SPID-based Method of Trojan Horse Detection

Xun-yi Ren   Gui-bing Qian

College of Computer Nanjing University of Posts and Telecommunications
Email: renxy@njupt.edu.cn

Abstract

The Trojans have become a hidden threat to computer security, how to identify various Trojan efficiently and accurately is a current research focus. In this paper, based on the research of SPID's attribute meters, we proposed a method to detect and identify Trojan that based on the SPID feature optimization. The method uses SPID attribute meters to analysis with common protocol, generating a model to identify Trojan. Then, get the combination of 12 attribute meters to identify Trojan by statistical the result of the recognition. Experimental results shows that the optimized combination of attribute meters have a high efficiency to identify Trojan based on keeping SPID detection accuracy.

Keywords: Trojan horse, SPID, detection, identification

1. Introduction

Trojan [1] is a destructive program, and it is the main threat to the security of the network and host. Trojans communicate with the remote control host by pretending to be a useful or interesting program. Trojan has very large hazards, so the Trojan detection technology becomes the hotspot of the information security.

Nowadays, Trojan detection technology can be divided into several methods [2]: (1) through network monitoring finding network traffic anomalies and then blocking it, or definite the rules that make the Trojans cannot communicate. (2) Signature technology: the method of this technology is the main technology of anti-virus software by matching features to detect the Trojan; (3) Real-time monitoring: Synchronization monitor the running process of the system (4) Behavior analysis: According to the program's dynamic behavior characteristics to identify Trojan.

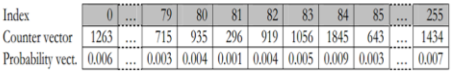
These methods have their pros and cons, The Trojan detection technology of SPID feature optimized is a web-based, real-time detection technology; it is different from the previous Trojan detection technology. Using the network characteristics attributes meters to generate protocol model library and statistical-based to identify Trojans has a high recognition rate and a wide range of adaptability. In this paper, we proposed a method to detect and identify Trojan based the SPID characteristics optimized. The method analysis common protocol by SPID attribute meters and then generates a protocol model library to identify Trojan. Then, get the 12 optimized characteristic combinations of attribute meters by the statistical results.

2. SPID

SPID [3] (Statistical Protocol Identification) is a method based on statistical for protocol identification.

SPID's main purpose is to recognize network communication with which protocol the application layer is, it is not coarse-grained traffic classification(such as P2P or web), but accurate identification of which protocol it is.

Each protocol model has a range of fingerprint feature [4], which is the probability distribution of the application layer payload data or flow characteristics (size, direction, time of arrival, etc.). Each attribute meter is 2*256 arrays (see the example in Figure 1). The first line is the Counter vector, and the second line of is the Probability vector. Counter vector value is a positive integer, and each of the values expressed the value of each package to obtain in the corresponding index of this in the attribute meter.



Index	0	...	79	80	81	82	83	84	85	...	255
Counter vector	1263	...	715	935	296	919	1056	1845	643	...	1434
Probability vect.	0.006	...	0.003	0.004	0.001	0.004	0.005	0.009	0.003	...	0.007

Figure 1. Fingerprint feature of one atrributemeter

Calculate the Kullback-Leibler [5](K-L)divergence of the session P with library model Q, calculate K-L value of the attribute meter value of P and Q, each attribute meter corresponds to a K-L value, last get the average K-L. For example, using attribute meter A, B, C to identify this session, calculate K-L value with default library model's attribute meter A, B, C, then get 3 K-L value, get the average K-Laver. K-L formula is as follows:

$$D_{K-L}(P_{attr} \| Q_{attr,\,prot}) = \sum_{i} P_{attr}(i) * \log \frac{P_{attr}(i)}{Q_{attr,\,prot}(i)} \tag{1}$$

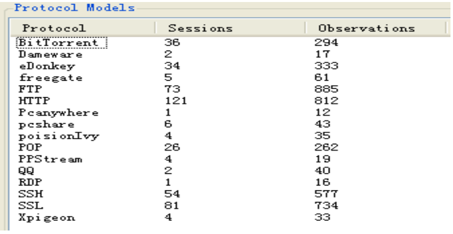
Compare the K-L_{aver} with Pre-set threshold (equal 2.04), less than the threshold value, and the smallest value can be recognized P as protocol Q.

3. SPID-based feature optimization of Trojan Horse identification

3.1. Selection and generation of model library

We select 16 protocols or application as model library by observing existing product of identifying network traffic and specific Trojan identification, such as PoisonIvy 、 xPigeon 、 PcShare 、 PCanywhere 、 DameWare 、 RDP 、 Freegate and other common protocol, etc. Now, we not only select the Trojans as a model library but also the traffic without obvious characteristics and common communications applications as a background protocol.

Consider packet capture analysis carried out on the Trojans different functions (such as screen shots, to transfer files) during the Trojans crawl, The test found no big difference in the recognition effect. Training packets, generate 16 existing protocol model, see Figure 2, Sessions are multiple sessions under different environmental, Observations are packet number observed. Because of the SPID limited, it must be the source ip and destination ip corresponds to a port number during the packet training, thus distinguish packet as a single session, and then training model library.



Protocol	Sessions	Observations
BitTorrent	36	294
Dameware	2	17
eDonkey	34	333
freegate	5	61
FTP	73	885
HTTP	121	812
Pcanywhere	1	12
pcshare	6	43
poisionIvy	4	35
POP	26	262
PPStream	4	19
QQ	2	40
RDP	1	16
SSH	54	577
SSL	81	734
Xpigeon	4	33

Figure 2. Model Library

3.2. Selection of SPID atrributemeters

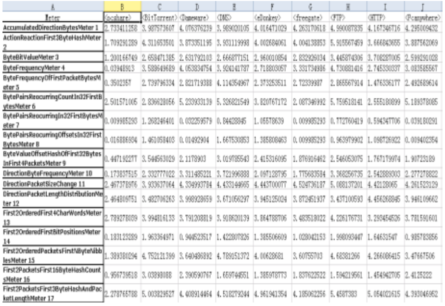
The SPID is based on the statistical methods and it has 34 attributemeters available for free combination. Thus, we have to select the best combination of attributemeters for accurate identification of Trojan. The best attributemeters selection procedure is as follows:

- Crawl known protocol packets P1,P2....P34 of model library;
- Calculate K-L value between specific protocol of the model library (for example, set PcShare to P1) with the each protocol in model library, the results shown in Figure 3 (due to the large amount of data, lists only a part). Column of Meter are 34 attributemeters, column of B,C...J are the K-L values of P1 compare with model library;
- Select the attribute meter A's KL value less than 2.04 of PcShare in model library. That is the PcShare column in Figure 3. Elected large discrimination of B1 with other protocols KL value in A;
- In turn, get B2, B3.... B34 in accordance with step one, two without PcShare;
- Selected attributemeters which has common measure, Selection rules: the same attribute meter occurrences is greater than or equal to 4, the results are shown in Figure 4;
- Select the attribute meter that meets 4th step. Now, we selected 12 attributemeters, the column of Meter number is greater than or equal to 4 in Figure 4.

3.3. Performance verification

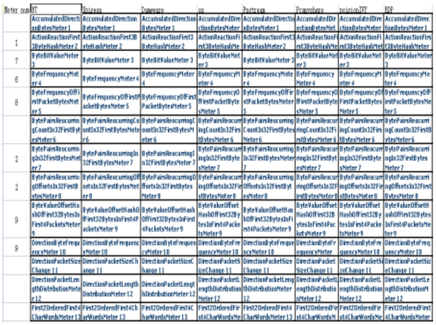
Configuration 12 common attributemeters we selected, running SPID Algorithm Proof-of-Concept 0.4.6 [3], capture network traffic packets P include PoisonIvy, PcShare through Wireshark.

First, test the effect of recognition of P1 to P34, take example of P1(PcShare), the results are shown in Figure 5. Area ① are P1 packet information including the source IP (Client IP), the destinate of the IP (Server IP), source port (C. Port)., destination port (S. Port), the observation of the number of packets (Observation) and the identified protocol (Protocol). Area ② are output K-Laver(Divergence)of the PcShare to model library as well as the discrimination compared to other protocols (Match Percentage). Regional ③ is the default model library.



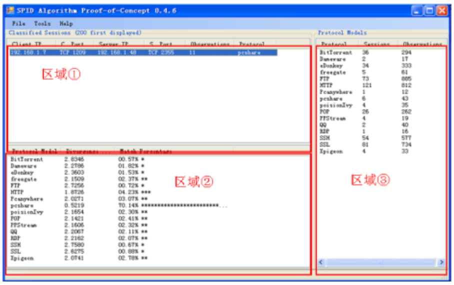
A	B	C	D	E	F	G	H	I	J
Meter	<pcshare>	<BitTorrent>	<Dameware>	<DNS>	<eDonkey>	<freegate>	<FTP>	<HTTP>	<Pcanywhere>
AccumulatedDirectionBytesMeter 1	2.733411258	3.987573607	4.076376239	3.989020105	4.016471029	4.263170618	4.990087835	4.167346716	4.295009432
ActionReactionFirst3ByteHashMeter 2	1.709291289	4.311653501	3.873351195	3.931119993	4.002684061	4.004138853	5.915567459	3.666843655	3.887562069
ByteBitValueMeter 3	1.200166749	2.658471385	2.631792103	2.666877151	2.960010854	2.832926034	3.445874306	3.708287005	2.599291028
ByteFrequencyMeter 4	0.03948913	3.588649689	4.053834754	3.924141787	2.718803057	3.331734986	4.730881416	2.745330337	3.083585567
ByteFrequencyOfFirstPacketBytesMeter 5	0.3502357	2.739796334	2.821719388	4.114354967	2.373253511	2.72339987	2.865567914	1.476336177	2.492689614
BytePairsReocurringCountIn32FirstBytesMeter 6	2.501571005	2.836628056	5.233933139	5.326821549	3.820767172	2.087346992	5.759518141	2.555180899	5.189378085
BytePairsReocurringIn32FirstBytesMeter 7	0.009985293	1.268246401	0.032259579	0.84428845	1.05578639	0.009985293	0.772760419	0.594347706	0.039180291
BytePairsReocurringOffsetsIn32FirstBytesMeter 8	0.016886934	1.461058403	0.01492904	1.667530853	1.385808463	0.009985293	0.963979902	1.098726922	0.009402354
ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	0.447192277	3.544563029	2.1178903	3.019785543	2.415316095	1.876916462	2.546053075	1.767179974	1.90723189
DirectionByteFrequencyMeter 10	0.173837515	2.332777022	3.311485221	3.721996888	2.097128795	1.775683584	3.368256735	2.542889303	2.277278822
DirectionPacketSizeChange 11	2.467378976	3.933637064	4.334993784	4.433144665	4.443700077	4.524736187	5.088137201	4.42128065	4.261523129
DirectionPacketLengthDistributionMeter 12	2.464809751	3.482706263	3.998928659	3.671056297	3.945125024	3.872451937	3.437100593	4.456268845	3.946109662
First2OrderedFirst4CharWordsMeter 13	2.789278039	3.994816133	3.791208819	3.918620139	3.864788706	3.483518022	4.226176731	3.293454526	3.781591601
First2OrderedFirstBitPositionsMeter 14	0.183123289	1.963364971	0.944523517	1.422807826	1.385506609	1.028042153	1.998093447	1.64631547	0.985783856
First2OrderedPacketsFirstNByteNibblesMeter 15	1.389380294	4.752121399	3.640496892	4.789151372	4.00628681	3.60755703	4.68381266	4.266086415	3.47667506
First2PacketsFirst16ByteHashCountsMeter 16	0.956739518	3.03898088	2.390590767	1.659744551	1.385978773	1.837622522	1.594219561	1.454942705	2.4125222
First2PacketsFirst3ByteHashAndPacketLengthMeter 17	2.278765788	5.003829527	4.408914464	4.518279244	4.961941354	4.185062256	5.4587383	5.054021615	4.393046952

Figure 3. K-L output of P1 to model library



Meter num	BT	[illegible]	Dameware	qq	Postream	Pcanywhere	poisionIVY	RDP
	AccumulatedDirectionBytesMeter 1	AccumulatedDirectionBytesMeter 1	AccumulatedDirectionBytesMeter 1	AccumulatedDirectionBytesMeter	AccumulatedDirectionBytesMeter 1	AccumulatedDirectionBytesMet	AccumulatedDirectionBytesMeter	AccumulatedDirectionBytesMeter 1
1	ActionReactionFirst3ByteHashMeter 2	ActionReactionFirst3ByteHashMeter 2	ActionReactionFirst3ByteHashMeter 2	ActionReactionFirst3ByteHashMe	ActionReactionFirst3ByteHashMeter 2	ActionReactionFirst3ByteHashMe	ActionReactionFirst3ByteHashMet	ActionReactionFirst3ByteHashMeter
7	ByteBitValueMeter 3	ByteBitValueMeter 3	ByteBitValueMeter 3	ByteBitValueMeter 3	ByteBitValueMeter 3	ByteBitValueMeter 3	ByteBitValueMeter 3	ByteBitValueMeter 3
6	ByteFrequencyMeter 4	ByteFrequencyMeter 4	ByteFrequencyMeter 4	ByteFrequencyMeter 4	ByteFrequencyMeter 4	ByteFrequency Meter 4	ByteFrequencyMeter 4	ByteFrequencyMeter 4
8	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5	ByteFrequencyOfFirstPacketBytesMeter 5
	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6	BytePairsReocurringCountIn32FirstBytesMeter 6
2	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7	BytePairsReocurringIn32FirstBytesMeter 7
2	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8	BytePairsReocurringOffsetsIn32FirstBytesMeter 8
9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9	ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter 9
9	DirectionByteFrequencyMeter 10	DirectionByteFrequencyMeter 10	DirectionByteFrequencyMeter 10	DirectionByteFrequencyMeter 10	DirectionByteFrequencyMeter 10	DirectionByteFrequencyMeter	DirectionByteFrequencyMeter 10	DirectionByteFrequencyMeter 10
	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11	DirectionPacketSizeChange 11
	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12	DirectionPacketLengthDistributionMeter 12
	First2OrderedFirst4CharWordsMeter 13	First2OrderedFirst4CharWordsMeter 13	First2OrderedFirst4CharWordsMeter 13	First2OrderedFirst4CharWordsMe	First2OrderedFirst4CharWordsMeter 13	First2OrderedFirst4CharWordsM	First2OrderedFirst4CharWordsMe	First2OrderedFirst4CharWordsMeter

Figure 4. Comprehensive 12 meters


SPID Algorithm Proof-of-Concept 0.4.6
File Tools Help
Classified Sessions (200 first displayed)
Protocol Models
区域①
区域②
区域③

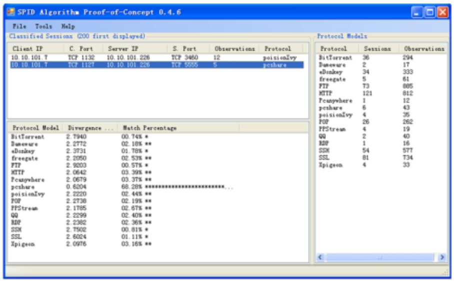
Client IP	C. Port	Server IP	S. Port	Observations	Protocol
192.168.1.7	TCP 1209	192.168.1.48	TCP 2355	11	pcshare

Protocol Model	Divergence ...	Match Percentage
BitTorrent	2.8346	00.57% *
Dameware	2.2786	01.82% *
eDonkey	2.3603	01.53% *
freegate	2.1509	02.37% **
FTP	2.7256	00.72% *
HTTP	1.8726	04.23% ***
Pcanywhere	2.0271	03.07% **
pcshare	0.5219	70.14% ************************...
poisionIvy	2.1654	02.30% **
POP	2.1421	02.41% **
PPStream	2.1606	02.32% **
QQ	2.2067	02.11% **
RDP	2.2162	02.07% **
SSH	2.7580	00.67% *
SSL	2.6275	00.88% *
Xpigeon	2.0741	02.78% **

Protocol	Sessions	Observations
BitTorrent	36	294
Dameware	2	17
eDonkey	34	333
freegate	5	61
FTP	73	885
HTTP	121	812
Pcanywhere	1	12
pcshare	6	43
poisionIvy	4	35
POP	26	262
PPStream	4	19
QQ	2	40
RDP	1	16
SSH	54	577
SSL	81	734
Xpigeon	4	33

Figure 5. PcShare recognition effect

The same method as above, test the packet p which includes PoisonIvy and PcShare, we know that the port 5555 of host computer 10.10.10.226 is PcShare, port 3460 is PoisonIvy, the result is shown in Figure 6.


SPID Algorithm Proof-of-Concept 0.4.6
File Tools Help
Classified Sessions (200 first displayed)
Protocol Models

Client IP	C. Port	Server IP	S. Port	Observations	Protocol
10.10.101.7	TCP 1132	10.10.101.226	TCP 3460	12	poisionIvy
10.10.101.7	TCP 1127	10.10.101.226	TCP 5555	5	pcshare

Protocol Model	Divergence ...	Match Percentage
BitTorrent	2.7940	00.74% *
Dameware	2.2772	02.18% **
eDonkey	2.3731	01.78% *
freegate	2.2050	02.53% **
FTP	2.9203	00.57% *
HTTP	2.0642	03.39% **
Pcanywhere	2.0679	03.37% **
pcshare	0.6204	68.28% ************************...
poisionIvy	2.2220	02.44% **
POP	2.2738	02.19% **
PPStream	2.1785	02.67% **
QQ	2.2299	02.40% **
RDP	2.2382	02.36% **
SSH	2.7502	00.81% *
SSL	2.6024	01.11% *
Xpigeon	2.0976	03.16% **

Protocol	Sessions	Observations
BitTorrent	36	294
Dameware	2	17
eDonkey	34	333
freegate	5	61
FTP	73	885
HTTP	121	812
Pcanywhere	1	12
pcshare	6	43
poisionIvy	4	35
POP	26	262
PPStream	4	19
QQ	2	40
RDP	1	16
SSH	54	577
SSL	81	734
Xpigeon	4	33

Figure 6. Effect of obfuscation packet P

We conducted a number of experiments, using 12 attributemeters can be able to maintain the original SPID detection accuracy, but detection speed significantly improved, therefore, confirmed that the proposed method is accurate and efficient.

4. Conclusion and prospect

What we proposed is a principle to early identify Trojan, generate model library by refining the traffic of Trojan as well as the remaining common protocol, it contains 16 protocols and applications. Then, using statistical observation method to extract the 12 best combinations of attribute meters can identify the protocol of this model library. Finally, give the recognition result, so can be further verify the feasibility of this principle. However, due to the less type of model library, it can be identify the protocol which is in. In addition, the model library training is less, but theoretically, it is more accurate if it has enough training. These problems are the focus work of the future.

5. References

[1] Ming Zhu, Sai Xu, Chunming Liu. “Analysis of Trojan Horse and Its Detection”, Computer Engineering and Applications, vol.2, no 28, pp.176-178. 2003

[2] D. Agrawal and et al, “Trojan detection using icfingerprinting”, *IEEE Symp. On Security and Privacy*, pp.296--310, 2007.

[3] Hjelmvik, E., John, W., “Breaking and Improving Protocol Obfuscation”, Technical Report No: 2010-05, Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden 2010.

[4] E. Hjelmvik. Wiki page of SPID. http://sourceforge.net/apps/mediawiki /spid/index.php

[5] S. Kullback and R. A. Leibler, “On information and sufficiency”, Annalyse of Mathematical Statistics, vol. 22, pp. 49–86, 1951.

[6] Xiaoqing Han,Jianfeng Wang,Wei Zhong, “Analysis and prevention of computer virus”, Beijing: Publishing House of electronics industry, pp.207-209, 2006.

Acknowledgement

Supported by National Natural Science Foundation of China (61073188), China Postdoctoral Science Foundation (20100471355)