

BotTalker: Generating Encrypted, Customizable C&C Traces

Han Zhang, Christos Papadopoulos
Computer Science Department
Colorado State University
Fort Collins, Colorado, United States
Email: zhang@cs.colostate.edu, christos@cs.colostate.edu

Abstract—Encrypted botnets have seen an increasing use in recent years. To enable research in detecting encrypted botnets researchers need samples of encrypted botnet traces with ground truth, which are very hard to get. Traces that *are* available are not customizable, which prevents testing under various controlled scenarios. To address this problem we introduce BotTalker, a tool that can be used to generate customized encrypted botnet communication traffic. BotTalker emulates the actions a bot would take to encrypt communication. It includes a highly configurable encrypted-traffic converter along with real, non-encrypted bot traces and background traffic. The converter is able to convert non-encrypted botnet traces into encrypted ones by providing customization along three dimensions: (a) selection of real encryption algorithm, (b) flow or packet level conversion, SSL emulation and (c) IP address substitution. To the best of our knowledge, BotTalker is the first work that provides users customized encrypted botnet traffic. In the paper we also apply BotTalker to evaluate the damage result from encrypted botnet traffic on a widely used botnet detection system - BotHunter and two IDS' - Snort and Suricata. The results show that encrypted botnet traffic foils bot detection in these systems.

I. INTRODUCTION

Botnets constitute one of the most serious threats in the Internet today, being responsible for malicious activities such as DDoS, spam, espionage and much more. Bots cost users and network operators billions of dollars every year.

In a recent alarming development, bots began using encrypted communication channels. Storm utilizes XOR for encrypted communications. Waledac's communication is encrypted by AES. All C&C traffic in Rustock is encrypted using RC4. In addition, an advanced hybrid peer-to-peer botnet with several new features to make the botnet hard to detect is designed in [18]. One of these features is that each bot randomly generates a symmetric encryption key for communication. In [17], Stinson describes how encrypted communication makes the botnet defeat detection systems.

Encrypted botnet traffic makes detection much harder because many detection systems rely on DPI. In addition, encryption makes it very hard to get ground truth necessary to develop detection algorithms. While bot communication can be captured in sandboxes from binaries harvested around the net, traces collected from sandboxes are fairly static. Researchers would like to adjust parameters such as encryption type, timing, etc., the same way a bot might act to alter its signature and foil detection.

To address these needs we designed and implemented BotTalker: a tool that researchers can use to generate customized encrypted botnet traffic. BotTalker emulates the actions a bot would take to encrypt communication and produces traces that look like they come from real botnets. It includes a highly configurable encrypted-traffic converter along with real, non-encrypted bot traces and background traffic. The converter is able to convert non-encrypted botnet traces into encrypted ones. It enables customization along three dimensions: (a) selection of real encryption algorithm, (b) flow or packet level conversion, SSL emulation and (c) IP address substitution. Many standard encryption algorithms are supported, including AES, DES, RC4 and more. The reason for providing packet/flow conversion and SSL emulation is to support a broad range of encryption methods. In packet conversion we emulate the case where a bot encrypts packets individually as they are transmitted. Flow conversion emulates the case where a bot transmits encrypted objects. SSL conversion emulates the case when the botnet exchange information via SSL connections. Finally, our converter provides the ability to obfuscate IP addresses. This is needed because many bot detection tools look for communication with blacklisted IP addresses, which can easily be subverted. To our best knowledge, BotTalker is the first work that provides user customized encrypted botnet traffic converted from real botnet traffic, rather than simulated traffic. BotTalker contributes to the community in the following ways.

- 1) Benefits IDS Developers: BotTalker provides IDS developers the ability to generate customized encrypted botnet traffic from real botnet traces to test their detection algorithms. The users can test whether the IDS works on botnets using strong encryption, weak encryption (e.g., XOR), or fails on all encrypted traffic.
- 2) Benefits IDS Customers: BotTalker helps customers (e.g., network administrators) evaluate a new IDS and make informed decisions. Customers test a new IDS by blending background traffic collected from their network with encrypted bot traffic generated by BotTalker, and then feed the traffic to the new IDS for performance testing.
- 3) Benchmark for IDS: Currently the detection systems use their own data sets for evaluation and the data sets are not public. It is impossible to compare the detection sys-

tems' performance directly. Since our system is public it can be used as a benchmark to evaluate different IDS'.

Our final contribution is to evaluate the effect of encrypted botnet traffic on three common security systems - BotHunter, Snort and Suricata. The results show that unfortunately, encrypted traffic foils bot detection in these systems.

II. BOT TALKER DESIGN

The goal of BotTalker is to emulate the actions a bot would take when encrypting communication and produce traces that are nearly identical to those generated by real bots. To achieve this goal, we first introduce several requirements that BotTalker needs to satisfy. Then we describe the architecture.

A. Requirements

- 1) Retain network level characteristics of original traces:
Network level characteristics include timestamps, packet size, flow size, etc. These can be typically calculated from traffic at the network level. Retaining these characteristics is important because many IDSs examine them to detect malicious behavior.
- 2) Use of real encryption algorithms:
Many real botnets use standard encryption algorithms because they are easy to use. For example, Rustock, Storm and Waledac use RC4, XOR and AES to encrypt communications respectively. BotTalker supports many standard encryption algorithms to encrypt the communication payload. We expand further on this point in Section VI.
- 3) Emulate various encryption schemes:
When a bot encrypts traffic it may encrypt one packet at a time, the entire flow, simply use an existing encrypted channel such as SSL. Packet level encryption emulates the case where packets are encrypted individually. For example, when the controller transmits a stream of commands to the bots via the same connection, it may put each command into a single packet, and each packet may be encrypted individually. Flow level encryption emulates the case where an entire object is transmitted, such as binaries, spam lists, etc. When a bot wants to transfer these encrypted objects, it usually encrypts them in memory first, then the entire encrypted data is chopped into multiple packets and sent out. SSL can be used when exchanging information via HTTPS, which is typically not blocked at firewalls. In fact, Bayer found that many malware already use SSL to protect their communications [6]. BotTalker provides the capability to emulate a botnet's communication via SSL.
- 4) Use real background traffic:

The use of real background traffic and real bot traffic is important to assess false positives and false negatives of detection algorithms. However, blending bot traffic and background traffic when such traces were collected in different networks has many challenges. For example, does one pick existing hosts from the background traffic and assign bot traffic to them? Or do we introduce new

hosts along with those present in the background traffic trace? BotTalker supports both approaches. However, in order to implement this correctly, BotTalker not only uses IP addresses to match the background traffic, but also adjusts bot traffic timestamps to match the new network's characteristics. This step requires that BotTalker estimates some characteristics of the background traffic (for example, delay) and uses those when adapting bot traffic to background traffic.

B. BotTalker Architecture

We designed BotTalker to address the above requirements. The architecture is shown in Figure 1. BotTalker includes a highly configurable encrypted-traffic converter along with background traffic and real, non-encrypted bot traces. The converter is the core module. There are two inputs to the converter: the bot trace to encrypt and a set of parameters (e.g., encryption algorithm, packet or flow level conversion). After generating the encrypted botnet traces, users may optionally blend it with the provided background traffic or with background traffic provided by the user. The final result is now ready to use.

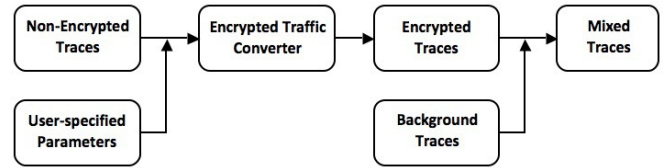


Fig. 1. BotTalker Architecture

BotTalker comes with a collection of bot traffic and background traffic. At the moment, it provides 140 Butterfly and Kraken bot traces[7], [13], as well as traffic from the 2009 DARPA dataset. We provide more details about the data sets in Section IV-D. Note that is trivial to add more traffic to the current collection.

TABLE I
BOT TALKER ENCRYPTION ALGORITHMS

Encryption Algorithm	Cipher Mode
XOR	N/A
RC4	N/A
RC4_40	N/A
DES	ECB, CBC, CFB, OFB
2-key Triple-DES	ECB, CBC, CFB, OFB
3-key Triple-DES	ECB, CBC, CFB, OFB
DESX	CBC
RC2	ECB, CBC, CFB, OFB
Blowfish	ECB, CBC, CFB, OFB
CAST	ECB, CBC, CFB, OFB
AES_128	ECB, CBC, CFB1, CFB8, CFB128, OFB
AES_192	ECB, CBC, CFB1, CFB8, CFB128, OFB
AES_256	ECB, CBC, CFB1, CFB8, CFB128, OFB

III. ENCRYPTION ALGORITHMS

Stream and block ciphers are two main approaches in encrypting data. Stream ciphers encrypt plaintext of variable length. Block ciphers encrypt fixed-length groups of bits called blocks. In our implementation, we support both stream and

block ciphers. In particular, four modes of block ciphers are supported, including Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB). Among these modes, ECB and CBC require the data size be a multiple of block size, while CFB and OFB do not. The various standard encryption algorithms provided by BotTalker are shown in Table I. We implement these encryption algorithms based on OpenSSL[2].

IV. SUPPORTED ENCRYPTION SCHEMES

In this section, we describe three encryption schemes that BotTalker supports. These are packet level encryption, flow level encryption and SSL.

A. Packet Level Encryption

The basic mechanism of packet level encryption is shown in Figure 2. There are three steps in packet level encryption.

- 1) Extract the payload from each individual packet
- 2) Encrypt the payload with the user specified encryption algorithm
- 3) Place the encrypted payload back in the packet

This approach essentially leaves the header intact and changes only the payload. One challenge, however, is that the size of the payload, once encrypted, could be different than the original payload. For example, with a stream cipher (XOR) or CFB and OFB modes of block ciphers the size of the payload remains the same. But with block ciphers using ECB and CBC the size of encrypted data is larger than the original due to padding. If we simply put the encrypted payload back into the packet the size of packet will change, which means that the sequence and acknowledgment numbers for all subsequent packets need to change. Moreover, if the size of the encrypted packet is greater than the MTU an additional packet has to be generated, and the timestamps of all subsequent packets have to be adjusted. To avoid this complicated scenario we compromise by simply trimming the encrypted payload to the same size as the original packet. We acknowledge that the encrypted stream is now different than what a bot may have produced, but we believe that it is more important to preserve the other characteristics of the stream (timing, header information, etc.) than strive for a completely accurate representation of the encrypted bytes. Our results described later in the paper indicate that this is not a serious problem.

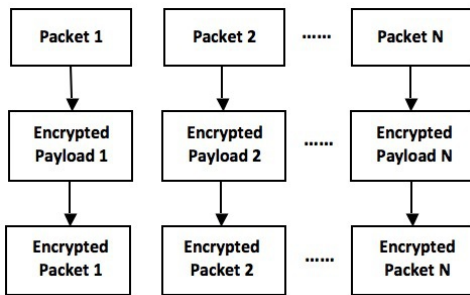


Fig. 2. Packet Level Encryption

B. Flow Level Encryption

Figure 3 depicts the process of flow level encryption. With flow encryption we assume that the endpoints exchange symmetric keys and encrypt each direction with a different key, but the same encryption algorithm. We also detect spurts of packets and encrypt them as a group so if the encrypted version is larger we don't alter packet timing. Flow level encryption is performed as follows:

- 1) Extract and concatenate the payload from groups of packets in each direction.
- 2) Encrypt each group or spurt using the user's specified encryption algorithm.
- 3) Place the encrypted payload back into the original packets.

With flow encryption we face the same problem as packet encryption scheme, namely the possibly larger size of encrypted data. We use the same solution: we trim the encrypted data to the same size as the original before we place it back into packets. In the flow case, however, we only trim the last packet in a spurt.

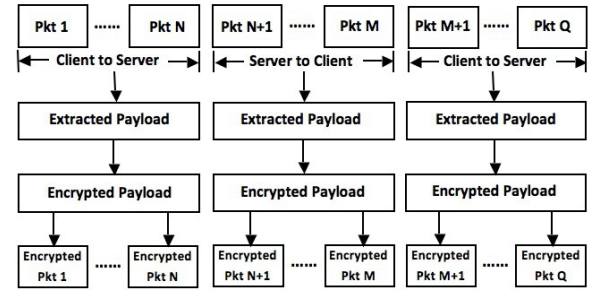


Fig. 3. Flow Level Encryption

C. SSL Emulation

The process of SSL emulation is shown in Figure 4. The figure indicates that the first step is similar with flow level encryption, which is to group packets in spurts on each direction, extract and concatenate the spurt's payload. Then we establish a SSL connection, transmit the extracted payloads in their original direction and capture the resulting packet trace. The IP addresses, port numbers and timestamps of the captured SSL traffic are of course different than the original. While it is easy to replace the IP addresses and ports, a big challenge is to adjust the timestamps in the SSL trace because the total number of packets is different. For example, the trace includes packets for setting up the SSL state.

We deal with this issue by calculating the *Client Reaction Time* (T_1), *Packet Inter-arrival Time on Server* (T_2), *Round Trip Time* (T_3) and *Packet Inter-arrival Time on Client* (T_4) from the original trace, as shown in Figure 5. Then we assign the timestamp of the first packet in the original trace to the first packet in the SSL trace. Finally, we use the above four values to estimate the timestamps for subsequent packets. For example, if the first packet is sent from client to server and the second is sent from server to client, then $Time(2nd\ pkt) = Time(1st\ pkt) + T_3$. We believe that this is a reasonably

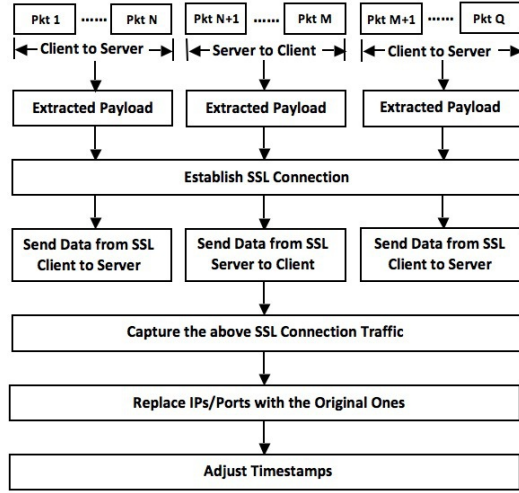


Fig. 4. SSL Emulation

accurate representation of a SSL connection, had the bots used one.

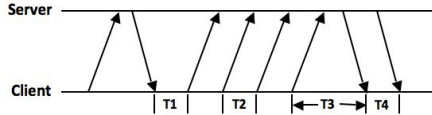


Fig. 5. SSL Timestamps Emulation

D. Resources

The current BotTalker distribution provides the following resources:

- 1) Botnet Traces: 140 Butterfly and Kraken botnet traces obtained from researchers at Georgia Tech [7], [13]. Each of these traces contains both Butterfly and Kraken traces, collected by running bot binaries in a sandbox.
- 2) Background Traces: Our collection of background traffic traces is limited due to privacy issues since these traces need to retain payload. We provide traces from a recently generated DARPA data set containing a variety of synthetic traffic. The data set was generated between the 3rd and the 12th of November, 2009. The traces are composed of 99.81% TCP, 0.162% UDP, 0.0230% ICMP and 0.0041% other traffic, the average bandwidth is 63.43 Mb/s. Note that BotTalker has the ability to use background traffic provided by the local network.

V. EVALUATING EXISTING DETECTION SYSTEMS

In this section we use BotTalker to generate encrypted botnet traffic and evaluate a widely used botnet detection system - BotHunter [8] and two IDS', Snort [3] and Suricata [4]. The results show that the encrypted botnet traffic foils bot detection in these systems.

A. Data Set

We generate 7 data sets by applying encryption and/or IP replacement based on *140BotSample*. Botnets can easily change C&C servers to evade blacklists, so we replace all IP

addresses (except DNS) to addresses that do not appear on blacklists. We name this dataset as *140BotReplaceIPs*. Then we generate three datasets to emulate botnets using encrypted communication. We encrypt all the traffic except SMTP and DNS by applying i) RC4 encryption with packet level encryption, ii) RC4 encryption with flow level encryption and iii) SSL emulation. We call these data sets *140BotPktLevel*, *140BotFlowLevel* and *140BotRC4SSL* respectively. Finally, we apply IP replacement on the above encrypted datasets to obtain new datasets, namely *140BotPktLevelReplaceIPs*, *140BotFlowLevelReplaceIPs* and *140BotSSLReplaceIPs*.

B. Evaluating BotHunter

Briefly, BotHunter is a botnet detection tool that relies on the fact that bot infection follows a distinct set of events in some loose order. BotHunter assigns different scores to these events and makes use of Snort to detect them. If the total score exceeds a threshold (0.8), then the host is labeled as bot. We reverse engineered BotHunter and calculated the scores for each event in an earlier work [19].

We ran BotHunter on the original bot trace *140BotSamples* and the seven generated data sets. Table II shows the results. From Table II we observe that 138 out of 140 bots are detected on the clear botnet traces. When BotHunter detects a bot it also reports the rules that fired. We categorize these rules into four groups: 1) DPI on HTTP traffic: these rules triggered 2 events with overall score of 1.0. 2) SMTP traffic: these rules triggered 1 event whose score is 0.5. 3) DNS traffic: these rules triggered 2 events with overall score of 1.3. 4) Blacklists: these rules triggered 1 event whose score is 0.1.

When we feed *140BotReplaceIPs* to BotHunter, 138 bots are detected as they trigger the events in groups 1 and 2 (some also trigger events in group 3). With encryption only, 127 bots are missed because they only trigger events in group 2 and 4 whose score is less than the threshold. The other 13 bots are detected because they trigger events in group 2, 3 and 4. These 13 bots are still detected when both encryption and IP replacement are applied as they trigger events in group 2 and 3. We note that even with three different encryption schemes, the results do not change. The reason is that all three encryption techniques foil DPI detection in group 1. In Section VI, we show how to utilize different encryption algorithms and schemes to evaluate IDS'. In conclusion, encrypted botnet traffic largely foils bot detection in BotHunter.

TABLE II
BOTHUNTER DETECTION RATE ON BOTNET TRACES

Traffic Encryption	IP Replacement	Detected Bots
Not Applied	Not Applied	138
Not Applied	Applied	138
Packet Level Encryption Applied	Not Applied	13
Packet Level Encryption Applied	Applied	13
Flow Level Encryption Applied	Not Applied	13
Flow Level Encryption Applied	Applied	13
SSL Emulation Applied	Not Applied	13
SSL Emulation Applied	Applied	13

C. Evaluating Snort and Suricata

In this section, we measure the effect of encrypted bot communication on two widely used IDS' - Snort (version 2.9.6.2) and Suricata (version 2.0.4), using the *Emerging Threats Open Ruleset* [1].

First we feed non-encrypted botnet traces *140BotSamples* to Snort. There are 14982 alerts generated. The number of alerts for each bot is shown in Figure 6. The x-axis is the bot index and the y-axis is the number of alerts. From the figure we can see that most of the alerts are related to SMTP, but they are triggered by the same rule. So we consider unique alerts in Figure 7. From Figure 7 we see that most of the unique alerts are DPI related. There are 28 unique alerts and they can be categorized into four groups: 1) SMTP: The alert is a result of *non mail servers sending a large number of emails*. 2) DNS: 9 unique alerts are due to i) DNS queries and replies to suspicious domain names, or ii) too many DNS NXDOMAIN responses. 3) Blacklist: The host talks to blacklisted IPs. 4) DPI: 17 alerts are triggered by looking for a suspicious string in the payload. 8 are clearly related to botnets, such as communicating with botnet controller, Butterfly bot download, etc. Another 4 are malware related, another 4 are related to executable file download, and the rest one is not suspicious.

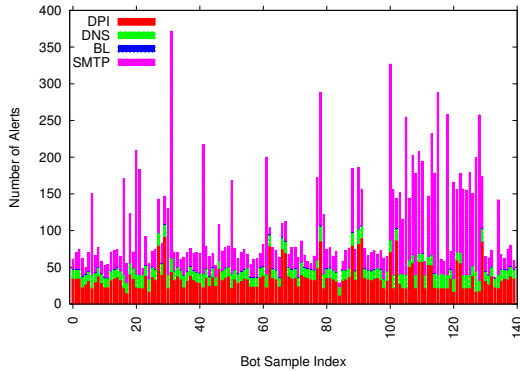


Fig. 6. Snort Alerts per Bot without Encryption

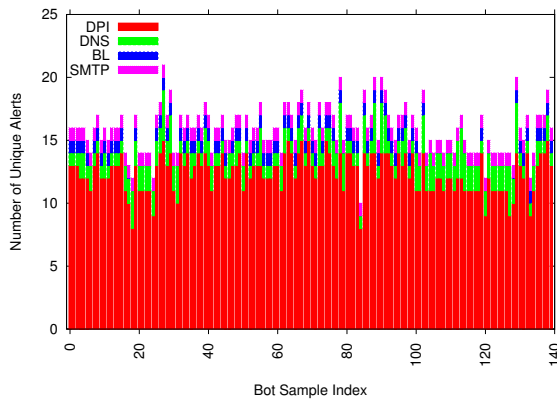


Fig. 7. Snort Unique Alerts per Bot without Encryption

We then feed the encrypted botnet traces *140BotPktLevel* to Snort. Only the alerts related to DNS, SMTP and Blacklist are triggered. 9926 alerts are generated, 34% fewer alerts than when encryption is not applied. Of the unique alerts only 11

remain, a 60.7% reduction. The number of alerts per bot and unique alerts per bot are the same as Figure 6 and Figure 7, except that the DPI alerts (red) are not visible.

We repeat these experiments with Suricata. 13482 alerts are triggered for *140BotSamples*, while 8638 (35.9% less) are triggered for *140BotPktLevel*. We show unique alerts for *140BotSamples* in Figure 8. There are 37 unique alerts, of which 21 rely on DPI, 9 rely on DNS, 3 rely on blacklists, 1 relies on SMTP and 3 are due to wrong stream handshake (we ignore them as they are not related to bots at all). 16 unique alerts are left (56.8% less) when the bot traffic is encrypted.

In conclusion, encrypted botnet communications suppresses around 35% of the total alerts and more than 55% unique alerts for Snort and Suricata for these particular traces.

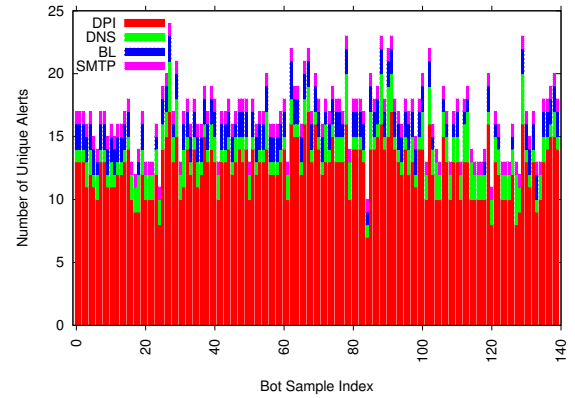


Fig. 8. Suricata Unique Alerts per Bot without Encryption

VI. DISCUSSION

A. Encryption Costs

Here we investigate the overhead introduced by encrypting data. Two chunks of data whose sizes are 1400B and 2MB are used to simulate the packet and flow that need to be encrypted. We encrypt the data 10,000 times and calculate the average overhead. The results indicate that most of the overhead for packet encryption is far lower than 1 millisecond, while for flow encryption the overhead is tens of milliseconds. There are some botnet detection algorithms that rely on timestamps [8], but don't rely to a granularity of milliseconds. Thus we conclude that the encryption costs of BotTalker are negligible.

B. Do We Need So Many Encryption Schemes?

PROVEX [12] is a botnet detection system that can detect encrypted C&C channels. It relies on the fact that many encrypted botnets use static keys, or their keys can be extracted from the transmitted data. With the help of sandboxes and reverse engineering, PROVEX is able to collect some botnet binaries and obtain a set of keys or key extraction functions from them. After that, PROVEX can detect encrypted C&C channels by i) extracting C&C message patterns from captured botnet binaries, ii) decrypting every packet using the set of keys or key extraction functions, and iii) matching the decrypted message with the extracted C&C patterns. However, PROVEX can only decrypt individual packets and doesn't provide stream reassembly mechanisms. As mentioned earlier,

when bots transfer a binary they encrypt the entire file first and then chop the encrypted data into packets. In this case, PROVEX may not work because some encryption algorithms require reassembling entire encrypted data to get full decrypted messages. However, if the botnet uses simple encryption algorithms (e.g. XOR), the first decrypted packet still has the correct original message and PROVEX can make use of it. This example shows the need to generate encrypted data sets using different encryption schemes with different encryption algorithms to test the detection systems' capabilities.

C. Do We Need So Many Encryption Algorithms?

Botzilla [11] can detect malware when the latter contact C&C servers. Botzilla assumes that in the communication with the servers, content may vary if they rely on host and network settings (e.g., OS, current time), but others don't change and can be used as signatures. Based on this assumption, Botzilla repeatedly runs the binaries in a controlled environment by changing network and host settings, and extracts the invariant content patterns from the traffic as signatures. Botzilla does not have any ability to decrypt encrypted messages and is not designed to detect encrypted communications. However, when the authors test it on Storm Worm, Botzilla is able to detect the communication even though it is encrypted. The reason is that Storm Worm uses a simple encryption algorithm (XOR) and does not conceal communication very well. If a strong encryption algorithm were used by the bots, Botzilla would most likely not detect it.

BotTalker's rich set of encryption algorithms allows us to evaluate PROVEX's performance. As PROVEX cannot distinguish which encryption algorithm and key is used in captured packets, it has to try all encryption algorithms and keys at its disposal. To test its performance we can generate a set of encrypted botnet traces using various encryption algorithms and keys, and then feed them to PROVEX.

VII. RELATED WORK

Previous work on generating malicious traffic includes [16], [15], [14]. In [16], the authors use alerts generated from a honeypot to label the datasets. Sommers introduces a method to generate various attacks in [15]. In [14], an attack description language is utilized to generate data sets for benchmarking IDS'. None of the above efforts, however, is able to generate botnet datasets. In [5], the authors introduce a system that can provide botnet traffic with ground truth by executing bot binaries in secure environments. However, it does not allow customization of encrypted botnet traffic. Our work is most closely related to SLINGbot [9], which is a tool that can generate user customized C&C traffic. There are several differences, however, between SLINGbot and BotTalker. First, SLINGbot doesn't emulate botnet encrypted communications although it supports certificates and password. Second, SLINGbot only generates C&C traffic. Besides the C&C traffic, the traces provided by BotTalker also include other traffic (e.g., propagation traffic), which can be used by detection systems, like BotHunter [8]. Rubot [10]

provides a framework that allows researchers to build bot code and then deploy the botnet onto a closed network. However, the bots' behavior and communication is emulated, which could be different from the real ones.

The most significant difference between BotTalker and all the above efforts is that they generate emulated traffic while BotTalker converts real C&C traffic to encrypted traffic. BotTalker preserves all the network level characteristics (timing information, packet size, etc.), which is important because many detection systems make decisions based on them.

VIII. CONCLUSIONS

Data sets with ground truth play an important role in developing detection systems. However, collecting such data sets is very hard. We address this issue by introducing BotTalker, a tool that can be used to generate customized encrypted botnet communication traffic. BotTalker works by emulating the actions a real bot would take to encrypt communication. In this paper we presented the design and implementation of BotTalker, and used it to evaluate the effect of encrypted botnet traffic on BotHunter, Snort and Suricata.

REFERENCES

- [1] Emerging threats. <http://http://www.emergingthreats.net>.
- [2] The open source toolkit for ssl/tls. <http://www.openssl.org>.
- [3] Snort. <http://www.snort.org>.
- [4] Suricata. <http://suricata-ids.org>.
- [5] P. Barford and M. Blodgett. Toward botnet mesocosms. In *Proc. 1st Work. on Hot Topics in Understanding Botnets*, 2007.
- [6] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. A view on current malware behaviors. In *Proc. 2nd USENIX Conf. on Large-scale Exploits and Emergent Threats*, 2009.
- [7] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proc. 15th ACM Conf. on Computer and Communications Security*, 2008.
- [8] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proc. 16th USENIX Security Symposium*, 2007.
- [9] A. W. Jackson, D. Lapsley, C. Jones, M. Zatkó, C. Golubitsky, and W. T. Strayer. Slingbot: A system for live investigation of next generation botnets. In *Proc. Cybersecurity Applications & Technology Conf. for Homeland Security*, 2009.
- [10] C. P. Lee. *Framework for Botnet Emulation and Analysis*. PhD thesis, Atlanta, GA, USA, 2009. AAI3364236.
- [11] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov. Botzilla: Detecting the "phoning home" of malicious software. In *Proc. ACM Symposium on Applied Computing*, 2010.
- [12] C. Rossow and C. J. Dietrich. Provex: Detecting botnets with encrypted command and control channels. In *Proc. 10th Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2013.
- [13] P. Royal. Analysis of the kraken botnet, 2008.
- [14] A. Shiravi, H. Shiravi, M. Tavallae, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 2012.
- [15] J. Sommers, V. Yegneswaran, and P. Barford. A framework for malicious workload generation. In *Proc. 4th Conf. on Internet Measurement*, 2004.
- [16] A. Sperotto, R. Sadre, F. Vliet, and A. Pras. A labeled data set for flow-based intrusion detection. In *Proc. 9th IEEE Int. Work. on IP Operations and Management*, 2009.
- [17] E. Stinson and J. C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *Proc. 2nd USENIX Work. on Offensive Technologies*, 2008.
- [18] P. Wang, S. Sparks, and C. C. Zou. An advanced hybrid peer-to-peer botnet. In *Proc. 1st Work. on Hot Topics in Understanding Botnets*, 2007.
- [19] H. Zhang, C. Papadopoulos, and D. Massey. Detecting encrypted botnet traffic. In *Proc. 16th IEEE Global Internet Symposium*, 2013.