Log File Analysis with Context-Free Grammars

Gregory Bosman | Stefan Gruner
Department of Computer Science
Universiteit van Pretoria
Republic of South Africa

Abstract. Classical ways of intrusion analysis from textual communication log files are either AI-based (such as by combinations of data mining with various techniques of machine learning), or they are based on regular expressions (such as the scanners implemented in the 'CISCO boxes'). Whereas AI-based heuristics are not analytically exact, methods based on regular expressions do not reach very far in Chomsky's hierarchy of languages. In this short chapter we describe work in progress on the topic of parsing traces of network traffic with context-free grammars. 'Green' grammars describe acceptable log files, whereas 'red' grammars represent already known specific patterns of intrusion attempts. This technique can complement or augment the aready existing AI-approaches with additional precision. Analytically it is also more powerful than CISCO's technique on the basis of regular expressions.

1 Introduction

Modern intrusion analysis systems, for the most part, rely on simple pattern recognition in order to differentiate and categorise malicious network traffic from benign [1] [7] [9]. Anomaly-based detection systems, which are those built to recognise traffic generated in the normal day-to-day operations of a network, often look no further than the header of packets in the data stream, determining information such as source and destination addresses, the identity of the application which generated the traffic, the port through which the connection is made, and the like, to determine whether a package should cause reason for alarm or not. Sometimes the analysis is taken one step further, scrutinising the packets as a stream, instead of just one by one. However this rarely extends further than simply checking the counts of the packet found also within the header to ensure the order is correct. Signature-based analysis systems operate in mostly the same way, in some cases analysing the data segments of network packets in addition, searching for signatures of, for example, injected code, which can usually be identified as a series of bits and identified with the use of a suitable regular expression.

In spite of all such analysis, however, there is still much potentially useful information which goes unnoticed, some of which can be drawn on from the procession of packets in a data stream. State-based analysis has been suggested before for the purpose of security in computing [4] [11]. Gudes and Olivier have argued in [11] that the 'state' of an application

should also be considered in determining the security of that application. They reason that certain messages recieved by an application, while perfectly safe if in one state, may compromise the security of that application should it be found in some other, more vulnerable state. Therefore, in analysis, the 'state' of the applications should be taken into account when trying to determine whether some incoming data was harmful or not. Gudes and Olivier, too, proposed the use of context-free grammars for the purpose of state-based analysis. This approach entails the construction of a grammar to represent the transitions from state to state internally within an application, thus also describing the actions that are allowable in any given state.

Memon had applied context-free grammars, too, in a log file categorization system [10]. His method of data analysis did not, however, take into consideration the sequencing of packets in a data stream, nor the idea of the 'statefulness' of an interaction between network applications. Instead, he proposed the use of grammars in categorisation for their adaptability, arguing that existing pattern-matching methods were too ad hoc, non-adaptive and become highly work-intensive, particularly when the systems relying on such methods require updates. Memon developed a grammar capable of representing a single packet in a data stream, including information such as packet protocol, source and destination. He also developed a method of grammar inference, capable of expanding an existing grammar modeled for such purpose to include additional packets to be regarded as benign, at the discretion of an administrator.

A grammar complex enough to capture and represent the statefulness of applications communicating over a network would likely require considerably more design efforts than those grammars described in Memon's work. There is, however, ongoing research into the application of artificial intelligence (genetic algorithms in particular) for the purpose of grammar inference from provided example sentences [2] [8] [14]. Currently these techniques allow only for the inference of rather simple grammars within reasonable time through the use of positive and negative examples; in intrusion detection these would be benign and malicious network traces respectively. More complex grammars, as required for the purpose we intend, require more design efforts which cannot yet be automated. However, the efforts of manually crafting a suitable intrusion analysis grammar will be worth the while when the grammar is used often enough after its construction.

The possibility of too many false alerts in automated intrusion detection systems was addressed in [6], where some algorithmic heuristics was shown to condense an un-managable amount of elementary alerts to a managable amount of so-called "meta-alerts". However the alert condensation technique in [6] appeared as somewhat 'ad hoc', (i.e.: without much theoretical underpinning). Similar work, also concerned about alert management and the "fusion" of alerts w.r.t. their similarity (near-match) to already existing "meta alerts", can be found in [13], too.

In the context of this chapter, however, we are not concerned about such managerial issues like as the sheer numbers of risen alerts; our concern is the further development of formal methods (here: context-free grammars) for the purpose of incident identification. The relevance of this undertaking with regard to potential forensic applications (in practice)

should be obvious and evident without many further words of motivation.

2 Method

In this explorative phase of our project we are only dealing with a subset of all possible malicious attack 'vectors'. Our grammar method, which fits well into the digital forensics context, is conceptually related to formal methods in the field of model-based testing [5] and may be regarded as a limited form of automated verification on the basis of specific samples. *Proof goal* is a decision whether or not an intrusion attempt has occurred on the network in a specific situation. To tackle the problem more effectively from two ends, two types of grammars will be needed:

- One 'green' grammar is designed to describe log files of harmless communications.
  - A log file that can be successfully parsed with a 'green' grammar is regarded as *hypothetically harmless* and passes the test.
  - A log file that cannot be parsed with a 'green' grammar is regarded as *suspicious* such that an alert will be raised.
- Several 'red' grammars are supposed to describe log files containing typical traces of already known patterns of malicious communication.
  - A log file that can be successfully parsed with a 'red' grammar is regarded as *harmful* such that an alert will be raised.
  - A log file that cannot be parsed with a 'red' grammar is regarded as *hypothetically harmless* and passes the test.

Using a specific browser to generate experimental traffic and log files we have designed the 'green' grammar (representing acceptable traffic) listed in the → *Appendix* below. In this grammar each terminal symbol represents a single packet of traffic recorded. The grammar as a whole models a string of packets of a 'conversation' as a simple web page is transfered. Ideally such a conversation would follow the following simplified pattern:

```
TRACE -> syn synack acko get acki PDU
PDU   -> data data acko PDU
       | data http acko
       | http acko
```

A communication at the browser opens with a 'handshake', i.e.: a sequence of packages with synchronisation or acknowledgement or both flags set. Subsequently a request is made for data transfer: the 'get' followed by a packet in which the acknowledgement flag is set (again), and then data transfer begins. This is represented by the protocol data unit (PDU) section. The rest is rather simple: data is transfered in two packets at a time, until the transfer is complete. After every two data packets incoming, an acknowledgement is sent out to indicate

there are no errors (for example with the order in which the packets are recieved) and that it is possible to continue with the transfer. When there is no more data, a final HTTP packet is recieved to indicate the end of the stream, followed by one last outgoing acknowledgement to the web server.

That would be an ideal model of a conversation between client and server. In practice, however, the streams tend to be interspersed with some additional 'mini' conversations every now and then which carry some kinds of meta information. For example, an address resolution protocol message may be sent in order to find the location of the web page initially. Also in nearly every trace there is at least one request for a domain name server which tends to jump straight into the middle of the main conversation with the web server. Therefore an appropriate 'green' grammar must be constructed in such a way that it can 'remember' the internal state of a conversation when it gets interrupted in such manner, such that it can return to that state again after the interrupt.

This implies that the entire conversation must complete in order to be recognized as a benign trace. If the conversation starts (with the handshake in this case), it must also terminate successfully. If it does not, then this can indicate that the browser could have been compromised (unless the communication was technically interrupted, for example by a browser 'crash' on the client's side). In such a case the corresponding trace, when parsed, will lead to a syntax error, and the corresponding communication trace will be classified as 'suspicious'.

Last but not least it is worth nothing that DNS requests and ARP also represent such 'mini' conversations as mentioned above. However, an ARP request is always completed without interruption in the traces corresponding to the grammar. DNS requests, on the other hand, are just as susceptible to interruption as the web browser conversation discussed above. At this point we can see that regular expressions alone (instead of context-free grammars) would not suffice for remembering a status of conversation before the occurrence of such interrupts.

3 Explorative Experiments

We presume that our readers are familiar with the principles and techniques of lexical and syntax analysis as they are well known from the classical computer science field of compiler construction. On these premises, with our 'green' grammar (as printed in the Appendix) and a controlled laboratory network setting we conducted a series of tests, both with various permissible and not permissible input strings, to assess the current effectivity of our technique. The experimental results are summarized in the following table, whereby the tool ANTLR [12] was used to generate the parser for that grammar.

Input	Total	Accepted	Rejected	Lex-Err.	Syntax-Err.
benign	100	94	6	6	0
malicious	100	0	100	28	72

Alpha error a.k.a. 'false positive' (wrong rejection of benign input): $6/100 = 6\%$. Alpha errors may be regarded as 'annoying' or 'inconvenient'; however they are not harmful from a security point of view.

Beta error a.k.a. 'false negative' (wrong acceptance of malicious input): $0/100 = 0\%$. Beta errors should, for obvious reasons, be interpreted as 'harmful' from a security point of view.

In spite of our seemingly 'nice' Beta value, residual problems with our current grammar can be inferred from the observation that not all parsing failures resulted from syntax errors, as it should have been the case if our grammar had been a-priori 'aware' of all possible input tokens. The lexical errors (prior to the parsing phase) show that an input stream can still contain a number of 'unknown' symbols with which our 'green' grammar could not cope. Thus we can roughly estimate the fit-for-purpose-quality of our intrusion detection grammar as follows:

In case Alpha (whereby ideally no parse error should have occurred at all), *all* errors occurred in the phase of lexical analysis. This implies that our grammar has caused 100% percent of the Alpha errors by not 'knowing' all the symbols of our input strings.

In case Beta (in which ideally only syntax errors should have occured), we have to admit that only 72% percent of our (correct) rejection decisions were based on parsing, whereas 28% percent of all rejections were based on decision-less lexical errors before the deciding parsing phase started. Therefore, contrary to what has been 'roughly' stated above, our Beta error should be estimated as higher than those 0%, namely up to 28% (in the worst case).

Thus there is still much room for improvements. On the basis of 'inspection with the naked eye' we conjectured that those errors seem to appear in the context of those occasional 'in between' communications which we have mentioned above in the introductory section.

For the experiments with malicious input streams we have so far restricted ourselves to very 'typical' scenarios such as buffer overflow attacks. In most of those cases communication terminated mid-stream, such that the parser could correctly recognise the absence of the stream-closing data packets. However we are also aware of at least one case where the lexical analyser could not recognize a termination symbol that had been sent properly to terminate a conversation. Some of those problems might even be linked to speed discrepancies between data download and analysis, such that the parsing unit may prematurely believe that an input stream has been 'hacked' whilst in truth a harmless 'download' is still on its way.

Last but not least we must also note from a science-philosophical point of view that our explorative experiments were predominantly 'qualitative' so far and did not yet meet the strictest criteria of experimental scientificness described by Bunge in [3].

4 Outlook

Critical readers may ask the question whether the additional complexity of context-free grammars, in comparison against regular expressions, is *practically* justified; (from a theoretical point of view is is clear that the class of all regular languages is a genuine sub class of the class of all context-free languages). To answer this question, more practical examples will need to be identified which cannot be elegantly treated by regular expressions, (whereby we can ignore the triviality that every *finite* string can of course be represented by its own, however cumbersome, ad-hoc regular expression). *Speed* of analysis had been the main reason for using regular expressions in the context of the 'CISCO boxes' because in that context the analysis of data streams proceeds on-line and on-the-fly; it is well known that regular epression analysis can be implemented very runtime-efficiently. Ex-post-facto log file analysis, on the contrary, does not operate under such strict real-time conditions.

Another practical problem faced by the parsing technique is posed by technically interrupted communication attempts, for example due to a 'collapsing' or 'freezing' browser on the client side of a volatile internet connection. On the server side this will leave a 'broken trace' in the log file which can possibly not be successfully parsed, no matter whether the interrupted communication attempt had been malicious or harmless. String traces of interrupted communication attempts may thus considerably increase the rate of 'inconvenient' Alpha-type errors. To tackle this problem, we would have to find a way of dividing a large trace into a set of shorter sub traces which could possibly be parsed individually. Alternatively we could think about modifying the detector grammars in such a manner that interrupted communications can be explicitly identified as such.

For the remainder of this project we shall work on a practically useful software framework which shall allow the user to 'plug in' case-specific grammars (of type 'green' as well as of type 'red') and thus to generate analysis tools for various application scenarios.

- The ANTLR parser generator [12] shall serve as the most important component of that software framework.
- We must also 'fine-tune' our method itself in the light of the experimental experiences we have described them in the previous section.
- Moreover, we must also 'cast' a larger number of well known patterns of intrusion attempts into case-specific 'red' grammars, in addition to our 'green' grammar which only specifies the acceptable communication patterns at the network interface.
- A suitable combination of 'green' and 'red' grammars should further diminish the frequency of both Alpha and Beta decision errors, though this would also require an additional meta decision policy for cases in which the decisions made by the 'green' parser and the 'red' parsers would not be mutually consistent from a logical point of view.
- Last but not least, some degree of "context-sensitivity" or supra-local information-capturing could be introduced by defining suitable *attributed* grammars —used in

compiler-construction to capture particular features of static semantics beyond mere syntax analysis— which we have not yet used in our un-attributed grammar approach so far.

Acknowledgements

Thanks to *Bruce Watson* for valuable discussions on this topic. He also informed us (as far as possible) about the regular expression technique in the context of the 'CISCO boxes', on which no papers are published because of commercial confidentiality.

References

[1] S. Axelsson, Intrusion Detection Systems: A Survey and Taxonomy. *Technical Report*, Chalmers University (Sweden), 2000.

[2] B.R. Bryant, M. Crepinsek, F. Javed, M. Mernick and A. Sprague, Context-Free Grammar Induction Using Genetic Programming. *Proceedings of the 42nd Annual Southeast Regional Conference*, pp. 404-405, 2004.

[3] M. Bunge, Experiment, chpt. 14, pp. 251-289 in M. Bunge, *Scientific Research II: The Search for Truth*. Springer-Verlag, 1967.

[4] T. Dean, S. Knight and S. Zhang, A Lightweight Approach to State Based Security Testing. *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, pp. 341-344, 2006.

[5] S. Gruner and B. Watson, Model-based Passive Testing of Safety-Critical Components, chpt. 16, pp. 453-483 in J. Zander, I. Schieferdecker and P.J. Mosterman (eds.), *Model-Based Testing for Embedded Systems*. CRC Press (Taylor & Francis), 2012.

[6] R. Harang and P. Guarino, Clustering of Snort Alerts to identify Patterns and reduce Analyst Workload. *Proceedings of MILCOM'2012*, pp. 1-6, IEEE, 2012.

[7] S. Jajodia and P. Ning, *Intrusion Detection Techniques*. Wiley, 2003.

[8] S. Lucas, Structuring Chromosomes for Context-Free Grammar Evolution. *Proceedings of the 1st IEEE Conference on Evolutionary Computation / IEEE World Congress on Computational Intelligence*, vol. 1, pp. 130-135, IEEE, 1994.

[9] T.F. Lunt, A Survey of Intrusion Detection Techniques. *Computers and Security*, vol. 12(4), pp. 405-418, Elsevier, 1993.

[10] A.U. Memon, Log File Categorization and Anomaly Analysis using Grammar Inference. *Technical Report*, Queen's University (Canada), 2008.

[11] M.S. Olivier and E. Gudes, Wrappers: A Mechanism to Support State-based Authorization in Web Applications. *Data and Knowledge Engineering*, vol. 43(3), pp. 281-292, 2002.

[12] T. Parr, *The Definitive ANTLR Reference*. Pragmatic Bookshelf, 2007.

[13] A. Valdez and K. Skinner, Probabilistic Alert Correlation. *LNCS*, vol. 2212, pp. 54-68, Springer-Verlag, 2001.

[14] P. Wyard, Context Free Grammar Induction Using Genetic Algorithms. *Proceedings of the IEEE Colloquim on Grammatical Inference: Theory, Applications and Alternatives*, pp. 11/1-11/5, 2002.

A Grammar for our Explorative Experiments

```
TRACE      -> e | TRANSF | ADRESO | DNSQRY
TRACED1    -> TRANSFD1 | ADRESOD1 | DNSQRY1
TRACET1    -> TRANSF1  | ADRESOT1 | DNSQRYT1
TRACET2    -> TRANSF2  | ADRESOT2 | DNSQRYT2
TRACET3    -> PDU      | ADRESOT3 | DNSQRYT3
TRACET1D1  -> TRANSF1D1 | ADRESOT1D1 | DNSQRY1T1
TRACET2D1  -> TRANSF2D1 | ADRESOT2D1 | DNSQRY1T2
TRACET3D1  -> PDUD1  | ADRESOT3D1 | DNSQRY1T3
TRACEP1    -> PDU1   | ADRESOP1   | DNSQRYP1
TRACEP2    -> PDU2   | ADRESOP2   | DNSQRYP2
TRACEP3    -> PDU3   | ADRESOP3   | DNSQRYP3
TRACED1P1  -> PDU1D1 | ADRESOD1P1 | DNSQRY1P1
TRACED1P2  -> PDU2D1 | ADRESOD1P2 | DNSQRY1P2
TRACED1P3  -> PDU3D1 | ADRESOD1P3 | DNSQRY1P3
TRANSF     -> syn TRACET1
TRANSF1    -> synack acko get TRACET2
TRANSF2    -> acki TRACET3
TRANSFD1   -> syn TRACET1D1
TRANSF1D1  -> synack acko get TRACET2D1
TRANSF2D1  -> acki TRACET3D1
ADRESO     -> arpq arpr TRACE
ADRESOT1   -> arpq arpr TRACET1
ADRESOT2   -> arpq arpr TRACET2
ADRESOT3   -> arpq arpr TRACET3
ADRESOT1D1 -> arpq arpr TRACET1D1
ADRESOT2D1 -> arpq arpr TRACET2D1
ADRESOT3D1 -> arpq arpr TRACET3D1
ADRESOP1   -> arpq arpr TRACEP1
```

```
ADRESOP2   -> arpq arpr TRACEP2
ADRESOP3   -> arpq arpr TRACEP3
ADRESOD1P1 -> arpq arpr TRACED1P1
ADRESOD1P2 -> arpq arpr TRACED1P2
ADRESOD1P3 -> arpq arpr TRACED1P3
DNSQRY     -> dnsq TRACED1
DNSQRY1    -> dnsr TRACE
DNSQRYT1   -> dnsq TRACET1D1
DNSQRYP1   -> dnsq TRACED1P1
DNSQRY1T1  -> dnsr TRACET1
DNSQRY1P1  -> dnsr TRACEP1
DNSQRYT2   -> dnsq TRACET2D1
DNSQRYP2   -> dnsq TRACED1P2
DNSQRY1T2  -> dnsr TRACET2
DNSQRY1P2  -> dnsr TRACEP2
DNSQRYT3   -> dnsq TRACET3D1
DNSQRYP3   -> dnsq TRACED1P3
DNSQRY1T3  -> dnsr TRACET3
DNSQRY1P3  -> dnsr TRACEP3
PDU        -> data TRACEP1 | http TRACEP3
PDU1       -> data TRACEP2 | http TRACEP3
PDU2       -> acko TRACET3
PDU3       -> acko TRACE
PDUD1      -> data TRACED1P1 | http TRACED1P3
PDU1D1     -> data TRACED1P2 | http TRACED1P3
PDU2D1     -> acko TRACET3D1
PDU3D1     -> acko TRACED1
```