PROVEX: Detecting Botnets with Encrypted Command and Control Channels

Christian Rossow †‡⋆ and Christian J. Dietrich †¶

†University of Applied Sciences Gelsenkirchen, Institute for Internet Security, Germany
‡VU University Amsterdam, The Network Institute, The Netherlands
¶Department of Computer Science, Friedrich-Alexander University, Erlangen, Germany
{rossow,dietrich}@internet-sicherheit.de

Abstract. Botmasters increasingly encrypt command-and-control (C&C) communication to evade existing intrusion detection systems. Our detailed C&C traffic analysis shows that at least ten prevalent malware families avoid well-known C&C carrier protocols, such as IRC and HTTP. Six of these families – e.g., Zeus P2P, Pramro, Virut, and Sality – do not exhibit any characteristic n-gram that could serve as payload-based signature in an IDS.
Given knowledge of the C&C encryption algorithms, we detect these evasive C&C protocols by decrypting any packet captured on the network. In order to test if the decryption results in messages that stem from malware, we propose PROVEX, a system that automatically derives *probabilistic vectorized signatures*. PROVEX learns characteristic values for fields in the C&C protocol by evaluating byte probabilities in C&C input traces used for training. This way, we identify the syntax of C&C messages without the need to manually specify C&C protocol semantics, purely based on network traffic. Our evaluation shows that PROVEX can detect all studied malware families, most of which are not detectable with traditional means. Despite its naive approach to decrypt all traffic, we show that PROVEX scales up to multiple Gbit/s line speed networks.

Keywords: Botnet Detection, Command & Control, IDS, Protocol Syntax

1 Introduction

Botnets have emerged as one of the most prevalent dangers to Internet users. Nowadays, most modern botnets employ encrypted C&C protocols to evade intrusion detection systems. As a result, the detection of botnet C&C flows by help of payload signatures becomes much more difficult if not impossible.

However, despite using encryption, some botnets still exhibit characteristic payload strings, often caused by a static key in combination with recurring C&C protocol keywords. In addition, for botnets using HTTP as carrier protocol – again, even in presence of encrypted C&C messages – HTTP characteristics can serve as recognition property in order to detect a C&C flow. For example, a characteristic sequence of HTTP URI parameters of a botnet's C&C protocol may be specific enough to recognize a corresponding C&C flow. For example, Perdisci has shown that by clustering HTTP traffic, behavioral features emerge which can be used to recognize HTTP-based botnets [16].

⋆ We thank our malware sample providers. This work was supported by the Federal Ministry of Education and Research of Germany (Grant 16BY1110, MoBE).

However, in contrast to these detectable botnets, we reveal that many prevalent botnets avoid HTTP and instead design their C&C protocol with TCP or UDP as carrier protocol. In addition, we discover botnet families that do not exhibit characteristic payload bytes and thus, cannot be detected using payload signatures. In these cases, to the best of our knowledge, no methodology exists to detect these kinds of C&C channels.

Therefore, in this paper, we address the problem of detecting C&C flows of botnets that neither exhibit characteristic payload strings nor fall for carrier-protocol-specific signatures. First, we use n-gram analysis to find those botnet families that do not exhibit characteristic payload strings in their encrypted network traffic. Second, we reverse engineer and re-implement their decryption routines, leveraging the fact that all of these botnets use symmetric encryption. Third, based on the plaintext C&C, we propose a methodology to automatically infer a probabilistic model of a botnet's C&C protocol syntax. Subsequently, we apply this model in our implementation ProVeX to detect C&C flows on arbitrary network traffic, scaling up to multi-gigabit network links.

To summarize, our contributions are:

- We identify six botnets which do not exhibit characteristic payload strings.
- We design a system to detect C&C traffic of botnets which employ encryption in their C&C protocols. We propose a methodology to automatically infer a probabilistic model based on key characteristics of the C&C protocol syntax.
- We implement and evaluate a recognition module that uses the previously inferred probabilistic vectorized signatures to detect C&C flows in arbitrary network traffic. Our implementation performs well and scales up to multi-gigabit network links.

2 Limitations of Payload Signatures

This section discusses why existing payload-based approaches fall short on detecting many types of today's C&C traffic. We first discuss the inherent limitations of payload signatures and then explore the current malware landscape for families that circumvent existing payload-based inspections.

2.1 Invariants in Network Traffic

Existing payload-based approaches can only detect malware if there are invariants in the network communication. Plaintext C&C protocols typically exhibit these invariants, for example, by C&C protocol keywords that can be used as part of payload signatures. Such invariants even appear in a few encrypted C&C protocols, especially those using encryption algorithms where each bot encrypts the C&C message with the same key. In this case, common plaintext parts – such as protocol keywords – result in common ciphertext parts, which can be leveraged by signatures on the encrypted C&C traffic [17]. However, more and more botnet C&C families do not use static keys and thus do not exhibit these characteristics. In other words, the ciphertext varies even for identical plaintext messages. For example, a bot can compute encryption keys dynamically, such as by deriving the key from the current date. Similarly, some malware families prepend random bytes as initialization vector to their C&C messages. If these messages are encrypted using cipher-block chaining or cipher feedback mode, the resulting ciphertexts do not exhibit invariants. These C&C protocols thus circumvent payload signatures.

Still, some C&C messages can be detected by properties of the carrier protocol. For instance, for HTTP-based C&C messages, the combination of request parameters may serve as signature [16]. Similarly, malware C&C with DNS as carrier protocol might be detected by identifying anomalies, such as high entropy in TXT resource records [5]. We define the *carrier protocol* as the underlying protocol of the malware-specific C&C protocol. In contrast to HTTP and DNS, both TCP and UDP offer hardly any possibility to define signatures based on the carrier protocol.

In this section, we shed light onto such malware families, that is, malware that effectively bypasses detection approaches on both the C&C protocol and carrier protocol layer. We inspect a dataset of C&C protocols of 28 prevalent malware families that we identified in our malware analysis environment SANDNET using a wide range of classification mechanisms. We identified C&C communication by help of our botnet tracking means, by matching C&C end points against our carefully assembled IP address and domain blacklists, by matching against payload signatures, by our traffic-analysis-based message length sequence approach [6], as well as communication periodicity analysis. In addition, we manually assigned malware family labels to the C&C communication flows and made spot checks to verify the correctness of labeled data. As motivated above, we focus on malware using non-descriptive carrier protocols and thus filter our dataset on ten prevalent families that use TCP or UDP as carrier protocols.

For each of these malware families, we then attempt to identify payload signatures by computing the n-grams among the C&C messages of all analyzed malware samples of one family. As shorter payload signatures typically cause false positives (see Section 4.1), we chose to require payload signatures of at least four consecutive bytes, i.e., $n = 4$. Then, we count the number of malware samples exhibiting a specific four-gram in all of its communication flows. We consider those C&C protocols as not reliably detectable by traditional payload signatures which do not exhibit at least one four-gram in the majority of malware samples.

Family	**Carrier**	**4-Gram**	**Encryption**	**Key Material**
Cutwail	TCP	✓	vXOR	Hardcoded
Fynloski	TCP	✓	RC4	Hardcoded (differs per sub-botnet)
Palevo	UDP	✗	cXOR	Derived from message length
Pramro	TCP	✗	RC4	Derived from message length
Ramnit	TCP	✓	RC4	Hardcoded
Sality	UDP	✗	RC4	Derived from message header
ZeroAccess	UDP	✓	rXOR	Hardcoded
Tofsee	TCP	✗	cXOR	Hardcoded
Virut	TCP	✗	vXOR	Known plaintext attack
Zeus P2P	UDP	✗	cXOR	Random byte

Table 1. N-gram analysis results for all malware families that used UDP/TCP as carrier protocol. vXOR = various custom XOR encr., cXOR = chained XOR encr., rXOR = rotating XOR encr.

Table 1 shows that only a minority of the analyzed malware families have C&C protocols with at least one invariant four-gram. The key material used in these C&C protocols is either static or does not vary much (e.g., because the message length serves as key and messages are of equal length). In these cases, applying the encryption algorithms on constant plaintext results in ciphertexts with common patterns. However, the derived four-grams are not necessarily sufficient for payload-based detection, as they may represent typical strings or keywords that can also be found in legitimate traffic. Consequently, the existence of four-grams does not prove that malware families can still be detected using traditional signature approaches. On the contrary, Table 1 shows that the majority of malware families *cannot* be detected by help of payload signatures. These six malware families effectively circumvent such detection, as none of them shows any invariant four-gram.

2.2 Encryption Case Studies

Table 1 also shows the encryption schemes that the malware families use to evade payload-based detection. Interestingly, most malware families effectively bypass payload signatures with custom XOR-based algorithms, which sometimes even only obfuscate the payload rather than using key-based encryption[1].

Consider, for example, the XOR-based encryption algorithm used by Zeus P2P (Listing 1.1). Using the first random byte as a key, all subsequent bytes are XORed with the preceding ciphertext byte. As in Zeus P2P messages typically also at least the third byte (which specifies the message padding length) must be considered random, two otherwise identical Zeus messages have up to 2^{16} ciphertext representations.

```
void zeus_encrypt(char *plain, char *cipher, int len) {
   plain[0] = random();  // first byte in plaintext is random
   cipher[0] = plain[0]; // use random byte as init. vector (IV)
   for(int i = 1; i < len; i++) {
       cipher[i] = plain[i] ^ cipher[i-1];
   }
}
```

Listing 1.1. Zeus P2P C&C encryption

Palevo uses an XOR-based encryption by deriving the initial key from the byte length of the C&C message (Listing 1.2). The payload length is additionally stored in an obfuscated way in each C&C message. Given messages of varying length, this way the ciphertext also shows no invariants, similar to Zeus P2P.

```
void encrypt_palevo(char *plain, char *cipher, int len) {
   /* derive initial key from C&C message length */
   char nextKey = (((len >> 8) + len) | 2) & 0xAA;
   for (uint32_t i = 0; i < len; ++i ) {
      cipher[i] = plain[i] ^ nextKey;
      nextKey = ~(cipher[i] << (i & 3));
   }
}
```

Listing 1.2. Palevo C&C encryption

Other malware families use well-known encryption algorithms such as RC4, but vary the key material. For example, Sality uses a CRC16 checksum and the C&C message length as key for en-/decrypting the C&C messages. Virut uses a known-plaintext attack

[1] For simplicity, we will still refer to these obfuscation algorithms as encryption algorithms.

to derive its key material from the first four bytes of an encrypted message, and then uses this key in a custom XOR-based encryption algorithm.

Albeit these encryption routines are quite simple, they avoid any invariances in the encrypted payload and thus effectively circumvent existing signature-based detection approaches. This is problematic, as most of these malware families are well-known to cause severe harm to millions of infected users. As such, one could assume that the evasive nature of their C&C protocols gave rise to their "success" in the malware business. With no reliable network-based detection method left, network administrators cannot alert the infected users of relevant malware families such as Pramro, Sality, Virut or Zeus. In the following section, we will therefore propose PROVEX, a novel payload-based detection method that can detect such types of C&C encryption using *probabilistic vectorized signatures*.

3 PROVEX: Detecting Encrypted C&C

3.1 C&C Detection by Payload Decryption

While we consider it impossible to detect the C&C messages of the aforementioned malware families in *encrypted* form, we hypothesize that their *decrypted* communication can be recognized. Luckily, nowadays most malware families deploy encryption routines with hard-coded or predictable key material (if any). Knowledge of the de-/encryption routines and the key derivation or the key material enables us to decrypt passively-acquired network traces. This naive approach requires to decrypt each captured frame — including arbitrary frames that do not belong to malicious C&C communication. Unfortunately, the decryption routines transform any arbitrary input and cannot verify per se if the decrypted data is a valid C&C message. In other words, for each decrypted frame, we need to examine if the decryption routine results in a reasonable plaintext C&C message. While this might sound trivial at first, we find that hardly any malware C&C protocol exhibits sufficiently characteristic and identifiable plaintext strings that can be used to verify the decryption result. Instead, many of the analyzed C&C protocols do not have invariant payload sequences even in the plaintext C&C messages, or the invariant sequences are too short to be distinctive. We propose the use of a *probabilistic* and *vectorized* signature. When used in combination, these two schemes circumvent the aforementioned shortcomings of traditional payload signatures.

We observed that binary C&C protocols often follow a well-defined syntax. That is, C&C protocols are similar to many legitimate network protocols in that they typically define message headers and payloads with *positional fields* or *tagged fields*. In a protocol with positional fields, the semantics of a certain field is given by its fixed offset in the message. For example, the IP address fields in the IP header are positional. In contrast, protocols such as HTTP use *tagged fields* which require a tag (e.g., a characteristic string) to specify the semantics of a field, such as a "User-Agent: Mozilla" tag value pair. We have observed both positional and tagged fields in the C&C protocols under evaluation. Bots such as Virut or Fynloski use very specific — and thus easily recognizable — tags to describe the exchanged data. However, the other C&C protocols use only positional fields. Such positional fields cannot be captured by traditional payload string signatures, since they lack characteristic tags. Therefore, we see the need for a new pattern-based signatures that can grasp the fact that even single byte values at certain offsets are characteristic for a C&C protocol. For example, C&C protocols may include the packet length in a two-byte-wide header field at a fixed offset, which we can use to verify packets. Similarly, bots accept multiple types of commands and include type identifiers at a fixed offset in the message header. In contrast to payload string signatures, we propose *vectorized* signatures that match on a combination of multiple one byte fields.

However, most of the C&C messages, or more precisely, most of the message fields' contents are invariant, which often renders it impossible to define an exact payload signature. Instead, the field's contents exhibits a sub-range of characteristic values. For example, a one-byte-wide field may be designed to denote the message type, which typically does not exhaust all 256 possibilities, because C&C protocols often exhibit significantly fewer message types. Similarly, boolean flags (0 or 1) exchanged in a one-byte-wide field are not invariant, but are characteristic in that they exhaust only two byte values. These are only two examples of many C&C protocol idiosyncrasies that we will leverage with *probabilistic* signatures. Such probabilistic signatures cover the likelihood of all possible byte values for a fixed offset in a C&C message. This new paradigm allows us to match all possible valid messages of a C&C protocol, instead of creating a signature over a (usually variable) message content.

3.2 Automatic Syntax Modelling

In this section, we propose PROVEX, a system to automatically learn and match the syntax of C&C protocol messages using *probabilistic vectorized signatures*. Our system PROVEX is based on the assumption that all messages of a certain C&C protocol and message type follow the same syntax. Thus, we will use the modeled syntax to verify if the decryption of an arbitrary network packet results in valid C&C communication. In particular, for each message type, PROVEX models the probability of a byte x occurring at a specific byte offset.

PROVEX takes as input (1) a list of C&C messages that were recorded per malware family, (2) a decryption function (and the key material or key derivation function, if any) that can be used to decrypt the messages, and (3) the position of the bytes indicating the message type (if any). The network traces can be recorded using multiple sensors (e.g., dynamic malware analysis environments like SANDNET [19], or real infection traces). Manual reverse engineering serves as a tool to provide the other two inputs. While systems to semi-automatically extract protocol semantics using dynamic taint analysis have been proposed, we explicitly restrict ourselves to a purely network-based learning approach to ease the reproducibility and increase the flexibility of our tool.

Figure 1 shows the training phase of PROVEX. In the first training step, given encrypted C&C messages from a malware family, we decrypt all C&C messages and group them according to their message type. PROVEX then identifies characteristic bytes in the plaintext C&C messages by calculating the distribution of byte values. In the last training step, PROVEX derives probabilistic vectorized signatures that can be used to verify if decrypted network packets stem from a certain malware family's C&C.

We define a *probabilistic vectorized signature* as follows:

$$psig = \langle (o_1, b_1, p_1), ..., (o_n, b_n, p_n), \quad (o_t, b_t), \quad (o_{pl}, l_{pl}, e_{pl}) \rangle,$$

whereas o defines the byte offset at which byte b occurs with a probability of $0 \leq p \leq 1$. The signature contains n of such byte probability 3-tuples, and the higher n, the higher

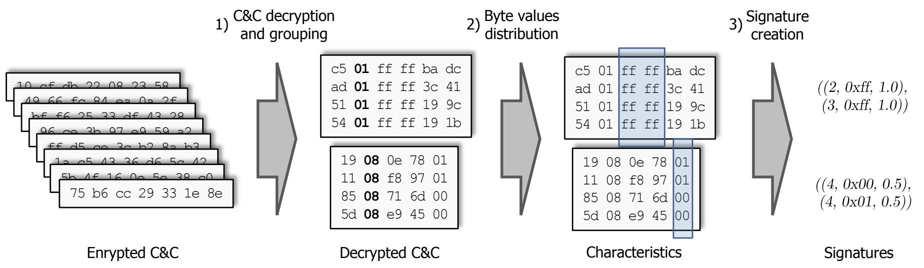
1) C&C decryption and grouping
2) Byte values distribution
3) Signature creation
((2, 0xff, 1.0), (3, 0xff, 1.0))
((4, 0x00, 0.5), (4, 0x01, 0.5))
Enrypted C&C
Decrypted C&C
Characteristics
Signatures

Fig. 1. Training phase of ProVeX.

is the accuracy of a signature. Static payload signatures can also be expressed with the probabilistic signatures using $p = 1$. The additional tuple (o_t, b_t) expresses for which message type b_t at offset o_t the signature was created. While we found that specifying message types on a single byte works well for current malware, this scheme can easily be generalized to message types of any length. If a C&C protocol does not use message types, we leave out the (o_t, b_t) tuple. The optional tuple (o_{pl}, l_{pl}, e_{pl}) expresses if a C&C protocol exhibits a C&C message length field, as explained later.

We developed ProVeX to automate the generation process of all probabilistic signatures per malware family. With the training dataset provided to ProVeX, we first decrypt all input traces. In order to generate a signature, we search for all C&C messages W of a certain malware family and message type. For each offset o, we count the number of occurrences c of each byte $0 \leq b < 256$ in all $w \in W$. We then compute the probability p that a byte value b occurs at offset $0 \leq o < len(w)$ in relation to all byte values at this offset, i.e., $p = c/|W|$, where $|W|$ is the number of messages with at least length $o + 1$. If a particular byte value b at offset o occurs significantly more often, i.e., if $p \geq T$, we include the tuple (o, b, p) in $psig_{m.t}$. A smaller threshold T allows to create more precise probabilistic signatures, whereas a larger threshold T minimizes the number of tuples included in the signatures. In our measurements, we set $T = 0.3$, i.e., we included a byte if it was present in at least 30% of the cases at a particular offset. Similarly, we excluded byte offsets o that did not represent a significant sample size to compute probabilities, that means, we required that at least 30% of the messages $w \in W$ had at least a length $o + 1$ before computing probabilities at offset o. In addition, we do not create 3-tuples for the message type offset o_t in order to avoid double-counting this particular message position. For efficiency reasons, we limit the number of 3-tuples included in a signature to the $g = 10$ tuples with the lowest offsets and discard all others.

For example, Figure 2 shows ten decrypted Zeus P2P messages of message type 0x05 randomly drawn from Sandnet. These messages represent UDP-based chunk download requests which belong to Zeus' update mechanism. The first 44 bytes (offsets 0 - 43) represent the Zeus header, in which only the message type field (offset 3) is invariant. The first payload byte (offset 44) indicates the download type (0x01 is a Zeus configuration file, 0x02 is a Zeus executable). Payload bytes 2-3 represent the chunk number, payload bytes 4-5 the chunk size, and the other bytes are random padding bytes. For byte offset $o = 44$, ProVeX learns two 3-tuples, as the byte value $b = 0x01$ and $b = 0x02$ both have an equal probability of $p = 0.5$. At offset $o = 45$, the least significant byte of the chunk number, the bytes are random and no 3-tuple is added to the signature. However, for the most significant byte of the chunk number, the 3-tuple

offset	0	1	2	3	...	44	45	46	47	48	49	50	51	52	53	54	55
packet A	6a	07	ad	05	...	02	16	00	50	05	68	a9	d7	3a			
packet B	15	a9	29	05	...	01	66	00	50	05	9b	c0	9c	a6	16	07	
packet C	8d	26	87	05	...	01	62	00	50	05	0c	45	c3	8e	47	35	ef
packet D	f9	fe	01	05	...	02	38	00	50	05	48	0d	3c	7d	11		
packet E	63	e1	d2	05	...	02	2e	00	50	05	3e	5c					
packet F	96	ab	c3	05	...	02	bc	00	50	05	c2	b7	65	5f	b9	22	9f
packet G	47	80	2f	05	...	01	0a	00	50	05	ad	98	07	60	51	78	83
packet H	fa	91	52	05	...	01	28	00	50	05	fc	81	78	76	4e	62	
packet I	a9	35	23	05	...	02	5f	00	50	05	c9	62	81	70	ad	1c	cc
packet J	07	f4	a9	05	...	01	0f	00	50	05	70	29	92	90	08		

Fig. 2. Randomly chosen Zeus P2P C&C messages of type 0x05.

$(o{=}46, b{=}0, p{=}1.0)$ is added, as Zeus downloads rarely exceed 256 chunks. Similarly, the bytes at offsets 47-48 are static, as the chunk length is always 0x550 (transferred in little-endian) = 1360 bytes. All the other offsets contain random bytes and are ignored during signature generation. To complete our signature, we have to consider that Zeus does not include a payload length field in the header. The created signature is:

$$psig = ((44, 1, 0.5), (44, 2, 0.5), (46, 0, 1.0), (47, 0x50, 1.0), (48, 5, 1.0), (3, 5))$$

This probabilistic vectorized signature covers both download types used by Zeus, and automatically identified single invariant bytes in the messages. In order to speed up the signature matching (as discussed later), we add a simple heuristic that avoids the decryption of irrelevant packets and thus reduces the computational cost. In particular, during our reverse engineering efforts, we stumbled upon C&C protocols which have their C&C payload length encoded in the C&C message. As part of the training phase, we thus analyze whether a C&C protocol fulfills this heuristic and if so, we include the C&C message's payload length bytes in our signatures. We store the offset of the payload length header o_{pl}, its field length in bytes l_{pl} and its endianness e_{pl}. Our heuristic correlates the binary value of all 1-grams, 2-grams and 4-grams in the encrypted packets in the training dataset with the packet lengths, and extracts the n-gram with the highest correlation score as payload length field (if the correlation is above a certain threshold). As bots may specify lengths by subtracting the C&C header lengths, we adjust the payload length computation if there is a consistent difference between the length field and the actual length. Note that this heuristic is solely added to increase performance and the general procedure works well even when skipping this heuristic. In essence, by verifying the lengths of encrypted packets, we can discard all invalid messages before the (computationally expensive) decryption.

Note that we do not need to infer any further semantics from the training data. Although this leaves us with no insights into the semantics of protocol fields or even protocol field boundaries, we will show that ProVeX performs well without such descriptive information. This level of abstraction significantly eases and speeds up the signature generation process, as only little of the C&C semantics need to be understood.

3.3 Probabilistic Signature Matching

We created the probabilistic signatures with the motivation to verify if a decrypted network packet belongs to a malware family. After capturing a frame, we first apply all payload-length-encoding heuristics and then dispatch a copy of the frame per decryption context and decrypt the copy with each of the valid decryption contexts. The decryption context consists of the decryption algorithm and (optionally) key material. A decryption context is valid, if the captured frame f_e matches the payload length specification in a decryption context, or if the decryption context does not include payload length specifications at all. Each decrypted frame f_d is then matched against all probabilistic signatures of the valid decryption contexts. A score s indicates the accordance of the frame with each probabilistic signature.

We compute the score s as follows. Given a frame f_d of length l, we search for the signature of the valid message type for f_d (if any), i.e., we ignore signatures for which $f_d[o_t] \neq b_t$. $f_d[0]$ follows the C notation of a byte array and refers to the first byte in the byte array f_d, $f_d[1]$ to the second byte, and so on. We then compute a score, per signature, initialized to $s = 0$ by summing up the byte probabilities that match the signature. That means, for all 3-tuples $(o, b, p) \in psig$, we add p to s iff $f_d[o] = b$. We ignore all tuples whose byte offset exceed the length of the current frame, i.e., all for which $o \geq l$. Intuitively, the more similar a frame is compared to the messages used during the training process, the higher is s. Low scores, on the other hand, indicate a low probability that the decrypted frame is a valid message for a given probabilistic signature. Note that these computations can be implemented in an extremely efficient manner, and the overhead of the few byte-wise comparisons per signature is negligible.

We use the score to classify if a packet can be considered known C&C communication. Instead of using only a fixed threshold, we also relate the achieved score to the maximum score that could have been achieved by the "perfect" packet. A relative threshold, in contrast to an absolute threshold, does not increase the probability that a signature matches if it contains many 3-tuples. Think of a signature which specifies a probability of $p = 1.0$ for $b = 0$ (i.e., a null byte) at all possible offsets. Assume that we specified a threshold that would raise alerts if at least four bytes matched the signature. For a random payload of 1400 bytes length with bytes drawn at random, the likelihood[2] that a score reaches an absolute threshold of $S = 4$ is about 80%. Thus, next to an absolute threshold S, we also require a high relative score to cope with signatures that are more likely to be matched at random (i.e., signature with many 3-tuples).

We compute the relative score r as $r = \frac{s}{maxscore}$. We compute the maximum score as $maxscore = \sum_{i=1}^{n} max(\{p_i \forall (o, b, p) \in psig, o = i\}) + l_{pl}$, i.e., we sum the maximum probability that can be achieved at each offset. The score is increased with the number of bytes specifying the payload length (if any, else $l_{pl} = 0$). Only if the relative score is at least as high as the relative threshold R, i.e., $r \geq R$, and the absolute score is at least as high as the absolute threshold, i.e., $s \geq S$, then the signature matches f_d. Depending on the specific context where ProVeX is used, the alert thresholds can be configured more conservatively or more aggressively. ProVeX allows to configure these parameters during runtime, and it is not necessary to relearn the probabilistic signatures. During our evaluation, we used the thresholds $S \geq 4.0$ and $R \geq 75\%$.

For example, assume that the packet in Figure 3 is matched against the signature we derived earlier from Figure 2. For every 3-tuple in the signature, we compare if the

[2] The likelihood of drawing exactly S null bytes in n random bytes can be calculated as: $p = (\frac{1}{256})^S * (\frac{255}{256})^{n-S} * \binom{n}{S}$. In a random frame with a typical length of $n = 1400$ bytes, the likelihood of matching exactly $S = 4$ predefined bytes is 15.7%. The likelihood of drawing *at least* four bytes is the sum of all probabilities p with $4 \leq S \leq 1400$, which is 79.5%.

offset	0	1	2	3	...	44	45	46	47	48	49	50	51	52	53	54	55
packet:	9a	1f	4c	$\underline{05}$	...	$\underline{02}$	91	$\underline{01}$	$\underline{55}$	$\underline{05}$	77	9a	cd	a2	fc		

Fig. 3. A random packet to be matched against a probabilistic vector signature for Zeus P2P.

message's byte at a given offset corresponds to the byte specified in the 3-tuple. All offsets that are covered by the signature are underlined in Figure 3. Overall, we find that the two 3-tuples $(44, 0x02, 0.5)$ and $(48, 0x05, 1.0)$ match, and compute a score $s = 1.5$. The maximum score is computed as $maxscore = (0.5, 1.0, 1.0, 1.0) + 0.0 = 3.5$, and hence, the relative score is $r = 43\%$. As a result, both, because the absolute (S) and the relative (R) thresholds are not reached, f_d does not match the signature and is discarded.

4 Evaluation

In our evaluation, we first measure the number of false positives and false negatives that PROVEX generates. Then, we interpret the results in a detailed qualitative analysis. Lastly, we evaluate the performance of PROVEX and show that it scales when applied to multiple Gbit/s network links.

4.1 Quantitative Evaluation

True Positive Evaluation We divide our True Positive (TP) evaluation in two parts. First, we evaluate our method with k-fold cross validation. Second, we analyze if our results generalize for settings other than the training environment.

For the k-fold cross validation, we assemble C&C streams of 50 different malware executions in SANDNET per malware family. We divide these 50 network traces into five disjoint folds consisting ten traces each. We then use every fold as training input for PROVEX and verify if the remaining 40 traces are correctly captured by the automatically derived probabilistic signatures. Note that we chose to use only a minority of the data for training in order to test if PROVEX also works for small numbers of input traces. Table 2 summarizes our TP evaluation results. The second column (*# sigs*) denotes the median number of probabilistic signatures that were derived by PROVEX. Per definition, the number of signatures is limited to the number of message types per malware family, but can be less if no clear patterns were found for certain message types. The third column (*CV TPR*) shows the average True Positive Rate that we measured during our 5-fold cross validation. Most malware families could be detected in all cases, and only three families missed 13–22% of the infections (cf. Section 4.2 for details).

A drawback of the cross validation is that all network traces stem from a single dynamic analysis environment. Consequently, the traces and thus also the derived signatures may include artifacts [20], such as IP addresses, user names, or Windows serial numbers, to name but a few. To verify if the signatures still capture infections in a completely different environment, we first trained signatures by using the 50 network traces from SANDNET that we used during the cross validation. We then executed three different samples per malware family in a second dynamic execution setup, which varies the aforementioned artifacts. In particular, we modified the external IP address, used a direct Internet connection instead of NATing, we changed the OS version, Windows

Family	# sigs	CV TPR	X-Env TPs
Cutwail	1	100%	3/3
Fynloski	11	78%	3/3
Palevo	5	87%	3/3
Pramro	1	81.5%	0/3
Ramnit	5	97%	3/3
Sality	2	100%	3/3
Tofsee	1	100%	3/3
Virut	1	100%	3/3
ZeroAccess	3	100%	3/3
Zeus P2P	5	100%	3/3

Table 2. True Positive evaluation results

user name, Windows serial and system language, we refrained from using virtualization, and we executed the three malware binaries at least one month later than the ones used during training phase. We use these secondary network traces to test if the signatures derived from SANDNET can detect malware infections in completely different network settings than the training environment.

The fourth column (*X-Env TPs*) in Table 2 shows that PROVEX detected these cross environment generated traces in all cases. For Pramro, PROVEX trained an artifact and produced a signature that did not generalize, but we were able to manually fix the signature (see Section 4.2). The results show that, even when using network traces from a uniform source, the created signatures can still capture communication for the vast majority of malware families. This shows that PROVEX usually only requires a single source of training input to generate meaningful signatures. However, in a few cases such as Pramro, it helps to train on malware traces from different environments. For example, Rieck et al. have shown that configuration artifacts can be mitigated by using multiple environments to generate training traces [17]. In Section 4.2, we will explain in detail why PROVEX can even be trained on traces despite artifacts in the datasets.

False Positive Evaluation We evaluate false positives (FPs) in a threefold approach. First, we show the statistical probability that the automatically-derived signatures match random payloads. Second, we deploy PROVEX in a university lab with supposedly legitimate network traffic. Third, we fuzz PROVEX with randomly generated payloads.

Signature Probability Evaluation: Legitimate network traffic with binary content is typically compressed, such as video/audio streams or other multimedia data, archived data or encrypted communication. Such binary streams have a high Shannon entropy and follow a normal distribution of byte values, as shown by Olivain et al. [15]. We leverage this observation to compute the statistical probability that a random packet triggers a signatures match. The probability that a random packet triggers a traditional n-gram payload signature is $(\frac{1}{256})^n$. For PROVEX, the probability that a random packet matches a signature depends on the score thresholds R and S. While a traditional signature thus only triggers, if exactly n bytes match, PROVEX can be tuned to match earlier, i.e., if a subset of the 3-tuples of the probabilistic signature match. In the next paragraph, we will derive the probability of random packet matches in PROVEX.

We first compute all possible (unordered) combinations $C_1 ... C_n$ of the probability 3-tuples in the signature, where $C_x \subseteq \{(o_1, b_1, p_1), ..., (o_n, b_n, p_n)\}$. Note that a combination can contain fewer 3-tuples than the original signature and that we did not include two 3-tuples with equal offset o in a combination. In order to find out which of these combinations can trigger the signature, we compute the score for each combination, i.e., $s_{C_x} = \sum_{i=1}^{h} max(\{p_i \forall (o, b, p) \in C_x, o = i\}) + l_{pl}$. We then ignore all combinations that do not score sufficiently high, i.e., we only consider combinations for which the score is greater than or equal to the absolute score threshold ($s_{C_x} \geq S$) and the relative score threshold ($s_{C_x} \geq maxscore_{psig} * r$). We denote the resulting set of V valid combinations as C_v, removing all combinations that merely represent supersets of other combinations, i.e., $\nexists\, C_1, C_2 \in C_v : C_1 \subset C_2$. Thus, C_v is the disjoint set of all possible payload combinations that would trigger the probabilistic signature.

With C_v, we can thus compute the probability that a signature matches. The probability that a combination $C_x \in C_v$ is triggered depends on the number of 3-tuples and the number of bytes specifying the payload length l_{pl} (if any), i.e., the total number of byte offsets covered by the tuple. As each byte value has a likelihood of $1/256$, the probability that a random payload matches C_x is $P_{C_x} = (\frac{1}{256})^{|C|+tlen+l_{pl}}$, where $|C|$ expresses the number of 3-tuples in C, $tlen$ is the length of the message type (usually 1 byte, or 0 if none), and l_{pl} denotes the number of bytes specifying the payload length (if any, else $l_{pl} = 0$). For example, a random payload matches a combination of four 3-tuples with a probability of $P = 2^{-40}$ (if no payload length bytes are included in the packet). The probability that a signature $psig$ is matched by a random packet can now be computed as the sum of the probabilities of all valid combinations, i.e., $P_{psig} = \sum_{i=1}^{V} P_{C_i},\ C_i \in C_v$. The probability P_{fam} that any of the signatures of a family is triggered is the sum of all signature probabilities of the family.

	$S = 4, R = 0.5$		$S = 4, R = 0.75$		$S = 4, R = 0.9$	
family	P_{fam}	*matches*	P_{fam}	*matches*	P_{fam}	*matches*
Cutwail	$2^{-34.5}$	6118	$2^{-52.6}$	1118	$2^{-67.7}$	43
Fynloski	$2^{-36.6}$	6518	$2^{-45.4}$	1426	$2^{-54.4}$	69
Palevo	$2^{-39.4}$	900	$2^{-47.0}$	186	$2^{-47.0}$	7
Pramro	$2^{-40.0}$	1	$2^{-40.0}$	1	$2^{-40.0}$	1
Ramnit	$2^{-34.4}$	3556	$2^{-43.8}$	649	$2^{-54.4}$	40
Sality	$2^{-39.9}$	638	$2^{-53.1}$	139	$2^{-69.2}$	10
Tofsee	$2^{-41.0}$	512	$2^{-64.5}$	56	$2^{-80.0}$	1
Virut	$2^{-41.0}$	256	$2^{-58.8}$	46	$2^{-80.0}$	1
ZeroAccess	$2^{-39.4}$	768	$2^{-57.2}$	138	$2^{-78.4}$	3
Zeus P2P	$2^{-38.4}$	1536	$2^{-56.2}$	276	$2^{-72.0}$	7

Table 3. Probability that random payload triggers a probabilistic signature per malware family.

Table 3 shows the statistical FP rates (P_{fam}) for the evaluated malware families and lists the number of possible combinations that trigger a signature (*matches*). We distinguish between three relative score thresholds: $R = 0.5$ (first column), $R = 0.75$ (second column, default for R) or $R = 0.9$ (third column). Clearly, using a higher threshold leads to significantly fewer FPs. A low R causes that many possible byte combinations trigger the signature, leading to FPs up to for every 2^{34}th random packet for $R = 0.5$. On a fully saturated 10 Gbit/s link (in worst case 15M packets/s), such an event occurs nearly every 20 minutes. A more conservative R mitigates the issue: approximately one in 2^{40} random packets triggers an alert (at worst a FP every 20 hours for $R = 0.75$). Some signatures, such as for Pramro, have so few 3-tuples such that the relative score does not have any effect. With the statistical FP evaluation and by tuning the thresholds R and S after training we can influence the number of false positives by PROVEX. We used $R = 0.75$ in all other experiments due to its reasonable FP rate.

Live-Network Evaluation: Next, we applied PROVEX on a university network consisting of 155 hosts, of which 69 are diverse workstations (Windows 7, Linux, iOS/Android) and 86 are servers (e.g., HTTP(S), SMTP(S), IMAP(S), DNS, VoIP, XMPP). We deployed PROVEX for 24 hours live on this network during a typical weekday. In total, we captured three FPs, all caused by a repeating UDP-based Echo Protocol (RFC 862) scan towards one of the Internet-facing servers. This Echo scan triggered one of the signatures generated for Palevo three times. When inspecting the packet, we found out that the captured message contained many null bytes, and after applying Palevo's decryption routine, the resulting plaintext contained many null bytes, too. This is an effect of the chained XOR encryption routines, as used by Palevo and Zeus. In practice, such false positives can easily be avoided by ignoring captured frames that predominantly consist of null bytes without negatively affecting the true positive rate.

Fuzzing Evaluation: Third, we used payload fuzzing to test if PROVEX accidentally triggers alarms. Fuzzing helps to randomize data which, in turn, is transformed into random plaintext when applying the decryption routines. We created 2^{35} random payloads of the length of the largest offset in any of the generated signatures. For every generated payload, we applied all decryption routines and matched the decrypted packets against the probabilistic signatures that were generated during the live-network evaluation. Fuzzing revealed that the signatures generated for Virut triggered 123 random packets. As Virut uses a known-plaintext attack to derive its key material from the first four bytes of an encrypted message, every decrypted packet — independent from the ciphertext — starts with "NICK". The signatures created by PROVEX covered these four bytes, significantly raising the chance that arbitrary payloads match the signature. With our knowledge of the C&C decryption routine, we manually excluded these invariant offsets from the Virut signatures. When repeating the experiments, none of the packets triggered the signature anymore, while at the same time Virut could still be detected. This underlines that only the specifics of Virut's cryptography — which we can easily deal with — led to false positives during the payload fuzzing evaluation.

4.2 Qualitative Evaluation

While the evaluation results manifest that PROVEX is effective, in this section, we aim to elucidate and illustrate *why* our methodology works in that we explain the semantics of the automatically-generated probabilistic signatures. We thus reverse engineered not only the C&C message encryption, but also the message processing logic, so that we can explain the semantics of the fields that our probabilistic signatures span.

For example, let us refer to the C&C messages of the Zeus P2P family [18]. Zeus' P2P C&C protocol consists of several message types. Each message carries a header which, among other fields, contains a message type ID, a TTL, a random session ID as well as the bot ID of the sender. However, there are not sufficiently many characteristic header fields for a signature, so ProVeX had to learn payloads that are specific for certain message types. For example, being a P2P bot, Zeus provides messages to request and reply peer lists from its neighbors. Peer list replies cover a list of up to ten peers, each of which consists of a bot ID, a port, an IP version flag and either an IPv4 (4 bytes) or an IPv6 address (16 bytes). Depending on the IP version of a peer, the IP version flag is zero for IPv4 or one for IPv6. These flags thus significantly contribute to the probabilistic signature (for this specific message type), because the two values (zero or one) by far do not exhaust all possible byte values. The probabilistic signatures derived by ProVeX enable us to detect Zeus P2P packets based on these IP version flags.

Similarly, the versions of Zeus P2P binaries and configurations are periodically exchanged and — if needed — synchronized over the P2P network. Version identifiers are four-byte integers which monotonically increase, whenever a new version is released by the bot masters. The most significant byte of such a version identifier — since it changes only every 194 days — is covered in one of our probabilistic signatures for Zeus. In order to keep a signature of such a corner case up to date, periodic retraining could be used. Note, though, that in all cases we observed retraining was not required, as at least one signature per malware family covered time-independent characteristics.

When training on network traffic of a contained environment, automatic approaches are likely to include artifacts of the environment [20]. As such, in case of Zeus P2P, outgoing messages always carry the bot ID of the contained machine as sender ID. The bot ID is derived in a deterministic manner, such that on the same hardware (or VM, resp.), the same bot ID is derived. As a result, when trained on traffic from one contained machine (without changing hardware), the probabilistic signature will learn and cover the sending bot ID. Unfortunately, such artifacts may lead to false negatives, as the bot ID covered in the signature is different in other networks. However, in all cases except for Pramro, ProVeX created at least one signature that did not include an artifact.

In case of ZeroAccess, too, peers are being exchanged by help of peer list requests and responses. While peer list requests have a characteristic field of four null bytes, peer list responses exhibit up to 16 two-byte-wide age fields which are zero in most cases. The trained probabilistic signatures successfully grasp these two specifics.

Tofsee exemplifies an interesting property which is advantageous for our probabilistic signatures. Here, the initial bootstrap message contains several counter values which are encoded as four-byte integers. However, due to the fact that nearly all of these values typically vary in a small range up to 200, three of the four bytes remain zero and contribute significantly to the probabilistic signature. This property of unused bytes in encoded counter values also holds for other families, such as Cutwail and Ramnit.

Cutwail C&C responses fulfill the message length heuristic. In addition, for example, the specific check-in response message type exhibits a magic substring ”addr” which is used to inform the bot about C&C servers. Our probabilistic signature thus exploits the message length heuristic as well as the magic tag for C&C server coordinates.

A Pramro C&C message is preceded with the payload length (two bytes) and a CRC16 checksum (two bytes). Each message starts with a static magic byte of 0x59. In order to evade detection, Pramro pads its messages with a random number of bytes (with random contents). In the first message sent from the bot to the C&C server the bot announces itself. The announcement response contains, among others, the external

IP address of the bot, as observed from the C&C server. In particular, this external IP address – an artifact of the contained environment – was included in the automatically trained probabilistic signature. While this would allow to detect Pramro-infected hosts behind the same NAT gateway as the contained environment, we consider this case as a failure because no general signature could be derived. All in all, Pramro turns out to be a difficult case where no probabilistic vectorized signature could be derived using the requirements as stated in Section 3.3. A signature for Pramro could cover the payload length encoding heuristic from the message header (two bytes) as well as the magic byte. Thus, Pramro does not exhibit at least $S = 4$ characteristic bytes. As an extension, to detect Pramro C&C messages, the signature could be extended to also include the CRC16 checksum check, similar to the payload length encoding heuristic.

In case of Ramnit, our signatures cover two magic bytes in the header of each C&C message. The header also includes the total payload length, encoded in four bytes, which the signatures use to verify the message length. Similarly, the very first announcement message sent by the bot, includes various counters, each encoded in four bytes, which exhibit characteristic zero bytes, especially in the more significant bytes. In addition, the message type 0xe8 is used to request a URL to check for HTTP transmissions, exhibiting a characteristic payload of three bytes 0x01 0x00 0x00.

In case of Sality, the plaintext C&C message payload is padded with a random number of constant bytes, which is exploited in the probabilistic signature.

In contrast to the above mentioned families, Virut does not exhibit a message type encoding field. Instead, Virut relies on an IRC-like protocol. In this case, the probabilistic signature finds characteristic strings, such as ”USER” in the decrypted message. Similarly, for Fynloski, our approach identifies characteristic ASCII strings such as ”info” and ”IDTYP”. These two cases show that our automatic signature generation can also cope with ASCII-based C&C protocols. However, some Fynloski botnets use custom encryption keys that we need to extract to detect such botnets. In Section 4.1, we used the keys of the three prominent Fynloski botnets and thus missed detection of the others.

4.3 Performance Evaluation

This section will evaluate the performance of ProVeX in order to test if our design can be used to botnet detection on high-speed networks. At first, the detection design of ProVeX may seem a non-scalable solution for live networks, as it requires to apply many decryption routines to network packets and relies on matching even more generated probabilistic signatures. When implementing ProVeX, we separated the training process from the detection process. The training process is written in Python, while the time-critical detection engine is written in C.

For our evaluation, we used a standard server with an AMD Opteron 6134 (8 cores, 2.3 GHz) and 16 GB RAM. The training process for all malware families completed in 27 seconds, a negligible overhead. After integrating the signatures to the detection process, we replayed traffic captured at the same network that was used for the False Positive evaluation. With a 1 Gbit network card, and without complex optimizations, ProVeX could capture 960–998MBit/s free of packet loss using only a single core. Note that ProVeX would even work with packet loss or sampled frames, as it does not rely on stream reassembly mechanisms. In fact, most malware families have long-lasting C&C communication spanning dozens of frames, such that packet sampling is perfectly suitable to capture the C&C traffic. In addition, the malware-specific detection routines can easily be performed in parallel, speeding up the process up to 10 Gbit/s.

Adding more decryption routines and signatures adds overhead to PROVEX. Our previous measurements hold for the ten families in our evaluation, but one would want to detect more malware families. We simulated the behavior of PROVEX when it tries to detect more than ten malware families, i.e., we created artificial load by looping over the detection phase (payload decryption and signature matching). In a 2x loop (simulating 20 malware families), PROVEX operated at 663 MBit/s. In a 5x loop (simulating 50 malware families), PROVEX could still capture 418 MBit/s. The performance is CPU-bound (single-core) and I/O-bound (multi-core). The time spent on signature matching is negligible, and our analysis with *perf* shows that mostly the decryption routines influence capturing speed. Parallel computing enables live captures on links with multiple Gbit/s, though. Similarly, a large number of frames (such as HTTP, etc.) can in practice be ignored before decryption. Similarly, adding pre-filters to the decryption routines would further speed up PROVEX, such as filtering on valid ranges of frame sizes [6] or payload entropies [15]. Consequently, despite its bruteforce methodology, PROVEX scales up to multiple Gbit/s in a setting with parallel CPU usage.

5 Discussion and Future Work

We have shown that PROVEX can reliably and efficiently detect encrypted malware C&C traffic and thus complements existing payload-based approaches that fall short of this task. In this section we will detail the limitations of PROVEX, including several discussions on possible evasion techniques.

All malware families that we reverse engineered used encryption routines with re-implementable decryption, such as various XOR variants and RC4. While asymmetric encryption is increasingly being used to *sign* C&C commands [18], hardly any malware family uses asymmetric cryptography to *encrypt* its C&C traffic. Asymmetrically encrypted C&C traffic would pose a performance challenge and if bot-specific key pairs were used, decryption would be much more difficult. Similarly, malware toolkits may form separate botnets that belong to the same malware family, but all use distinct keys which renders PROVEX less scalable, due to the performance impact of an increased number of keys. Finally, when using session keys (e.g., by Diffie-Hellmann key exchange), network traffic decryption becomes tricky. However, the Virut example shows that session keys can sometimes be derived online. In addition, botnets must rely on the cooperation of untrusted parties which makes key management challenging for bot masters. Including key material for asymmetric encryption (such as public keys of the botmasters) in malware binaries make host-based malware detection signatures easier, and key exchange protocols themselves may exhibit detectable characteristics. In SANDNET, we do not see a shift towards malware using stronger cryptography. On the contrary, malware families such as Virut have successfully remained operable since more than 10 years with relatively simple XOR-based encryption. Another limitation of PROVEX is that it requires characteristic bytes at fixed offsets. Although all C&C protocols included fixed-length headers, some C&C protocols exhibit payload field boundaries of dynamic size (e.g., tagged fields). To cope with these cases, PROVEX could realign C&C messages, for example, using string alignment algorithms like Needleman–Wunsch [12]. As we have shown, PROVEX detected the current C&C protocols without realignment.

In a few cases, such as with Pramro, the number of characteristic bytes is low. We leave it up to future work to include additional features in the probabilistic signatures, such as the Shannon entropy over payload windows, checksum computations, or computing valid byte *ranges* (instead of values). However, from the perspective of malware authors, designing a generic C&C protocol without introducing fields with characteristic presentations is a hard task. A major advantage of our current signature design is that the packet matching phase is computationally cheap, making PROVEX a scalable system. We encourage follow-up research that explores scalable machine learning techniques that can be used to automatically train and match decrypted C&C communication, which may leverage additional features not thought of by us.

With PROVEX, we introduce a new paradigm for a NIDS, in that we propose to decrypt all network packets in a brute-force-like manner. From a practical perspective, this introduces manual effort to extract the C&C decryption routines for each malware family. However, we designed PROVEX such that no knowledge about the message semantics is required, which relieves us from the most time-consuming task during the reverse engineering process. While we experienced that manually extracting C&C routines can scale, systems that assist in automating the identification and extraction of crypto routines have been proposed [1, 7, 9, 11], which furthermore support this process.

6 Related Work

A wide area of research explores botnet detection, concisely summarized by Rossow et al. [20]. Sommer and Paxson show the difficulties of applying many of these systems in real networks [21], which are often exacerbated by the lack of human understanding of the derived detection models. In addition, Hadiosmanović et al. show limitations of n-gram-based attack detection for binary protocols [8]. The new signature format generated by PROVEX can be interpreted by human analysts (and modified, if necessary) and allow to detect malware families that lack characteristic substrings. Our proposed method aims to identify C&C traffic of currently undetectable malware families.

A whole body of related work exists on the automatic extraction of protocol specifications and message formats of (unknown) protocols [2–4, 10, 13, 17, 22]. Our approach significantly diverts in that we specifically target two shortcomings of existing work.

First, in contrast to most existing work, our approach aims at protocols with encrypted messages. Given the fact that all recent botnets employ some kind of encryption of the C&C messages, nowadays, this requirement is crucial in order to detect C&C traffic of prevalent botnets, such as Cutwail, Pramro, Palevo, Sality or Zeus P2P.

Second, our approach specifically avoids *stateful* models. *Stateless* signatures allow to operate on a per-frame basis, instead of having to reassemble streams and keeping states over several frames. While our main motivation for stateless models lies in performance, i.e., being able to cope with carrier-grade network link rates up to 10 Gbit/s or even *sampled* network traffic, we also underline that all of the C&C protocols mentioned in this work can successfully be detected with stateless models.

On a broader scale, automatic protocol modeling and signature generation has been covered in many ways. Replayer [13] aims at replaying dialogs of a certain protocol. Botzilla [17] automatically extracts signatures in form of characteristic recurring payload substrings from network traces of repeated execution of a malware sample. While related to our work, Botzilla does not address encrypted protocols which do not exhibit characteristic strings in the ciphertext. Furthermore, PROVEX detects C&C messages even if they do not show substrings that can be found using exact matching.

While ReFormat by Wang et al. [22] targets the system-level automatic reverse engineering of encrypted messages, their approach is a useful extension to our work in that it helps in identifying and re-implementing the en-/decryption process. However, Wang et al. do not cover the recognition aspects that we propose in this work.

Newsome et al. discuss the limitations of contiguous strings as payload signatures and propose Polygraph [14] as possible improvement. Using Bayesian signatures and their probabilistic nature, they show that worms can be detected even if they exhibit only short substrings (Polygraph was evaluated with substring length $n \geq 2$). We decrypt C&C traffic to detect even encrypted communication and show that PROVEX can even match on single characteristic bytes only, abstracting from the notion of substrings.

Krueger et al. [10] target multi-stage attacks which involve several communication transactions (request/response pairs) until reaching the attack's goal. As a result, stateful models are inferred. However, as we show in this work, in the context of malware, C&C protocols can be modeled in a stateless manner, allowing for a higher efficiency.

Discoverer by Cui et al. [4] is a system to infer protocol semantics from network traces of (unknown) protocols. Similarly, Polyglot [2] extracts protocol information using system-level dynamic instrumentation. In contrast to these systems, we do not aim at an exact protocol specification or at fully understanding the message format. Instead, PROVEX identifies characteristic byte values at specific offsets in the decrypted message, without the need to understand field semantics and field boundaries.

7 Conclusion

We proposed a payload-based NIDS PROVEX that can detect a class of C&C communication that has not sufficiently been covered by traditional payload-based inspection systems: encrypted C&C traffic encapsulated in non-descriptive carrier protocols (such as UDP/TCP). Our work was motivated by an increasing number of malware families using encryption routines to evade traditional payload-based detection methods. The way PROVEX works is almost fully-automated. From a number of network traces, and given decryption algorithms and key material, we automatically derive probabilistic vectorized signatures that can be used to detect C&C traffic on multiple Gbit/s links. With this novel method, we scrutinize paradigms that were followed by payload-based NIDSs for a long time. First, PROVEX attempts to *decrypt* C&C traffic in order to see if the plaintext messages belong to C&C communication, an approach – to the best of our knowledge – never explored in a live-traffic NIDS. Second, the probabilistic model covered in signatures generated by PROVEX allow to match on payload patterns that *likely* belong to a certain malware family — a supporting tool to capture certain C&C protocol semantics without the need for manual analysis. Third, the vectorized signatures cover one-byte-wide values as opposed to continuous substrings, which allows to capture even short characteristic C&C protocol fields. Admitting its shortcomings in inverting certain kinds of encryption routines that may arise in the future, we have shown that PROVEX performs well for many prevalent malware families in practice, measured both, in terms of detection accuracy and scalability.

Bibliography

[1] J. Caballero, N. M. Johnson, S. McCamant, and D. Song. Binary Code Extraction and Interface Identification for Security Applications. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2010.

[2] J. Caballero, H. Yin, Z. Liang, and D. X. Song. Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, November 2007.

[3] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. *Proceedings of the 30th IEEE Symposium on Security and Privacy (S&P)*, May 2009.

[4] W. Cui. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. In *Proceedings of the 16th USENIX Security Symposium*, August 2007.

[5] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. van Steen, and N. Pohlmann. On Botnets that Use DNS for Command and Control. In *Proceedings of European Conference on Computer Network Defense (EC2ND)*, September 2011.

[6] C. J. Dietrich, C. Rossow, and N. Pohlmann. *CoCoSpot*: Clustering and Recognizing Botnet Command and Control Channels Using Traffic Analysis. In *A Special Issue of Computer Networks On Botnet Activity: Analysis, Detection and Shutdown*, July 2012.

[7] F. Gröbert, C. Willems, and T. Holz. Automated Identification of Cryptographic Primitives in Binary Programs. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2011.

[8] D. Hadiosmanović, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle. N-gram Against the Machine: On the Feasibility of the N-gram Network Analysis for Binary Protocols. In *Proceedings of the 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, September 2012.

[9] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda. Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In *Proceedings of the 30th IEEE Symposium on Security & Privacy (S&P)*, May 2009.

[10] T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning Stateful Models for Network Honeypots. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec)*, October 2012.

[11] F. Leder, P. Martini, and A. Wichmann. Finding and Extracting Crypto Routines from Malware. In *Proceedings of the International Performance Computing and Communications Conference (IPCCC)*, December 2009.

[12] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[13] J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: Automatic Protocol Replay by Binary Analysis. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, November 2006.

[14] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *Proceedings of the 26th IEEE Symposium on Security & Privacy (S&P)*, May 2005.

[15] J. Olivain and J. Goubault-Larrecq. Detecting Subverted Cryptographic Protocols by Entropy Checking. Research Report LSV-06-13, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2006.

[16] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the USENIX Symposium on Networked Systems Designs and Implementation (NSDI)*, April 2010.

[17] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov. Botzilla: Detecting the Phoning Home of Malicious Software. In *Proceedings of the 25th ACM Symposium on Applied Computing (SAC)*, March 2010.

[18] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos. P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets . In *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)* , San Francisco, CA, May 2013.

[19] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In *Proceedings of ACM EuroSys BADGERS*, April 2011.

[20] C. Rossow, C. J. Dietrich, C. Kreibich, C. Grier, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook . In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2012.

[21] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 31st IEEE Symposium on Security & Privacy*, May 2010.

[22] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, September 2009.