

ISNIDS

A Network Intrusion Detections System
Inspired by the Human Immune System

John M. Hall
john_hall4@hotmail.com

Abstract

The Immune System Network Intrusion System (ISNIDS) performs negative selection, gene library evolution, and clonal selection to adaptively detect attacks across a network. It also performs simplified immune response. This paper describes the architecture and implementation including the rationale behind design decisions. It also discusses a preliminary assessment of the performance of ISNIDS. Finally, it analyzes the development effort and suggests future investigations. This implementation finds that although there are some substantial research areas, immune system based intrusion detection systems offer significant promise.

Introduction

This intrusion detection system was designed with three attacks in mind. The goal was to detect and respond to these attacks automatically, effectively, and quickly using an immunological approach. These general attacks were doorknob rattling, password guessing, masquerading, and illegal data access attempts. In order to understand these attacks, we discuss specific attack scenarios below.

The first attack is doorknob rattling. This attack is performed when someone outside the network performs a portscan on the server. For the purposes of this ids, we will not consider situations where the scan is distributed or spoofed. To simplify the attack further, the scan will be done fairly quickly. While this is an attack, it is not serious in most environments. The ids should not overreact to this attack.

The second attack is password guessing. In this attack, an attacker has managed to connect to a telnet type service on the server. They are now trying common passwords. To be more realistic, we will assume the attacker knows the usernames to the accounts on the system. This should be considered a serious threat.

The third attack is masquerading. In this scenario, the attacker has managed to either guess or find using other means a username and password. The attacker has also managed to connect to a telnet type service on the server. The attacker is now exploring the system using this knowledge. For the sake of implementing this attack, we will assume the attacker is issuing commands to view the passwd file. This should also be considered a serious threat.

The final attack is illegal data access. In this scenario, the attacker is attempting to locate private files on the server. In our implementation of this attack, this will be done using a web browser. We will look for a predetermined list of potential file names. In our environment this is not considered a serious threat.

Several issues were ignored to simplify development of this system. These limitations would seriously hinder the use of this ids in a production environment. Bandwidth and processing requirements were ignored. The size of objects passed across the network was not considered and the algorithms used for analyzing the data were not optimized. In addition as there was no worst case analysis of these values, a denial of service attack could easily be mounted. The robustness of the system was not fully addressed. For example, when network connections fail, the system does not retry those connections in a reasonable manor, and indeed does not even allow the system to start. The security of the system was not considered. For example, the network connections were not encrypted. Finally many configurable variables were hard coded into the executables.

Architecture

ISNIDS is a network ids that performs anomaly detection using immune system methods. This system was separated into two components. The primary and the secondary ids. These components communicate across the network. The primary ids is centralized and is responsible for creating detectors. This is done using negative selection. In order to adapt to new attacks, the primary ids also evolves its gene library. The secondary ids is distributed and is responsible for data gathering, data reduction, detection, and response. It also forwards successful detections to the primary ids to perform clonal selection. This architecture is similar to the artificial immune model proposed by Kim and Bentley [9]. Their model offered significant promise. This system is intended to show the results of actually implementing such a system. It also show the practical concerns that need to be taken into account.

In terms of the human immune system, the centralized primary ids represents the thymus and to some degree the bone marrow. The decentralized secondary ids represents the mobile components of the immune system. In particular the secondary ids represents the flow of immune system detectors throughout the body.

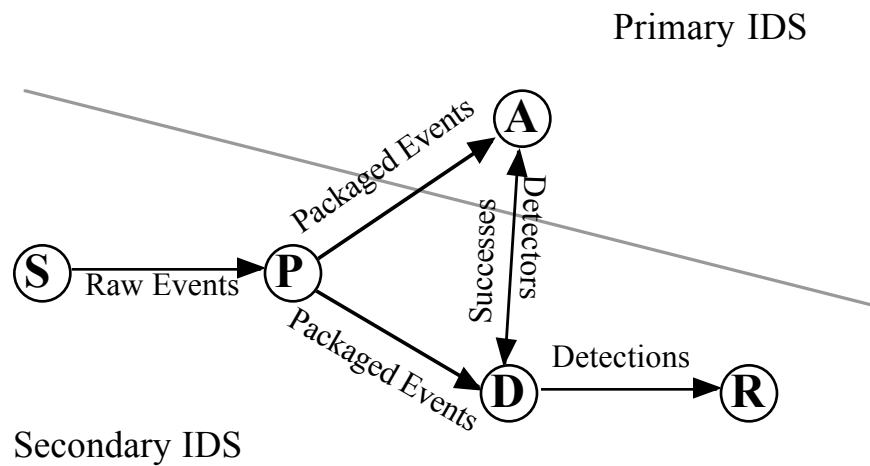


Figure 1. Overview of ISNIDS

The architecture of ISNIDS is shown in figure 1. The secondary ids consists four components, the sensors, the packager, the detector, and the response. The primary ids consists of only an analysis component. The sensors collect audit information and convert it to a common event format. The packager performs data reduction by grouping the events into sessions. The analysis component uses these sessions to create detectors. The detector matches current sessions to its detectors. Finally, the response component automatically responds to attacks. Ideally, once the secondary ids had a set of detectors, it could continue to function if the primary ids failed.

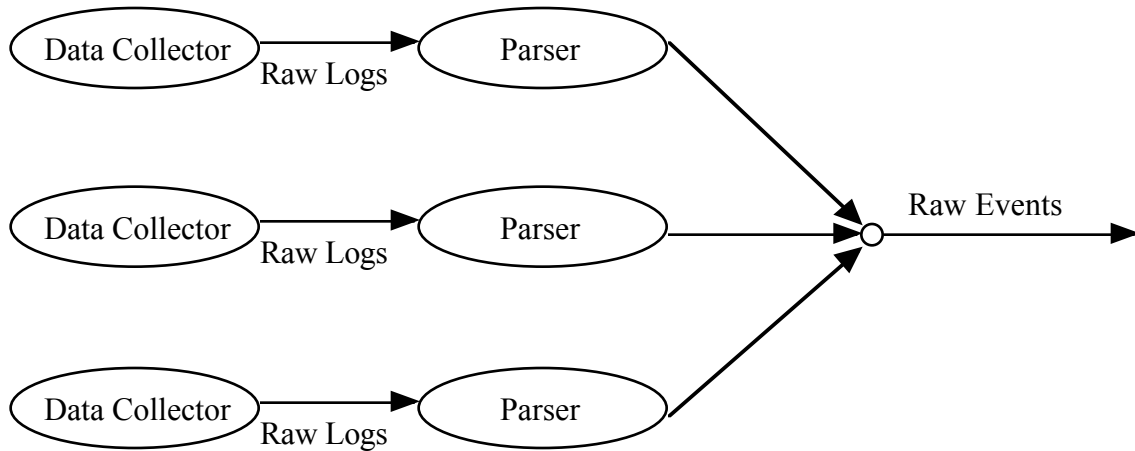


Figure 2. Design of Sensor Component

The sensors component shown in figure 2 reads the raw audit data. It then parses this information into a common event format. The audit data is gathered from several sources. The first is the firewall output. This includes all packets dropped by the firewall. The second data source is the standard messages log. This log includes login successes and failures. The third data source is the standard secure log. The incoming connections information is gathered from this. The fourth data source is the Apache access log. Information about web results is gathered from this. The fifth data source is the Apache error log. Information about file, access, and protocol web errors is gathered from this log. The final data source, which was not implemented in this version, is from a raw network packet dump. Ideally this would include information about every packet on the network.

The common event format includes a condensed version of the log information. The format includes the source IP address, the destination IP address, the username, the network protocol, the time-to-live, the service involved, the authentication result, the HTTP result, and the HTTP error. It is possible that any given field will not have a value. These are the same attributes that will eventually be used in our detectors.

In the human immune system, these events represent portions of antigens. These events are like individual proteins inside of an antigen. In the human immune system, these proteins are naturally grouped to form the entire antigen. In the context of an ids, these proteins need to be reconstructed to form the entire antigen.

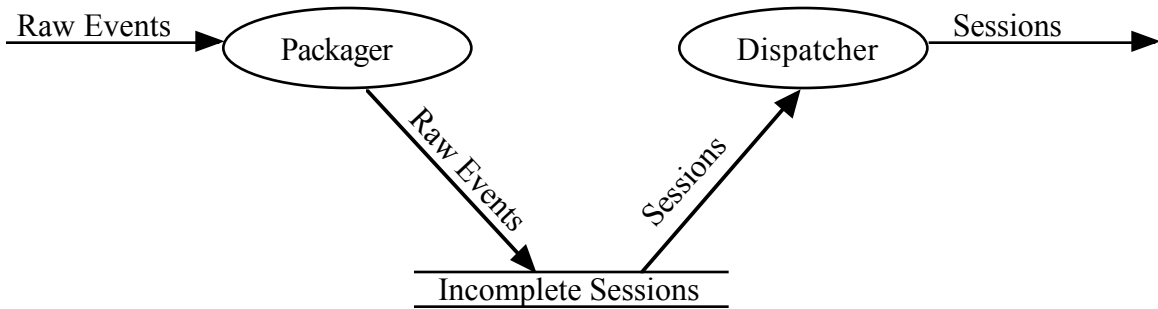


Figure 3. Design of Packager Component

The packager component shown in figure 3 performs data reduction by grouping events into logical sessions. The ideal way to do this is an open research issue. In our system, we merge events within a given time frame from the same user or source IP. We also merge events that occur sufficiently close together. The session generated is actually a summary of all the attributes in all the events. The dispatcher will forward sessions that are either completed. However, it will also forward sessions at regular intervals as they are being constructed. This ensures that an attack doesn't escape detection simply because it has not ended.

In the human immune system, these sessions represent the surface of our antigens. The proteins inside of the antigen can be detected from outside, but many of the details about how they are organized is lost. This simplified view of the antigen is what makes matching in both the human body and in ISNIDS possible

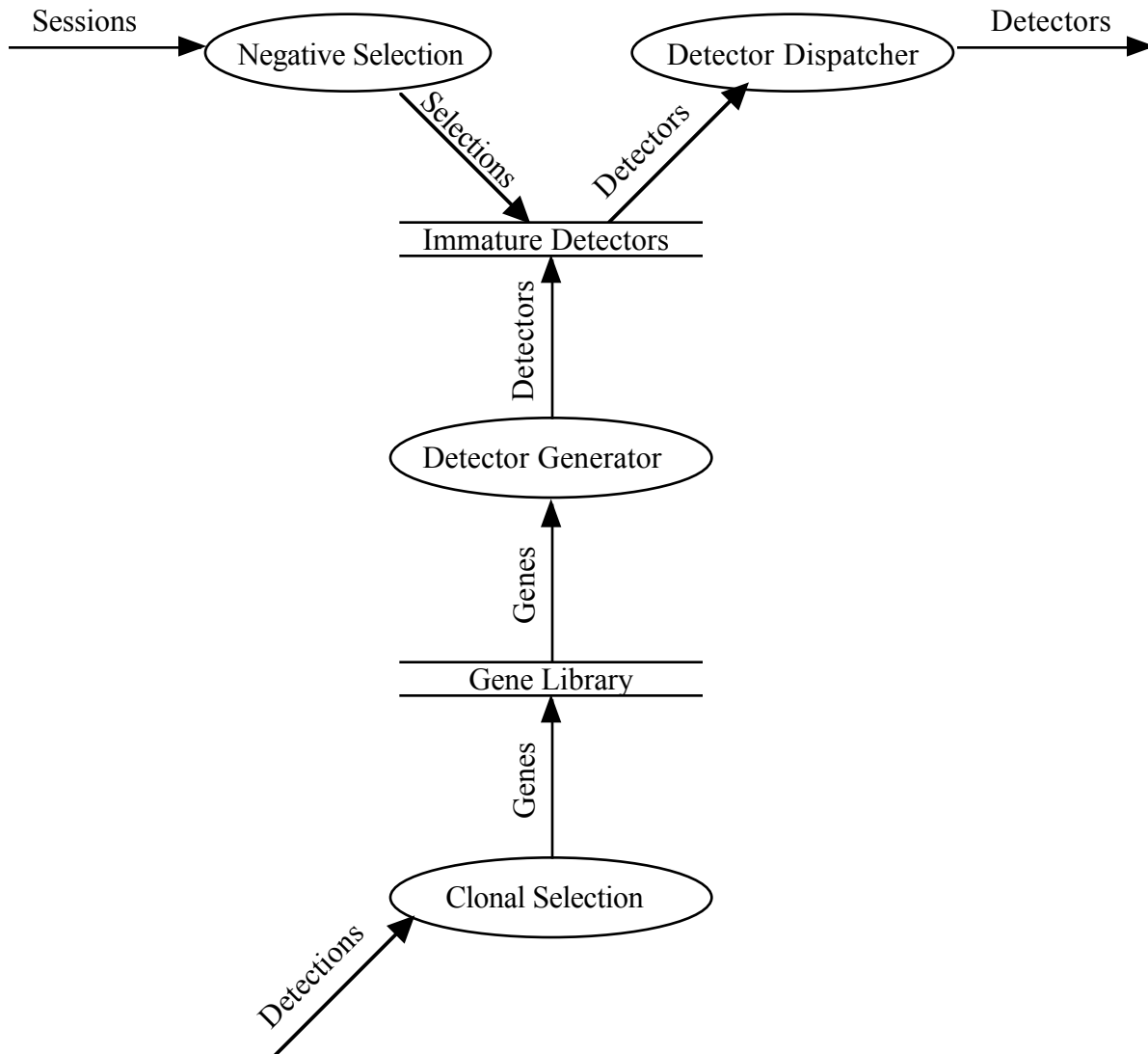


Figure 4. Design of Analysis Component

The analysis component shown in figure 4 is the most complicated component in our system. Its purpose is to generate detectors. The detectors contain genes to match particular instances of the attributes in sessions. In addition, detectors perform approximate binding. The analysis takes all the sessions generated in the packager component in each secondary ids. We use these sessions to perform negative selection on the immature detectors. The full implementation details of the negative selection process are discussed in the implementation section. When a detector matures, it is passed back to the detection component in one secondary ids. When a secondary ids successfully matches an antigen, it passes the session and the detector that matched it to the clonal selection component. The analysis component adds the individual genes from the detector and the attributes from the session to the gene library and increases their fitness. This causes the gene library to evolve to generate better detectors. These detectors are generated in such a way that any detector may match attributes from any or all of the data sources.

The analysis component is similar to the thymus and bone marrow in the human immune system. The bone marrow generates detectors from the gene library. Self cells are passed through in a negative selection process. T-Cells move to the thymus to mature. Here they encounter more self cells and continue the negative selection process. Detectors that survive negative selection are released into the body to detect pathogens. The human body also performs clonal selection by promoting the growth of detectors that match pathogens.

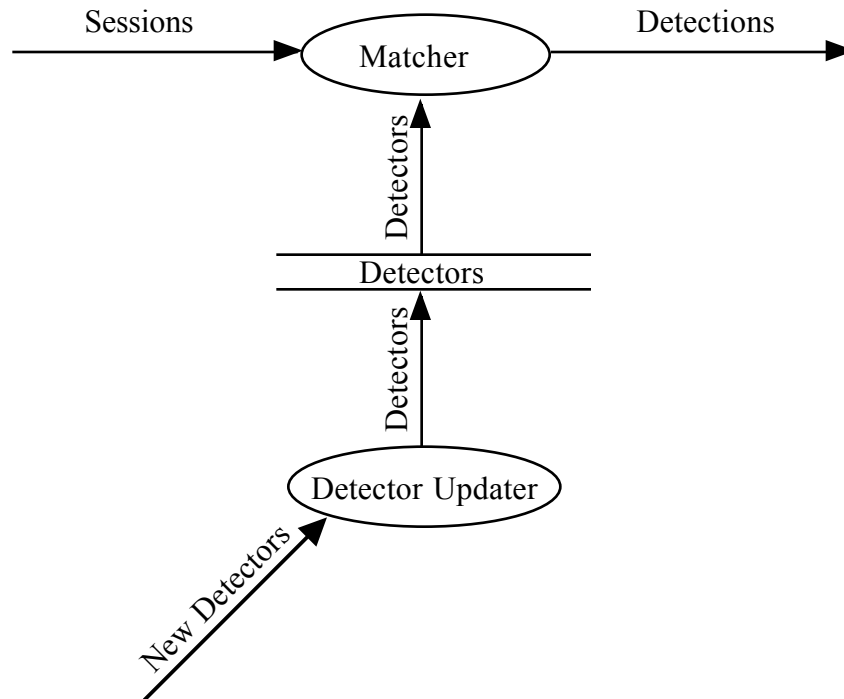


Figure 5. Design of Detection Component

The detection component matches sessions and detectors. The detectors are received from the primary ids. When a detection is made, the session and the detector that detected it are sent to both the primary ids and to the response component. This component differs from the artificial immune system [9] in that detection is done on the exact same session data as analysis.

In the human immune system, detection is done throughout the body. The detectors generated in the bone marrow and thymus circulate the body trying to match pathogens. The primary difference between the immune system and this detection scheme is that the antigens are brought to the detectors instead of the detectors to the antigens.

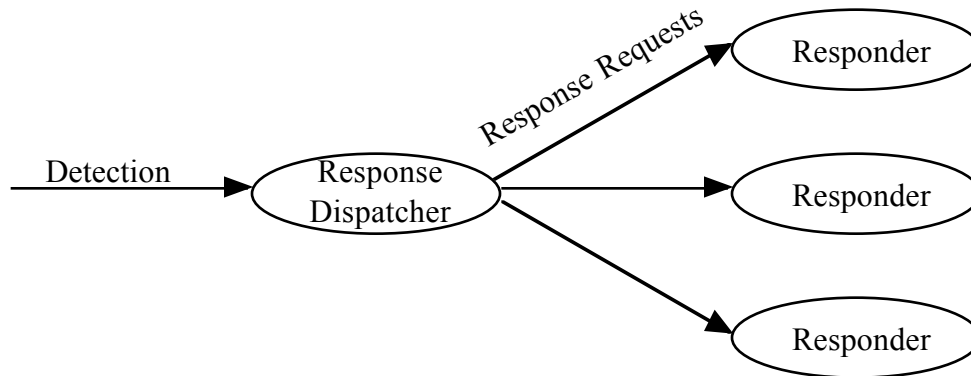


Figure 6. Design of Response Component

The final component in ISNIDS is the response component. The response dispatcher chooses between the available responses. In this version, the response dispatcher will simply choose the lowest impact available response that is appropriate for the given detection. This will allow the system to try the least impact response first. If the problem is not solved, the system will continue to try responses until the attack is no longer seen. Hopefully this means the attack, and not the ids, has been defeated. This means that any response that has the potential for fixing the problem will be tried. The responses chosen for this system are tracing the source of the attack, disable services using the firewall, blocking IP addresses at the firewall, and shutting down the system. The other responses considered for future systems are increasing logging, emailing a warning, increasing the system's sensitivity, and disabling user accounts.

The chosen responses map to the considered attacks. These attacks would be handled differently in the context of a military, a commercial, and a student project ids. The first attack, doorknob rattling, would likely be handled preemptively in a military context. The system would probably trace the source of the attack and retaliate with a mild computer based counterattack such as a denial of service attack. In a commercial system, this attack would probably not be considered a serious threat. Likely responses include tracing the source using more legal means, and blocking addresses at the firewall. Our system includes both of these recommended commercial responses.

The second chosen attack was password guessing. In a military context, this is probably considered a serious threat. An ids would probably consider shutting down non-critical systems. It would probably trace the source of the attack, disable the attacked user's account, and probably also retaliate with both computer and possibly even non-computer counterattacks. A commercial system would probably trace the source of the attack, block addresses at the firewall, and possibly disable services. It might also disable the attacked user's account. Our system includes the first three of these recommended commercial responses. Disabling user accounts was not included in this version.

The third attack, masquerading, indicates that the system has been compromised. In a military context would consider this a very serious attack. An ids would probably shut down non-critical

systems, and perhaps even some more critical ones. It would trace the source of the attack and disable the attacked user's and perhaps some other similar users' accounts. It would probably retaliate with severe computer countermeasures. In the extreme case, it may completely lock down this system. It would also probably recommend retaliation using non-computer means. A commercial system would also consider this a serious threat. The ideal responses would be tracing the source of the attack, disabling the user's account, blocking addresses at the firewall, and in the extreme case it may also completely lock down the system. Our system includes all of these commercially recommended responses except disabling of user's accounts.

The fourth attack, illegal data access attempts, can be considered serious attacks in some environments and doorknob rattling in others. In the military context, the severity of these attacks is probably considered somewhat severe. A military ids would likely secure sensitive data by either encrypting it, moving it to an offline location, or perhaps even by destroying it. Depending on the severity of the attack the ids may retaliate by attacking the source of the attack. In the extreme case, it may lockdown the system. A commercial ids would probably handle this attack by tracing the source of the attack. It may also try to block the attack at the firewall, or disable services used to access the data. In an extreme case, this system may also lockdown the computer. Our system includes all of the commercially recommended responses.

Implementation

The initial release of ISNIDS is available at <http://www.uidaho.edu/~hall0393/isnids.tar.gz>. It contains 13,679 raw lines of code in 224 files. The archive is approximately 1.4 MB. This includes all the training and test data. The code is designed to compile without warnings under BSD (Specifically MacOS X version 10.2), but should compile under other unix variants with minimal changes.

To configure the ids to work in your network, you will need to edit the "common/include/IdsSettings.h" file to include the IP address of where you will run the primary ids. To build the entire project run "make" from the root directory. The build is configured to compile debug versions of the projects. To ignore debug asserts, type 'r' when running an executable. The build will create a "primary/source/primary.exe" and a "secondary/source/secondary.exe". These two executables will be linked to from the test directory. The build will also create an "a.out" test program for each component.

The primary and secondary ids should be run from the "test/primary" and "test/secondary" directories. To process saved events, update the links (.log files) in the "test/secondary/test" directory to point to the saved files. To process live events, update these links to point to fifos created with mkfifo. Configure your system to write the audit events to these queues.

The implementation of several components warrant additional discussion. The packager component was originally missing from the architecture. Early experiments indicated that raw network events did not contain enough differences between attacks and normal behavior for the

ids to perform effectively. The packaging method used should be investigated further. The current implementation will group events with the same user within a long time frame together. Events from the same address are grouped together if they occur within a shorter time of each other. Finally, an event is added to a group if it occurs within a short time of other events. Overlapping groups are merged. This method would not work correctly in a busy network environment.

Another implementation detail of interest is how the genes in detectors match sessions. All genes can match every value and nonexistent values. They can also match some set of values. In this implementation, this set is typically determined by a mask. The mask specifies how many bits of the value must match for the detector to match. A detector will match a session if the session has any values that match each gene in the detector. This approximate binding scheme allows our detectors to match a large number of sessions.

One of the largest problems facing the implementation of an immunological based ids is negative selection. In the human body, this is a massively distributed process. Kim and Bently [10] found that generating a detector from network data was a computationally infeasible operation. They gave up on their experiment when after 24 hours they had failed to generate a detector with five genes that could survive negative selection. ISNIDS solves this by setting the masks of all new detectors to their most general values. Every time negative selection finds a detector that matches a portion of self, the mask of a random gene is made more specific until it no longer matches. A detector will only be discarded if it exactly matches a set of values in a session.

Primary IDS

```
CDetectorIncubator::kTimeUntilMature=4096  
CDetectorIncubator::kDesiredNumberOfDetectors=8192  
CGeneLibrary::kMaxAllelesPerGene=4096
```

Secondary IDS

```
CDetectorSet::kMaxDetectorSetSize=2048  
CIncompleteSession::kSessionTimeForAllEvents=20s  
CIncompleteSession::kSessionTimeForSameSourceEvents=6 min  
CIncompleteSession::kSessionTimeForSameUserEvents=20 min  
CIncompleteSession::kTicksUntilView=15-20 s  
CIncompleteSession::kTicksUntilDelete=10 min
```

Figure 7. Chosen ISNIDS Constants

There are many constants embedded in the system that influence how ISNIDS works. The most interesting of these are shown in figure 7 and discussed here. `kTimeUntilMature` is the number of iterations of negative selection that a given detector must survive through in order to mature. This was originally set far too low. In a busy network, this should be set higher.

`kDesiredNumberOfDetectors` is the number of immature detectors that can exist at the same time.

This value is set to a large value to support a larger number of secondary ids. In a large network, this should be set even higher. `kMaxAllelesPerGene` is the number of values we will save for every gene. Because gene fitness is represented by including the gene multiple times in the database, this value should be large. A more complex network should use a larger value. `kMaxDetectorSetSize` determines how many detectors the secondary ids will use. To improve the performance of the secondary ids, use a smaller value. To decrease the number of false negatives, use a larger number. `kSessionTimeForAllEvents` is the time we allow between events that otherwise seem unrelated to be grouped into the same session. In a busy network this should be set smaller. If it does not make sense in your environment to group these events, this value should be set to 0. `kSessionTimeForSameSourceEvents` and `kSessionTimeForSameUserEvents` are similar values used when events have the same source IP address or same user. `kTicksUntilView` is the time after a session is modified until it is sent to the rest of the system. This value should be set lower in more critical applications. Finally, `kTicksUntilDelete` is the time we keep an incomplete session before we consider the session complete.

- 0 impossible (will not block this attack)
- 1 system completely disabled
- 2 network completely disabled
- 3 network disabled
- 4 all network services disabled
- 5 some network services disabled
- 6 reserved
- 7 some communications blocked
- 8 some mild external behavior (ie: trace)
- 9 some internal notification (ie: email admin)
- 10 no significant impact (ie: increase tracing)

Figure 8. Response Impacts

- 0- Normal (Convenient loose firewall)
- 1- Strict (Extra checks on user traffic)
- 2- Partially Disabled (Some services off)
- 3- Fully Disabled (All services off)
- 4- Locked (All network traffic dropped)

Figure 9. Firewall Levels

The actual implementation of the responses is also interesting. As mentioned above, the response dispatcher will choose the response with the least impact. The different levels of impact are shown in figure 8. The impact of responses will change over time based on the history of that response. The firewall response for example has several levels based on the current state of the firewall. The responses will also remember previous actions. For example, if the trace response has recently traced an IP address, then it will not consider itself a valid response if the same address is detected again. The system shutdown response is configured to have the most impact.

Assessment

Any immunologically based ids such as ISNIDS should be good at detecting new attacks and learning from previous attacks. ISNIDS performs distributed data gathering, detection, and response. As a result, performance should be reasonably good overall, though there is more load on the individual hosts than a more centralized method. The division of the ids into two components, a primary and a secondary ids, gives the ids itself some resistance to attack. Once the secondary ids have received detector sets from the primary ids, they can continue to detect attacks even if the primary ids fails or is attacked. The secondary ids is distributed so an attacker would need to disable all of these to successfully subvert the system. This would be especially true if the raw network sensor had been implemented.

There are several types of attacks that ISNIDS is not likely to perform well against. These include “low and slow” attacks and many denial of service attacks. Most portions of a low and slow attacks are likely to be packaged in different sessions. As a result it is likely there will not be sufficient information in each session to distinguish the attack from normal system usage. This is similar to the problem detecting some types of denial of service attacks. If the attack is flooding the system with a large number of what look like normal sessions, the system will not detect the increased quantity of this traffic.

We tested ISNIDS using two different approaches. We tested for false positives by allowing the ids to run on actual events from our network. The ids output for the first 30 detections is shown in appendix A. The ids detected a total of 444 attacks. We believe 5 of these are actual detections are shown in appendix B. A subset of the detections were manually checked by looking through a copy of the audit data. The second portion of our testing procedure was for false negatives. We tested this by manually running two variations of the attacks described above. Prior to these tests, we trained the ids with 1 months training data. Appendix C shows some interesting detectors generated. Appendix D attempts to show what the typical detectors looked like by showing the first 24 generated.

The false positive test used an additional two and a half weeks of audit data. Interpreting the detections in appendix B, the attacks detected appear to mostly be doorknob rattling attacks. The false positive rate appears low. We do not believe any of the shown entries are false positives. However, as we mentioned above, we believe 5 of the detection were. This is a large number of attacks, but these are mostly true positives.

	Attack	Packaging Time	Detectors Matched	Time to Detect
1	Nearby Portscan	1:20	14	1:42
2	Remote Portscan	1:20	15	1:43
3	Password Guessing Through Tunnel	:20	4	:20
4	Password Guessing From Inside	:20	-	-
5	Masquarding Through Tunnel	:20	-	-
6	Masquarding From Inside	:20	9	:20
7	External Attempts to Locate Private Data	:20	1	:20
8	Internal Attempts to Locate Private Data	:20	3	:21

Figure 10. Test results.

Figure 10 shows the results for our 8 false negative tests. ISNIDS matched 6 of the eight attacks. In most cases, many detectors matched each attack. Since an immunological ids performs probabilistic detection, we can look at this number as some indicator of how likely it is that another instance of the ids will also detect the attack. It is also interesting to note that packaging time is responsible for the majority of the detection time. Perhaps, the dispatch rate should be increase to detect attacks more quickly.

Forensics was not a large consideration during design and development of ISNIDS. The ids does output most information about the sessions, detectors, and detections created. However, while deciphering this input is easier then deciphering the raw audit events, it is not as simple as it could be. In addition, this versions of ISNIDS does not perform traceability. The ideal ids would allow every data item to be traced back to its source. Using this traceability would allow an administrator much more information about an intrusion. In fact, it may even allow the administrator (or an expert witness) to understand why the ids decided that there was an attack.

While ISNIDS was not designed for performance, its performance seems adequate for a small network such as mine. Nonscientific time trials indicate that ISNIDS can handle about 1000 events per minute when the primary and secondary ids are on the same machine. This is more than reasonable in an environment like mine where approximately 500 events are generated per day and there is little concern about a denial of service attack. Some optimizations will be necessary before ISNIDS is ready for a production environment.

Observations and Future Work

This project obviously taught me quite a few things about immune system based ids. This section is a short summary of the issues and successes encountered in this project. Many of

these issues lead to future work.

Gene selection was a difficult problem. The current architecture also makes it very difficult to add or remove genes. Simplifying this process is necessary before we can easily compare the effectiveness of different genes. It would be very nice if we could design a common representation for a gene. This would also seem to make the ids more like the human immune system in that the human immune system represents every detector as a string of proteins.

Training was important. Originally, only about 2000 sessions were used to create the detectors. This was too few. This original detector set matched almost two-thirds of other sessions. I tried some less than optimal solutions including requiring multiple detectors to match. This is not a good long term solution. The current training set consists of around 4000 sessions. Along with other changes made, the detection rate is much more reasonable. This does have the side effect of training the system to expect more anomalous behavior as normal.

Diversity in the gene library is essential. The original gene library was built from the training data. This resulted in detectors that matched almost no detections. Based on this library, only 2 of the false negative tests above passed. The initial generation of the library is not optimal. My current belief is that to detect novel attacks well, it needs to include more randomness.

Currently ISNIDS has a centralized analysis component and decentralized detection components. I believe it may make sense to add a centralized detection and a decentralized analysis component. I can't quite draw the parallels of a centralized detection component to the human immune system, but such a component would help the system to detect global attacks much better. The decentralized analysis component, on the other hand, resembles the bone marrow. We could even build a two layer hierarchy of analysis components. The top level would create t-cells and the middle layer would create b-cells. The lower layer would still perform detection. I found the audit reduction step of building sessions very effective. If we can find a logical way to group sessions, then the top layer ids could build detectors for those more generic groups. We could also choose which sessions were passed all the way to the top layer randomly, or by how well each sessions performed during negative selection.

As I mentioned, I found that building sessions worked well. However, there may be ways to do better. We condensed 15,071 log entries to 3394 events to 2618 sessions in our original training data. To see whether the same group was sent out multiple time while it was being created, I increased the time I waited before dispatching a session. This resulted in 2476 sessions. I concluded that this was not a major factor. I returned the dispatch time to its original value and tried increasing the times we used to group similar events. I figured this would allow us to determine if there were a large number of small groups. This resulted in 2019 sessions. I assumed the cause of small sessions was that the sessions were distinct and should not be merged. Therefore I believe there are a large number of small sessions generated. This seems especially true in a system like mine where there are few legitimate users.

The response mechanism currently implemented in ISNIDS is not as good an immune response as it could be. In an immune system context, the current mechanism implements the two standard types of response: weaken the intruder and strengthen self. It also implements global and local responses. However, the system does not classify them by these attributes. Another weakness of the current implementation is that choosing an appropriate attack does not take into account whether the attack is still in progress. The current mechanism also does not allow responses to be cooperative between ids.

The negative selection algorithm used was very effective at creating detectors. With early versions of the gene library, about 7 detectors were destroyed for every 1000 that matured. Later versions of the gene library lowered that to less than 1 for every 4000. That being said, using masks with counting values does not seem to work as well as it does for other values. the problem is that most of the sessions contain small values for these. As a result, the mask size is quickly reduced. I considered a range type gene with more flexible bounds than a mask offers, but did not implement it in this version.

Appendix A

ISNIDS output when run on events from November 8th to November 26th. (First 30 detections)

Setting Firewall to level 0 with 0 blocked IPs.

Detection matching 15 detectors.

We've found 1 anomalous events.

Invoking Response!

Increasing the sensitivity of the ids.

Detection matching 12 detectors.

We've found 2 anomalous events.

Invoking Response!

Invoking firewall response!

Setting Firewall to level 1 with 0 blocked IPs.

Detection matching 15 detectors.

We've found 3 anomalous events.

Invoking Response!

Starting Trace response.

Tracing IP: 211.125.183.3

Detection matching 15 detectors.

We've found 4 anomalous events.

Invoking Response!

Starting Trace response.

Tracing IP: 209.51.163.99

Detection matching 15 detectors.

We've found 5 anomalous events.

Detection matching 13 detectors.

We've found 6 anomalous events.

Invoking Response!

Starting Trace response.

Tracing IP: 195.120.133.123

Invoking Response!

Starting Trace response.

Tracing IP: 62.181.67.44

Detection matching 11 detectors.

We've found 7 anomalous events.

Detection matching 11 detectors.

We've found 8 anomalous events.

Detection matching 16 detectors.

We've found 9 anomalous events.

Invoking Response!

Starting Trace response.

Tracing IP: 63.148.99.247

Invoking Response!

Invoking firewall response!
Setting Firewall to level 1 with 1 blocked IPs.
Invoking Response!
Starting Trace response.
Tracing IP: 207.6.95.231
Detection matching 16 detectors.
We've found 10 anomalous events.
Detection matching 13 detectors.
We've found 11 anomalous events.
Invoking Response!
Invoking firewall response!
Setting Firewall to level 1 with 2 blocked IPs.
Invoking Response!
Starting Trace response.
Tracing IP: 61.188.179.36
Detection matching 18 detectors.
We've found 12 anomalous events.
Detection matching 15 detectors.
We've found 13 anomalous events.
Detection matching 16 detectors.
We've found 14 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 203.93.204.71
Invoking Response!
Starting Trace response.
Tracing IP: 210.136.6.27
Invoking Response!
Starting Trace response.
Tracing IP: 200.171.236.99
Detection matching 16 detectors.
We've found 15 anomalous events.
Detection matching 12 detectors.
We've found 16 anomalous events.
Detection matching 13 detectors.
We've found 17 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 200.165.203.39
Invoking Response!
Starting Trace response.
Tracing IP: 65.173.60.5
Invoking Response!

Starting Trace response.
Tracing IP: 12.252.125.14
Detection matching 12 detectors.
We've found 18 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 61.222.96.196
Detection matching 14 detectors.
We've found 19 anomalous events.
Detection matching 17 detectors.
We've found 20 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 210.78.17.189
Invoking Response!
Starting Trace response.
Tracing IP: 211.253.110.214
Detection matching 12 detectors.
We've found 21 anomalous events.
Detection matching 9 detectors.
We've found 22 anomalous events.
Detection matching 15 detectors.
We've found 23 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 61.222.247.138
Invoking Response!
Starting Trace response.
Tracing IP: 129.101.171.14
Invoking Response!
Starting Trace response.
Tracing IP: 211.191.40.205
Detection matching 12 detectors.
We've found 24 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 65.43.136.11
Detection matching 12 detectors.
We've found 25 anomalous events.
Detection matching 18 detectors.
We've found 26 anomalous events.
Invoking Response!
Starting Trace response.

Tracing IP: 61.159.250.73
Invoking Response!
Starting Trace response.
Tracing IP: 203.199.92.19
Detection matching 12 detectors.
We've found 27 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 143.246.86.213
Detection matching 15 detectors.
We've found 28 anomalous events.
Detection matching 15 detectors.
We've found 29 anomalous events.
Invoking Response!
Starting Trace response.
Tracing IP: 218.108.179.159
Invoking Response!
Starting Trace response.
Tracing IP: 218.76.94.34
Detection matching 15 detectors.
We've found 30 anomalous events.

Appendix B

ISNIDS detections log when run on events from November 8th to November 26th.

Event 1 (Sat Dec 7 06:34:55 2002): SESSION Source: 216.127.20.40 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 2 (Sat Dec 7 06:34:55 2002): SESSION Source: 24.123.109.7 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 3 (Sat Dec 7 06:34:55 2002): SESSION Source: 211.125.183.3 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 4 (Sat Dec 7 06:34:56 2002): SESSION Source: 209.51.163.99 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 5 (Sat Dec 7 06:34:56 2002): SESSION Source: 195.120.133.123 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 6 (Sat Dec 7 06:34:56 2002): SESSION Source: 62.181.67.44 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 7 (Sat Dec 7 06:34:56 2002): SESSION Source: 63.148.99.247 Http File Errors: 1
DETECTOR User: ""(or empty) NwProtocols: -NONE- Logoff Count: 0-1 Http Proto Errors:
0-1 Http File Errors: 0-1 Http Access Errors: 0-1

Event 8 (Sat Dec 7 06:34:56 2002): SESSION Source: 63.148.99.247 Http File Errors: 1
DETECTOR User: ""(or empty) NwProtocols: -NONE- Logoff Count: 0-1 Http Proto Errors:
0-1 Http File Errors: 0-1 Http Access Errors: 0-1

Event 9 (Sat Dec 7 06:34:56 2002): SESSION Source: 207.6.95.231 Http File Errors: 1
DETECTOR Source: 206.10.58.128:26 NwProtocols: -NONE- Http Proto Errors: 0-1

Event 10 (Sat Dec 7 06:34:56 2002): SESSION Source: 207.6.95.231 Http File Errors: 1
DETECTOR Source: 206.10.58.128:26 NwProtocols: -NONE- Http Proto Errors: 0-1

Event 11 (Sat Dec 7 06:34:56 2002): SESSION Source: 61.188.179.36 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 12 (Sat Dec 7 06:34:57 2002): SESSION Source: 203.93.204.71 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 13 (Sat Dec 7 06:34:57 2002): SESSION Source: 210.136.6.27 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 14 (Sat Dec 7 06:34:57 2002): SESSION Source: 200.171.236.99 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 15 (Sat Dec 7 06:34:57 2002): SESSION Source: 200.165.203.39 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 16 (Sat Dec 7 06:34:57 2002): SESSION Source: 65.173.60.5 Http Proto Errors: 1
DETECTOR Source: 80.205.92.136:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http File
Errors: 0-1 Http Access Errors: 0-1

Event 17 (Sat Dec 7 06:34:57 2002): SESSION Source: 12.252.125.14 Http Proto Errors: 1
DETECTOR Source: 4.62.115.112:28 NwProtocols: -NONE- Logoff Count: 0-1 Http Proto
Errors: 0-1 Http File Errors: 0-1

Event 18 (Sat Dec 7 06:34:57 2002): SESSION Source: 61.222.96.196 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 19 (Sat Dec 7 06:34:57 2002): SESSION Source: 210.78.17.189 Http Access Errors: 1
DETECTOR Source: 210.78.128.0:20 NwProtocols: -NONE- Http Proto Errors: 0-1 Http File
Errors: 0-1

Event 20 (Sat Dec 7 06:34:57 2002): SESSION Source: 211.253.110.214 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 21 (Sat Dec 7 06:34:58 2002): SESSION Source: 61.222.247.138 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 22 (Sat Dec 7 06:34:58 2002): SESSION Source: 129.101.171.14 Http File Errors: 1
DETECTOR User: ""(or empty) NwProtocols: -NONE- Logoff Count: 0-1 Http Proto Errors:
0-1 Http File Errors: 0-1 Http Access Errors: 0-1

Event 23 (Sat Dec 7 06:34:58 2002): SESSION Source: 211.191.40.205 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 24 (Sat Dec 7 06:34:58 2002): SESSION Source: 65.43.136.11 Http Proto Errors: 1
DETECTOR Source: 80.205.92.136:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http File
Errors: 0-1 Http Access Errors: 0-1

Event 25 (Sat Dec 7 06:34:58 2002): SESSION Source: 61.159.250.73 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 26 (Sat Dec 7 06:34:58 2002): SESSION Source: 203.199.92.19 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 27 (Sat Dec 7 06:34:58 2002): SESSION Source: 143.246.86.213 Http Proto Errors: 1
DETECTOR NwProtocols: -NONE- Http Proto Errors: 1-1

Event 28 (Sat Dec 7 06:34:59 2002): SESSION Source: 218.108.179.159 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 29 (Sat Dec 7 06:34:59 2002): SESSION Source: 218.76.94.34 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Event 30 (Sat Dec 7 06:34:59 2002): SESSION Source: 211.181.210.190 Http Proto Errors: 1
DETECTOR Source: 211.196.228.36:30 NwProtocols: -NONE- Http Proto Errors: 0-1 Http
File Errors: 0-1 Http Access Errors: 0-1

Appendix C

Some interesting detectors generated

```
// This detector caught my eye because at first glance it looks like all sessions with successful
// logons are attacks. After more thought, I realised that the protocol information is coming
// from the firewall. Apparently I don't generate many packets that my firewall blocks.
// (I must generate some in either the 1-8191 or the 16383-65535 protocol range) My ids
// decided that such an access with a successful logon must be a masquerader. (I'm really pleased
// with this rule because is not the type of rule you could easily generate for a misuse based ids.)
DETECTOR Source: 0.0.0.0:0 NwProtocols: TCP Protocol: 8192-16383 Logon Count: 0-3
```

```
DETECTOR User: "-ALL-" Source: 128.0.0.0:1 NwProtocols: UDP TTL: 104-107
```

```
DETECTOR User: "-ALL-" Source: 24.202.0.0:15 NwProtocols: UDP TTL: 114-115 Logon
Count: 0-1
```

```
DETECTOR User: "-ALL-" Source: 61.117.67.244:31 NwProtocols: TCP TTL: 40-43 Http
Proto Errors: 1-1
```

```
DETECTOR User: "-ALL-" Source: 202.123.215.0:24 NwProtocols: TCP TTL: 32-47 Http
Proto Errors: 1-1
```

```
DETECTOR User: "root"(or empty) NwProtocols: -NONE- Logon Failure Count: 1-1
```

```
DETECTOR Source: 48.0.0.0:4 NwProtocols: TCP TTL: 64-127 Logon Failure Count: 0-1
```

Appendix D

First 24 detectors generated

DETECTOR User: "-ALL-" Source: 62.146.0.0:15 NwProtocols: UDP TTL: 114
DETECTOR User: "-ALL-" Source: 62.248.128.0:18 NwProtocols: UDP TTL: 110
DETECTOR User: "-ALL-" Source: 80.199.0.0:16 NwProtocols: UDP
DETECTOR User: "-ALL-" Source: 193.216.208.0:21 NwProtocols: UDP TTL: 112-119
DETECTOR User: "-ALL-" Source: 200.0.0.0:11 NwProtocols: UDP Protocol: 0-32767 TTL:
114-115
DETECTOR User: "-ALL-" Source: 63.205.96.0:19 NwProtocols: UDP TTL: 112-115
DETECTOR User: "-ALL-" Source: 61.192.0.0:11 NwProtocols: UDP TTL: 50-51
DETECTOR User: "-ALL-" Source: 200.131.224.0:19 NwProtocols: UDP TTL: 112-127
DETECTOR User: "-ALL-" Source: 210.118.144.0:20 NwProtocols: UDP TTL: 110
DETECTOR User: "-ALL-" Source: 62.83.0.0:16 NwProtocols: UDP TTL: 104-107
DETECTOR User: "-ALL-" Source: 209.91.154.0:23 NwProtocols: UDP TTL: 116-119
DETECTOR User: "-ALL-" Source: 80.8.0.0:14 NwProtocols: UDP TTL: 112-119
DETECTOR User: "-ALL-" Source: 151.16.0.0:12 NwProtocols: UDP TTL: 115
DETECTOR User: "-ALL-" Source: 24.200.95.128:28 NwProtocols: UDP TTL: 118
DETECTOR User: "-ALL-" Source: 65.184.0.0:13 NwProtocols: UDP TTL: 96-111
DETECTOR User: "-ALL-" Source: 67.105.113.0:24 NwProtocols: TCP TTL: 108 Http Proto
Errors: 0-0
DETECTOR User: "-ALL-" Source: 220.82.174.192:27 NwProtocols: TCP Protocol: 24-31
DETECTOR User: "-ALL-" Source: 212.45.32.0:20 NwProtocols: UDP TTL: 107
DETECTOR User: "-ALL-" Source: 151.38.96.0:21 NwProtocols: UDP TTL: 116-119
DETECTOR User: "-ALL-" Source: 148.243.215.0:24 NwProtocols: UDP TTL: 118-119
DETECTOR User: "-ALL-" Source: 219.138.24.0:21 NwProtocols: UDP TTL: 114
DETECTOR User: "-ALL-" Source: 207.30.200.0:21 NwProtocols: UDP Protocol: 0-32767
TTL: 115

Bibliography

[1] Forrest, Stephanie, Alan S. Perelson, Self-Nonself Discrimination in a Computer, 1994, <http://citeseer.nj.nec.com/forrest94selfnonself.html>.

[2] Forrest, Stephanie, Brenda Javornik, Robert E. Smith, Alan S. Perelson, Using Genetic Algorithms to Explore Pattern Recognition in the Immune System, 1993, <http://citeseer.nj.nec.com/forrest93using.html>.

[3] Forrest, Stephanie, Steven A. Hofmeyr, Engineering an Immune System, 2001, http://www.cs.unm.edu/~forrest/ism_papers.htm.

[4] Forrest, Stephanie, Steven A. Hofmeyr, Anil Somayaji, Computer Immunology, 1996, <http://citeseer.nj.nec.com/forrest96computer.html>.

[5] Forrest, Stephanie, Steven A. Hofmeyr, Anil Somayaji, Thomas A. Longstaff, A Sense of Self for Unix Processes, 1996, <http://citeseer.nj.nec.com/forrest96sense.html>.

[6] Hofmeyr, Steven A, An Immunological Model of Distributed Detection and Its Application to Computer Security, 1999, <http://citeseer.nj.nec.com/hofmeyr99immunological.html>.

[7] Hofmeyr, Steven A., Stephanie Forrest, Architecture for an Artificial Immune System, 2000, <http://citeseer.nj.nec.com/374069.html>.

[8] Kephart, Jeffrey O., A Biologically Inspired Immune System for Computers, 1994, <http://citeseer.nj.nec.com/kephart94biologically.html>.

[9] Kim, Jungwon, Peter Bentley, An Artificial Immune Model for Network Intrusion Detection, 1999, <http://citeseer.nj.nec.com/262970.html>.

[10] Kim, Jungwon, Peter Bentley, An Evaluation of Negative Selection in an Artificial Immune System for Network Intrusion Detection, 2001, <http://citeseer.nj.nec.com/447684.html>.

[11] Kim, Jungwon, Peter Bentley, The Human Immune System and Network Intrusion Detection, 1999, <http://citeseer.nj.nec.com/kim99human.html>.

[12] Kim, Jungwon, Peter Bentley, Negative Selection and Niching by an Artificial Immune System for Network Intrusion Detection, 1999, <http://citeseer.nj.nec.com/kim99negative.html>.

[13] Kim, Jungwon, Peter Bentley, Towards an Artificial Immune System for Network Intrusion Detection: An Investigation of Dynamic Clonal Selection, 2002, <http://citeseer.nj.nec.com/531326.html>.