

A Combinatorial Approach to Network Covert Communications with Applications in Web Leaks

Xiapu Luo[§], Peng Zhou[§], Edmond W. W. Chan[§], Rocky K. C. Chang[§] and Wenke Lee[†]

The Hong Kong Polytechnic University[§], Georgia Institute of Technology[†]

{csxluo, cspzhouroc, cswwchan, csrchang}@comp.polyu.edu.hk, wenke@cc.gatech.edu

Abstract—Various effective network covert channels have recently demonstrated the feasibility of encoding messages into the timing or content of *individual* network objects, such as data packets and request messages. However, we show in this paper that more robust and stealthy network covert channels can be devised by exploiting the *relationship* of the network objects. In particular, we propose a combinatorial approach for devising a wide spectrum of covert channels which can meet different objectives based on the channel capacity and channel undetectability. To illustrate the approach, we design WebLeaks and ACKLeaks, two novel covert channels which can leak information through the data and acknowledgment traffic in a web session. We implement both channels and deploy them on the PlanetLab nodes for evaluation. Besides the channel capacity, we apply the state-of-the-art detection schemes to evaluate their camouflage capability. The experiment results show that their capacity can be boosted up by our combinatorial approach, and at the same time they can effectively evade the detection.

I. INTRODUCTION

Network covert channel is an important element in various attacks (e.g., password leaks [1], data leaks [2], large-scale DDoS attacks [3], and botnets [4]) and privacy enhancement systems (e.g., for censorship circumvention [5]–[7]). There are two general approaches to designing network covert communication: encoding covert messages in a network object (e.g., IP packets) and encoding messages in the timing information of *individual* network objects (e.g., inter-packet delay). These two types of channels are usually referred to as *storage channels* and *timing channels*, respectively.

Although the two approaches are different, both share the commonality that messages are encoded in *individual* network objects which could be protocol fields, time intervals, and others. Exploiting only individual objects could result in low capacity and vulnerability to detection. For example, the timing channels can be detected based on the statistical anomalies caused by their encoding methods [8], [9]. In this paper, we show that a novel exploitation of the relationship among network objects will result in more robust and stealthy covert channels. Using the combinations of objects to encode messages, for example, will exponentially increase the channel capacity. It is also less susceptible to anomaly detection, because the messages are not encoded by manipulating the values of single objects.

Exploiting the relationship of network objects for covert communication is not entirely new though. Several works propose to encode messages in the order of packets (e.g.,

IPSec packets [10] and TCP packets [11]). Moreover, El-Atawy and Al-Shaer propose to rearrange packets according to the statistical distribution of packet reordering measured in the Internet [12]. The prevalence of packet reordering, however, is highly dependent on the network paths [13]. In our earlier work [14], we propose Cloak that encodes messages into different combinations of TCP packets and TCP flows [14]. Nevertheless, none of the existing works, to the best of our knowledge, explores the *whole* design space of exploiting the relationships among network objects to build covert channels. The existing methods have exploited only a small design subspace. For example, Cloak is just an instance of applying the approach to TCP flows and packets.

To develop a general combinatorial method, we first identify two fundamental properties that affect the inter-relationship among network objects: *distinguishability* and *sequence detectability*. Distinguishability means that a network object can be differentiated from others, whereas sequence detectability concerns whether the order of a sequence of network objects can be discriminated. In this paper, we design a large number of different message encoding methods, each characterized by a different design objective in terms of distinguishability and sequence detectability. We also propose addition and multiplication principles to generalize these methods. On one extreme, we can design a covert channel with maximal capacity by making all network objects distinguishable and all sequences of network objects detectable. On the other, we can also minimize the channel detectability by making all network objects indistinguishable. Between them, many other covert channels can be designed to meet different objectives.

As applications of the combinatorial approach, we propose two novel covert channels—WebLeaks (in Section III) and ACKLeaks (in Section IV)—that can leak information through the data and acknowledgment (ACK) traffic in a web session (i.e., web leaks). Both do *not* modify any packets and therefore can evade existing detection systems inspecting packet payload [15], [16]. WebLeaks is a cross-layer covert channel that exploits the combinations and permutations of HTTP requests and web pages, and that of TCP packets and TCP flows for message encoding. ACKLeaks, on the other hand, embeds covert messages into pure TCP ACK packets, without affecting the upper layer application. We implement WebLeaks and ACKLeaks, and conduct extensive PlanetLab experiments to evaluate their performance. We also apply the state-of-the-art detection schemes inspecting timing information [8], [17], [18]

to evaluate their camouflage capability. The experiment results show that both channels' data rates can be boosted up by our combinatorial approach, and at the same time they can effectively evade the detection.

In Section II, we introduce the threat model, the combinatorial approach, and effective algorithms for message encoding and decoding. After that, we describe the design of WebLeaks and ACKLeaks using the combinatorial approach in Sections III and IV, respectively. In Section V, we report the PlanetLab experiment results for evaluating their channel capacity and camouflage capability. We then highlight related works in Section VI and conclude this paper in Section VII.

II. THE COMBINATORIAL APPROACH

A. Threat model

There are three roles in a general context of network covert communications: encoder, decoder, and warden. An encoder sends covert messages to a decoder located outside the encoder's network by transforming the messages into some properties of network objects that can be observed by the decoder. The network objects can be TCP/IP packets, traffic flows, and application-level requests, and the properties can be the packets' content, inter-packet time, packets' sequencing, and so on. The decoder, after receiving a copy of the network objects, extracts from them the covert messages. On the other hand, a warden, located on the edge of the encoder's network, inspects all the traffic coming from and going into the network to detect covert communication activities.

B. Riders and carriers

In the ensuing discussion, we use *rider* to refer to a network object and *carrier* to a container that accommodates a sequence of riders. For example, a network flow (a carrier) carries one or more packets (riders). For the timing channel, a time interval (a carrier) contains zero or one (e.g., [17]), exactly one (e.g., [19]), or multiple (e.g., [20]) packets (riders). A covert channel can employ more than one type of riders and/or carriers to increase capacity and camouflage capability. We first propose nine methods for covert message encoding using a single type of riders and a single type of carriers in Section II-C, and then consider multiple types of riders and carriers in Section II-D.

We use R to denote the number of riders in a covert channel. Assuming that each carrier accommodates at least one rider¹, then the number of carriers, denoted as I , can vary from 1 to R (i.e., $I \in [1, R]$). That is, R is the only fixed parameter in this combinatorial approach. The riders and carriers can be further classified according to two fundamental properties:

Distinguishability All riders and carriers are generally distinguishable on the object level, and their distinguishability could be made based on an unique "ID." A rider/carrier is regarded distinguishable if its ID is unique and preserved when received by the decoder. For example, the IPID could be used

¹This assumption can be easily removed to allow some carriers not to always contain rider.

to distinguish a group of consecutive IP packets. However, some protocol fields (e.g., IPID) may be "erased" by packet scrubbers [21]. Therefore, packets relying on these protocol fields for distinguishability may be made indistinguishable. Similarly, time intervals of various lengths could be used to distinguish riders and carriers.

Sequence detectability Distinguishable riders/carriers are regarded sequence detectable if any sequencing of the riders and carriers arranged by an encoder can be detected correctly by a decoder. For example, a sequence of TCP packets in a TCP connection can be detected by their arrival times and sequence numbers. A sequence of TCP connections can also be detected based on their arrival times. However, IP packets can be made sequence undetectable if packet scrubbers randomize their IPIDs.

C. Nine methods for message encoding

Based on the distinguishability and sequence detectability of riders and carriers, we can consider a large number of rider-carrier cases for a given R . To simplify the discussion, we propose the nine main cases ①–⑨ in Table I which are obtained by first considering whether the riders/carriers are distinguishable and then whether the distinguishable riders/carriers, if any, are sequence detectable. Our combinatorial approach is to encode covert messages in the different *arrangements* of the riders by distributing them into the I carriers and possibly sequencing them. Denote the total number of arrangements by $\mathbb{T}_i, i = 1, \dots, 9$. Based on Information Theory [22], the capacity of encoding messages using these arrangements is $\mathbb{C}_i = \lfloor \log_2 \mathbb{T}_i \rfloor$. We have derived \mathbb{T}_i for all nine cases, shown in Table I, and their proofs are sketched in [23].

The nine methods can meet different objectives for covert communications. First, making riders or carriers distinguishable increases the channel capacity (e.g., $\mathbb{C}_2 > \mathbb{C}_1$ and $\mathbb{C}_4 > \mathbb{C}_1$), as illustrated in Figure 1. Moreover, exploiting the sequence information in riders or carriers further increases the channel capacity (e.g., $\mathbb{C}_3 > \mathbb{C}_2$ and $\mathbb{C}_7 > \mathbb{C}_4$). Second, for most encoding methods, their \mathbb{T} s increase exponentially with R . A large \mathbb{T} may ease the design in terms of providing both covertness and capacity, because the encoder may use only a subset of the rider-carrier arrangements that can better mimic the normal traffic pattern but exclude those that are vulnerable to warden detection.

The nine methods are general enough to include the previously proposed covert channels based on packet ordering and TCP packet-flow combinations. The covert channels based on the orders of packets [10]–[12] are special scenarios of method ③ in Table I. Particularly, for one carrier (i.e., $I = 1$) and R riders that support sequence detectability, we have $\frac{R!}{I!} \binom{R-1}{I-1} = R!$. Similarly, the Cloak channels [14] that transform messages into the combinations of TCP flows (carriers) and TCP packets (riders) could be obtained from our combinatorial approach. For example, Cloak³'s capacity is the same as that of method ⑤ by fixing the number of carriers to I (i.e., $I!S(R, I)$). Note that method ⑤ offers a

TABLE I: The nine rider-carrier cases based on their distinguishability and sequence detectability. $\mathcal{P}(R, I)$ is the number of ways of partitioning an integer R into I integers. $\mathcal{S}(R, I)$ is the number of ways of splitting R distinguishable elements into I indistinguishable sets.

		Riders		
		Indistinguishable	Distinguishable and sequence undetectable	Sequence detectable
Carriers	Indistinguishable	$\mathbb{T}_1 = \sum_{I=1}^R \mathcal{P}(R, I)$ ①	$\mathbb{T}_2 = \sum_{I=1}^R \mathcal{S}(R, I)$ ②	$\mathbb{T}_3 = \sum_{I=1}^R \frac{R!}{I!} \binom{R-1}{I-1}$ ③
	Distinguishable and sequence undetectable	$\mathbb{T}_4 = 2^{R-1}$ (i.e., $\sum_{I=1}^R \binom{R-1}{I-1}$) ④	$\mathbb{T}_5 = \sum_{I=1}^R I! \mathcal{S}(R, I)$ ⑤	$\mathbb{T}_6 = 2^{R-1} R!$ (i.e., $\sum_{I=1}^R R! \binom{R-1}{I-1}$) ⑥
	Sequence detectable	$\mathbb{T}_7 = \sum_{I=1}^R I! \binom{R-1}{I-1}$ ⑦	$\mathbb{T}_8 = \sum_{I=1}^R (I!)^2 \mathcal{S}(R, I)$ ⑧	$\mathbb{T}_9 = \sum_{I=1}^R I! R! \binom{R-1}{I-1}$ ⑨

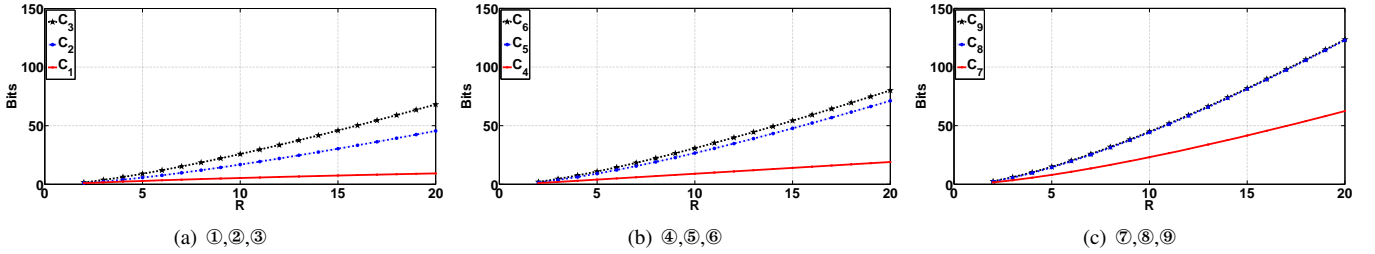


Fig. 1: A comparison of the channel capacity for the nine message encoding methods.

much higher capacity than Cloak³, because it uses a variable number of carriers to increase the total number of rider-carrier arrangements.

D. Generalization

In this section, we generalize the covert channel design for single types of riders and carriers to multiple types. The motivation to devising such covert channels is three-fold. First, exploiting relationships among several types of riders and carriers can increase the channel capacity. Second, employing more types of riders and carriers, instead of requiring riders and carriers to be distinguishable or sequence detectable, makes the channel less detectable. Third, as the types of riders/carriers can be based on different features of the same set of network objects, the covert channel can enjoy higher capacity without dispatching more network objects and have better camouflage capability due to less footprint.

Assume that an encoder has N types of riders, denoted as \mathbb{R}_i ($i = 1, \dots, N$), and M types of carriers, denoted as \mathbb{I}_j ($j = 1, \dots, M$). Then she has $N \times M$ rider-carrier type combinations, denoted as $\mathcal{L}_i = (\mathbb{R}, \mathbb{I})$, $i = 1, \dots, N \times M$. For a given \mathcal{L} , if the combination is feasible in practice, the encoder can use the proper methods in Table I to create a covert channel that has $\mathbb{T}^{\mathcal{L}}$ different arrangements.

We refer to a covert channel employing k ($k > 1$) type combinations of riders and carriers (i.e., \mathcal{L}_i , $i = 1, \dots, k$) as a *composite covert channel*, and use $\Psi = \{\psi_1, \dots\}$ to represent all the possible arrangements of riders and carriers. A ψ may contain multiple types of riders and carriers, depending on the design of a covert channel and the potential restrictions among different types of riders and carriers. Since it is very

difficult, if not impossible, to enumerate all possible composite covert channels and compute the size of their Ψ (i.e., $|\Psi|$), we apply the *addition principle* and the *multiplication principle* in Enumerative Combinatorics [24] to count the total number of arrangements in a composite covert channel:

- 1) **Addition principle** If each ψ in a composite covert channel relies on only one \mathcal{L} , the total number of its arrangements is the sum of the number of arrangements provided by each \mathcal{L} : $|\Psi| = \sum_{i=1}^k \mathbb{T}^{\mathcal{L}_i}$.
- 2) **Multiplication principle** If each ψ in a composite covert channel depends on all \mathcal{L} s, the total number of arrangements is the product of the number of arrangements offered by each \mathcal{L} : $|\Psi| = \prod_{i=1}^k \mathbb{T}^{\mathcal{L}_i}$.

Generally the multiplication principle can result in higher capacity than the addition principle. In practice, both principles may be used because some restrictions may force a ψ to have some but not all \mathcal{L} s. Based on the multiplication principle, we propose in Section III a *cross-layer* covert channel, named WebLeaks, which exploits the information in both HTTP messages and TCP packets.

E. Encoding and decoding procedures

To use the nine methods, the encoder and decoder must pre-agree on the riders, the carriers, R , and a codebook for encoding/decoding. Since it is not always feasible to keep a large codebook due to storage and security issues, we propose efficient encoding and decoding procedures to replace the codebook. The basic idea behind the encoding algorithms is to enumerate all carrier-rider arrangements and then index them consecutively. In the field of combinatorics, this index is known as *rank* and the “encoding” algorithms as *unrank*

⑦, ⑧ (or methods ③, ⑥), and method ⑨ will not be used. The encoder can select other methods. Moreover, only the carrier-rider arrangements that are consistent with the normal traffic behavior will be used.

Strategy 3: If carriers and riders could be sequence detectable, all methods can be used. An encoder could select method ⑨ to maximize the channel capacity. Similar to before, the rider-carrier arrangements that deviate from the normal traffic pattern will not be used.

Although an encoder may not know the detection features used by the warden, she just needs to mimic the normal behavior of network objects. For example, WebLeaks to be presented in Section III mimics normal user behavior of Internet surfing and ACKLeaks in Section IV mimics the normal TCP/IP behavior of sending TCP ACK packets.

F. Unranking and ranking algorithms

It is challenging to design algorithms to rank and unrank the nine cases in Table I and to the best of our knowledge there are no existing algorithms for them. We propose a divide-and-conquer approach to design the ranking and unranking algorithms. It is motivated by Ts' expressions in Table I, which can be constructed based on four building blocks: permutation (i.e., !), combination (i.e., $\binom{\mathcal{X}}{\mathcal{Y}}$), integer partition (i.e., $\mathcal{P}(\mathcal{X}, \mathcal{Y})$), and set partition (i.e., $\mathcal{S}(\mathcal{X}, \mathcal{Y})$). Since ranking and unranking algorithms for these building blocks exist, our approach is to "glue" them together for the nine cases in Table I. We first briefly introduce the four building blocks and then elaborate on the construction of the ranking and unranking algorithms.

Permutation: Let $\text{RkPm}(\Gamma, \mathcal{X})$ and $\text{UkPm}(V, \mathcal{X})$ be the ranking and unranking algorithms for $\mathcal{X}!$, respectively, where $\Gamma = [\gamma_1, \dots, \gamma_{\mathcal{X}}]$ is a permutation of \mathcal{X} elements, and V is the rank. There are several existing ranking and unranking functions for $\mathcal{X}!$, and the time complexity of traditional unranking algorithms is $\mathcal{O}(\mathcal{X}^2)$ [25]. We employ the ranking and unranking algorithms with time complexity $\mathcal{O}(\mathcal{X})$ in [25].

Combination Let $\text{RkCm}(\Theta, \mathcal{X}, \mathcal{Y})$ and $\text{UkCm}(V, \mathcal{X}, \mathcal{Y})$ be the ranking and unranking algorithms for $\binom{\mathcal{X}}{\mathcal{Y}}$, respectively, where $\Theta = [\theta_1, \dots, \theta_{\mathcal{Y}}]$ is an instance of selecting \mathcal{Y} elements from \mathcal{X} available elements, where $0 \leq \mathcal{Y} \leq \mathcal{X}$. We adopt the ranking and unranking algorithms 2.7 and 2.8 in [26].

Integer partition Let $\text{RkIP}(\Lambda, \mathcal{X}, \mathcal{Y})$ and $\text{UkIP}(V, \mathcal{X}, \mathcal{Y})$ be the ranking and unranking algorithms for the integer partition $\mathcal{P}(\mathcal{X}, \mathcal{Y})$, $1 \leq \mathcal{Y} \leq \mathcal{X}$, respectively. $\Lambda = [\lambda_1, \dots, \lambda_{\mathcal{Y}}]$ is the value in the k th partition ($\sum_{k=1}^{\mathcal{Y}} \lambda_k = \mathcal{X}$). We utilize the ranking and unranking algorithms 3.8 and 3.9 in [26].

Set partition Let $\text{RkSP}(\Pi, \mathcal{X}, \mathcal{Y})$ and $\text{UkSP}(V, \mathcal{X}, \mathcal{Y})$ be the ranking and unranking algorithms for the set partition $\mathcal{S}(\mathcal{X}, \mathcal{Y})$, $1 \leq \mathcal{Y} \leq \mathcal{X}$, respectively. $\Pi = [\pi_1, \dots, \pi_{\mathcal{X}}]$ is a division of \mathcal{X} riders into \mathcal{Y} carriers. We use the RankSetPtns and UnrankSetPtns functions in [27].

Figure 2 illustrates the unranking procedure for the nine cases. The inputs to this procedure consist of the rank of an

arrangement (i.e., A), the number of riders (i.e., R) and the method's index (i.e., $1 \leq m \leq 9$) which is shown by a circled number in Figure 2.

The unranking procedure uses four functions to determine the rider/carrier distinguishability (i.e., $\text{IsRidDis}/\text{IsCarDis}$) and sequence detectability (i.e., $\text{IsRidSeq}/\text{IsCarSeq}$) based on the index of adopted method. Besides the unranking algorithms of the four building blocks (i.e., UkPm , UkCm , UkIP , and UkSP), the procedure uses two important functions: GetNumCar and SplitVal .

As listed in Algorithm 1, GetNumCar computes the number of carriers (i.e., I) and the value (i.e., Val) that determines the combination, and the permutation and the order of riders and carriers. Table I shows that the outer layer operation of all Ts is a summation of $\mathfrak{F}(R, i, m)$ for $i = 1$ to R , where $\mathfrak{F}(R, i, m)$ represents the m th method's result. For example, $\mathfrak{F}(R, i, m) = \mathcal{P}(R, i)$ when $m = 1$. Given the rank of an arrangement (i.e., A), we obtain I that fulfils Eq. (1).

$$\sum_{i=1}^{I-1} \mathfrak{F}(R, i, m) < A \leq \sum_{i=1}^I \mathfrak{F}(R, i, m). \quad (1)$$

At the same time, we calculate $Val = A - \sum_{i=1}^{I-1} \mathfrak{F}(R, i, m)$. Lines 2-4 of Algorithm 1 show the corresponding operations.

According to the adopted method, SplitVal divides Val into several parts, each of which represents the rank of the combination/permutation/order of riders and carriers. When $m = 7$, SplitVal returns two values: $U = Val \bmod I!$ and $V = \lfloor Val/I! \rfloor$. U is the rank of a permutation of I carriers, and V is the rank of dividing R indistinguishable riders into I distinguishable carriers.

When $m = 5$, SplitVal outputs two values: $U = Val \bmod I!$ and $V = \lfloor Val/I! \rfloor$. U is the rank of a permutation of I carriers, and V is the rank of arranging R distinguishable riders into I indistinguishable groups. When $m = 8$, SplitVal generates three values: U_1 , U_2 , and V . Letting $U = Val \bmod (I!)^2$, we have $V = \lfloor Val/(I!)^2 \rfloor$, $U_1 = U \bmod I!$, and $U_2 = \lfloor U/I! \rfloor$. V is the rank of arranging R distinguishable riders into I indistinguishable groups. Both U_1 and U_2 are the ranks of permutations of I carriers. They represent different operations in practice. U_1 is related to carrier labeling, whereas U_2 involves in a permutation of carriers.

When $m = 3$, SplitVal calculates two values: $U = Val \bmod R!/I!$ and $V = \lfloor (Val - U)I!/R! \rfloor$. $U \times I!$ is the rank of a permutation of R riders, and V is the rank of dividing R indistinguishable riders into I distinguishable carriers. Similarly, when $m = 6$, SplitVal yields two values: $U = Val \bmod R!$ and $V = \lfloor Val/R! \rfloor$. U is the rank of a permutation of R riders, and V is the rank of dividing R indistinguishable riders into I distinguishable carriers. When $m = 9$, SplitVal gives three values: U_1 , U_2 , and V . Letting $U = Val \bmod (R!I!)$, we have $V = \lfloor Val/R!I! \rfloor$, $U_1 = U \bmod R!$, and $U_2 = \lfloor U/R! \rfloor$. V is the rank of dividing R indistinguishable riders into I distinguishable carriers. U_1 is the rank of a permutation of R riders, and U_2 is the rank of a permutation of I carriers.

As an example, using method ⑨ with $R = 5$, we have $\mathbb{T}_9 = 31320$ different rider-carrier arrangements, and $\mathbb{C}_9 = \lfloor \log_2 31320 \rfloor = 14$ bits. To transmit a 14-bit binary segment “10000000000000,” we first convert it into an integer (i.e., 8192) and then invoke the unranking algorithm for method ⑨ with inputs of $A = 8192$, $R = 5$, and $m = 9$. The outputs of the unranking algorithm include (1) the number of carriers (i.e., $I = 4$); (2) the way of arranging five riders into four carriers (i.e., $\Theta = \{\text{two riders in the first carrier, one rider in the second, third and fourth carrier each}\}$); (3) the order of carriers (i.e., $\Gamma_I = \{2, 4, 3, 1\}$); and (4) the order of riders (i.e., $\Gamma_R = \{1, 5, 3, 4, 2\}$).

Input: A, R, m

Output: I, Val

```

1  $I \leftarrow 0; Val \leftarrow 0;$ 
2 while  $A \geq 0$  do
3    $I \leftarrow I + 1;$ 
4    $Val \leftarrow A; A \leftarrow A - \mathfrak{F}(R, I, m);$ 
5 return  $I, Val;$ 

```

Algorithm 1: The procedure of `getNumCar`.

Figure 3 shows the ranking procedure for the nine cases. Its inputs comprise the observed riders and carriers, R , and a method’s index. Function `ExtractArrange` determines I and obtains the rider-carrier arrangements according to the requirements of the building blocks’ ranking algorithms. We use $(\Gamma_s, \Theta_s, \Lambda_s, \Pi_s)$ to denote the permutations, combinations, integer partitions, and set partitions extracted from the observed riders and carriers through the function `ExtractArrange`. It is worth noting that the number of $\Gamma_s/\Theta_s/\Lambda_s/\Pi_s$ depends on the adopted method.

Given a set of $(\Gamma_s, \Theta_s, \Lambda_s, \Pi_s)$ and m , we first apply the corresponding building blocks’ ranking algorithms to compute the rank (i.e., Val in Figure 3) for a certain I . We then use function `IterateT` to sum up the values of $\mathfrak{F}(R, i, m)$ for $i = 1 \dots, I - 1$ and obtain the final rank (i.e., A in Figure 3).

III. WEBLEAKS

WebLeaks exploits the relationships among network objects on both HTTP and TCP layers. WebLeaks does *not* modify any network object, thus evading content based detection schemes against HTTP-based covert channels and TCP-based covert channels [15], [16], [28]. Moreover, WebLeaks mimics the normal user behavior of browsing web pages and therefore can circumvent detection systems based on HTTP’s and TCP’s timing patterns [8], [15], [17], [18].

On the HTTP layer, WebLeaks regards web pages that contain at least R URLs located in the same web server as carriers. It is not difficult to find such web pages (e.g., portal web sites’ front pages), and in each web page we select R URLs in the same web server as riders. Web pages and URLs are distinguishable according to their hash values. They are also sequence detectable based on the visit sequence (i.e., the sequence of HTTP requests sent by the encoder).

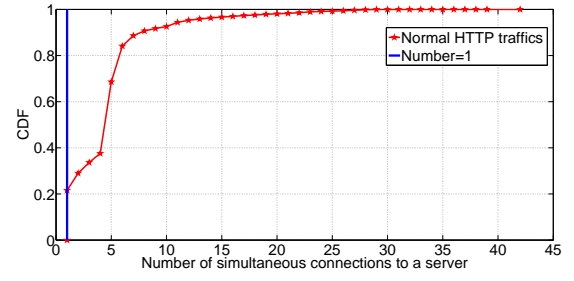


Fig. 4: CDF of the number of simultaneous TCP connections to a web server.

On the TCP layer, if WebLeaks visits R_i (where $1 < R_i \leq R$ and $\sum_i R_i = R$) URLs on the i th web server according to the arrangement of web pages and URLs, it regards TCP flows to that web server as carriers and packets carrying HTTP requests to those URLs as riders. It is common for a client to establish multiple concurrent TCP connections to a server. Figure 4 plots the CDF of the number of simultaneous TCP connections to a web server. We obtained the data by using Firefox 3.6.12 with default settings to visit the front pages of the top 600 web sites ranked by www.alexa.com 30 times. In most cases, the client issued multiple concurrent TCP connections to a server.

The TCP flows are distinguishable according to the source ports selected by the host, and the TCP packets are also distinguishable based on the HTTP requests carried in those packets. Although TCP flows can also be considered as sequence detectable according to the time when a packet is sent through a TCP flow, WebLeaks does not use such information, because it excludes the visit sequence of web pages. Since a WebLeaks’s arrangement comprises web pages, URLs (i.e., HTTP request packets), and TCP flows, its number of arrangements, according to the multiplication principle, is the product of the number of arrangements on the HTTP layer and that on the TCP layer.

The encoding method Figure 5 illustrates the encoding method in WebLeaks. On the HTTP layer, there are five riders (i.e., URLs labeled as $u \in \{1, \dots, 5\}$) in each of the two carriers (i.e., web pages labeled as $w \in \{I, II\}$). For brevity, we use $w.u$ to represent a URL u in a web page w . In this example, method ⑨ is adopted on the HTTP layer and $R = 5$. Hence, we have $\mathbb{T}_9^{HTTP}(5) = 31,320$ different arrangements of web pages and URLs and let $\mathbb{C}_9^{HTTP} = \lfloor \log_2(31320) \rfloor = 14$ bits. To deliver a 14-bit covert message “00000111110001” (i.e., 497), the encoder invokes the unranking algorithm for method ⑨ and obtains the following arrangement: two riders in the first carrier and three riders in the second carrier. The orders of the carriers and riders are $\{I, II\}$ and $\{3, 5, 1, 2, 4\}$, respectively.

Accordingly, the encoder schedules the visit sequence (i.e., sending HTTP requests) as $\{I.3, II.5, II.1, I.2, II.4\}$, which is also labeled through the boxed numbers next to those selected URLs in Figure 5. Note that the orders of web servers (i.e., the carriers on the HTTP layer) are represented according to the sequence of the first two visits.

Given an arrangement on the HTTP layer, the encoder organizes the TCP packets carrying HTTP requests to the selected URLs and the TCP flows connected to the web servers hosting those URLs. Following the example in Figure 5, the encoder visits two URLs in web server `specials.msn.com` (i.e., $R = 2$ in the TCP layer) and three URLs in `www.bbc.co.uk` (i.e., $R = 3$ in the TCP layer). Since WebLeaks adopts method ⑤ in the TCP layer, we have $T_5^{TCP}(2) = 3$ and $T_5^{TCP}(3) = 13$ different arrangements of TCP flows and TCP packets for each web server, respectively.

As a result, the encoder can encode one bit (i.e., $\lfloor \log_2(3) \rfloor = 1$) of information and three bits (i.e., $\lfloor \log_2(13) \rfloor = 3$) of information into the arrangements of TCP flows and TCP packets for the respective web servers. In this example, to visit the URLs in `specials.msn.com`, the TCP packet carrying the HTTP request for $I.3$ goes through TCP flow i and that for rider $I.2$ is sent through TCP flow ii . On the other hand, the HTTP requests for the URLs $\{II.1, II.4, II.5\}$ in `www.bbc.co.uk` are transmitted through TCP flow i . Since each arrangement in WebLeaks includes all riders and carriers on both HTTP and TCP layers, we can apply the multiplication principle to compute the capacity of WebLeaks in this example and obtain 18 bits (i.e., $\log_2(2^{14} \times 2^1 \times 2^3) = 18$).

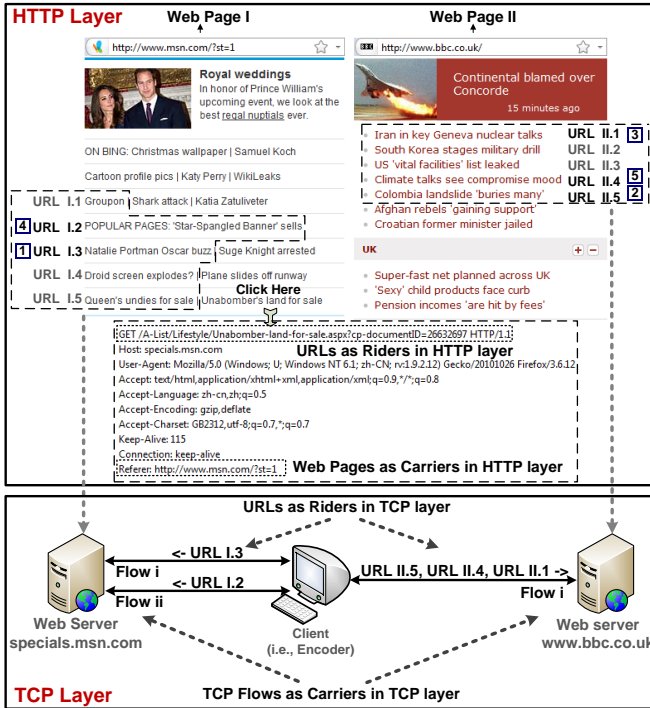


Fig. 5: An example of WebLeaks.

Evading detection The existing detection schemes, most of which are based on the content of HTTP traffic [15], [16], [28], cannot discover WebLeaks, because WebLeaks does *not* change the packet content. Although the latest detection scheme employs the `referer` field to correlate an HTTP request with its previous URLs [28], it still cannot detect WebLeaks, because all the URLs visited by WebLeaks have

the correct `referer` values (i.e., the web page serving as the carriers). Other detection methods employ the timing information related to HTTP requests (i.e., inter-request delays (IRD)) and TCP/IP packets (i.e., inter-packet delays (IPD)) as features to discover HTTP-based and TCP-based covert channels [8], [15], [17], [18], [28]. To evade such detection, WebLeaks mimics the normal browsing behavior by dispatching HTTP requests according to the IRD and IPD distribution extracted from traces of normal web sessions.

WebLeaks may induce anomalies in the number of visits to a URL, if it keeps on using the same set of web pages and URLs. To eliminate such anomalies, we propose a *web group hopping* (WGH) method, which is motivated by WebShare's site-hopping algorithm [29]. Using this approach, the encoder and the decoder agree on a set of web pages that can be divided into N_W groups, each of which includes R web pages. The web pages in different groups may be overlapped, depending on the frequency of a web page occurring in the normal trace. For example, if `Google.com` is visited many times by a normal user, it will appear in many groups. After using one group to send a covert message, the encoder will employ a new group to send the next message. To avoid using a fixed sequence of group indices, we let the index of the $(i + 1)$ th group be $G_{i+1} = H(G_i, Msg)$, where H is a keyed hash function that generates the pseudo-random value in the range of $[1, N_W]$. G_i is the index of the i th group, Msg is the covert message sent through the i th group of web pages, and the key is pre-shared by the encoder and decoder.

The encoder and the decoder can update those web pages after a period of time (e.g., several days) to avoid the unavailability of some web pages. They also pre-agree on a set of popular and long-lasting web sites (e.g., MSN, CNN, etc.) and the method to select the web pages. More precisely, the encoder and decoder use the pervasive random beacon [30] in the Internet to agree on a random integer and then employ a keyed hash chain to generate a sequence of random integers. Part of the random integers are mapped to the indices of web sites using a hash function. The remaining random integers are mapped to the indices of web pages in the selected web sites. Since both encoder and decoder use the same criteria to verify whether the web pages are properly selected, they will finally agree on a set of new web pages.

Figure 6 shows the CDF of the number of visits to a URL in 1000 successive HTTP requests compiled from a 4-week (Aug. 7 to Sept. 3, 2006) web access log collected by IRCaches [31]. It also includes the CDFs for WebLeaks with and without using the WGH method. WebLeaks uses method ⑤ and $R = 10$ to transmit 10,000 covert messages. It is clear from the figure that the URL visit pattern from WebLeaks channel without WGH could be classified as anomalies, because more than 60% of the URLs appear more than ten times in the 1000 successive URL requests. In contrast, the pattern from the WebLeaks channel with WGH displays almost the same distribution as that for the normal HTTP traffic.

An advanced detection scheme may profile a sequence of URLs (i.e., several URLs and the traversal probabilities from

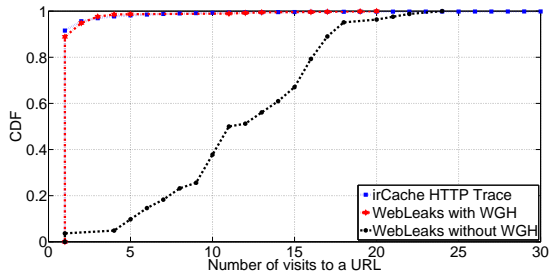


Fig. 6: CDF of the number of times of visiting a URL for normal HTTP requests and WebLeaks with and without WGH.

one URL to another URL) to detect anomalies. WebLeaks can evade such detection scheme by using a normal sequence of URLs to represent one bit of information. Although such mapping may decrease the capacity, WebLeaks can compensate it through several ways, such as (1) using multiple types of riders and carriers, (2) increasing R , and (3) selecting riders and carriers that are distinguishable or sequence detectable and are not regarded as anomalies by the warden.

IV. ACKLEAKS

Unlike Clack [32] that modifies the acknowledge numbers in TCP ACK packets and other TCP-based covert channels affecting data packets [14], ACKLeaks embeds messages in a sequence of pure TCP ACK packets. The distinct advantage of this approach is to allow web clients to communicate with others covertly simply by requesting legitimate data from web servers. Moreover, ACKLeaks can evade content-based detection methods and can be implemented by exploiting the existing TCP connections established by legitimate users.

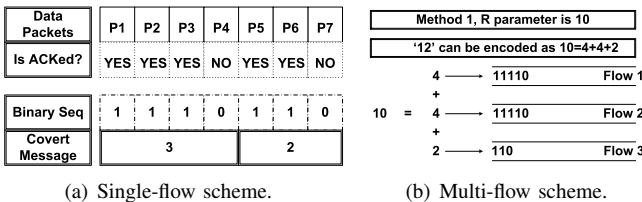


Fig. 7: Two encoding schemes of ACKLeaks.

The encoding method In the simplest form, ACKLeaks embeds messages into the number of consecutive ACKs sent in a single TCP connection. Although the basic idea seems simple, there are many challenges to tackle before it can be in practice. We first introduce the basic design and then elaborate on the mechanisms to enhance the performance of ACKLeaks, such as increasing capacity, correcting errors, avoiding deadlock, and evading detection.

Figure 7(a) gives an example of a single-flow ACKLeaks, in which the encoder acknowledges data packets P_1 , P_2 , and P_3 by replying with three ACKs, one for each data packet, but does not acknowledge P_4 . After that, it acknowledges P_5 and P_6 but not P_7 . In this case, each ACK is a rider, and each sequence of contiguous ACKs is a carrier, and the

absence of each ACK serves as a delimiter for two consecutive carriers. Since ACK values are generally not unique, we regard them as indistinguishable riders. But the carriers are distinguishable. Each covert message is represented by the arrangement of riders in carriers. The single-flow ACKLeaks actually implements the binary (d,k)-constraint sequence code [33].

When multiple TCP connections are available, we can design ACKLeaks with TCP connections as carriers. Figure 7(b) shows a multi-flow ACKLeaks using method ① with $R = 10$. To send a covert message “12,” the encoder first unrankes the message to obtain the corresponding rider-carrier arrangement: four riders each in the first and second carriers, and two riders in the third carrier. She then dispatches the ACKs to the connections according to the arrangement output. **Error detection and correction** Since ACK losses introduce errors to the ACKLeaks channel, we propose a coding scheme, denoted by (R, m, n) , to detect and correct errors. The basic idea is to use m consecutive ACKs to represent a single rider and n consecutive unacknowledged TCP data packets to denote a delimiter. Consequently, the minimum Hamming distance for each rider representation is m , thus allowing ACKLeaks to detect $m - 1$ ACK losses and correct $\frac{m-1}{2}$ ACK losses for each rider [34]. Moreover, using n unacknowledged data packets as a delimiter can help detect and correct at most $n - 1$ ACK losses which are wrongly decoded as a delimiter. **Avoiding deadlock** We implement ACKLeaks by discarding ACKs from a local host’s TCP connection to a remote sender and inserting additional ACKs. Dropping ACKs, however, can cause the remote sender to retransmit the unacknowledged data packet, which in turn elicits the local host to send a duplicate ACK for the retransmitted packet. If such ACK is used as a delimiter, a deadlock will occur and the TCP flow will stall. We therefore propose an R -shift algorithm to address this issue. When the encoder detects a duplicate ACK from the local host, she lets it go and drops the $(m \times R)$ th ACK after the duplicate ACK. The decoder can extract the message by applying $\text{mod } m \times R$ to the observed ACK sequence, because the longest ACK sequence must not exceed $m \times R$ in ACKLeaks.

Evading detection We define *correspondency* for TCP data packets and ACKs as the number of consecutive TCP data packets required to trigger an ACK. We denote a correspondency of K by χ_K . ACKLeaks may induce anomalous correspondency. Specifically, the (R, m, n) scheme will cause two kinds of correspondency— χ_1 and χ_{n+1} —because the encoder acknowledges $m \times R$ data packets (i.e., χ_1) and then skips n packets before acknowledging a new data packet (i.e., χ_{n+1}). Let $P_{(R,m,n)}(K)$ be the probability of inducing χ_K in a flow for ACKLeaks. For example, given $m = n = 1$, $R = 5$, and method ①, a covert message “2” is encoded as “1110110” that contains three χ_1 s and two χ_2 s. We therefore have $P_{(5,1,1)}(1) = 3/5$ and $P_{(5,1,1)}(2) = 2/5$.

Generally, $P_{(R,m,n)}(K)$ is different from $P(K)$, the probability in the normal traffic. To evade detection, we propose a new algorithm, referred to as *Fixed Carrier Quantity with*

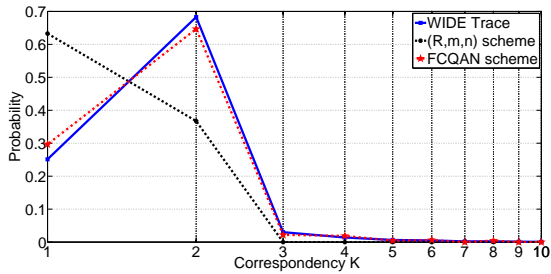


Fig. 8: Distribution of χ_K for normal traces and ACKLeaks.

Alterable n (FCQAN), to let ACKLeaks mimic the normal distribution of correspondence. FCQAN is motivated by two observations: $P_{(R,m,n)}(1)$ is determined by the number of carriers (i.e., I), and different n s produce different χ_{n+1} s. FCQAN takes a two-step approach to make ACKLeaks’s correspondence similar to that in the normal trace. In the first step, FCQAN fixes I such that $P_{(R,m,n)}(1)$ is close to $P(1)$. In the second step, before generating a delimiter, the algorithm computes n , such that $P_{(R,m,n)}(K)$ approximates $P(K)$ for $K > 1$. More precisely, we first calculate $P(K)$, $K > 1$, and its CDF, and then generate a random value $v \in [0, 1]$. If $\sum_{i=2}^K P(i) \leq v < \sum_{i=2}^{K+1} P(i)$, we set $n = K$.

Figure 8 shows the correspondence distributions for the (R, m, n) coding scheme, FCQAN, and normal HTTP traffic obtained from the WIDE data set [35]. The WIDE data set contains *all* traffic going through its sample point-F from 30 March 2009 to 2 April 2009, and the size of packet header traces is around 433 GB. The ACKLeaks channel uses $R = 10$, $m = 1$, and method ①. As shown, the correspondence distributions for FCQAN and the normal traces are very similar. However, the (R, m, n) scheme’s distribution deviates significantly from the other two, as already explained before.

V. PERFORMANCE EVALUATION

We implement WebLeaks and ACKLeaks in C language on Ubuntu (Kernel v2.6.28) and evaluate their performance and camouflage capability through PlanetLab. The performance is measured in terms of goodput (i.e., the average rate of information leak in bits/second). To evaluate their camouflage capability, we apply the state-of-the-art detection schemes to WebLeaks and ACKLeaks.

A. Implementation

To ease the evaluation, we implement the decoder as a web proxy. On the HTTP layer, the encoder constructs appropriate HTTP requests for selected URLs. On the TCP layer, the encoder establishes the required number of TCP connections to the decoder and sends HTTP requests through different TCP connections. We use the same R in both HTTP and TCP layers, because all URLs can be regarded as coming from the same server. Note that the decoder can be a passive sniffer, instead of a web proxy, placed at any location to monitor the HTTP requests. For the ease of explanation, we use a method

pair (a, b) to refer that WebLeaks uses method a on the HTTP layer and method b on the TCP layer.

ACKLeaks uses `libpcap` to capture all incoming data packets and stores their sequence numbers, acknowledgement numbers, and packet lengths in a linked list. Based on this list and the selected method, ACKLeaks knows which data packets should be acknowledged and records the verdict in the corresponding linked nodes. The encoder uses `iptables` to drop unnecessary ACK packets and employs `raw socket` to generate new ACK packets if necessary. Moreover, ACKLeaks incorporates both single-flow and multi-flow schemes to apply the combinatorial framework.

B. Performance

We install the WebLeaks and ACKLeaks encoders in our campus network and their decoders in six PlanetLab nodes in different geographical locations as listed in Table II.

TABLE II: The PlanetLab nodes used in our experiments.

PlanetLab nodes	RTT	PlanetLab nodes	RTT
137.189.98.30 (HK)	3.91 ms	203.178.143.10 (JAP)	56.72 ms
130.194.252.9 (AU)	119.4 ms	143.215.131.197 (US)	221.8 ms
142.150.238.12 (CA)	247.4 ms	193.55.112.41 (FR)	256.5 ms

Figure 9 shows WebLeaks’s goodput under different settings. We observe that the goodput increases with R and decreases with RTT. This result is in accordance with the analysis in Table I. Specifically, Figure 9(a) shows that the goodputs for all evaluated method pairs increase with R on the path to the PlanetLab node in Hong Kong; Figure 9(b) illustrates the same trend on the paths to other PlanetLab nodes. Moreover, using methods with distinguishable or sequence detectable carriers or riders increases the goodput. As shown in Figure 9(a), method pair (⑨, ⑤) produces higher goodput than other method pairs. In our evaluation, WebLeaks achieves the highest goodput (i.e., more than 100 Kbps) on the path to Hong Kong with $R = 50$ and method pair (⑨, ⑤).

We evaluate both single-flow and multi-flow schemes for ACKLeaks. For comparison purpose, we also include a basic non-combinatorial coding scheme that embeds covert messages into the number of consecutive ACKs and uses unacknowledged TCP data packets as delimiters. Figure 10(a) shows the goodputs of single-flow ACKLeaks, multi-flow ACKLeaks, and the basic coding scheme. Although single-flow ACKLeaks with method ① has least goodput comparing to the other ACKLeaks, its goodput is still higher than that of the basic scheme. Besides, Figure 10(a) shows that method ④ with distinguishable carriers obtains a better goodput than method ①. This result is expected according to our theoretical analysis. Moreover, for the same method ④, the multi-flow scheme can further improve the goodput, because the pure ACKs are dispatched to multiple TCP connections in parallel, thus reducing the time for sending a covert message.

Unlike WebLeaks, Figure 10 shows that ACKLeaks’ goodput decreases with larger R , because the R -shift algorithm described in Section IV introduces a longer delay as R

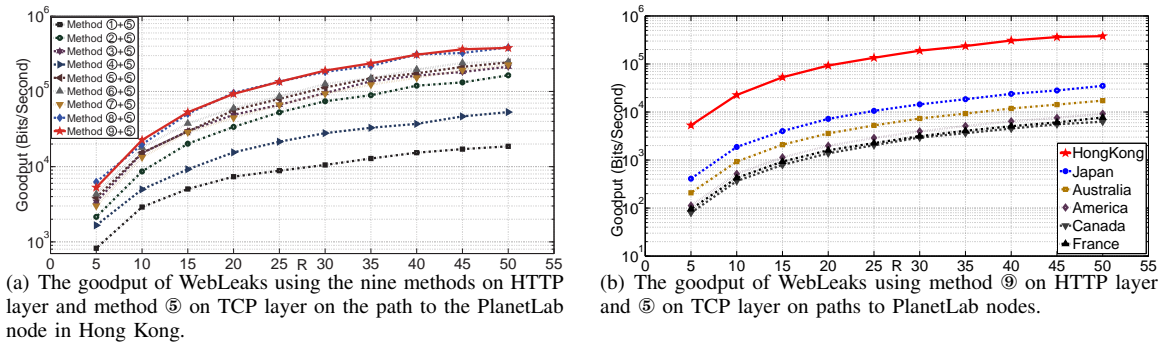


Fig. 9: The goodput of WebLeaks.

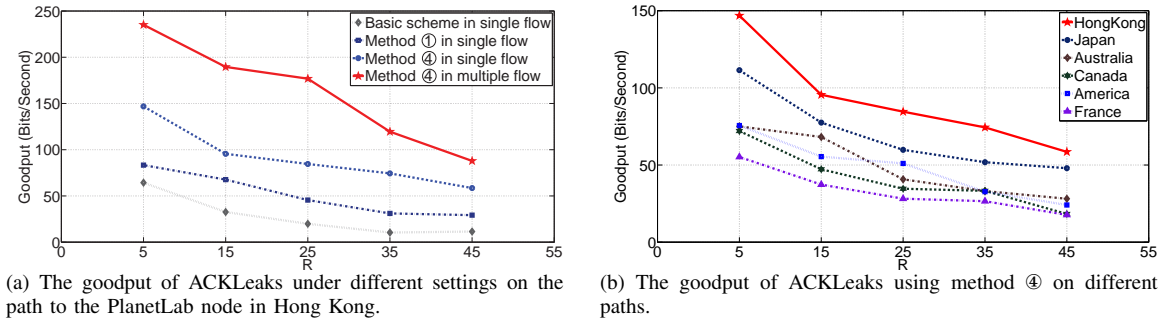


Fig. 10: The goodput of ACKLeaks.

increases. Moreover, the goodput decreases with longer RTT because of the inflated waiting time for ACKLeaks to observe new TCP data packets from the sender.

C. Camouflage capability

Since WebLeaks and ACKLeaks do not change any packet content, they can evade those content-based detection methods [15], [28], [36]. In Section III, we discuss how WebLeaks can evade the detection based on the number of visits to certain URLs and the transition patten of visiting URLs. In Section IV, we show that ACKLeaks can evade the detection based on the correspondency between TCP data packets and TCP ACK packets. To further evaluate WebLeaks's and ACKLeaks's camouflage capability, we apply the state-of-the-art timing-based detection methods to their traces. The experiment results show that WebLeaks can also evade all these detection methods designed for HTTP-based covert channels [15], [28] and for TCP/IP packets [8], [17], [18].

The experiment result shows that WebLeaks can evade the Web Tap system that detects covert channels in HTTP traffic [15]. Web Tap employs five filters: single request size, request time-of-day, request regularity, bandwidth limit, and delay time. Since WebLeaks can easily adjust its traffic to evade the detection based on the first four filters according to the discussion of Web Tap's vulnerabilities in [15], we only examine the delay time filter in our experiment.

From the web logs captured by five IRCache servers between 07/Aug/2006 to 03/Sep/2006, we obtain IRD series of legitimate HTTP requests for a specific web site (according

to their domain names) for each particular source IP address, resulting in an aggregate series of around $n = 52$ million IRDs for all the source IP addresses. Similar to [15], we obtain a subset of $\sqrt{n} = 7155$ elements in the aggregate series to compute the derivative of IRDs and the running average of the derivative.

After that, we instruct WebLeaks to mimic the subset of IRDs and measure the resultant IRD for every pair of HTTP requests injected by WebLeaks using method pair (⑨,⑤) and $R = 10$. To reduce the processing time, we consider only those IRDs smaller than 120 seconds, which cover 82% of the subset. Figure 11 shows the CDF of the derivative of IRDs from IRCache's HTTP traces and that from WebLeaks. It shows clearly that WebLeaks can strictly mimic normal HTTP IRDs and evade the detection.

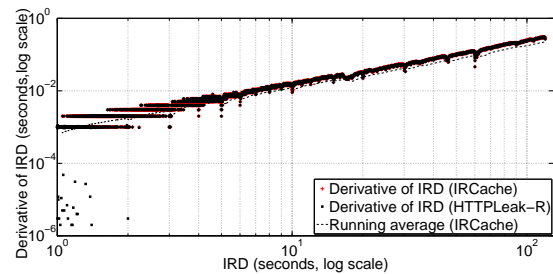


Fig. 11: CDFs of the IRDs' derivatives.

WebLeaks, as a cross-layer covert channel to embed partial information into TCP flows, can also evade detection methods

based on the TCP-layer IPDs due to two main reasons. First, based on the measurement of 143,333 HTTP requests from the top 2000 web sites ranked by `www.alexacom`, we found that the sizes of more than 99% HTTP requests are less than the default TCP MSS (i.e., 1460 bytes) [37]. That is, almost all requests can be transmitted in one packet. Therefore, a successful mimicking of IRDs implies a successful mimicking of IPDs. Second, even if some requests are sent in more than one packet, WebLeaks will not cause abnormal IPDs, because it does not alter the packets' timing information.

To evaluate ACKLeaks's camouflage capability, we implement the Regularity test [17], ϵ -similarity test [17], Kolmogorov-Smirnov (KS) test [18], EN test [8] and CCE test [8], and applied them to the ACKLeaks traces. Following the parameter settings in their original papers, we let the Regularity test's non-overlapping window size ω be 100 [17] and ϵ -similarity test's ϵ be 0.01 [17]. For the KS test, the normal distribution of the inter-ACK packet delays (IADs) is built from a set of around 10,000,000 IADs obtained from 10,000 HTTP flows randomly selected from the WIDE data set [35], following what Gianvecchio et al. did in [18].

We use the traces of downloading a 10-MB file 100 times from the PlanetLab node in Japan as normal HTTP samples. Since ACKLeaks only affects TCP ACK packets in a TCP flow, we use each method to compute a score based on the IADs in individual normal HTTP flow. We then start ACKLeaks when downloading the same file 100 times and save the traces as ACKLeaks samples.

Cabuk et al. [17] classify a flow as a covert channel if it has a score smaller than a threshold in the Regularity test or a score larger than another threshold in the ϵ -similarity test [17]. Gianvecchio et al., on the other hand, classify a flow as a covert channel if its KS test score is larger than a threshold. They also found that a covert timing channel has a score less than a threshold in the EN test or a score that is either lower or higher than thresholds in the CCE test [8].

Following these suggestions in their original papers, we select the threshold of each detection algorithm by letting its false positive rate be no more than 0.01. As shown in Table III, the detection rates for ACKLeaks are very low (less than 0.03). In other words, ACKLeaks can successfully evade these detection methods.

TABLE III: Detection rates on ACK packets per flow.

Tests	Normal HTTP	ACKLeaks
	False Positive	Detection Rate
Regularity ≤ 0.271	0.01	0
ϵ -similarity ≥ 88.97	0.01	0.03
KSTEST ≥ 0.481	0.01	0
EN ≤ 6.388	0.01	0.02
CCE ≥ 1.417	0.01	0

VI. RELATED WORK

Most storage channels use specification-based approaches to locate possible covers [38], [39]. Existing literature and tools on storage channels have already covered many popular

protocols (e.g., IP [10], [39], TCP [10], [36], [40], SSH [41], and HTTP [6]) and applications (e.g., Web counter [29], VoIP [42]).

Existing timing channels employ either the absolute time interval as the timing reference [17], or a single packet or a group of packets as the timing reference [18]–[20], [43], [44]. For example, the IP timing channel delivers bit 1 by sending one packet during a predefined time interval and bit 0 by keeping silent [17]. Jitterbug encodes messages into the inter-arrival time between two consecutive packets [19]. Through mimicking TCP's burstiness, TCPScript leaks information through the number of back-to-back packets in each burst [20].

Infranet lets an encoder select one out of K URLs to deliver $\log_2 K$ -bit information [5]. There are two major differences between Infranet and WebLeaks. First, Infranet considers neither the order of URLs nor the combinations of URLs and web pages, thus having much lower capacity than WebLeaks. Second, Infranet does not exploit TCP layer information to increase capacity. Interested reader may refer to [45] for a more thorough survey on covert channels.

To detect hidden data in protocols, models have been built to capture anomalies in the statistical features of packet headers [36] and the timing information of packets [8], [17], [18]. Detection schemes against HTTP-based covert channels scrutinize the content and timing information of HTTP requests [15], [16], [28]. Borders and Prakash propose a framework to quantify information leaks due to HTTP-based covert channels [28]. Besides detection, another approach is to neutralize covert channels by performing active operations on the traffic, for example, the protocol scrubbers [21].

Quantifying the capacities of nine methods is motivated by the *Twelvefold Way* problem in the field of Enumerative Combinatorics [24] and its extension [46]. They refer to a number of counting problems that count all the possible ways of putting balls into urns and their results. We select the cases that can be employed to design network covert channels instead of applying all cases in [24], [46]. Moreover, we design new ranking and unranking algorithm to realize the transforming between combinations and their index.

Port knocking can also benefit from our combinatorial approach. More precisely, since the original port knocking method needs to select a sequence of packets as the secret for user authentication [47], it can have a much larger secret space by adopting our combinatorial method.

VII. CONCLUSIONS

Unlike most existing network covert channels that consider only *individual* network objects, we proposed a general combinatorial approach that exploits the relationship among network objects to devise a wide spectrum of covert channels. Based on two fundamental features of network objects, our approach provides *nine* main methods and two generalization principles for network covert communications. To illustrate our approach, we applied it to the design of two novel covert channels: WebLeaks and ACKLeaks. Both of them can leak information

without modifying any network object. We implemented them and conducted extensive experiments to evaluate their performance and applied the state-of-the-art detection algorithms to evaluate their camouflage capability. The experiment results show that the combinatorial approach boosted up their data rates and helped them evade the detection.

ACKNOWLEDGMENTS

We appreciate the anonymous reviewers for their quality reviews. We thank Duane Wessels for allowing us to access the NLANR traces. This work is partially supported by grants GU-G386 and G-U669 from The Hong Kong Polytechnic University. This material is based upon work supported in part by the National Science Foundation under grant no. 0831300, the Department of Homeland Security under contract no. FA8750-08-2-0141, the Office of Naval Research under grants no. N000140710907 and no. N000140911042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Department of Homeland Security, or the Office of Naval Research.

REFERENCES

- [1] A. Young and M. Yung, "Deniable password snatching: on the possibility of evasive electronic espionage," in *Proc. IEEE Symp. Security and Privacy*, 1997.
- [2] NISCC Monthly Bulletin, "Phishing trojan plays ping-pong with captured data," www.niscc.gov.uk/niscc/docs/re-20060831-00629.pdf, August 2006.
- [3] P. Henry, "Covert channels provided hackers the opportunity and the means for the current distributed denial of service attacks," CyberGuard Corporation, 2000.
- [4] F. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *Proc. ESORICS*, 2005.
- [5] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger, "Infranet: Circumventing censorship and surveillance," in *Proc. USENIX Security*, 2002.
- [6] M. Bauer, "New covert channels in HTTP: Adding unwitting Web browsers to anonymity sets," in *Proc. ACM WPES*, 2003.
- [7] S. Burnett, N. Feamster, and S. Vempala, "Chipping away at censorship with user-generated content," in *Proc. USENIX Security*, 2010.
- [8] S. Gianvecchio and H. Wang, "Detecting covert timing channels: An entropy-based approach," in *Proc. ACM CCS*, 2007.
- [9] D. Figueiredo, B. Liu, V. Misra, and D. Towsley, "On the autocorrelation structure of TCP traffic," *Computer Networks*, no. 3, 2002.
- [10] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. ACM Workshop on Multimedia Security*, 2002.
- [11] R. Chakinala, A. Kumarasubramanian, R. Manokaran, G. Noubir, C. Pandu Rangan, and R. Sundaram, "Steganographic communication in ordered channels," in *Proc. Inform. Hiding*, 2006.
- [12] A. El-Atawy and E. Al-Shaer, "Building covert channels over the packet reordering phenomenon," in *Proc. IEEE INFOCOM*, 2009.
- [13] X. Luo and R. Chang, "Novel approaches to end-to-end packet reordering measurement," in *Proc. ACM/USENIX IMC*, 2005.
- [14] X. Luo, E. Chan, and R. Chang, "Cloak: A ten-fold way for reliable covert communications," in *Proc. ESORICS*, 2007.
- [15] K. Borders and A. Prakash, "Web Tap: Detecting covert Web traffic," in *Proc. ACM CCS*, 2004.
- [16] N. Schear, C. Kintana, Q. Zhang, and A. Vahdat, "Glavlit: Preventing exfiltration at wire speed," in *Proc. ACM HotNets-V*, 2006.
- [17] S. Cabuk, C. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *Proc. ACM CCS*, 2004.
- [18] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia, "Model-based covert timing channels: Automated modeling and evasion," in *Proc. RAID*, 2008.
- [19] G. Shah, A. Molina, and M. Blaze, "Keyboards and covert channels," in *Proc. USENIX Security*, 2006.
- [20] X. Luo, E. Chan, and R. Chang, "TCP covert timing channels: Design and detection," in *Proc. IFIP/IEEE DSN*, 2008.
- [21] D. Watson, M. Smart, G. Malan, and F. Jahanian, "Protocol scrubbing: Network security through transparent flow modification," *IEEE/ACM Trans. Net.*, 2004.
- [22] T. Cover and J. Thomas, *Elements of Information Theory*, 2nd ed. Wiley-Interscience, 2006.
- [23] X. Luo, P. Zhou, E. Chan, R. Chang, and W. Lee, "A combinatorial approach to network covert communications with applications in web leaks (full version)," <http://www.comp.polyu.edu.hk/~csrchang/Leak11.pdf>, 2011.
- [24] C. Charalambides, *Enumerative Combinatorics*. Chapman Hall CRC, 2002.
- [25] W. Myrvold and F. Ruskey, "Ranking and unranking permutations in linear time," *Information Processing Letters*, vol. 79, pp. 281–284, 2001.
- [26] D. Kreher and D. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC press, 1998.
- [27] H. Wilf, "East Side, West Side: An introduction to combinatorial families with Maple programming," <http://www.cis.upenn.edu/~wilf/lecnotes.html>, 2002.
- [28] K. Borders and A. Prakash, "Quantifying information leaks in outbound web traffic," in *Proc. IEEE Symp. Security and Privacy*, 2009.
- [29] X. Luo, E. Chan, and R. Chang, "Crafting web counters into covert channels," in *Proc. IFIP SEC*, 2007.
- [30] H. Lee, E. Chan, and M. Chan, "Pervasive random beacon in the Internet for covert coordination," in *Proc. Inform. Hiding*, 2005.
- [31] NLANR, "Web caching project," <http://www.ircache.net>.
- [32] X. Luo, E. Chan, and R. Chang, "CLACK: A network covert channel based on partial acknowledgment encoding," in *Proc. IEEE ICC*, 2009.
- [33] E. Zehavi and J. Wolf, "On runlength codes," *IEEE Trans. Inform. Theory*, vol. 34, 1988.
- [34] T. Moon, *Error correction coding: Mathematical methods and algorithms*. John Wiley & Sons, 2005.
- [35] WIDE data set, <http://tracer.csl.sony.co.jp/mawi/>.
- [36] S. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proc. Inform. Hiding*, 2005.
- [37] X. Luo, P. Zhou, E. Chan, W. Lee, R. Chang, and R. Perdisci, "HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows," in *Proc. ISOC NDSS*, 2011.
- [38] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating steganography in Internet traffic with active wardens," in *Proc. Inform. Hiding*, 2002.
- [39] N. Lucena, G. Lewandowski, and S. Chapin, "Covert channels in IPv6," in *Proc. PET Workshop*, 2005.
- [40] J. Giffen, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *Proc. PET Workshop*, 2002.
- [41] N. Lucena, J. Pease, P. Yadollahpour, and S. Chapin, "Syntax and semantics-preserving application-layer protocol steganography," in *Proc. Inform. Hiding*, 2004.
- [42] T. Takahashi and W. Lee, "An assessment of VoIP covert channel threats," in *Proc. SecureComm*, 2007.
- [43] V. Berk, A. Giani, and G. Cybenko, "Detection of covert channel encoding in network packet delays," Dartmouth College, Tech. Rep. TR2005536, 2005.
- [44] S. Sellke, C. Wang, S. Bagchi, and N. Shroff, "Covert TCP/IP timing channels: Theory to implementation," in *Proc. IEEE INFOCOM*, 2009.
- [45] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys and Tutorials*, March 2007.
- [46] R. Proctor, "Let's expand Rota's twelvefold way for counting partitions," <http://www.math.unc.edu/Faculty/rap/30FoldWay.pdf>, June 2006.
- [47] M. Krzywinski, "Port knocking: Network authentication across closed ports," *SysAdmin Magazine*, vol. 12, 2003.