Crafting Web Counters into Covert Channels

Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang

Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, SAR, China
{csxluo|cswwchan|csrchang}@comp.polyu.edu.hk

Abstract. Almost all the previously proposed network storage channels write covert messages in the packets' protocol fields. In contrast, we present in this paper a new network storage channel *WebShare* that uses the plentiful, public Web counters for storage. Therefore, the physical locations of the WebShare encoder and decoder are not restricted to a single path. To make WebShare practical, we have addressed a number of thorny issues, such as the "noise" introduced by other legitimate Web requests, and synchronization between encoder and decoder. For the proof-of-concept purpose, we have experimented a WebShare prototype in the Internet, and have showed that it is practically feasible even when the Web counter and the encoder/decoder are separated by more than 20 router hops.

1 Introduction

Using a network covert channel, two Internet hosts can communicate with each other such that others are not even aware of its existence. Therefore, it is not surprising that a network covert channel has been perceived as a serious threat to the national security and enterprise security, because it can be a handy tool for data exfiltration and large-scale attack coordination. On the other hand, network covert channels have found useful applications in leaking news from some countries, and allowing privacy-concerned communities to bypass censorship and privacy intrusion devices [1, 2].

Similar to the classic covert channels in trusted computer systems, network covert channels are broadly classified into *network storage channels* and *network timing channels* [3, 4]. Since the network timing channels are not the focus of this paper, we will not further consider them in the rest of this paper. In a network storage channel, an encoder embeds messages into a storage location that can be read by a decoder. The storage locations are usually packets' header fields in the previously proposed storage channels. Moreover, to further conceal the covert channel, the encoder could send the cover traffic to a third part. Therefore the decoder has to eavesdrop the communication path from the encoder to the third party. However, eavesdropping an Internet link generally requires a special setup and is not reliable [5]. As a result, we turn to the traditional sense of storage channels investigated for multilevel secure systems [6, 7].

That is, we select a publicly shared state in the Internet as a storage. In this paper we use Web counters as the storage and demonstrate its practicality by presenting *WebShare*, a new network storage channel based on Web counters. Danezis [8] first mentioned the idea of using Web counters but did not provide any details to realize it.

Furthermore, there are other publicly shared states that could also be exploited for covert communications. In [9], Rowland proposes to encode into the sequence number (SN) of a TCP packet that is "bounced" to a decoder by a TCP server; thus, the SN can be considered as a TCP-layer shared state. Although this method is simple, it can be easily detected based on spoofed source IP address and an unusually large number of TCP SYN packets. As another example, Danezis has recently proposed exploiting the IPID field in the IP header as a shared state [8]. However, the global IPID counter is not supported by all systems. Moreover, the global IPID counter has been exploited for scanning idle ports since 1998 [10]; consequently, many IDSes and firewalls are able to detect and defeat it. As we will show later, the plentiful Web counters used by WebShare do not suffer from the above problems.

Although the basic idea of WebShare is simple, we will show in the rest of the paper that we have overcome many inherent technical issues in the design of WebShare. Notable ones include allowing the encoder and decoder to maintain only loose time synchronization, using multiple counters to increase the capacity, and proposing a site-hopping approach to further camouflage WebShare and boost the capacity. We have also experimented a WebShare prototype in the Internet and have evaluated its performance under different parameter settings. The initial results show that WebShare is practically feasible.

We structure the rest of this paper as follows. Section 2 introduces WebShare and details our approaches to resolving a number of design issues. In section 3, we report the experiment results and findings. Section 4 briefly summarizes the existing works on network storage channels and the defense methods. We finally conclude this paper in section 5.

2 WebShare

To communicate covertly through a WebShare channel, both the WebShare encoder and decoder agree on the start time T_0, and the Web counter to use. As will be explained in subsection 2.2, WebShare does not require strict time synchronization between the encoder and decoder. However, to simplify the discussion, we assume for the time being that the times are perfectly synchronized, and we will later propose methods to mitigate the impact of time desynchronization.

The decoder first fetches the Web counter value before T_0. After that, the encoding period T_E and decoding period T_D alternate between themselves, and the periods are also known to them beforehand. Moreover, define a *run* as $T_A = T_E + T_D$. During each T_E, the encoder requests the corresponding Web counter once for transmitting bit `1` but does not send any for bit `0`. In the next T_D, the decoder fetches the Web counter value for message decoding.

In terms of the threat model, we assume that the encoder's incoming and outgoing traffic is all under a passive warden's close scrutiny for the purpose of covert channel detection. We further assume that the Web sites whose counters are used by WebShare do not conduct cooperative intrusion detection [11]. This assumption is reasonable, because the Web sites exploited by WebShare are randomly selected from a vast number of public Web sites. It is therefore very difficult, if not impossible, for them to share the Web access information. Furthermore, the WebShare traffic may not easily trigger an alarm in these Web sites, because it may not increase the counter values frequent enough.

2.1 Noise handling

As the Web counter value could also be increased by other legitimate visitors, additional *noise* will be introduced to the covert messages. Therefore, it is not sufficient to simply send 0 or 1 HTTP request as discussed before. Instead, we set a high enough threshold Q^* for determining whether the increase in the counter value is due to a covert message. The choice of Q^* obviously depends on the Webpage's popularity, and we will elaborate the impact of Q^* on the decoding accuracy in section 3.

We adopt the following notations for explaining the WebShare encoding and decoding algorithms when taking into account of the channel noise.

- VC_i: the bit (`0` or `1`) sent during the ith T_E.
- VW_i: the Web counter value at the end of the ith T_E.
- VE_i: the number of HTTP requests sent for VC_i.

To transmit bit `0` during the ith T_E, the encoder does not send any HTTP requests, i.e., $VE_i = 0$. To transmit bit `1`, the encoder sends a number of contiguous HTTP requests, such that $VW_i - VW_{i-1} \geq Q^*$. To account for possible request losses, it is prudent to set $VE_i > Q^*$. At the end of the ith T_D, VC_i is decoded to bit `0` if $VW_i - VW_{i-1} < Q^*$, and to bit `1`, otherwise. Figure 1 shows the alternating pattern of T_E and T_D, and the change in VW_i when both sides are perfectly synchronized in time.

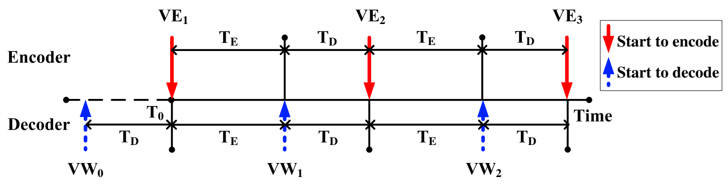


Fig. 1. The alternate encoding and decoding periods in WebShare when the encoder and decoder are perfectly synchronized in time.

To conduct a more formal analysis, let $\mu = \frac{VE_i}{T_E}$ be the encoder's average request sending rate for encoding bit `1` and λ be other visitors' average request arrival rate. Then the following inequalities must be satisfied in order to decode correctly.

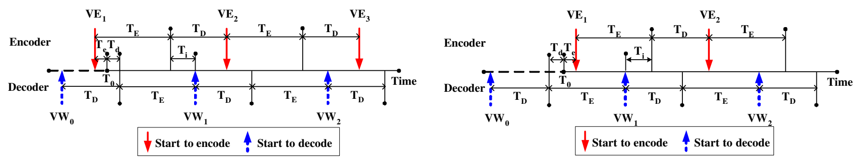
$$(T_E + T_D)\lambda < Q^*. \tag{1}$$

$$(T_E + T_D)\lambda + T_E(1 - P_{loss})\mu \geq Q^*, \tag{2}$$

where P_{loss} is the probability of a request loss. Note that the encoder could dispatch all requests at the beginning of T_E or send them out with the constant rate μ.

2.2 Mitigating the effect of time desynchronization

In this subsection we show that WebShare requires only loose time synchronization which is made possible by tuning the values of T_E, T_D, and Q^*. We first let T_e (or T_d) denote the difference between the encoder's (or decoder's) local time and the standard time, and $T_i = |T_e - T_d|$. In order not to affect the next run's decoding, T_i should be less than $\min\{T_E, T_D\}$. The special case of $T_e = T_d = 0$ is therefore the same as the case of perfect synchronization. In the following, we discuss two extreme cases when $T_i = |T_e| + |T_d|$ which serves as the upper bound for the time difference between the encoder and decoder. The same result also applies to other cases, i.e. $T_i < |T_e| + |T_d|$.



(a) $T_e < 0 < T_d$. (b) $T_d < 0 < T_e$.

Fig. 2. Two time desynchronization scenarios for WebShare.

Figure 2(a) depicts one of the desynchronization scenarios where $T_e < 0 < T_d$. As shown, the encoder starts earlier than what the decoder expects. We can also observe from the figure that each T_E is sandwiched between two consecutive decoding epoches. For example, although VE_1 requests are dispatched before $T_0 + T_d$, its effect is still registered by $VW_1 - VW_0$. Hence, it is not difficult to see that if Eqs. (1)-(2) hold, the decoding can still be done correctly.

Figure 2(b) shows an opposite scenario: $T_d < 0 < T_e$. As a result, the decoder might interpret part of the HTTP requests from the previous run as the current run's when the requests are sent out with a constant rate μ. To decode correctly, we need to consider all four possible cases for any two adjacent bits: {`0 0`}, {`1 1`}, {`0 1`}, and {`1 0`}. The same analysis can be done for the first bit in a covert message as if it were preceded by a `0`.

For the case of {`0 0`}, since the encoder does not dispatch HTTP requests, the decoder could correctly extract the two bits if Eq. (1) is satisfied. For the case of {`0 1`}, Eq. (3) must also be fulfilled to decode the bit `1` correctly:

$$(T_E - T_i)(1 - P_{loss})\mu + (T_E + T_D)\lambda \geq Q^*. \tag{3}$$

For the cases of {`1 1`} and {`1 0`}, note that the first bit `1` could be correctly decoded if Eq. (3) is fulfilled. The requirements, however, are different for the second bits. For the case of {`1 1`}, VE_i is given by the HTTP requests leftover from the previous run and a portion of the HTTP requests from the current run. Hence, the decoder could extract the correct value if Eq. (2) is satisfied. For the case of {`1 0`}, Eq. (4) must be fulfilled:

$$T_i(1 - P_{loss})\mu + (T_E + T_D)\lambda < Q^*. \tag{4}$$

The remaining issue is to mitigate the adverse impact of P_{loss}, λ, and T_i on the channel quality. First, we can estimate P_{loss} from the loss rate of the normal requests and mitigate its effect by increasing the values of μ and the related parameters, i.e. T_E and Q^*. Similarly, we can also estimate λ (see section 3 for the methodology) and alleviate its impact by increasing the value of Q^*. As for T_i, we can minimize its impact by employing the network time protocol (NTP), or by exploiting the random beacons widely available in the Internet, e.g., stock indices [12]. Hence, T_i could be made as small as 100ms. To further mitigate the impact, the encoder could transmit the HTTP requests in bursts at the beginning of T_E. In this way, HTTP requests will not show up during T_i, thus minimizing the impact of T_i for the cases of {`1 0`}, {`0 1`}, and {`1 1`}.

2.3 Increasing the bit rate

WebShare's data rate is limited by the frequency of incrementing the Web counter. We employ three approaches to improve it. The first approach is to dispatch VE_i requests for encoding `1` through VE_i parallel HTTP connections, HTTP request pipelining in a single HTTP connection, or a mixture of the two.

The second approach is to transmit multiple bits in parallel. To do so, the encoder and decoder pre-agree on a set of ordered Web counters. During each T_E, the encoder sends 1 bit of information to each Web counter. The decoder can therefore retrieve multiple bits from the set of counters in the next T_D.

The third approach is based on *multilevel quantization.* Take a uniform quantization as an example. To convey an M-bit message, where $M > 1$, we partition the increased value of the Web counter into M intervals with an interval size of Q^*. If the increased value falls into the interval of $[iQ^*, (i+1)Q^*)$, $0 \leq i < M - 1$, then the message is decoded as i. If the number is larger than or equal to $(M-1)Q^*$, then the message is decoded as $M - 1$. As a result, the encoder could deliver $\log_2 M$ bits within a run.

2.4 Site-hopping

Recall that one of the methods of increasing WebShare's data rate is to use a set of Web counters. Using a fixed set of Web counters repeatedly, however, could increase the vulnerability of being detected. To remove this static behavior, we propose to change the set of Web counters dynamically. This idea is similar to frequency hopping in the spread spectrum communication [13]; therefore, we name it as *site-hopping*. For example, the encoder and decoder can use two sets of nonoverlapping Web counters alternately.

Besides the advantage of further camouflaging WebShare, this approach in fact helps increase the channel throughput, provided that any two consecutive sets of S counters do not have any overlap. To see why, consider the previous example again. Now it is possible to parallelize the encoding and decoding operations: while the encoder is sending S bits to the first (or second) set of Web counters, the decoder can simultaneously read S bits from the second (or first) set of Web counters.

To deploy the site-hopping approach, we have to resolve two important design issues. The first is to let both the encoder and decoder agree on the same set of Web counters each time. However, because of the additional overhead, we do not prefer to use the covert channel to communicate this information. The second is to ensure that any two adjacent sets of S Web counters do not overlap, which is required for achieving the parallelism and for reducing the decoding errors.

We propose a novel, and yet simple, approach based on enumerative combinatorics [14] to resolve the two issues. Assume that both the encoder and decoder have agreed on the list of N, where $N >> S$, available Web counters and a shared secret key K_0. We partition the N Web counters into two groups with $N_1 = LS$ counters and $N_2 > S$ counters, respectively. Therefore, for a given order of N_1 counters, we could send L S-bit segments of the message using nonoverlapped sets of counters. After sending the first L segments, we could consider a different order of the N_1 counters, and perform the similar steps. However, there may be overlapping between the last set of S counters for the ith N_1-bit block and the first set of S counters for the $(i+1)$th N_1-bit block. Our solution to this problem is to use a randomly selected set of S Web counters from the second group as a separator between the two adjacent blocks. Therefore, at least S bits of information can be transmitted before revisiting a Web counter.

To agree on the same set of counters for each S-bit message, both the encoder and decoder must first agree on the exact order of the N_1 counters to use for each N_1-bit block, and there are $NC_P = N_1!$ of them. Both must also agree on the set and order of the S counters from the second group after sending each N_1-bit block; there are a total of $NC_C = \begin{pmatrix} N_2 \\ S \end{pmatrix} S! = \frac{N_2!}{(N_2-S)!}$ such sequences. From the field of enumerative combinatorics, there exist unranking algorithms for permutations [15] (or binomial coefficients [14]) that map a positive integer uniquely to each permutation (or combination). For this purpose, we have designed an algorithm to index the NC_C sequences. Therefore, if both the encoder and decoder could come up the same indices for permutations or combinations, they could use the unranking algorithms to agree on the same set of Web counters. We use the following procedures to generate the indices securely. Let Idx_P and Idx_C be the indices for the permutations and combinations, respectively. Let H_P and H_C be good hash functions that output pseudo-random values in the range of $[1, NC_P]$ and $[1, NC_C]$, respectively. The indices can be computed randomly according to the following rules:

$$Idx_{P,i-1} = H_P(K_{i-1}) \;\; and \;\; Idx_{C,i-1} = H_C(K_{i-1}),$$
$$K_i = Idx_{P,i-1} \oplus Idx_{C,i-1}, \quad i > 0.$$

3 Experiment Results

We have prototyped WebShare encoder and decoder using Perl 5.8 under Linux kernel v2.6.8. Altogether we have conducted experiments on 220 randomly selected Web counters, whose hosting servers are located in ten different geographical locations. The Web counters increase their values on receiving an HTTP request. The encoder and decoder run on different machines in our campus network; their round-trip times (RTTs) to each Web counter are therefore very similar. Moreover, to make T_i as small as possible, we use the NTP to synchronize the encoder's and decoder's clocks. We have also examined the impact of T_i on the decoding accuracy, and the results validate our arguments in section 2.

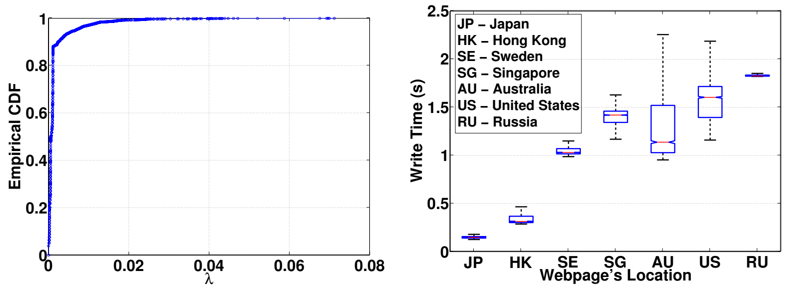
3.1 The choice of Q^*

As recalled, one of the major factors affecting the decoding accuracy is the unavoidable noise from legitimate visitors. This factor directly affects the choice of Q^*, which determines whether the counter increment should be accepted as bit 1. To obtain a suitable value of Q^* for our target Webpages, we have performed a one-week measurement to study their popularity. During the measurement period, a node at our campus network queried the counter values of the 220 Webpages every hour. Figure 3(a) shows the empirical CDF (ECDF) of the average rate of requesting the Web counters, denoted by λ requests/second. Over 95% of the measured Web counters have their λs smaller than 0.01, while the λs for all the Web counters are no greater than 0.08. Thus, it is reasonable to choose $Q^* = 2$ to mitigate the noise-induced errors.

3.2 An evaluation of WebShare

Distribution of Web counters' write times We report the distributions of seven Web counters' *write times*, and we select these Web counters randomly from the original set. The write time is defined as the duration between a node's transmission of a TCP SYN packet (for initiating an HTTP connection) to a Web server and its reception of the counter's value from the server's responses. The write time could affect the channel accuracy, and the choices of T_E and T_D.

We measure the write times for the seven Web counters and obtain a total of 2,880 samples; each measurement is conducted with a single HTTP request. Figure 3(b) shows the box-and-whisker plot of the write time measurements, and we identify the Web counters by their geographical locations. Each box includes the lower quartile, median, and upper quartile values of the write time samples. The whiskers extended from the box represent 1.5 interquartile range of the samples. We have summarized the statistics and their respective hop counts in Table 1. Although the mean write time for each Web counter is smaller than 2s, the measured write times could range from 0.120s to 82.684s. Moreover, the variations of the write times for some Web counters, such as those in SG, AU, and US, are much larger than others. As shall see shortly, the write time variations can adversely affect Webshare's accuracy.



(a) ECDF of legitimate visitors' average arrival rates (λ).

(b) Distribution of the seven Web counters' write times.

Fig. 3. Some measured characteristics of the Web counters under consideration.

Choices of T_E and T_D We have measured WebShare's performance under various configurations of T_E and T_D ($T_E, T_D \in \{0.25, 0.50, 1.00\}$s) using the seven Web counters. For these experiments, we set $Q^* = 2$, and the encoder conveys a random 16-bit message to the decoder. Moreover, the encoder employs one Web server for each measurement, and sends $VE_i = 3$ requests in parallel in order to mitigate the error due to encoding delays and request losses. For each configuration of (T_E,T_D), we measure the raw bit error rate (BER) on the decoder side for 30 times, in terms of the Hamming distance. The acceptable BER depends on various factors, such as the forward error correction code in use and the application requirement.

Table 1. Measured BERs for seven Web counters under different (T_E, T_D). The four values under the column of "Write Time" represent the lower limit of and the upper limit of the 95% confidence intervals for the sample means, the sample means, and the standard deviation, respectively.

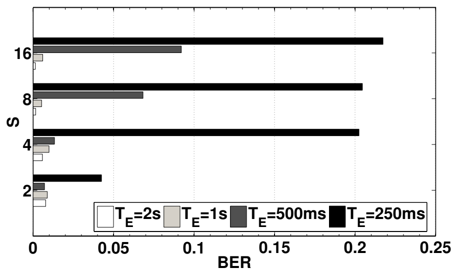
Locations	Hops	Write Time (s)	T_E with $T_D = 1$s		T_D with $T_E = 1$s		$T_E = T_D$
			250ms	500ms	250ms	500ms	= 1s
JP	16	.1695/.1928/.1811/.3192	.0479	.0146	.4708	.4667	0
HK	14	.7353/.7935/.7644/.9876	.0750	.0167	.4708	.4625	.0063
SE	17	1.042/1.0582/1.0501/.2211	0	.0125	.4688	.4604	.0208
SG	16	1.5631/1.6352/1.5991/.9870	.1667	.0958	.4646	.4562	.1146
AU	23	1.6296/1.8069/1.7182/2.4280	.3083	.2875	.4813	.4250	.2604
US	18	1.6632/1.7743/1.7188/1.5214	.1500	.0333	.4729	.4292	.0729
RU	15	1.8297/1.8496/1.8397/.2713	.3979	.3521	.4604	.4396	.0531

As shown in Table 1, the channel accuracy depends greatly on the Webpages' write times. Notice that when $T_E = T_D = 1s$, WebShare performs very well with the BERs of less than 3% for some locations, such as JP, HK, and SE. We have verified that the errors are mostly due to the background legitimate requests and dropping of the encoder's and decoder's requests. However, Webshare shows poorer performance for some other locations, especially for AU, SG, and US whose write times exhibit very high variations, or whose mean write times are greater than T_E and T_D. A simple way to relieve this problem is to ensure that the values of T_E and T_D are greater than the Webpages' write times.

Besides, we observe that it is more likely to incur a higher BER for $T_D < T_E$; however, a small T_E generally has less impact on the channel performance. We conjecture that the errors may be due to the interference from the encoder's next counter update. Depending on the design of the Web counter and the Web server's program design, the server may not produce the HTTP response immediately after the counter update. On the other hand, even if T_E is small, the Web server can still produce the response for the decoder's request based on the current counter value, as long as T_D is long enough and no later request interferes the current value. Thus, it's prudent to assign a longer T_D in order to increase the decoding accuracy.

Performance gain of the site-hopping approach As discussed in Section 2.4, the site-hopping approach helps enhance both the throughput and the channel accuracy. To measure the performance gain obtained by the site-hopping approach, we adopt the same experiment settings as the last section: $Q^* = 2$ and $VE_i = 3$, and the encoder conveys a random 16-bit message to the decoder. To avoid overlapping between two adjacent sets of Web counters, the encoder and decoder agree on two distinct sets of randomly selected $S \in \{2, 4, 8, 16\}$ Web counters from the original 220 Web counters. The encoder issues HTTP requests to the S Web counters concurrently during each T_E.

Figure 4 shows the measured BERs for $T_E \in \{0.25, 0.50, 1.00, 2.00\}$s and S. As the results show, the site-hopping approach can improve the WebShare channel accuracy. For example, when $T_E \geq 1$s, all measured BERs are no greater than 1%. Even when $T_E = 250$ms, the channel's BER with $S = 2$ can stay below 5%. These results confirm that site-hopping can effectively mitigate the interference from the encoder's next counter update. Furthermore, since the encoder transmits at most S bits in parallel, and both the encoding and decoding processes are conducted in parallel, we expect that the site-hopping approach can improve the channel throughput by a factor of $[(T_E + T_D)S]/T_E$. Our experiment results are indeed close to the expected results. For instance, when $T_E = 250$ms and $T_D = 1$s, Webshare with site-hopping achieves a throughput of 57.816 bits/s, whereas that without site-hopping is only 0.789 bits/s.



S	T_E			
	250ms	500ms	1s	2s
2	.0425	.0071	.0088	.0078
4	.2025	.0133	.0099	.0058
8	.2046	.0683	.0053	.0016
16	.2174	.0921	.0060	.0014

Fig. 4. Measured BERs for WebShare with site-hopping, $Q^* = 2$, and $VE_i = 3$ for various values of S and T_E.

4 Related Works

There have been quite a number of TCP/IP-based network storage channels proposed for the past few years. In the network layer, many methods have been proposed to hide data in the IP packets and ICMP packets. Virtually all possible fields in the IP headers have been exploited for storage covert channels [9, 16, 17, 18]. The fields in the TCP header are also equally exploited for embedding storage covert channels [9, 19, 20, 17]. On the application layer, HTTP not only has been used as a substrate for tunneling other protocols [21, ?], but also utilized to implement covert channels [1, 22, 2, 23], some of which have been deployed to facilitate anonymous communications [1, 23].

On the defense side, neural network and support vector machines have been adopted to detect storage covert channels based on the ISN of TCP flows [24, 25, 17]. Moreover, statistical approaches have been proposed to detect covert channels over HTTP [26, 2]. Besides detection, another approach is to neutralize covert channels by performing active operations on the traffic.

The basic idea of frequency-hopping has also been used in the Infranet system to avoid widespread discovery and blocking [27, 23]. However, there are important differences between our site-hopping approach and the ones in [27, 23]. First, all Web counters in WebShare are essentially "victims." In other words, the encoder and decoder just need to find and use these Web counters.

However, the proxies in the Infranet system need extra cooperation and installation. Second, our pseudo-random sequence algorithm, which is based on enumerative combinatorics, not only could generate pseudo-random sequences, but also guarantee that there are no overlapping in the consecutive sets of members. The algorithm in Infranet, however, does not provide this feature.

5 Conclusions

In this paper, we have proposed WebShare, a new network storage channel using Web counters to relay covert messages. A WebShare decoder can be located anywhere to read the messages written by an encoder. We have shown that WebShare requires only loose time synchronization between the encoder and decoder. The channel data rate can also be increased by various schemes, such as using multiple counters and the site-hopping technique. Moreover, we have demonstrated its feasibility by prototyping the WebShare encoder and decoder and performed extensive experiments in the Internet. We have measured its decoding accuracy and studied the impact of different parameters on its performance.

Besides, we have conducted an information-theoretic analysis of WebShare's capacity and have proposed a new detection system designed for WebShare. However, due to the paper limit, we could not present them in this paper. We are in the process of reporting them in the forthcoming paper.

Acknowledgment

The work described in this paper was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. PolyU 5080/02E) and a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

References

1. M. Bauer. New covert channels in HTTP: Adding unwitting Web browsers to anonymity sets. In *Proc. ACM Workshop on Privacy in the Electronic Society*, 2003.
2. K. Borders and A. Prakash. Web Tap: Detecting covert Web traffic. In *Proc. ACM CCS*, 2004.
3. DoD US. Department of defense trusted computer system evaluation criteria (orange book). Technical Report DoD 5200.28-STD, National Computer Security Center, Dec. 1985.
4. V. Gligor. A guide to understanding covert channel analysis of trusted systems (light pink book). Technical Report NCSC-TG-030, National Computer Security Center, Nov. 1993.

5. E. Cronin, M. Sherr, and M. Blaze. The eavesdropper's dilemma. Technical Report MS-CIS-05-24, University of Pennsylvania, February 2006.
6. R. Kemmerer. Shared resource matrix methodology: A practical approach to indetifying covert channels. *ACM Transactions on Computer Systems*, 1(3), 1983.
7. C. Tsai and V. Gligor. A bandwidth computation model for covert storage channels and its applications. In *Proc. IEEE Symp. Security and Privacy*, 1988.
8. G. Danezis. Covert communications despite traffic data retention. http://homes.esat.kuleuven.be/∼gdanezis/cover.pdf, 2006.
9. C. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday: Peer-reviewed Journal on the Internet*, 2(5), 1997.
10. Fyodor. Idle scanning and related IPID games. http://www.insecure.org/nmap/idlescan.html.
11. F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proc. IEEE Symp. Security and Privacy*, 2002.
12. H. Lee, E. Chang, and M. Chan. Pervasive random beacon in the Internet for covert coordination. In *Proc. Information Hiding Workshop*, 2005.
13. M. Simon, J. Omura, R. Scholtz, and B. Levitt. *Spread Spectrum Communications Handbook*. McGraw-Hill, 2002.
14. D. Kreher and D. Stinson. *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC press, 1998.
15. W. Myrvold and F. Ruskey. Ranking and unranking permutations in linear time. *Information Processing Letters*, 79:281–284, 2001.
16. K. Ahsan and D. Kundur. Practical data hiding in TCP/IP. In *Proc. Workshop on Multimedia Security*, 2002.
17. S. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In *Proc. Information Hiding Workshop*, 2005.
18. C. Abad. IP checksum covert channels and selected hash collision. http://www.gray-world.net/papers/ipccc.pdf, 2001.
19. J. Giffen, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging through TCP timestamps. In *Proc. PET Workshop*, 2002.
20. J. Rutkowska. The implementation of passive covert channels in the Linux kernel. In *Proc. Chaos Communication Congress*, 2004.
21. K. Moore. On the use of HTTP as a substrate. RFC 3205, Feb. 2002.
22. Gray-World Team. Covert channel and tunneling over the HTTP protocol detection: GW implementation theoretical design. http://www.gray-world.net/projects/papers/cctde.txt, 2003.
23. N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting Web cenorship with untrusted messenger discovery. In *Proc. PET Workshop*, 2003.
24. J. Seo T. Sohn and J.Moon. A study on the covert channel detection of TCP/IP header using support vector machine. In *Proc. ICICS*, 2003.
25. E. Tumoian and M. Anikeev. Network based detection of passive covert channels in TCP/IP. In *Proc. IEEE LCN*, 2005.
26. D. Pack, W. Streilein, S. Webster, and R. Cunningham. Detecting HTTP tunneling activities. In *Proc. IEEE Annual Information Assurance Workshop*, 2002.
27. N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing censorship and surveillance. In *Proc. USENIX Security Symp.*, 2002.