

Applying Artificial Intelligence to Automated Detection of DNS attacks

Brian Russell
Department of Computer Science
Rutgers University
morbius@paul.rutgers.edu

Abstract

This paper describes the work done as part of an independent study project to design an automated mechanism using Artificial Intelligence to construct data extraction analyzers and polymorphic expert modules that would have the ability to provide statistical likelihood information on attacks launched through the Domain Name Service (DNS).

1. Introduction

In the realm of the Internet, attackers are constantly devising new ways of attacking other systems and system administrators are constantly struggling to find better ways to intercept these attacks. An automated mechanism that could detect such attacks before they can do any significant harm would be of considerable value.

The problem of monitoring distributed network traffic to recognize potential malfeasance is challenging enough to draw on a broad array of approaches. A suitable architecture would therefore support and isolate different choices of learning mechanisms and experience data, while combining the separate ideas into a unified decision maker. Polymorphic expert modules would achieve the former while a unified merger module would accomplish the latter. The unified merger module would allow independently developed expert modules to contribute to the performance of the system as a whole.

This paper is organized as follows. Section 2 describes the analyzer algorithms that extract various kinds of DNS data for use by expert modules. Section 3 describes the polymorphic expert modules and the algorithms therein. Section 4 describes the calculation of attack probabilities by the expert modules. Section 5 describes the MySQL database where DNS message information is kept. Section 6 describes the results of the project and Section 7 provides the conclusion.

2. DNS Analyzers

The initial work done with DNS message data was the creation of a series of analyzers. The first two analyzers parsed DNS messages directly. All the remaining analyzers used the MySQL database on 204.147.180.70 to extract various kinds of data in an effort to find evidence of attacks. The last analyzers calculated prior probabilities and percentiles for different aspects of DNS. There were 15 analyzers in all and here's what each of them does:

1. A sanity checker that parsed DNS messages directly from UDP packet payloads. It checked the query, response, authority and additional DNS records to detect packets that deviated from DNS specifications. It was machine independent and ran on little endian as well as big endian machines.
2. An augmented sanity checker that also checked for responses that do not match queries (a sign of potential cache poisoning), and grouped queries and responses together by DNS ID number. This analyzer also

parsed DNS messages directly from UDP packet payloads.

3. First analyzer to get DNS data from the MySQL database. It grouped domain names together with IP addresses or canonical names nontransitively.

4. Calculates a probability distribution of domain name suffixes broken down by the first byte of all possible IP addresses.

5. Calculates proportion of matching IP addresses across the entire database.

6. Calculates probability distribution of domain name suffix length broken down by the first byte of all possible IP addresses. The data obtained was slightly distorted by misspelled domain names, usually when an end user failed to put a dot into a domain name.

7. Calculates the number of 4 byte IP address changes for every owner in the database.

8. A depth-first exhaustive search for IP addresses broken down by owner. Done as a transitive search through canonical names.

9. Probability distribution of IP addresses, calculated transitively, broken down by owner.

10. Alternate implementation of 9, above.

11. Transitive probability for IP addresses, interactively calculates transition score for input IP addresses for any previously input domain name.

12. Probability distribution of raw traffic levels for 10 minute intervals over 24 hours to determine diurnal patterns of DNS use, if any.

13. Calculates percentiles for raw traffic levels.

14. Gets IP address set for hard-coded owner names and canonical names from ANSWER records.

15. Calculates percentiles for specialized traffic levels. The traffic level is determined by a command line argument.

3. Expert Modules

The expert modules were intended to run collectively as a daemon process, sleeping and periodically waking up to monitor recent history on the subject machine to determine the probability of some sort of DNS subversion attack. Information on recent DNS history came from queries to the MySQL database where information about DNS activity would be stored by a packet sniffer. The driver code for the expert modules was implemented to run either in a test mode that simply cycled through the database, extracting DNS data in 10-minute intervals or as a daemon that extracted only the data on DNS activity since the last time it extracted data for examination.

There were two versions of expert modules implemented. This first version was a testbed for some expert module algorithms and did not cleanly separate one expert module from another. There were algorithms for monitoring raw traffic levels, the amount of IPv6 traffic and TCP traffic implemented in the first version. The level of traffic was mapped into a percentile gleaned from the analyzers. The operational belief for the level of raw traffic is that normal, benign traffic levels vary, but excessively high levels of traffic might correlate to an attack, possibly Denial of Service. The level of IPv6 traffic and TCP traffic was used as input to a modified sigmoid function. The operational

assumption here is "any is bad, more is worse". The modified sigmoid made it possible to treat a zero level as entirely benign, and any nonzero level of traffic as potentially hostile.

The second version of expert modules used a cleaner architecture to cleanly separate the mechanics of each expert module within derivations of a common C++ base class. This afforded the necessary polymorphism to make it possible to adjust or modify the algorithms of any expert module as necessary without disturbing the other expert modules. The algorithms for raw traffic levels, IPv6 traffic and TCP traffic from the first version were reimplemented here and new expert modules to monitor how domain names mapped to IP addresses, specialized traffic levels and anomalous time-to-live values were also created.

The domain name to IP mapping monitor module assumed the presence of a specialized data file for a single domain name that contained information about any number of canonical names and any number of associated IP addresses associated with each canonical name. By providing the name of a different data file for each domain name, multiple instances of this expert module could be used polymorphically to generate attack information about different popular domain names as a means of detecting cache poisoning. In operation, the domain name expert module would get a set of canonical names and IP addresses for a given domain name and compare that information to canonical name and IP information taken from the input file using an asymmetrical similarity metric for set member comparison. The similarity metric function used was:

$$\text{sim}(A, B) = \frac{f(A \text{ and } B)}{f(A \text{ and } B) + f(A - B)}$$

Where A and B are two sets and A is the set of IP addresses taken during the most recent interval and B is the set of IP addresses for the same canonical name taken from the input data file. Set A is considered benign if it is a subset of B, but if A contains any IP addresses not in B, then set A may have been obtained from a poisoned cache. The function f compares IP addresses, byte by byte and produces the smallest number of bytes different as measured from left to right. IP addresses that differ only in the rightmost byte are quantitatively more similar than two IP addresses that differ in the leftmost byte.

There were separate data files created for www.ebay.com, www.google.com and www.amazon.com. The DNS data for the first two domain names had multiple canonical names and a distinct set of IP addresses for each canonical name. The DNS data for the third domain name did not have canonical names, but simply associated different IP addresses with the domain name in multiple DNS responses. These domain names proved to be sufficiently varied in their DNS response behavior to demonstrate that the IP address and canonical name information (if any) for any domain name could be used in the domain name to IP address mapping monitor expert module.

The specialized traffic modules were implemented with a similar strategy. The constructor specified only the name of a data file for the specific name column of the dns_section_line table. Specialized traffic modules for different name values (e.g. PTR, AAAA, etc) could be used as separate expert modules.

The TTL expert module is the first result of thinking like an attacker. If someone wanted to poison a cache, it follows that the attacker would want the cache to stay poisoned for a long time. The TTL expert module looked

for abnormally long time-to-live values in DNS response records. Typical time-to-live values for ANSWER records in DNS responses are two days, as supported by examining DNS response data. Any TTL values greater than 172800 (the number of seconds in two days) are correlated to a possible cache poisoning attack.

Both implementations of the expert modules get the data required for each kind of expert module by making queries of the MySQL database. The first version simply implemented each expert module as a function and the second version implemented each expert module as a derivation of a base C++ class. The latter implementation localizes the queries within each expert module.

4. Calculation of Attack Probabilities

The goal of the expert modules is to calculate the conditional probability of attack given the traffic evidence that each expert module collects. The goal of the merger is to merge the probabilities output from the expert modules into the probability of an actual attack. Bayes' theorem can be applied to both goals to produce the following formula:

$$p(\text{attack}|\text{traffic}) = \frac{p(\text{traffic}|\text{attack})p(\text{attack})}{p(\text{traffic}|\text{attack}) + p(\text{traffic}|\text{no attack})}$$

If the factors on the right side of the formula could be determined, then the probability of an attack given observed traffic could be calculated. The factor $p(\text{attack})$ could be expressed as a fraction with the number of attack packets (or records) observed over the total number of packets (or records) observed. The number of attack packets (or records) exhibiting certain characteristics divided by the total number of packets (or records) exhibiting the same characteristics can be

used to represent $p(\text{attack}|\text{traffic})$. Similarly, the number of benign packets (or records) exhibiting certain characteristics divided by the total number of packets (or records) exhibiting the same characteristics represents $p(\text{traffic}|\text{no attack})$.

5. Using the Database

The expert modules have been implemented to use MySQL queries to a database to get information on DNS activity. There are pros and cons to this approach.

One the plus side, getting different types of information is as easy as constructing a new query.

One the down side, only DNS messages that can be parsed can be entered into the database. It's impossible to detect DNS subversion that grossly violates the DNS definition to the point where the attacking data cannot be parsed and entered into the database, which greatly reduces the ability to detect overt malfeasance. A second problem addresses the efficiency of some queries. It is very desirable to detect responses that have no matching query, since these unmatched responses may be attempts to poison a cache. Unfortunately, while a query to extract this information is easy constructed, in practice it takes a prohibitively long time to complete. In experiments on 204.147.180.70, no such query ever completed in less than several hours.

6. Results

Each of the expert modules calculates a value between 0.0 and 1.0 based on the data that each such module examines. These values are not $p(\text{attack}|\text{traffic})$, although they were intended to be. The reasons for this difference follow.

There is no way to calculate $p(\text{attack}|\text{traffic})$ in any of the expert modules from percentiles or sigmoid functions because of a lack of training data. Referring to the equation based on Bayes' theorem, $p(\text{attack})$ could not be determined because the proportion of attack packets (or records) could not be determined. Similarly, $p(\text{traffic}|\text{attack})$ and $p(\text{traffic}|\text{no attack})$ could not be determined, again because no such training data was available.

The inability to solidly determine the factors involved in a formula for $p(\text{attack}|\text{traffic})$ would be greatly mitigated if a closed feedback loop could be established. If such were possible, then the probabilities for $p(\text{attack})$, $p(\text{traffic}|\text{attack})$ and $p(\text{traffic}|\text{no attack})$ would have been initial probabilities given to an algorithm that would have been able to modify those parameters over time from the actual DNS data it examined while in operation, essentially learning its own parameters over time. Unfortunately, there is no way for any algorithm to get the feedback that makes such learning possible.

This second limitation can be illustrated by an example of a working feedback loop. Consider an automated agent that controls a water valve and receives inputs to increase or decrease the water flow. This agent might not know which way to turn the valve, but it would have the ability to monitor the flow rate. When the agent receives a request to increase the flow of water, it could turn the valve in one direction and through its ability to monitor the flow rate, observe the result and determine if its action was successful. Determining the success or failure of its actions on the valve, the agent would then learn which way to turn the valve to increase or decrease the flow of water as needed. In the case of the DNS subversion detector, there is no mechanism that the detector could use to determine if an attack had actually

occurred, which makes intuitive sense since such attacks are intended not to be detectable. This suggests a paradox for DNS subversion detection: If no mechanism existed that could determine that an attack occurred, then the parameters of a learning mechanism could never be modified and there would be no learning. If a mechanism existed that could determine that an attack occurred, then that mechanism should be used instead of a learning mechanism that could never do better than the monitoring mechanism.

7. Conclusion

We live in a hostile world where various forms of subversion and attacks occur across the Internet. The ability to detect such malfeasance early would be both beneficial and lucrative. The most effective automated detection mechanisms would have the ability to learn from ongoing data collected and be able to adapt to changing conditions without human intervention. Such learning is predicated on the idea of a feedback loop that would allow a mechanism to evaluate and modify its own behavior. Such a feedback loop has not been demonstrated to exist, at least as yet.

REFERENCES

Paul Albitz and Cricket Liu, DNS and BIND, 4th Edition, O'Reilly & Associates, 2001.

Michael Moncur, MySQL, Your visual blueprint to open source database management, Wiley, Publishing, Inc., 2003.

Christopher Irving, "The Achilles Heal [sic] of DNS", SANS Institute, 2003.

Michael Littman, Greg Keim, Noam Shazeer, "A probabilistic approach to solving crossword puzzles", *Artificial Intelligence* 134 (2002).

Peter Turney, Michael Littman, Jeffrey Bigham, Victor Shnayder, "Combining Independent Modules to Solve Multiple-Choice Synonym and Analogy Problems".