of automated costimulation signal.

This general approach was employed by Qiao and Weixin (2002) to successfully reduce the false positive rate. They built an intrusion detection model comprised of detectors and monitor agents. The detectors were based on Lisys, and the monitor agents were used to monitor the state of important subsystems in the network. When a detector is activated, a start signal awakens each monitor. If any of the monitors detects an anomaly, a costimulation signal is sent to the activated detector, causing it to become a memory detector sensitive to any similar datapaths encountered in the future. Qiao and Weixin's (2002) research succeeded in reducing the false positive rate, however their detection system does not build directly upon the existing Lisys implementation and does not consider the issue of response. The present research which has led to the development of the Synapse system defines a new architecture that builds directly on Lisys and addresses the issue of response.

3.1 The Correlation Engine, Automated Costimulation Signals and the Console

The correlation engine is the central component of the new architecture; it coordinates the entire event correlation process within the system. At its most basic level, the correlation engine provides an interface between Lisys detectors and third-party monitor agents. It accepts requests from Lisys to correlate unusual TCP connections with the activity reported by these monitor agents. If the correlation is successful, an automated costimulation signal is generated, fulfilling the overall purpose of the correlation engine. It is important to note that the correlation engine makes the entire costimulation mechanism transparent to the monitor agents. The consequence of this is that the monitor agents themselves do not need to be modified to participate; rather, they simply need to be inserted into the system and have the correlation engine informed of their existence so their output logs can be queried.

The costimulation signal plays a key role in reducing the false positive rate, and its automation reflects the functioning of the natural immune system. The correlation engine described above is responsible for automating the costimulation signal. After Lisys has flagged an anomalous connection, a request is sent to the correlation engine causing it to query each monitor agent in turn. The engine will only generate a costimulation signal to indicate that an anomalous connection is correlated with other anomalous activity as detected by a monitor agent if there is at least one monitor agent that reports behaviour on the system that is consistent with an attack scenario for that connection.

The console presents the administrator with a global view of alerts from all correlation engines on the subnet onto a single display. The information displayed on the console enables an administrator to monitor suspicious events on the network in real-time as they unfold. Such information includes details of unusual connections, outcomes of interrogating monitor agents (including whether a costimulation signal was sent), and instances of memory detector activation.

We note that, consistent with the original Lisys approach, there is one Lisys detector located at each node. As a result, we likewise have a correlation engine deployed at each node. We expect to investigate how deployment of just one Lisys detector at, for example, a firewall, compares with this approach.

3.2 Monitor Agents

Monitor agents are used to retrieve corroborative evidence that helps to determine whether the anomalous connection really is of malicious intent. Often, monitor agents will be no more than third-party IDS systems that are capable of contributing a unique assessment of security-relevant activities for a given host. One beneficial side-effect of introducing such monitor agents is that the detection system now becomes a hybrid system, inheriting all the advantages afforded by the coverage that is found with heterogeneous sensors (Ranum, 2001). To best exploit the concept of coverage in deploying monitor agents, it is important to choose those agents which provide highly complementary strengths and weaknesses. Accordingly, for proof of concept the present research has used two very different IDS systems as monitor agents. The first monitor agent used is Snort, a signature-based network intrusion detection system (NIDS) capable of identifying suspicious packets as they traverse the network (see `http://www.snort.org/`). The second monitor agent used is the advanced intrusion detection environment (AIDE) (see `http://sourceforge.net/projects/aide`). AIDE is a free replacement for Tripwire (see `http://sourceforge.net/projects/tripwire/`), and is an anomaly-based host intrusion detection system (HIDS) that monitors the integrity of important files on the host.

3.3 Response Capability

Intrusion detection research so far has flirted with a variety of response mechanisms, including those that are capable of tearing down connections, killing processes and dynamically updating firewall rules. Despite the range of responses provided, the general strategy has been to respond as early as possible to frustrate the attack in progress. Ironically, it is this drive to respond early that exposes a weakness in the system. Although early response strategies may prevent or at least mitigate an attack before its full impact is achieved, this works against the collection of information for evidence purposes. The present research aims to overcome this limitation by adopting a response strategy that transparently redirects the attack to an isolated host in order to study the attacker's activities. The notion of a honeypot, a host whose value lies in being probed, attacked or compromised, lends itself particularly well to this scenario. Honeypots have become a significant area of research, and are designed to learn the tools, tactics and motives involved in computer and network attacks (see `http://project.honeynet.org/`). By deploying the isolated host as a honeypot, it now becomes possible to both mitigate the effects of the attack while at the same time study the attack itself.

3.4 Summary

Figure 1 illustrates how the system is deployed in a network. The system provides:

- attack or intrusion detection through auto- correlation of Lisys alerts with monitor agent alerts; and
- dynamic attack redirection to an isolated honeypot environment to allow further and safe investigation of the behaviour of the attack.

Note that the 'Client' in the figure is the attacker. On detection of an anomalous connection (e.g., a not-recently-seen connection between 'Client' and 'Server' machines), the correlation engine polls monitor agents and on detecting a costimulation signal from one of the monitor agents it recognizes that an attack is occurring. At this point the correlation engine sends the costimulation signal back to the detection node that originally flagged the anomaly, which then promotes the detector to a memory detector. The correlation engine then causes the client-server session to be hijacked - in a manner which is transparent to the client - to the Honeypot machine where two things happen: the effect of the attack is mitigated; and the attack may be carefully examined.

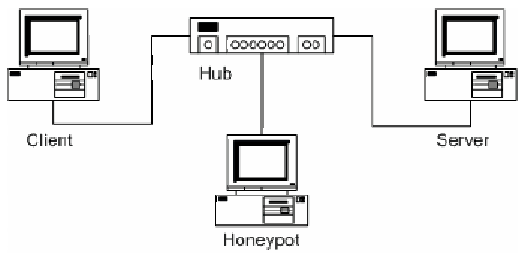

Figure 1: Deployment of the system.

Note also that if a legitimate TCP session is redirected (i.e. as a result of a false positive), the behaviour of the client is dependent entirely on how it is implemented in software. Obviously, just as it is important to minimise responses where connections are terminated, it is similarly important to minimise occurrences of the redirection of legitimate sessions.

4 Implementation of Synapse

The architecture outlined above has been implemented as a software package called Synapse which incorporates and adapts Lisys. In addition to inheriting all the benefits of the Lisys framework, Synapse offers the ability to place an increased level of confidence in diagnoses made. It displays data in real-time indicating all unusual TCP connections found by Lisys, and whether or not monitor agents agreed with Lisys about whether the connection really was unusual.

4.1 The Correlation Engine

As described in the previous section, the correlation engine provides an interface between detectors and monitor agents, and coordinates the event or alert correlation process. The implementation details of the correlation engine are presented in Figure 2. If a detector recognises a novel TCP connection, a signal is triggered to make the correlation engine on the target host interrogate local monitor agents. If any one of the agents detect an anomalous condition, the correlation engine will send a costimulation signal back to the detector. Additionally, the correlation engine sends a report to the console indicating the relevant connection parameters as well as details of the correlation outcome (i.e. whether the detector was costimulated or not). Note that each participating host in the network runs a correlation engine. This is consistent with the model of distributed protection that is found in the natural immune system, and is used so that no single point of failure exists.

In the example in Figure 2, an attacker makes a connection from external host 20.20.20.5 to the FTP service on internal host 131.181.6.133. Recall that all connections made to or from the network are captured in Tcpdump line format and are then broadcast to all other participating detection nodes on the network. Although this process is not explicitly depicted in the example, it can be inferred that the detection node at 131.181.6.142 has received the Tcpdump line representing the attacker's connection. At this stage, the Tcpdump line is converted by the detection node into a 49-bit string resembling the base representation, and is then compared with each detector in the node's detector set. A local detector is indeed activated during this comparison, and a query is accordingly sent to the correlation engine on the target host of the attacker's connection to confirm the anomaly. The correlation engine then requests the status of each monitor agent in turn. If the correlation engine receives any indication of anomalous conditions from at least one monitor agent, it will send a costimulation signal back to the detection node that originally flagged the anomaly. This signal has the effect of promoting the detector to a memory detector, essentially installing a fingerprint against which to match future connections that are similar to those of the attacker's connection.

4.2 The Automated Costimulation Signal

In order to properly understand the changes necessary to automate the costimulation signal, it is useful to look in detail at how the current implementation of Lisys employs this signal. In Lisys, a human operator is used to provide the costimulation signal. There

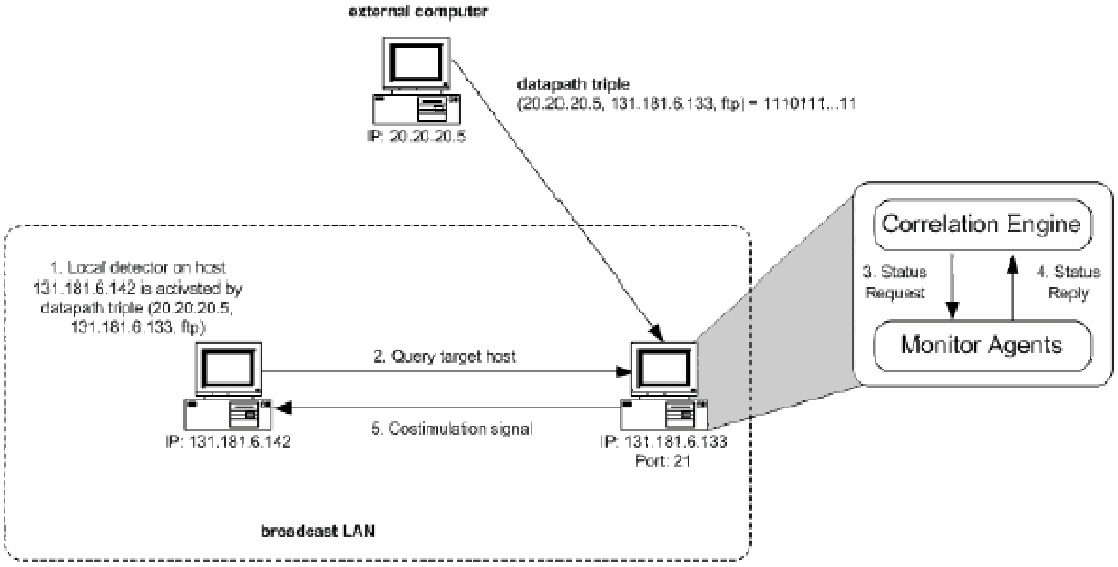

Figure 2: The correlation engine.

are two interfaces by which a human operator can do this: the CGI interface and the web server interface. The CGI interface allows the operator to select from a list of anomalies those datapaths that warrant a costimulation signal. The web server interface fulfils the same purpose as the CGI interface, but instead pushes a list of hyperlinked anomalies to the operator's inbox. A simple web server is then deployed for the sole purpose of accepting any HTTP requests made by the operator, indicating that the current anomaly should be costimulated. For the purposes of the current discussion, we will focus on the web server interface, which is referred to in the Lisys documentation as the costimulation server.

Conceptually Lisys can be considered to consist of two main subsystems: the detection subsystem and the broadcast subsystem. The detection subsystem, which represents the largest slice of the Lisys architecture, is implemented in the NodeServer class (so-called because it wraps around a detection node and exists to serve requests to interact with that node). It is this NodeServer class that is responsible for generating hyperlinks in response to a mature detector exceeding its activation threshold. Periodically, such links are emailed to a specified recipient list. If a human operator clicks on the link, confirming that the Tcpdump line really is an anomaly, the costimulation server will send a costimulation signal back to the node server.

In the implementation of Synapse, the costimulation server is no longer needed. It has essentially been replaced by the correlation engine. Before, the logical flow of traffic was from the NodeServer class, to an email inbox, and then (if clicked) to the costimulation server. Now, the logical flow of traffic is directly from the NodeServer class to the correlation engine there is no intermediary involved.

This has important implications for the change in format of the messages that are now sent from the NodeServer class. There is no need to wrap the information in an HTTP hyperlink, as no human operator needs to click on the anomalous Tcpdump line. Furthermore, the destination address of this information (which used to be the costimulation server but is now the correlation engine) need not be included in the new request format, because it is used as a parameter in creating the TCP socket connection that connects directly with the correlation engine. Finally, the node server address is no longer needed either. The NodeServer class used to attach its own address to the HTTP request so that the costimulation server would know how to contact the node server that flagged the anomalous connection in the first place. In Synapse, the establishment of a socket connection allows the correlation engine to automatically keep track of the peer address.

Until now we have overlooked one important consideration. In the original implementation of Lisys, only a single costimulation server was used, and it remained at a single address. The address of this server was listed in a configuration file, and loaded at runtime. Now, the situation is different. The replacement for the costimulation server, the correlation engine, resides on all participating hosts. What is needed is a way to determine the appropriate correlation engine to connect to. Figure 2 shows that the correlation engine queried is that which resides on the local endpoint of the anomalous connection. This makes sense – after all, what we are trying to do is look for any signs of disruption on the machine that is thought to be involved in an anomalous connection in order to gain a greater sense of whether there is really an intrusion on the machine (i.e. reduce the false positive rate).

Note that it is possible for an anomalous connection to occur between two internal computers. In such an instance, it would be reasonable to connect to both internal computers, and either could then conceivably send a costimulation signal. Importantly, however, only one costimulation signal should be recognized (even if both computers detect anomalous conditions). This is because, due to the way Lisys is implemented, recognizing two identical costimulation signals causes unintended side-effects. Even though both costimulation signals would identify the same Tcpdump line to be used in costimulation, each will be compared in separate iterations with all the detectors in the detector set on the host that originally

flagged the anomaly. The detector that best matches the Tcpdump line in each case will be promoted to a memory detector – resulting in two memory detectors in response to a single anomaly. The current implementation resolves this issue by always connecting to the target of a connection. Figure 3 is a screen shot to illustrate automated costimulation and the information directed to the Synapse console. The first alert indicates that a suspicious telnet connection was detected between hosts 131.181.6.142 and 131.181.6.133. In this instance, however, no monitor agents on the target host found any unusual conditions, and accordingly no costimulation signal was sent. The second alert indicates that a suspicious FTP connection was detected between the same two hosts above. This time, a monitor agent on the target host detected a buffer overflow exploit directed at the FTP server, and accordingly a costimulation signal was sent. The final alert illustrates the utility of this costimulation signal the memory detector created when the signal was sent now alerts to the console whenever it is activated by similar datapaths in the future.

4.3 Interfacing with Monitor Agents

Monitor agents were introduced in the previous section as being the mechanism by which evidence is retrieved to help determine whether the anomalous connection is malicious. Two monitor agents were put forward as being suitable candidates for deployment: Snort and AIDE. We consider here only the Snort monitor agent. Snort offers an array of reporting facilities, and is capable of dumping alerts to both text files and databases, among other formats. The correlation engine interfaces with Snort through its database output log, as this is a simple and efficient way to search for information. After configuring Snort to alert to the database, it becomes a relatively trivial matter to have the correlation engine extract the relevant data using an SQL statement. If this returned data indicates that Snort has also flagged suspicious behaviour for the connection under consideration, the correlation engine will send a costimulation signal back to the node server.

4.4 Response Capability – Attack Redirection

In order to successfully redirect an existing attack session to an isolated host, careful thought must be given to the mechanisms that are required to carry out such a task.

In particular, the nuances of the transport layer protocol in use cannot be ignored. The transport layer protocol is responsible for establishing, maintaining and tearing down sessions; hence, any attempt to manipulate the session must be done within the constraints of such a protocol. The transport layer protocol under consideration in the present research is the TCP protocol. This is because the emergence of TCP/IP as the predominant protocol stack has cemented the use of TCP as the transport layer protocol of choice for reliable, end-to-end data transfer. The underlying mechanics of the TCP protocol do not, under normal circumstances, provide for the case whereby packets in an existing session are redirected midstream to a different destination endpoint. Simple packet redirection at the network layer is not sufficient, because the connection management procedures of TCP would be ignored. In practical terms, the daemon that is listening on the alternative destination endpoint would disregard packets redirected to it because it would not recognise them as belonging to any existing session it has under its control. Even if such a daemon was modified to handle this scenario through low-level raw socket I/O, the scalability of this approach would be limited because each daemon that accepts redirected packets would have to be modified accordingly. One alternative approach to packet redirection is to develop purpose-built software that actively takes over the session from the original daemon so that control can then be passed to the impostor daemon. Such software would have to send spoofed response packets to the attacker that appear to come from the original daemon, and similarly would have to intercept attacker request packets so they could be passed to the impostor daemon. Additionally, it would have to deal with a number of issues that arise as a result of desynchronising the original TCP session. The sections that follow will reveal that session hijacking software is particularly well-suited to dealing with these very issues.

Session hijacking is a technique that is particularly well suited to the goal of attack redirection and is facilitated by a technique commonly known as ARP cache poisoning. Although historically employed by the blackhat community to take over a connection from a client once it has authenticated with a server, the underlying logic found in session hijacking software enables it to perform the operations that will be necessary to redirect an attack. There are many tools available that will perform session hijacking; two of the most popular being Hunt (see `http://packetstormsecurity.nl/sniffers/hunt/`) and JUGGERNAUT (see `http://staff.washington.edu/dittrich/talks/qsm-sec/P50-06.txt`). Hunt was chosen for further development in the present research largely because of its richer feature set and superior extensibility when compared to other open source hijacking tools. Hunt is capable of advanced connection management functions, providing the user with the ability to set connections of interest, watch connections, reset connections, and hijack connections without the proliferation of ACK storms. Hunt can also resynchronise the TCP session with the original client so the connection does not need to be reset.

In order to adapt Hunt to redirect attacks to a honeypot, there are two main design goals that must be achieved. Firstly, Hunt must be able to take over the session from the server, instead of from the client. Secondly, Hunt must be able to provide a proxy interface between the attacker and the impostor daemon. Although these modifications are sufficient to implement attack redirection in isolation, we must integrate the software into Synapse. In Synapse, an anomalous connection is flagged by the correlation engine after it has consulted with monitor agents. For Hunt to be integrated into such a system, the correlation engine must inform it of the anomalous connections that need to be redirected to the honeypot. In this fashion, Hunt can automatically redirect connections that are deemed to be intrusions.

Figure 4 illustrates a proof of concept example where Hunt has been modified to allow a user to manually redirect a specified connection to a third host. A telnet connection is first established to the peer, and the hostname command is issued. At this stage, the user interacts with the modified Hunt to redirect the connection. Now, when the command is issued again, the output reflects the successful redirection of packets to the impostor daemon on the third host. The example is completed with a simple directory listing.

5 Evaluation

To provide proof of concept of the Synapse system of both the automated costimulation and the response mechanism which hijacks the exploit for further examination - we have deployed the well known Bobek

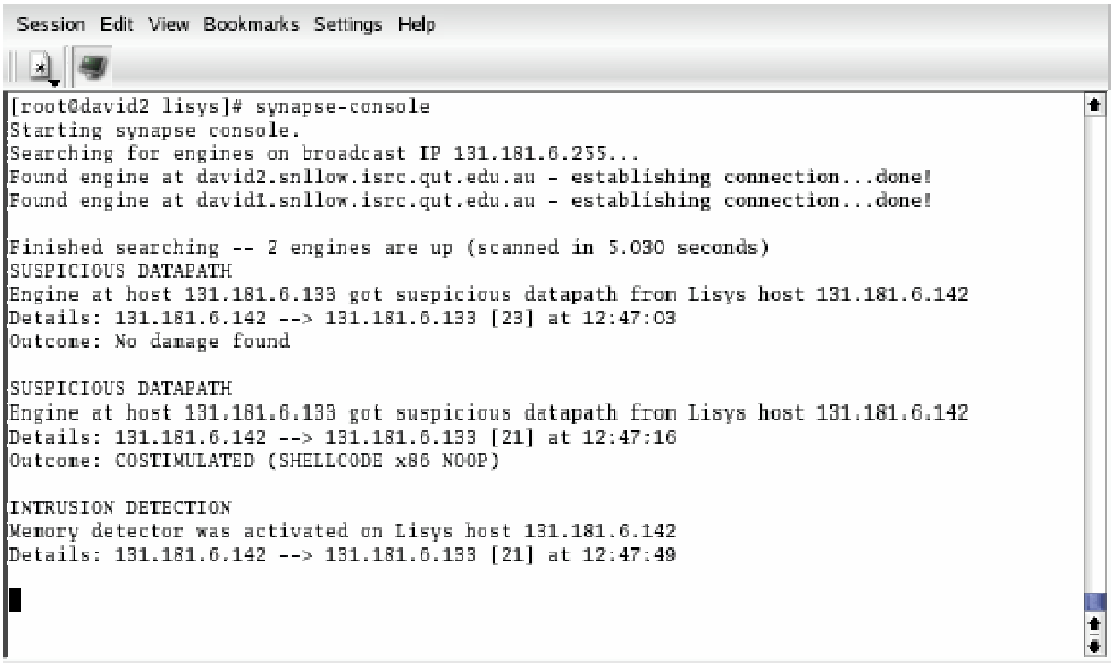

Figure 3: The Synapse console.

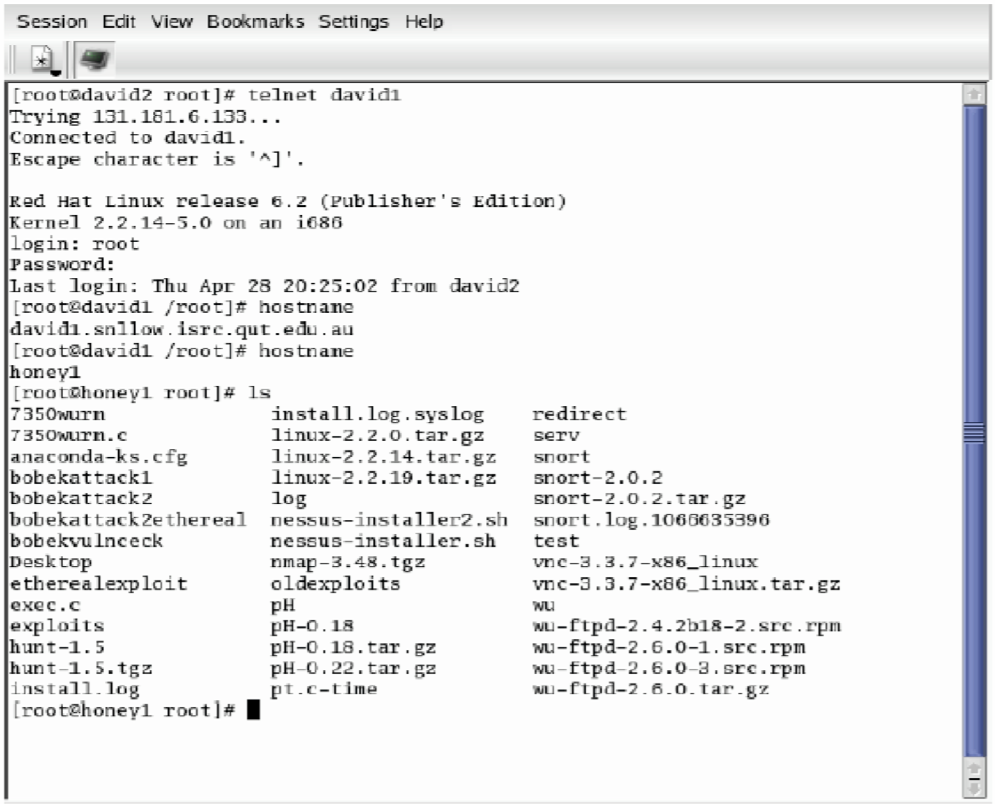

Figure 4: An illustration of client redirection via session hijacking.

attack. The Bobek attack is one of a number of published exploits that takes advantage of a format string vulnerability in the Washington University FTP daemon version 2.6.0. The specific vulnerability is documented in CERT Advisory CA-2000-13 (Havrilla, 2000). It is a remote root exploit that can be triggered using the "anonymous" account. This essentially enables remote users who possess the exploit to gain root privileges on boxes running the affected daemon. The exploit works by injecting specially-crafted character format strings while executing the "site exec" command, causing the return address to point to malicious code loaded by the "pass" command.

In summary, the attack when deployed triggers a Lisys alert to the correlation engine which then leads to polling of Snort. A positive response from Snort then causes activation of the response such that the exploit session is successfully diverted from the server under attack to the so-called honeypot where analysis of the now quarantined or mitigated exploit is then possible.

In more detail, the attack proceeds as follows:

a. the attack is launched

b. a Lisys detection node (i.e. a population of detectors at a host in the network) flags the attack connection as an unusual/anomalous connection, based purely on the datapath triple (source IP + destination IP + destination port). The underlying process that has occurred here is that the Lisys broadcast subsystem, which broadcasts the Tcpdump line of every relevant TCP connection on the network, has broadcast the Tcpdump line corresponding to the attack connection. The detection node receives this Tcpdump line, compresses it into a 49-bit string (the so-called 'base representation' discussed earlier) and compares it to each mature detector in the detector population. (Note that each such detector is considered to represent nonself). Now, the connection corresponding to the attack has matched at least one mature detector, which in turn causes a CORRELATE_REQ message to be sent to the correlation engine on the victim host.

   The word relevant in the above refers to the fact that certain traffic exists for which no stable definition of self can be sought. Specifically, www traffic is filtered out by default because it is acceptable for many new connections to be established to the server each day. Also, all internal Lisys/Synapse related traffic is likewise not considered by the detection system.

c. The correlation engine on the victim host then polls Snort, and finds that Snort has indicated an intrusion has just happened also. The correlation engine then sends the COSTIMULATE message to the Lisys detection node that originally flagged the anomaly.

d. The Lisys detection node that receives the COSTIMULATE message promotes the detector that most closely matches the 49-bit datapath triple (representing the attack connection) to a memory detector.

e. The correlation engine, after having sent the COSTIMULATE message, then immediately sends the HIJACK message to Hunt which is deployed on the honeypot. Hunt then proceeds to take over the connection, and all subsequent interaction is between the attacker and the impostor daemon on the honeypot with Hunt acting as a proxy between the two.

Figures 5, 6 and 7 illustrate some of the network traffic resulting from the above attack and confirm successful detection of the exploit and its redirection. The figures depict a selection of packet frames that illustrate successful redirection of the attacker to the honeypot.

The frames in the figures show that packets have been redirected (there are no packets per se that explicitly confirm successful detection of the exploit). In summary, Figure 5 is the last packet before redirection, travelling from attacker to victim). It allows us to see that (1) the real MAC address of the victim host is 00:50:04:ab:98:ab and (2) that the acknowledgment number of the attacker is 16755.

Figure 6 is the first packet after redirection, also travelling from attacker to victim. It allows us to see that (1) after redirection, the attacker addresses the frame to the nonexistent MAC address ea:1a:de:ad:be:02 (even though the packet is still destined for the victim host's IP address) this shows the ARP spoofing worked, and (2) the acknowledgment number of the attacker is still 16755 (it hasn't received any data yet).

Figure 7 is the second packet after redirection, this time travelling from victim to attacker. It allows us to see that (1) Hunt is successfully spoofing packets back to the attacker as though they are from the real victim notice that, although the source IP address is still the victim's real IP address, the source MAC address ea:1a:de:ad:be:02 is nonexistent and was inserted by Hunt in that fashion. And (2) the sequence number has been correctly spoofed by Hunt as 16755 (equal to the attacker's previous acknowledgment number).

6 Conclusions

The Synapse system provides attack and intrusion detection through auto-correlation of Lisys alerts with monitor agent alerts, and dynamic attack redirection to an isolated honeypot environment to allow further and safe investigation of the behaviour of the attack. Its success in doing so has been demonstrated using the well known Bobek exploit.

Future work will proceed in the following directions:

- Consistent with the original Lisys approach, there is currently one Lisys detector located at each node. We intend to investigate how deployment of just the one Lisys detector at, for example, a firewall, compares with this approach.
- Enhancing the honeypot environment to provide a more accurate rendition of the 'victim' server environment based on honeyd,
- Incorporation of a richer set of monitor agents into Synapse, and
- Enhancing the session redirection:
  - by capturing a window of packets previous to an exploit to allow attack replay; and
  - by providing finer control over preventing two machines from communicating than the present ARP cache poisoning (which has potential to provide denial of service).

7 References

AIDE. (2005). Project Info aide. Retrieved September 5, 2005, from the World Wide Web: `http://sourceforge.net/projects/aide`.

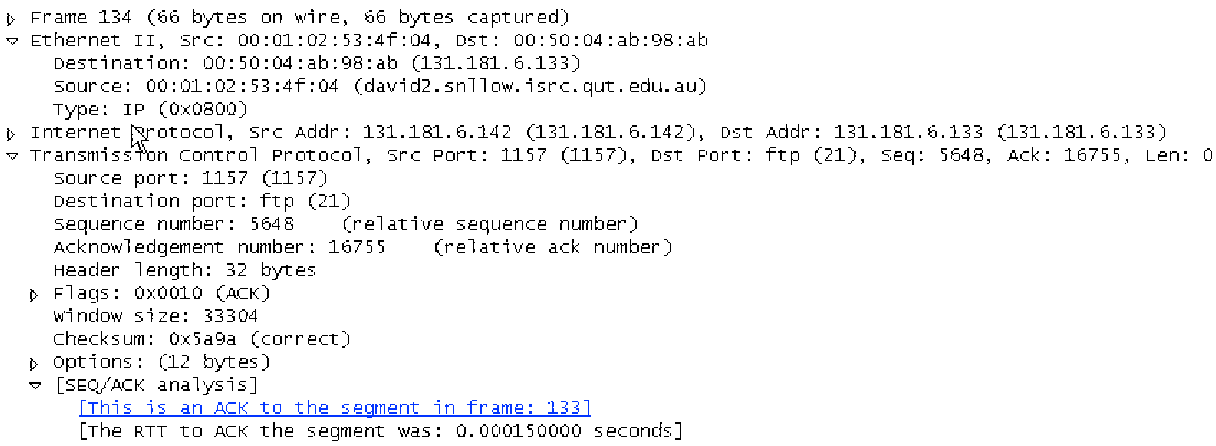

Figure 5: The last packet sent before redirection (from the attacker to the victim).

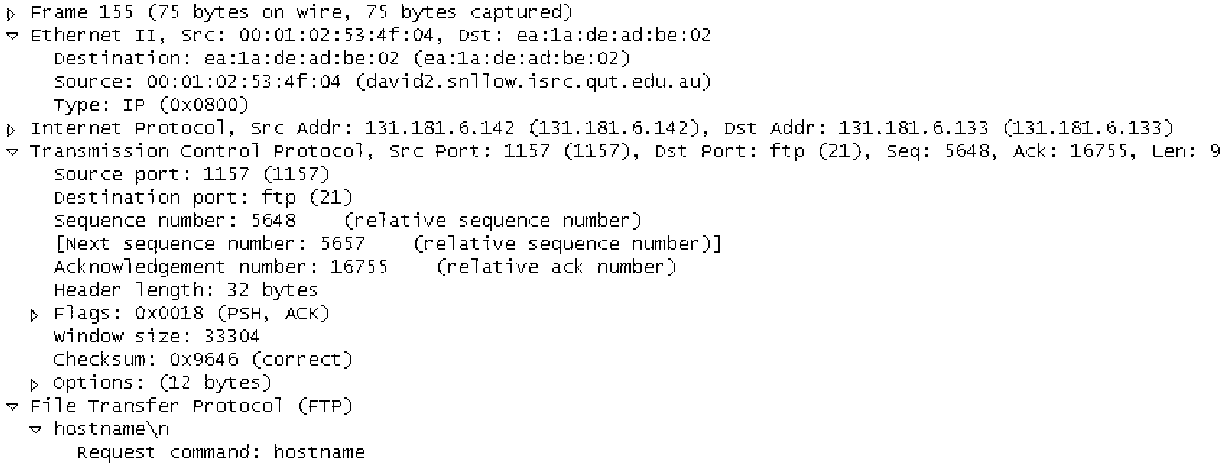

Figure 6: First packet sent after redirection (from the attacker to 'victim'), redirected to the honeypot.

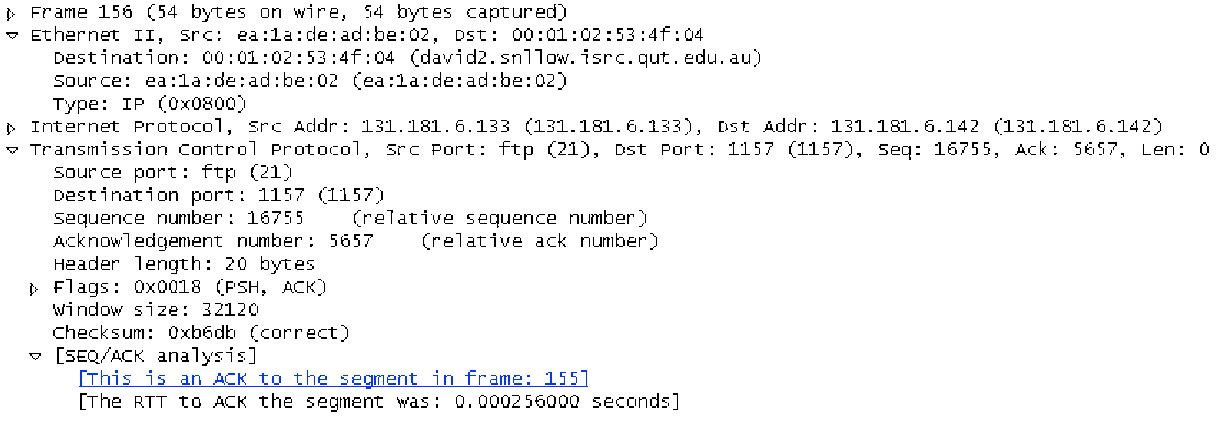

Figure 7: Second packet sent after redirection (from Hunt on the honeypot to attacker).

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

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for Unix processes. Paper presented at the 1996 IEEE Symposium on Computer Security and Privacy, Oakland, CA.

Haines Joshua, Dorene Kewley Ryder, Laura Tinnel, Stephen Taylor (2003). Validation of Sensor Alert Correlators. IEEE Security & Privacy, vol. 1, 2003.

Havrilla, J. S. (2000). CERT Advisory CA-2000-13 Two Input Validation Problems In FTPD. Retrieved September 3, 2005, from the World Wide Web: `http://www.cert.org/advisories/CA-2000-13.html`.

Hofmeyr, S. A. (1999). An Immunological Model of Distributed Detection and Its Application to Computer Security. Unpublished PhD thesis, University of New Mexico, Albuquerque.

Hofmeyr, S. A., & Forrest, S. (2000). Architecture for an Artificial Immune System. Evolutionary Computation, 8(4), 443-473.

Hunt. (2005). Packetstorm back to your roots. Retrieved September 3, 2005, from the World Wide Web: `http://packetstormsecurity.nl/sniffers/hunt/`.

JUGGERNAUT. (2005). JUGGERNAUT. Retrieved September 3, 2005, from the World Wide Web: `http://staff.washington.edu/dittrich/talks/qsm-sec/P50-06.txt`.

McHugh, J. (2001). Intrusion and intrusion detection. International Journal of Information Security, 1:14-35.

Qiao, Y., & Weixin, X. (2002). A Network IDS with Low False Positive Rate. Paper presented at the Congress on Evolutionary Computation, Honolulu, HI.

Ranum, M. J. (2001). Coverage in Intrusion Detection Systems. Retrieved August 15, 2003, from the World Wide Web: `http://www.nfr.com/resource/downloads/Coverage\_in\_IDS.pdf`.

Snort. (2005). Snort the de facto standard for intrusion detection/prevention. Retrieved September 3, 2005, from the World Wide Web: `http://www.snort.org/`.

Somayaji, A., & Forrest, S. (2000). Automated Response Using System-Call Delays. Paper presented at the 9th USENIX Security Symposium, Denver, CO.

The Honeynet Project (2005). The Honeynet Project. Retrieved September 8, 2005, from the World Wide Web: `http://project.honeynet.org/`.

Tripwire. (2005). Project Info Open Source Tripwire. Retrieved September 3, 2005, from the World Wide Web: `http://sourceforge.net/projects/tripwire/`.

Verwoerd, T., & Hunt, R. (2002). Intrusion detection techniques and approaches. Computer Communications, 25(15), 1356-1365.