SSL Anomaly Detection with Self-Organizing Maps

Jeffrey J. Guy
Johns Hopkins University
jjg@jhu.edu

***Abstract*—Using session-based features as inputs to a self-organizing map, your author develops an SSL anomaly detector to highlight any traffic that does not resemble HTTPS traffic. Additionally, it is shown that a map trained with traffic from one user browsing the web for a few hours is sufficient to filter 98.6% of HTTPS traffic from a large enterprise network with thousands of hosts.**

I. Introduction

MOST ENTERPRISE networks use firewalls and application-level gateways (ALGs) to enforce policies on permitted outbound traffic. Firewalls are used to allow or deny traffic per port, ALGs are used to provide network administrators more granular control for the major protocols. The most common ALG is the web proxy: also used to cache popular requests, they are often configured to filter traffic to prohibited domains.

In a well-configured network, there will not be a TCP or UDP port allowed outbound through the firewall to an arbitrary internet address. All DNS is routed through the organization's internal DNS server; all HTTP/S through the web proxy and all email through the email server. After providing HTTP, DNS and email services to every user, the remaining communications needs can usually be handled on a case-by-case basis, with documented firewall exceptions. The adage ”deny all, allow by exception” applies.

In order to be successful, attackers must write their malicious code to tunnel over one of these common application protocols. The idea is not new: open source implementations exist for DNS tunnels [15], HTTP tunnels [16], SSL tunnels [17], and many more. By tunneling arbitrary traffic over one of these “safe” protocols, attackers can circumvent an organization's security policy.

As ASCII-based clear-text protocols, HTTP and DNS tunnels have numerous features that could differentiate legitimate traffic from malicious traffic. As an encrypted stream, SSL tunnels have significantly fewer features to distinguish legitimate encrypted HTTP traffic from illicit arbitrary traffic wrapped in SSL. There are very few products in industry to highlight unusual SSL traffic; if malicious code were to land in a network and connect out via an SSL tunnel, network administrators have little chance to detect it. This technique, dubbed a *reverse command shell* is commonly used by attackers; the open source exploitation framework Metasploit [18] includes several implementations of SSL reverse command shell shellcode.

This paper demonstrates techniques to classify SSL traffic to discover potential SSL tunnels. **Section II** describes the approach, **section III** discusses related work, **section IV** described the methodology in detail and **section V** presents the results. Conclusions are in **section VI**.

II. Approach

A. Motivation

The communication between an HTTP client and HTTP server is specified by RFC 2616. After establishing a TCP connection, the client sends something similar to:

```
GET /path/to/resource.jpg HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0...
```

If `resource.jpg` exists on the server and is available for retrieval, the server sends something similar to:

```
HTTP/1.1 200 OK
Content-Type: image/jpg
Date: Sun, 25 Apr 2010 17:55:12 GMT
Content-Length: 5656
Connection: close

(5656 bytes of resource.jpg)
```

In HTTP versions 1.1 and higher, the client may use the same TCP socket to request multiple resources to minimize the overhead of establishing TCP connections. After all resources have been requested, the socket closes.

In contrast, there is no protocol specification for a reverse command shell. The client application only needs to communicate with the attacker's server. The specifics of the attacker's wire protocol will vary with his objective, but there is one commonality we might exploit: the attacker gained access to the victim computer for a reason. From the server, he will issue commands to the client and receive some kind of response. After the client establishes the TCP connection, the client will send something like:

```
HELO
```

The server will respond with some command:

```
length: 7
dir c:\
```

And the client will respond with the results:

```
length: 5600
Volume in drive C is DISK\nVolume
Serial...
```

In the attacker's protocol, the connection directionality is the same as HTTPS, but the dataflow semantics are precisely opposite. In HTTPS, small requests are sent outbound and large responses are returned. In the attacker's protocol, small requests are sent inbound and large responses are returned.

Protocol	Bytes to client	Bytes to server	Duration
HTTPS	Large	Small	Short
Attacker's SSL tunnel	Small	Large	Long

TABLE I
SUMMARY OF THE ANECDOTAL DIFFERENCES BETWEEN HTTPS AND AN SSL TUNNEL

If the attacker's server automates commands to the clients, the TCP connection times will be similar to that of HTTPS. However, if the attacker uses the command shell as a remote command line, he is likely to use a single TCP socket for the entire session. This will cause the duration of the TCP session to increase, potentially substantially.

B. Technique

From a collection of packet captures with SSL traffic, the packets are reassembled into TCP sessions and three features extracted: *bytes from server*, *bytes from client* and *duration in seconds*. Using a set of known-good HTTPS traffic, I train a self-organizing map. The resulting map is used to classify SSL sessions into "normal" and "abnormal" classes using a threshold distance from the best-matching node in the map. A detailed presentation is in **Section IV**.

III. Related Work

A. Network traffic analysis

There has been significant work in recent years on techniques protocol identification and fingerprinting. Border and Prakash are often credited with the first attempts to detect HTTP tunnels in their 2004 work. [1] They used simplistic filters from the a variety of sources: formats of the HTTP header requests, request inter-arrival times, size of a single HTTP request, time of day of requests, cumulative size of outbound requests and request regularity. After collecting data of 30 legitimate users surfing the web for 30 days, they used the data to establish one-sided confidence intervals, such that the probability of any legitimate request exceeding the threshold was very small.

Dusi, et al [2] recently proposed their *Tunnel Hunter* technique. Use packet size, inter-arrival time and direction to develop a protocol classifier. They train the classifier using traffic collected from their university perimeter, then establish a variety of tunnels out of the university network and use their classifier to highlight their traffic.

Tunnel Hunter's use of packet size, inter-arrival time and directionality stands in contrast to Border's large and diverse set of features. The use of size, arrival times and direction is increasingly common, due to the remarkable results given how narrow the features are relative to how much information is available. In 2004, Wright et al [12] developed a classifier using profile Hidden Markov Models to differentiate between different application level protocols. In 2006, Liberatore and Levine [5] used those features with Jaccards coefficient and a NaiveBayes classifier to identify the web sites a user was visiting by watching HTTP requests in an SSH-encrypted tunnel. As early as 2001, Song et al [11] demonstrated the use of packet interarrival times in SSH logins to significantly reduce the keyspace necessary for a brute force attack against a user's password.

This paper's use of flow-based features instead of packet-based features is unique. The flow features chosen can be constructed from the typically used packet-based features, but consolidating those into flow metrics reduces computation and data storage requirements at the expense of granularity. As demonstrated in the results, the granularity of packet-based features may be an unnecessary complication for this problem.

B. Self-organizing maps in traffic analysis

The application of self-organizing maps to network traffic analysis and anomaly detection is also an area of frequent research. In 2003, Ramadas et al [9] studied a SOM architecture in an IDS at the perimeter of Ohio University. They used average packet size, directionality and inter-arrival time to train a SOM per network service. They then threw exploit packets past the classifier and used a threshold distance metric to identify abnormal traffic. Our work is similar, but with different features and a more focused objective.

Heywood et al [6] used SOMs with log file data from unix workstations in a hierarchical SOM, Labib et al [7] used just source/destination IPs and protocol as inputs into a SOM to highlight any communication to new hosts. In 2000, Rhodes et al [10] developed a SOM-based DNS classifier with a clever six-feature input that described how many bytes in each packet fit a particular character class (such as alphabetic, numeric, control, and non-ASCII).

IV. Methodology

The approach is discussed in three parts. First is a discussion of the datasets used and feature extracted, the second is a discussion of the SOM's design and finally the SOM's training and subsequent use as a classifier.

A. Dataset and feature selection

To present full experimental results, three datasets are required: known-good HTTPS traffic, known-bad SSL tunnel traffic and a large collection of unknown traffic to test the classifier's performance on a large corpus.

To develop the training set of known-good HTTPS traffic, `tcpdump` was used to collect all `tcp/443` traffic while a user did normal web browsing over a two day period. Sites and activites included HTTPS email to Gmail, a corporate Exchange webmail interface, online banking, custom HTTPS applications and eCommerce transactions from several online vendors. The resulting capture file was 1145 unique `tcp/443` sessions.

To develop a set of known-bad SSL tunnel traffic, `stunnel` [17] and `netcat` [20] were used to launch `/bin/bash`. Over this connection, a series of normal unix commands were executed to simulate an attacker exploring the system and several files transferred to simulate the exfiltration of sensitive data. `tcpdump` was again used to collect the raw traffic, the resulting capture file contained one session per simulated command shell. Both the known-good and known-bad raw captures are available in libpcap format at http://www.jjguy.com/packets.tgz (24MB)

The LBL anonymized enterprise captures [8] were used to test the classifier's performance against a large, unknown dataset. Two days were selected at random (Oct 6 2005 and Oct 7 2005) resulting in 11084 unique `tcp/443` sessions.

For all traces, a modified version of `tcpflow` [21] was used for stream reassembly. The publicly available version was modified to extract the three desired features (bytes to client, bytes to server, duration in seconds) during reassembly. A patch with the modifications for tcpflow v0.21 is available at http://www.jjguy.com/tcpflow.patch.

B. Self-organizing map

The SOM is a 10 x 10 rectangular mesh of nodes as described by Kohonen. [4] Each node is associated with a 3-dimensional vector to store the three features. The chosen features are naturally unbounded, so they are transformed by the logistic function to bound the feature space between (-1, 1):

$$f(x) = \frac{1}{1+e^{-rx}}$$

The value r is set to 10^{-6} for byte counts and 10^{-2} for connection duration. This provides granularity to distinguish amongst the most common values. The best matching node is the closest node as determined by standard Euclidean distance calculation between the weights.

C. Training and classification

All weights of all nodes were initialized to random values, uniformly distributed across the feature space. The 1145 known-good sessions were used to train the map. The known-bad and unknown sessions were compared against the trained map. If the distance from the input vector to the best matching node was greater than a threshold value k, the input vector was flagged as abnormal traffic. For the results in **Section V**, $k = 0.25$.

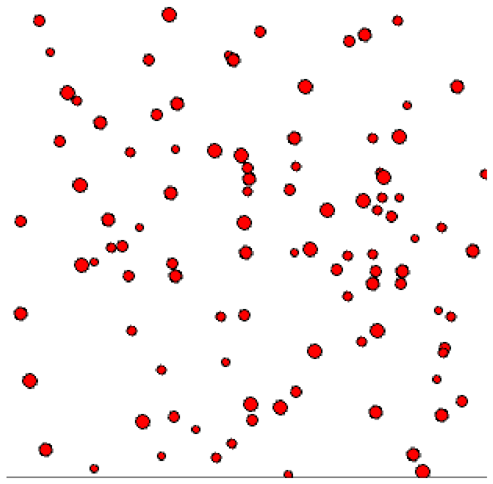
Fig. 1. Map randomly initialized before training; top-left is (0,0), bottom-right is (1,1). y-axis represent bytes to client, x-axis bytes to server and circle diameter session duration in seconds.

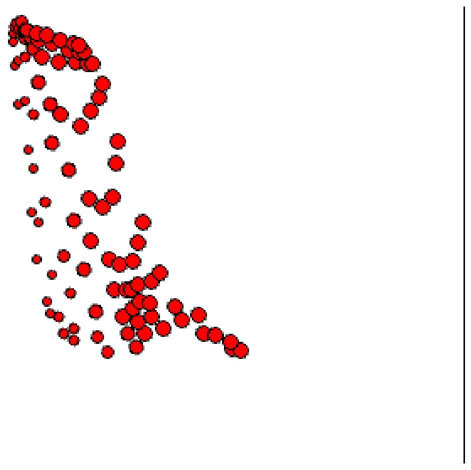
Fig. 2. A map after training with the known good dataset

To visualize the map, nodes were plotted as red circles on a two-dimensional x/y scale, with the y-axis representing *bytes to the client*, x-axis representing *bytes to the server* and the diameter of the circle representing *duration in seconds*. **Figure 1** shows a map after initialization, but before training. **Figure 2** is the map after being trained on the known good sessions.

V. Results

A. Known bad testing

Using a profile of an attacker connected to a reverse SSL command shell for 83 minutes, exploring the system and transferring 3 files between 10 and 20 megabytes produced the a session with 42,966,322 bytes to server, 1524 bytes to

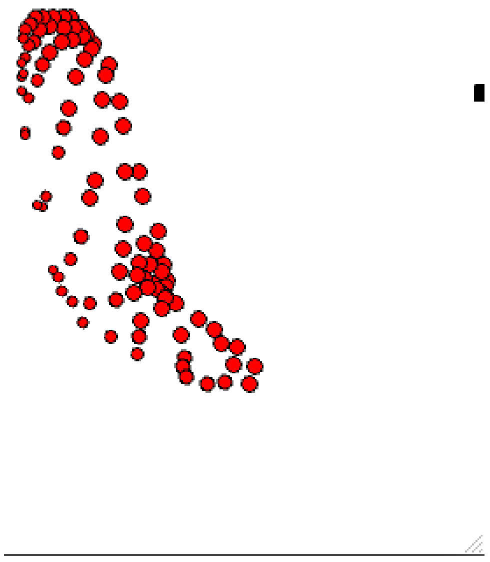

Fig. 3. Trained map, with the known-bad session plotted as a black rectangle.

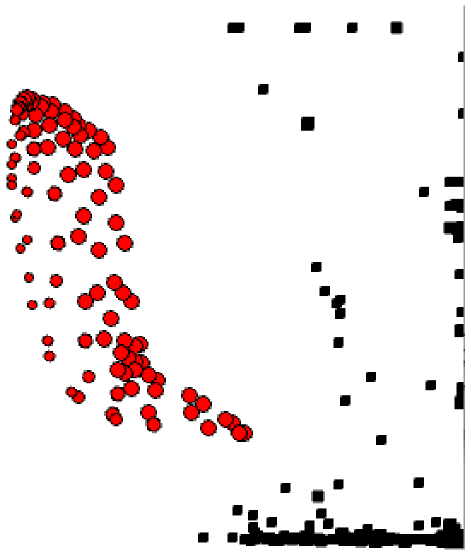

Fig. 4. Trained map, with abnormal sessions from the LBL data plotted as black rectangles.

client and 5008 seconds. This input vector was a distance of 0.36 from the closest matching node in the trained map. **Figure 3** shows a trained map as described above with the known-bad point mapped as a black rectangle.

The point is clearly far outside the profile of normal HTTPS traffic and has an equally large distance from the best matching node. If we isolate just the *bytes to server* metric, the maximum value in the trained map equates to about 1 megabyte sent from client to server. Thus, if an attacker exfiltrates anything over 1 megabyte, he is in danger of being flagged as abnormal traffic. Of course, as evidenced in **Figure 3** the map nodes (circles) with the largest *bytes to server* also have large *bytes to client*. For the attacker's profile of small commands sent from the server, he will be limited to just hundreds of kilobytes before being in danger of being flagged as abnormal traffic. This danger is further increased by the duration measure.

There is a subjective analysis in how bad known-bad needs to be; the exfiltration of 42 megabytes of data over the course of an hour is a certainly a less frequent occurrence than an attacker spending 10 minutes and transferring no files. However, in a corporate enterprise environment, data theft is the amongst the most long-lasting and damaging results of an intrusion.

B. Large scale performance

The dataset used to train the map was clearly narrow. While the profile should mimic most HTTPS traffic, how well will it scale on a enterprise network with thousands of concurrent hosts and a more diverse use of SSL traffic? Using LBL's enterprise captures, [8] all `tcp/443` sessions were compared to a trained map. Any session with a distance to the best matching node greater than the threshold $k = 0.25$ was flagged as abnormal traffic. Of the 11084 sessions in the LBL capture, 154 (1.38%) were flagged as abnormal. **Figure 4** shows a trained map with all abnormal LBL sessions plotted as black rectangles. The results of both are summarized in **Table II**.

Dataset	Total sessions	Count abnormal (%)
Known good	1145	n/a
Known bad	1	1 (100%)
LBL unknown	11084	154 (1.38%)

TABLE II
RESULTS OF SOM CLASSIFIER ON EACH DATASET

Since the LBL data is anonymized and payloads stripped, there does not exist sufficient data to validate the abnormal sessions. It is assumed they are all legitimate network traffic and simply represent different applications using SSL. There is a large cluster of sessions with high bytes in both direction (those in the bottom right of Figure 4), while the rest represent large uploads from client to server. Of course, as a national lab LBL is a candidate for attack by other nation-states. In 2007, law enforcement officials were investigating intrusions at LBL and suspected nation-state attackers[1].

Even if all abnormal sessions are legitimate, a false positive rate of 1.38% is better than or consistent with other techniques doing similar analysis. For instance, Wright's 2004 HMM protocol detector classified HTTP with a 3.1% failure rate and SMTP with a 2.1% failure rate. Dusi's 2009 *Tunnel Hunter* technique classified chat tunneled over SSH with a 9.1% failure rate. More direct comparisons are difficult due to different objectives of the available techniques.

VI. Conclusion

HTTPS traffic follows a startlingly narrow profile. Using a profile from one user normally browsing the web for just a

[1]http://abcnews.go.com/TheLaw/Technology/story?id=3966047

few hours, the self-organizing map filtered 98.6% of sessions from a large, enterprise network with thousands of hosts.

A. *Effectiveness*

However, used alone the technique is insufficient. While 154 abnormal sessions is significantly fewer than the 11,000 sessions analyzed, manual analysis of 154 connections every two days is a non-trivial effort. The technique may be effective as a data reduction methodology, to feed the resulting session information to a more computationally-intensive task. It may also be more effective combined with hand-crafted filters of known-good servers to remove common sessions with abnormal characteristics from custom applications. Finally, it may be worthwhile to capture session data from other applications using SSL and train the map with both HTTPS profiles. In the LBL data, it may account for the application with the sessions clustered in the bottom right of Figure 4.

B. *Feature selection*

The effectiveness of session-based features instead of packet-based features is worth further consideration. Wright, et al's 2006 paper [14] developed protocol profiles using packet-based features to produce protocol detectors. It is worth further exploration to discover if there are similar distinctions in session-based features between major protocols. With network speeds constantly increasing, the reduced data storage and computation time for session features will enable wider application if successful.

C. *Attacker avoidance*

If an attacker was aware an organization was using session profiling as described above, could he avoid detection? Moving the reverse command shell to another protocol could be mitigated by good outbound filtering and other tools. The attacker could limit the data exfiltrated and duration of his connections, but that would negatively impact his objective for gaining access in the first place. The attacker could programmatically limit the data transferred in any single connection to his own threshold value and force a new connection, but the SOM could add a fourth feature to track count of multiple connections over time between client and server. Finally, the attacker could multiplex his tunnel between multiple external servers each acting as a proxy to a single hidden server reassembling the traffic, but this is a significant increase in complexity.

References

[1] K. Borders, A. Prakash, *Web tap: detecting covert web traffic*. CCS04: Proceedings of the 11th ACM conference on Computer and Communications Security, Washington DC, USA, 2004, pp. 110120.

[2] M. Dusi, M. Crotti, F. Gringoli, L. Salgarelli, *Tunnel Hunter: Detecting Application-Layer Tunnels with Statistical Fingerprinting*, Elsevier "Computer Networks" (COMNET), Vol.53, No.1, pp. 81-97, Jan. 2009.

[3] J Horton, R Safavi-Naini, *Detecting Policy Violations through Traffic Analysis*, 22nd Annual Computer Security Applications Conference (AC-SAC '06), Miami Beach, Florida, USA, December 2006, 109-120.

[4] T. Kohonen. *Self-Organizing Maps.* Berlin: Springer, 1995.

[5] M. Liberatore, B. N. Levine, *Inferring the source of encrypted http connections* CCS 06: Proceedings of the 13th ACM conference on Computer and Communications Security, Alexandria, Virginia, USA, 2006, pp. 255263.

[6] Peter Lichodzijewski, A. Nur Zincir-Heywood, Malcolm I. Heywood, *Host-based intrusion detection using self-organizing maps,* Proceedings of the 2002 IEEE World Congress on Computational Intelligence, 2002.

[7] K. Labib, R. Vemuri, *NSOM: a real-time network-based intrusion detection system using self-organizing maps*, Networks and Security, 2002.

[8] R. Pang, M. Allman, V. Paxson, and J. Lee. *The devil and packet trace anonymization.* SIGCOMM Comput. Commun. Rev., 2006.

[9] Manikantan Ramadas, Shawn Ostermann, and Brett Tjaden, *Detecting Anomalous Network Traffic with Self-organizing Maps* RAID 2003, LNCS 2820, pp. 3654, 2003.

[10] Rhodes B., Mahaffey J., Cannady J., *Multiple Self-Organizing Maps for Intrusion Detection* Proceedings of the NISSC 2000 conference.

[11] Dawn Song, David Wagner, and Xuqing Tian. *Timing analysis of keystrokes and SSH timing attacks.* In Proceedings of the 10th USENIX Security Symposium, August 2001.

[12] Charles Wright, Fabian Monrose, and Gerald M Masson. *HMM profiles for network traffic classification (extended abstract).* In Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pages 915, October 2004.

[13] Charles V. Wright, Fabian Monrose, Gerald M. Masson, *On Inferring Application Protocol Behaviors in Encrypted Network Traffic*. Journal of Machine Learning Research 7 (2006) 2745-2769,

[14] C. Wright, F. Monrose, and G. Masson. *Using visual motifs to classify encrypted traffic.* Proceedings of the ACM Workshop on Visualization for Computer Security, 2006

[15] dns2tcp, open-source DNS tunnel software, http://www.hsc.fr/ressources/outils/dns2tcp/index.html.en

[16] HTTPTunnel, open-source HTTP tunnel software, http://www.nocrew.org/software/httptunnel.html

[17] STunnel, open source SSL tunnel software, http://www.stunnel.org

[18] The Metasploit Project, http://www.metasploit.com/

[19] tcpdump, originally a product of LBL's Network Research Group, http://ee.lbl.gov/

[20] netcat, the network swiss army knife, http://netcat.sourceforge.net/

[21] The TCP Flow Recorder, http://www.circlemud.org/ jelson/software/tcpflow/