IMPROVING STATEFUL INSPECTION LOG ANALYSIS

Cristiano Lincoln Mattos (lincoln@cesar.org.br)
Centro de Estudos e Sistemas Avançados do Recife - CESAR
Universidade Federal de Pernambuco – CIn/UFPE
Tempest Security Technologies

Fabio Silva (fabio@cesar.org.br)
Centro de Estudos e Sistemas Avançados do Recife - CESAR
Universidade Federal de Pernambuco – CIn/UFPE

Evandro Curvelo Hora (evandro@cesar.org.br)
Centro de Estudos e Sistemas Avançados do Recife - CESAR
Universidade Federal de Pernambuco – CIn/UFPE
Universidade Federal de Sergipe – DCCE/UFS
Tempest Security Technologies

Marco Antonio Carnut (kiko@cesar.org.br)
Centro de Estudos e Sistemas Avançados do Recife - CESAR
Universidade Federal de Pernambuco – CIn/UFPE
Tempest Security Technologies

ABSTRACT

This paper presents a method for analyzing firewall log files that recognizes related connections from application level protocols, much like "stateful inspection" firewalls such as Linux's IPTables or Checkpoint's Firewall-1 do for allowing/denying traffic. Both a general framework and specific examples are discussed, and analysis results from sample data. Implications and potential for security applications are also presented.

1 INTRODUCTION

Log analysis is an often forgotten activity, perhaps because of the massive amount of logs generated by firewalls and the lack of good automated tools to aid in their analysis and interpretation. Most analysis tools limit themselves to tally up connection counts; since many application protocols originate and/or receive several connections, this is neither a convenient nor didactic way of presenting the results of such an analysis.

It would be much better if the analysis tool could identify related connections from common application protocols, tallying them separately or ignoring them. The resulting report would be much more readable, allowing for easier identification of normal and anomalous behavior.

This paper is organized as follows: section 2 details how the connection tracking process in typical stateful inspection firewalls interact with the log generation process, highlighting some particularities that arise from this interaction, especially in Linux's IPTables and Checkpoint's Firewall-1. Section 3 describes how the ideas of stateful connection tracking and correlation can be applied successfully in log analysis to expose a few kinds of security-related anomalies. Section 4 outlines some ideas for implementing these techniques directly on firewalls, intrusion detection systems and other network devices subject to real-time processing constraints. Section 5 presents conclusions and ideas for future work and implementations.

2 TYPICAL STATEFUL LOG GENERATION

Stateful inspection firewalls log their activities depending on the protocol involved: for TCP and UDP, they log the first packet that causes the "connection" to be evaluated against its rule base; subsequent packets of accepted connections are not logged since they are already in the state table. The return packets of these connections are not logged either, and are allowed to pass. Rejected connections are always logged since they necessarily cause the rule base to be consulted.

A connection is identified by its traditional 4-tuple: (source address, source port, destination address, destination port). This allows the concept of "connection" to be extended to UDP, which, being a datagram protocol, lacks the concept of "connection"; in this paper, we refer to them as UDP "sessions".

The state tables have inactivity timeouts. Idle TCP connections are removed from the state table after a certain period (typically tens of minutes). The same applies to UDP sessions, but with a much shorter timeout – tens of seconds, typically. A connection that resumes transmitting after these periods is rechecked against its rule base and logged again.

For TCP, either a FIN or a RST packet removes the connection from the state tables. It is interesting to note that while connection initiations are logged, connection terminations are not. This is unfortunate, since interesting data could be obtained from them, such as the duration of the connection. This obviously doesn't apply to UDP, since it lacks explicit termination; they are dealt with by the aforementioned timeout rules.

ICMP packets, however, are typically logged in the traditional stateless fashion: each packet generates a log entry, without trying to prevent further log entries by relating it with previous exchanges. This is somewhat inconsistent, since there is usually enough information in ICMP packets to be able to relate requests and replies, despite the fact that they were not designed to provide connection services.

Checkpoint's Firewall-1 has an extra "excessive log" filtering feature built-in: nearly identical packets arriving within a certain time frame (62 seconds, by default) will not be logged. This prevents long-lasting ping trains, commonly used by system administrators and dynamic routing protocols for connectivity testing, to flood the logs with repetitive uninteresting data. IPTables can do that kind of rate limitation using token bucket filters, but it's not enabled by default.

It is also appropriate to remind that logging is optional, being enabled or disabled on a rule-by-rule basis. The more pervasive the logging policy is, the better the results will tend to be.

In fact, Firewall-1 goes beyond, allowing each rule to be logged in on of several logging styles: "none" (no logging), "short" (logs only source and destination addresses and ports), "long" (same as "short" plus translated addresses and VPN events) and "accounting" (same as "long" plus total amount of data transferred). The more detailed the logging, the bigger the speed and space requirements.

2.1 Related connection logging

When a connection is accepted by a rule with certain "special" destination ports (21/tcp, for example, corresponding to the control port of the FTP service), it is put "on watch": all packets in this connection are inspected looking for control information about new endpoint (addresses & ports) negotiations. In our example with the FTP protocol, that would be the "PORT" command. This new endpoint is added to a separate table, along with a destination endpoint. Together they become a special temporary rule that is checked even before the rule base and allows those connections to pass even without explicit mention in the rule base. This is what we call "related connections".

When one of these related connections is initiated, the "related connections special rule table" is checked, a match is found and the connection is automatically accepted. However, Firewall-1 doesn't log the acceptance of this connection, maybe because they're supposed to be accepted anyway. This is unfortunate from the point of view of the log analyzer, since valuable information about the exact connections that took place is lost. IPTables, however, can be set up in a way that logs these connections.

When the master connection is shut down, either by timeout or by explicit termination, all its related "special rules" are deleted, thus closing the "holes" they opened in the firewall. Notice that it doesn't finish any ongoing related connections; it merely prevents the creation of new ones. None of this is logged, though. It should also be stressed that the deleted rule is a *temporary one* kept in memory – none of this modifies the security policy rule table in any way.

While this process has been described for FTP only, it readily generalizes for several other protocols. The principle is the same: watch and interpret the control connection searching for new endpoint negotiations, adding them to the special "related connections dynamic rule table" and making sure to get rid of them when the control connection finishes.

Note that this approach requires one handler for each application protocol, since it is necessary to understand the protocol messages in sufficient detail to extract the endpoint negotiations. Both Firewall-1 and IPTables have handlers for several popular protocols that require related connections, such as Sun RPC, RealAudio, etc.

It is unfortunate that neither Firewall-1 nor IPTables log the name of the application protocol handler or the endpoints of the related control connection – if that information was available, it would be possible to reconstruct exactly which related connection was generated by which control connection.

From the preceding discussion, it follows that application protocols handled specially by the firewall will usually have only its control connection logged, preventing any correlation with its related connections – FTP, RealAudio, etc., being the prototypical examples. Several other protocols and network interactions, however, exhibit related connection behavior without being subject to any special processing. These are worthy cases for stateful correlation.

3 THE TECHNIQUE

3.1 Connection correlation

The following section of the log analyzer configuration file makes a good example of the technique:

```
1 port=80/tcp name=http-reverse-conn
2  master: record srcip, dstip
3  related-X: match srcip_r=dstip,
      dstip_r=srcip, dstport_r=6000/tcp
4  related-http: match srcip_r=dstip,
      dstip_r=scrip, dstport_r=80/tcp
5  related-ftp: match srcip_r=dstip,
      dstip_r=scrip, dstport_r=21/tcp
6  related-generic: match srcip_r=dstip,
      dstip_r=scrip, dstport_r=*/*
```

(The line numbers are for reference only in this text; they don't actually need to be present in the actual configuration file).

The first line specifies the name of the event ("http-reverse-connection") and the port on which the master connections should be watched: 80/tcp. The second line specifies the which data from the master connection should be recorded in the "state table"; in this case, the source and destination IP addresses. We could simplify it by storing everything about the connection, but, since the state tables tend to grow quite large, it is more memory-efficient to store only what is effectively needed.

The remaining lines specify several cases of related connections that we would be interested to hear about:

- The third line describes an attempt to connect to the X Windows port of the machine that originated the HTTP request. It often happens in Unix machines after a successful exploitation faulty CGI applications. Reading from the notation, it says "tag with the name 'related-X' all connections coming from the same IP of the destination of the master connection, going to the same IP that originated the master connection and whose destination port is 6000/tcp".
- The fourth and fifth directives go along similar lines, but for ports 80/tcp (HTTP) and 21/tcp (FTP). Readers with background on common exploits and intrusion detection will recognize this traffic behavior as arising from a successful exploitation of a common IIS vulnerability where the attacker connects elsewhere to download trojan horses or remote control programs.
- The sixth line is a "catch-all" for reverse connections: it would flag any connections originating from the web server originally contacted to the client that originally made the contact. The "*" stands for "any".

While none of these kind of traffic are proof of a security breach, they are uncommon enough to raise suspicions and deserve the attention of the administrators.

It can be argued that the condition "dstip_r=srcip" is too restrictive – an attacker could download his backdoors from a machine other than the one he/she used to send the exploit. This condition could be relaxed if it is felt that it wouldn't generate too many false alarms.

On the other hand, if we make some assumptions about the security policy and the network architecture, we could generalize the match condition without significantly increasing its potential for false positives: if we assume that the HTTP servers are on a DMZ and the security policy forbids connections originating from the DMZ going to the Internet (a well-known Good Thing), we could write:

```
6  related-generic: match srcip_r=dstip,
      dstport_r=*/*, action=reject or
      action=drop
```

That is, flag only the connections that were blocked – the fact that it was blocked is indication that it is against the security policy.

This kind of correlation analysis is especially useful when doing forensic investigations in incident response scenario: it easily pinpoints reverse connections and other anomalies that are telltale signs of unauthorized activity, automating the tedious manual process of relevant evidence collection.

There are several kinds of traffic that can be correlated in this fashion. Although most of it is not directly security-related, the mere act of properly grouping them together and displaying it nicely encourage the system administrators to actually read the log summaries and thus conform to the classical "know thy traffic" security tenet. The following subsections illustrate some cases:

3.2 ICMP Messages Correlation

It would be useful to correlate the ICMP messages with the packets that originated them. The fragment below shows such a configuration in our tool for a simple UDP ⇔ ICMP correlation.

```
1 port=*/udp name=unreachables
2  master: record srcip, dstip,
                  ipid optional
3  port-unreach: match dstip_r=srcip,
      type=3-3/icmp, ipid_r=ipid
4  net-unreach: match dstip_r=srcip,
      type=3-0/icmp, ipid_r=ipid
5  host-unreach: match dstip_r=srcip,
      type=3-1/icmp, ipid_r=ipid
6  frag-needed: match dstip_r=srcip,
      type=3-4/icmp, ipid_r=ipid
7  admin-prohib: match dstip_r=srcip,
      type=3-13/icmp, ipid_r=ipid
   ...
```

The first line defines the "unreachables" tag and state table for all UDP sessions. The second line tells it to record only the source and destination IPs and the IP identification number. The following lines identify several kinds related ICMP control messages that could arise out of this packet: port, host or destination unreachable, communication administratively prohibited (commonly sent by packet filtering routers), etc. In this example, the IP identification number is used to relate the replies with the packets that originated them. Since certain log file formats don't record the ID field of the IP header, the "optional" keyword is used to make the log analyzer try to relate the packets even in its absence. Without the "optional", the analyzer would simply discard this whole section due to lack of information to perform the correlation.

Even simple things such as correlating pings prove useful and have interesting security implications: (the example below was shortened for clarity – we could promptly add the same ICMP correlation rules we did above for UDP):

```
type=8-0/icmp name=pings
 echo-request: record srcip, dstip, ipid
 echo-reply: match srcip_r=dstip,
    dstip_r=srcip, ipid_r=ipid optional,
    type=0-0/icmp, atmostonce
 ...
```

In the above example, the “atmostonce” keyword tells the analyzer that the replies must match the request at most once. “At most” because the reply may get lost or not be reported in the log. The tool considers an anomaly to see two or more replies to the same packet. Badly configured routing, broadcast address and other bizarre network effects might cause this and have been observed in practice. If the condition that requires the match of the IP IDs is relaxed, it might be used to detect ICMP tunnelers such as Loki – an interesting security event worth being flagged.

3.3 Connection/Session Counting and Graphing

Besides the “record” directive, the specification of the master connection allows for other kinds of processing. The example below implements a port scan detector:

```
type=*/tcp name=portscan-detector
 histogrm: match port=*/tcp or port=*/udp
       count dstport group_by srcip
       graph if count > $treshold
```

This setting does the following: for each source IP, the analyzer builds a hash table that counts the number of different destination ports in the TCP connections and UDP sessions it originated. If the number of connections is greater then `$treshold` (a macro that we once set to expand to 60 and never more changed it), it produces a histogram graph of the distribution of the ports. The original idea was to make a real histogram graph to be saved as a GIF file to be viewed in a web page, but since one of the requisites of our first version was to be text-only, it produces a three-line report like the one shown in Figure 1.

The first line lists the total number of connection attempts that matched, the source address and the total number of unique ports.

The second line is the privileged port number space from 0 to 1023, each character representing 16 ports. The “-” characters means that this “slot” of 16 ports received no “hits” or connection attempts. Numbers and letters are hex digits representing the number of hits each slot had.

The third line is the full port number space from 0 to 65535, each character representing a slot of 1024 ports. Again, a “-” represents no hits in that slot. The numbers and letters, however, have a different meaning: they are the count of hits in the slot divided by 32, represented in Radix-32. In other words, “1” means anything from 1-32 hits, “2” means 33-64 hits, up to “W”, meaning 993 to 1024 hits. This is a way to make a compact text-only histogram.

This scan is an example picked from our real world logs. Experience has shown that this kind of scan is usually generated by the options of the NMAP tool in TCP `connect()` scan mode, plus some other “probing around” – that is, when we scan ourselves using NMAP, the shape of the histogram is quite similar.

The current version of the tool does not show the exact targets of the port scans, although we can get that information from other subreports. We are currently working on making the syntax for specifying nested subgroupings generating their own counts, histogram graphs and subreports – and making them fully graphical. What becomes clear is the vast space for analysis criteria.

A non-obvious characteristic of this port scan detection scheme is that it does not expect the scan to be in increasing port number order like many other port scan detector tools do. Modern port scan detectors randomize the order they try the ports, but since our technique counts the total number of distinct ports, it catches these cases perfectly well.

The somewhat arbitrary decision that a port scan is when we get connections to more than 60 distinct ports is certainly debatable, but perfectly configurable. While analyzing single-day log files, it has been found to be quite acceptable. It is planned that future versions of our tool will allow for complex expressions to calculate this threshold.

It is interesting to apply the tool and these techniques for very large log files – actually, we are working a version in which the state tables are stored on disk as B-trees, so as to be able to analyze month-long logs and bigger. Our preliminary results show several unexpected features, like distributed slow port scans.

4 REAL-TIME APPLICATIONS

The ideas described above were used to implement a batch log analysis tool: the log files of a certain period, typically a whole day, were collected and a report was produced. While this makes for interesting reading, it’s natural to think of the next steps:

- Firewall devices could already analyze and report their data in this “stateful/correlating”

```
1920: 200.231.88.116  (  1761)
   CDEEDD0E897187536425563087553473305-6C84D6C86C6A6135759553366538
   ECC7---39-------------------------------------------------------
```

Figure 1

way. Actions could even be taken based on conclusions regarding the correlation analysis. The difficulty with this idea stems from the fact that the state tables take a lot of memory. Strict expiration and discard policies for the table entries should be applied to keep them within reasonable bounds. It could also be argued that the increased memory demand could make the firewall more vulnerable to resource exhaustion attacks. Performance might also become a problem in very fast networks and slow processors.

- Intrusion Detection Systems might be a better candidate for this kind of analysis. Some of them already perform a some kind of stateful analysis and correlation, but most of them usually limit themselves to analyze the packet contents in search of known common attack signatures.
- At the very least, firewalls should log more data, like connection termination; related connections caused by application-layer handlers; the exact identification of the application handler and the endpoint negotiation it detected; perhaps even the complete transport and network headers and the beginning of the payload – the goal being to make the log analyzer capable of accurately reconstruct the actions taken by the firewall and the interaction between the communicating parties.

5 CONCLUSIONS AND FUTURE WORK

This work showed that the same techniques used by stateful firewalls to filter the traffic can be applied to the field of log analysis. The characterization technique has been applied to expose several kinds of security-related incidents, such as reverse connections and covert channels. Many other types of anomalies, not necessarily security-related, can also be flagged.

Some inconsistencies and omissions in the way common stateful inspection firewalls generate their logs have been presented, especially regarding the stateless handling of ICMP packets, the omission of related application-level connections (FTP being the typical example). Most log files fail to provide enough information to accurately reconstruct their actions and some improvements were suggested.

It has also been shown that state tables can be used to draw histograms or perform statistical characterization of the traffic that could be used to detect security-related probing, such as port scanning, or anomalous traffic patterns.

The tool implementing these techniques makes its analysis in batch mode, operating on a large text-mode log file. It was originally conceived both as a forensic analysis tool and a daily summarizer to be run along the log file rotation and archival process. However, it has been shown that the correlation techniques may also be implemented directly in the firewalls or in intrusion detection systems. A worthwhile goal in sight would be to produce patches to IPTables or Snort to achieve this.

Another avenue of work being pursued is the statistical characterization of port scans and signature-based recognition of the tools that produced the scan – we would like our tools to be able to say something along the lines of "this anomaly is consistent with a NMAP connect() scan".

6 REFERENCES

iptables(8) man pages.

Phoneboy's Firewall-1 FAQ: www.phoneboy.com

Project Loki: ICMP Tunneling www.phrack.org

Snort IDS: http://www.snort.org

AMOROSO, Edward. *Intrusion Detection – An introduction to Internet surveillance, correlation, trace back, trap, and response.* AT&T Laboratories. Intrusion.Net Books, 1999, ISBN 0966670078.

CHECKPOINT Software Technologies Ltd., *Checkpoint Firewall-1 Architecture and Administration*, 1998, Part No 71300001400.

NORTHCUTT, Stephen, *Network Intrusion Detection: An Analyst's Handbook, 2nd Edition*, 2000, New Riders Publishing, ISBN 0735710082.

SPITZNER, Lance, *Understanding the FW-1 State Table: How Stateful is Stateful Inspection?*, http://www.enteract.com/~lspitz/fwtable.html

STEVENS, W. Richard, *TCP/IP Illustrated, Volume 1 – The Protocols*, Addison-Wesley, 1994, ISBN 0-201-63346-9.