ProtoLeaks: A Reliable and Protocol-Independent Network Covert Channel

Arne Swinnen, Raoul Strackx, Pieter Philippaerts, and Frank Piessens

Dept. of Computer Science, University of Leuven

Abstract. We propose a theoretical framework for a network covert channel based on enumerative combinatorics. It offers protocol independence and avoids detection by using a mimicry defense. Using a network monitoring phase, traffic is analyzed to detect which application-layer protocols are allowed through the firewalls. Using these results, a covert channel is built based on permutations of benign network objects, such as FTP commands and HTTP requests to different web servers. Any protocol that offers reliability guarantees can be plugged into the framework. This includes any protocol that is built on top of the TCP protocol. The framework closely mimics the behavioral statistics of the legitimate traffic, making the covert channel very hard to detect.

Keywords: timing channel, ordered channel, adaptive covert communication

1 Introduction

Protection of private data on a corporate network is challenging due to the large amount of data that is typically involved, the number of systems on the network that are potentially compromised by malware, and users that are often security-unaware. As a last line of defense, intrusion detection systems (IDSs) and firewalls monitor outgoing network connections to prevent sensitive information from leaking out of the network. Usage of unknown protocols or other suspicious behavior patterns can be detected. Covert channels, however, can also be built using benign channels.

Traditionally, network covert channels are classified into storage and timing channels [1], where most existing covert channels fall into the former category. *Storage channels* attempt to hide covert data inside header or footer fields of specific protocols or within payload fields of messages themselves [2]. In general, a high capacity can be achieved by this method, but once these channels are documented a network administrator is able to locate them easily and take appropriate countermeasures by means of content-based detection schemes [3]. Classic *timing channels* hide information by modifying timing mechanisms. In a networked environment this is often realized by varying packet rates [4] or changing inter-packet delays [5]. Active channels introduce unseen traffic, as opposed to passive channels which only alter timings of existing packets. Detecting

timing channels is hard since only anomaly-based detection schemes are applicable, which can be bypassed by mimicking monitored legitimate network traffic properties. The major disadvantage of timing channels is the low throughput and the common need for absolute time synchronization between sender and receiver, which complicates practical implementations. They also are often not immune to dynamic network conditions such as packet duplication, packet loss or noise.

Recently a new family of network timing covert channels was presented based on enumerative combinatorics [6]. This combinatorial approach exploits the relationship of network objects. The absolute time synchronization constraint between sender and receiver is relaxed to relative time synchronization while higher capacity compared to classic timing channels can be achieved. The method is based on two fundamental properties which affect inter-relationship among network objects: *distinguishability* and *sequence detectability*. Distinguishability means that one network object can be differentiated from one another, whereas sequence detectability implies whether the order of a sequence of network objects can be discriminated. These properties determine the number of unique arrangements which can be constructed from the pool of network objects. Once these arrangements are known, a mapping between arrangements and bits is constructed. These encoding and decoding algorithms are based on functions which map combinations to positive integers and vice versa, which are known as *ranking* and *unranking* functions. This new technique is very suited for cases where the sequence of the chosen network objects is inherently subject to variation, for example human browsing behavior [6] or the packet reordering phenomenon in the internet [7].

The covert channel presented in this paper exploits the fact that different reliable protocol packets or sessions are not always sent in the exact same sequence over time. A human being does not always execute his actions in the same order, nor do daemon applications execute in predefined sequences. It is the first network covert channel, as far as we know, that offers protocol independence. *ProtoLeaks* is the first ordered channel that makes the distinction between network objects and network object instances. Furthermore, it utilizes a new encoding method based on permutations of network objects with repetitions, which achieves higher capacity than methods presented in related work. In addition, a known encoding method based on permutations of network objects without repetitions is thoroughly analyzed, which results in the identification of some desirable properties for mimicking legitimate traffic. This is achieved through a brand new mimicking algorithm based on a machine learning clustering technique. Non-overlapping clusters of legitimate transmit times are constructed to represent a model of legitimate traffic, after which the second encoding scheme can be exploited to generate traffic according to these clusters.

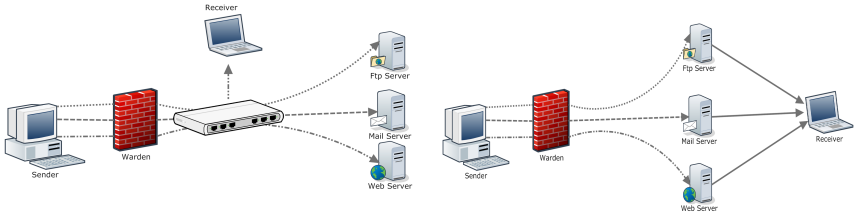
The remainder of this paper is structured as follows. In Section 2 possible threat models are described. An outline of the channel design is given in Section 3. Section 4 discusses how legitimate traffic can be mimicked. Experimental

results are presented in Section 5. Section 6 highlights related work and Section 7 concludes this paper.

2 Threat Model

ProtoLeaks encodes covert data in sequences of network object instances containing different protocol payloads or having distinct destinations. Therefore, the receiver must be able to notice all these instances inserted into the network by the sender in the same order. Figure 1 depicts two plausible locations of sender and receiver. In the first scenario (Fig. 1(a)) the receiver is able to monitor all packets traveling between the distinct servers and the sender by eavesdropping on a common routing path. This can be achieved in practice by sniffing network traffic. In the second scenario (Fig. 1(b)) the receiver controls all the distinct servers. They report every arrival of a packet to the receiver before acknowledging the arrival of the packet to the sender, to make sure the order of arrival is maintained at the receiver's end. This second scenario is the most likely scenario to be used in practice, because of the impracticalities of network sniffing on larger networks. However, in the first scenario the distinct servers that are used must not be owned by the receiver but can be chosen freely, which facilitates deployment.

We assume the presence of a warden (i.e. an intrusion detection system) that resides on the network and guards against any network covert channels. The warden is active and stateful, which implies he can inspect and modify observed packets and remember previously seen packets as well. He may also block protocols based on the time of day e.g. sending emails after midnight may not be permitted.



(a) Receiver eavesdrops on common routing path

(b) Servers report packet receipt to receiver

Fig. 1. Two possible communication scenarios between sender and receiver

3 Channel Design

Given the threat model of the previous section, we now describe our combinatorial approach. In Section 4 we will show how this approach can be combined with

a network monitoring phase to mimic legitimate traffic to circumvent a stateful active warden.

3.1 Combinatorial Approach

Pairs of reliable protocols and server addresses are used as network objects o in *ProtoLeaks*. Packets or sessions are possible examples of network object instances o_i, which can be seen as realizations of network objects on the wire. They are the building blocks for the covert channel described in this paper. These network objects as well as their possible instances satisfy both the fundamental properties of distinguishability and sequence detectability. They are distinguishable since one can always detect and compare the reliable protocol carried by two instances by analyzing their payload. Network protocol analyzers such as Wireshark[1] already offer this functionality. Because only reliable protocols are considered, sequence detectability is guaranteed when one instance is sent only after the previous instance has arrived its destination successfully.

It is important to stress that *ProtoLeaks* does not constrain the payload of a protocol-compliant packet of a network object instance in any way. It only deals with permutations of network objects. For example, when considering the FTP protocol, a USER command and a PASS command sent to one specific FTP server at address x are both fine examples of instances of network object o = *(FTP, Ftp Server Address x)*. In fact, if one has an FTP protocol-specific covert channel to his disposal, it could be embedded in *ProtoLeaks* trivially. However, to stay undetected, it is advised to issue commands fully compliant with the considered reliable protocol. Mimicking typical protocol-specific scenarios when a stateful protocol is chosen, is also required to avoid detection by the warden. By obeying these guidelines, a warden cannot raise suspicion based on the contents of the command itself, nor on the logical flow of commands. In the case of the FTP protocol, a typical login scenario requires that USER and PASS commands are sent consecutively to the FTP server x via the same TCP connection. When sessions were chosen as network object instances, a complete login scenario is a perfect candidate for a network object instance o_i of the aforementioned network object o. When packets were chosen as network object instances, these commands can be interpreted as two network object instances o_u and o_p. The sender must keep the TCP connection open after sending o_u. When a new instance of object o has to be sent and the TCP connection is still available, o_p should be sent to complete the login scenario. If the TCP connection is not available any more, the scenario should be restarted from scratch on a freshly established connection. The network object instance o_u is then sent again.

The advantage of session network object instances is that connections do not have to remain available between consecutive sends of instances of the same network object. For every new object instance, a whole session is completed. The major downside of this type of instance is loss of capacity compared to packet

[1] `http://www.wireshark.org`

object instances, since a complete session will obviously use more bandwidth than a single packet.

3.2 Encoding and Decoding

The basic encoding and decoding process is depicted in Fig. 2. The sender first reads β bits from its covert data stream. Subsequently, this bit stream is transformed to a decimal number and handed over to the unranking function. This function always takes a decimal number and an ordered list of n distinguishable and sequence detectable network objects, which is agreed upon by both parties beforehand, and spits out a new sequence of these objects of length l, representing the provided decimal number. This permutation of network objects is given to the sender's network component which is responsible for sending appropriate instances of them in the given order. As packets are only send after reception of the previous package was acknowledged, these l object instances are observed in the same order at the receiver's end. The receiver translates this permutation of network object instances to a permutation of network objects again and provides this permutation to the ranking function. This function takes this permutation of length l and the same ordered list of n distinguishable and sequence detectable objects and returns a decimal number. Finally, this decimal number is converted back to a bitstream of β bits and added to the received covert data stream.

Recall that a network object in *ProtoLeaks* is a unique pair of a reliable protocol and a server address. A network object instance is packet or session compliant with the object's protocol sent to the object's server address. Note that symbols β and l as well as the ranking and unranking functions are specific to the chosen permutation flavor. The number of bits β directly depends on the total number of permutations α and is given by the following equation:

$$\beta = \lfloor \log_2 \alpha \rfloor \tag{1}$$

In the remainder of this section, two practical encoding schemes based on distinct flavors of permutations are presented and compared. The power encoding scheme allows repetitions of network objects and is aimed at high capacity. The factorial encoding scheme does not allow repetitions but exhibits useful properties to mimic legitimate traffic.

The Power Encoding Scheme In this encoding scheme, permutations of network objects containing duplicates are employed. Allowing repetitions in permutations of network objects means that one permutation of network objects may contain multiple entries of one network object o. This implies that multiple network objects instances of that same network object o may be scheduled consecutively. Since the actual payload of the protocol-compliant instances may differ, this will not raise suspicion of wardens when the payload is chosen carefully. The total number of permutations with repetitions α is given by the following equation:

$$\alpha = n^l \tag{2}$$

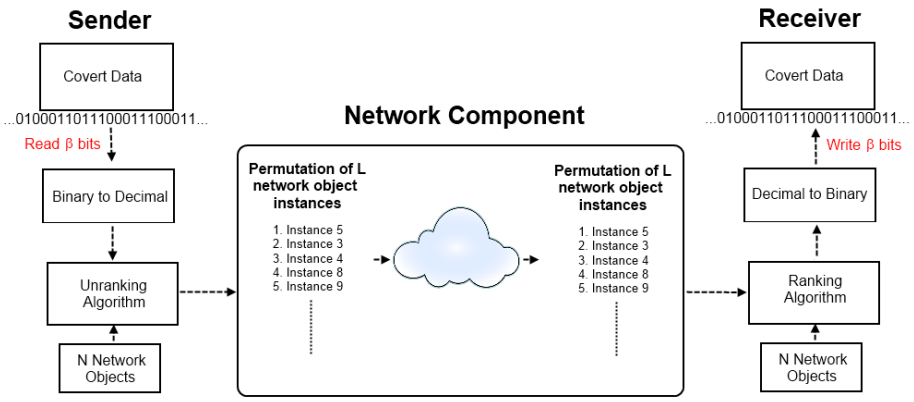


Fig. 2. Basic encoding - decoding process

The symbol n represents the number of available network objects. The symbol l indicates the length of the permutations and is free to choose. When n is not a power of two, choosing a greater value for l increases the maximum number of bits β that can be represented by one permutation, because of the binary logarithm in (1). This number of bits can be calculated by substituting (2) in (1).

No ranking and unranking functions for permutations with repetitions were located in existing literature. We found that constructing a bijective function between permutation with repetitions and binary numbers is possible when interpreting a permutation as a number of base n. Each of the network objects is first mapped to a positive integer in the range $[0..n-1]$. This mapping is known by sender and receiver. A permutation is then transformed to a number of base n by replacing every network object in the permutation by its corresponding value in this mapping. Translation between a permutation and β bits can then be reduced to a case of base conversion between numbers of base n and base 2.

The Factorial Encoding Scheme The number of permutations without repetitions α is given by the following equation:

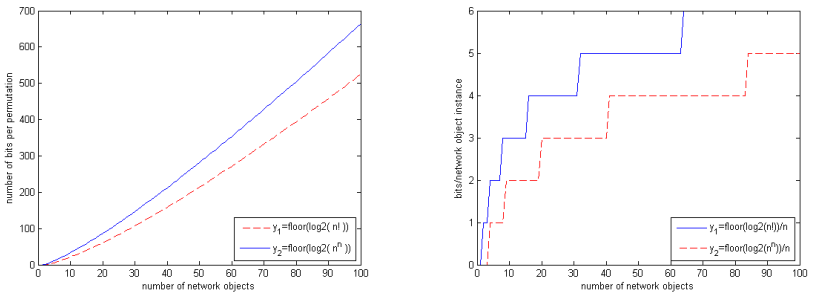
$$\alpha = n! \tag{3}$$

The symbol n still represents the number of available network objects. The length of one permutation is always equal to n. The maximum number of bits β that can be represented by one permutation can again be calculated by substituting (3) in (1). Ranking and unranking functions for permutations without repetitions are readily available in existing literature. An algorithm for ranking and unranking in linear time was given by Myrvold [8].

Comparison of encoding schemes The capacity of both encoding schemes in terms of *bits/permutation* and *bits/network object instance* is given in Fig. 3(a) and Fig. 3(b) for $n \in [0..100]$. In (2), a length $l = n$ was chosen in order to compare permutations of the same length from both schemes. It is clear that the power encoding scheme achieves higher capacity. This is due to the employment of permutation with repetitions in this scheme.

However, the factorial encoding scheme exhibits some other desirable properties useful for mimicking legitimate traffic. First, it guarantees that the same amount of instances from every available network object are generated during the covert communication. This is because each permutation contains exactly one instance of each available network object. In this way, the sender has more control over the generated traffic. In the power encoding scheme, the amount of generated network object instances of one network object depends on the distribution of the covert data. Only when the data is distributed uniformly, the same amount of network objects instances will be generated over time, which is an assumption that cannot be made in general.

Second, a trivial command channel between sender and receiver can be constructed. Since permutations with repetitions of network object instances are never generated for covert data transmission by the factorial encoding scheme, these kind of permutations can be used to signal commands from sender to receiver instead. In this way, permutations without repetitions are reserved for the covert data channel and all other permutations are available to the sender to issue commands to the receiver, effectively constructing a unidirectional command channel. The number of available permutations for the command channel is given by $n^n - n!$ and is visible for varying values of n as the distance between both functions in Fig. 3(a).



(a) Capacity in bits / permutation (b) Capacity in bits / instance

Fig. 3. Comparison of capacity of presented encoding schemes

4 Mimicking Legitimate Traffic

Because *ProtoLeaks* encodes data in sequences of packet or session instances but does not enforce anything on the contents of the chosen payloads of these instances, it can trivially bypass content-based detection schemes by carefully construction these payloads. However, anomaly-based detection schemes can still notice peculiarities in generated outgoing traffic by the covert channel. Therefore, mimicking legitimate traffic properties is a must to stay undetected. This is achieved by the mimicking algorithm described in this section.

The algorithm expects access to a log file containing legitimate traffic of one day of the week to perform analysis on. The choice for granularity of one weekday was made based on the findings of Danzig in [11]. He noticed that protocol traffic distributions often differ even between days of the week. This log file can be obtained by the sender in a preliminary network monitoring phase. A logical scenario would be that the sender first sniffs outgoing legitimate traffic for at least a week before moving on to the communication phase. During the communication phase, he keeps on sniffing outgoing legitimate data to obtain recent logs. Logs of multiple weeks could also be combined to obtain a better fingerprint of overall network traffic for longer periods.

The algorithm also assumes the availability of an ordered collection of network objects O. This collection is divided in ordered subcollections O_p based on the reliable protocol p employed by network object $o \in O$. O_{http} thus represents an ordered subcollection of network objects relying on the HTTP protocol.

The mimicking algorithms exhibits four desirable properties. First of all, overall increase of original legitimate traffic observed in the log file due to generated traffic of the covert channel can be limited to a certain (strictly positive) percentage. This percentage is expected as a parameter γ by the mimicking algorithm. Limiting the overall traffic increase is vital to remain undetected for wardens monitoring this property. Second, the mimicking algorithm makes sure that the covert channel only utilizes reliable protocols which have been observed in legitimate traffic. This bypasses security policies restricting outgoing protocols. Third, ratios between these identified reliable protocols are maintained. For example, if there is twice as much HTTP traffic than FTP traffic present in the legitimate traffic log file during a certain time period, the mimicking algorithm will generate two HTTP requests for every FTP request. In combination with the overall traffic increase limitation, this effectively limits traffic increase of each identified reliable protocol to the same percentage. Fourth, legitimate traffic transmit times are mimicked by the algorithm. In a typical company network, the most significant portion of traffic is observed during daytimes. Generated traffic not exhibiting this behavior could be detected trivially.

The algorithm is based on a combination of a clustering technique and the factorial encoding scheme described in Sect. 3. It consists of four main steps, which are outlined in the following subsections.

4.1 Protocol Identification and Clustering

During the first step, reliable network protocols are identified in the legitimate traffic log file. For each of these protocols, outgoing network object instance transmit times are extracted and the *k-means* clustering algorithm[9] is applied to this data. Each identified cluster represents a time period, obtained by taking the earliest and last outgoing instance transmit time in the cluster. The boundaries between the identified clusters are interpreted by the mimicking algorithm as moments in time when the shape and/or regularity of legitimate traffic of the considered reliable protocol substantially changes. The identification of clusters is a way to model the dynamic behavior of legitimate traffic.

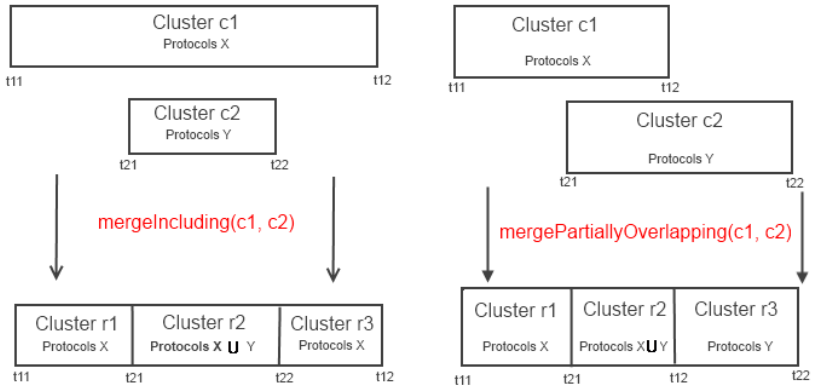
The standard k-means clustering algorithm expects a parameter indicating how many clusters should be identified. Since we don't know this parameter in advance but on the contrary are interested in this value, a cluster validation criterion is used. The k-means algorithm is executed for varying values of the parameter indicating the desired number of clusters, and each result is validated by calculating its average *silhouette* value [10]. The silhouette value of a member of a cluster is the distance from this member to the cluster's center. The clustering result with the smallest average silhouette value for all clusters is chosen as the optimal clustering for the considered protocol by the mimicking algorithm.

Finally, for each identified cluster c of protocol p, the observed outgoing legitimate instances $C_c[p]$ during the time period described by the cluster are counted. Subsequently, this value is divided by 100 and multiplied by the traffic percentage limit parameter γ to obtain $M_c[p] = \lfloor C_c[p] * \frac{\gamma}{100} \rfloor$, the maximum number of instances that can be generated in the time period described by c. This value is necessary in the next step of the algorithm.

4.2 Transformation to Temporal Clusters

In this step, the collection of clusters for each reliable protocol identified in the previous step is transformed into a new collection of temporal clusters. In the resulting collection, each cluster contains a list of reliable protocols observed throughout the time period the cluster describes.

There are two situations in which two clusters may overlap in time. These situations are handled by subroutines *mergeIncluding* and *mergePartiallyOverlapping*, which are explained graphically in Fig. 4. These routines take two overlapping clusters and return a collection of three non-overlapping clusters, each containing an appropriate protocol list P. However, a useless cluster can be introduced by these subroutines. An useless cluster c is a cluster whose maximum number of network object instances $\sigma_c = \sum_{p \in P_c} M_c[p]$ that can be generated in the time period described by c is equal to zero. Such a cluster does not signal a significant change of shape or regularity in legitimate traffic. To filter out these clusters, σ_c is calculated for every new cluster that is introduced in these subroutines. When $\sigma_c = 0$, the cluster c is discarded.



(a) mergeIncluding(c1, c2) (b) mergePartiallyOverlapping(c1, c2)

Fig. 4. Functions handling the merging of two overlapping clusters

4.3 Permutation scheduling

In this step a permutation schedule is composed. The main idea is to use the data channel of the factorial encoding scheme to send covert data during time periods described by clusters and to use the command channel of the scheme to signal transitions between consecutive clusters. Recall that boundaries between clusters are moments in time when the shape and/or regularity of legitimate traffic leaving the sender's network substantially change. A transition to a new cluster indicates a change of observed reliable protocols P and/or a significant in- or decrease of the maximum number of instances $M[p]$ of a protocol p.

For each cluster c, a distinct list of network objects O_c to build permutations from is assembled. This list is based on the ratios between the available protocols P_c in this cluster. First, $nbPerm = \max_{p \in P_c}(\lceil \frac{M_c[p]}{O_p} \rceil)$ is calculated. Recall that O_p represents an ordered subcollection of all available network objects O based on the reliable protocol p. Then, ratios of protocols in the cluster are defined as $R_c[p] = \lfloor \frac{M_c[p]}{nbPerm} \rfloor, \forall p \in P_c$. Now, for every protocol $p \in P_c$, the first $R_c[p]$ network objects of O_p are taken. The collection of all these chosen network objects is called O_c. The ratios between network objects based on distinct reliable protocols p in this collection are equal to the ratios of network object instances observed in legitimate traffic.

The schedule is now created as follows. For every cluster c a number of permutations based on O_c representing covert data are generated. The number of permutations is equal to $nbPerm$. Since each cluster c utilizes a different collection O_c, transition to a new cluster must be signaled to the receiver. This is done by exploiting the command channel available in the encoding scheme. This transition command is represented by a permutation containing repetitions. The smallest possible permutation containing two repeated network objects of O_c is chosen, to limit the traffic overhead.

4.4 Packet scheduling

In this final step, the permutations are effectively transmitted over the network. The clusters are sorted according to start time and for each cluster c the total number of network instance objects to be transmitted are calculated. This number is equal to the number of scheduled permutations $nbPerm$ in this cluster multiplied by the size of O_c, the length of one permutation. Two more instances are added to this number, to account for the transition command permutation.

In order to not introduce any regularity in generated traffic, transmit times for network objects instances of permutations are generated based on legitimate instance transmit times. First, an inverse cumulative distribution function is fitted on these transmit times observed in the time period represented by the cluster. Hereafter, the *inverse transformation method* [11] is used to generate a number of instance transmit times from this distribution. After instance transmit times for all clusters have been generated, the actual network object instances are sent over the wire chronologically according to these transmit times.

On the receiver's side, there are two states. In the initial state, the network object list O_c utilized by the currently active cluster c at the sender's side is discovered. The receiver can detect this list by exploiting the property that a permutation in the factorial encoding scheme does not contain repetitions. When a network object instance is observed that has been seen before, the receiver knows the second permutation is started and can deduce O_c from the first permutation that has been received. This permutation still represents covert data, so no bandwidth is lost. At this point the receiver enters its second state, pure covert data decoding. This state is left when a permutation of O_c containing repetitions is observed. This is the transition command from the sender. The receiver then goes back to its initial state.

5 Evaluation

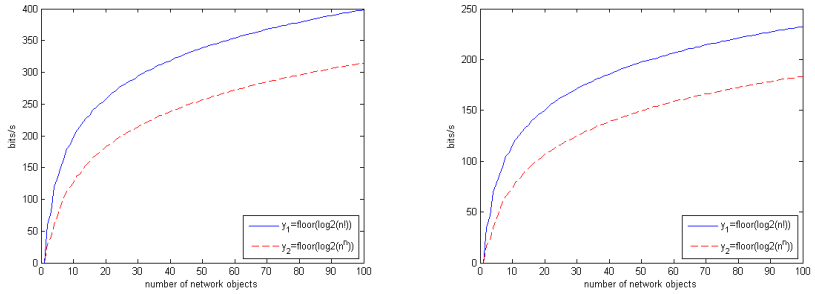
The sender's encoder and the receiver's decoder were implemented in C++ on a Windows 7 machine having Matlab and Wireshark installed. They support five protocols, namely HTTP, FTP, IMAP, POP and SMTP. Packets as well as sessions can be chosen as network object instances. In both cases, the encoder executes a typical scenario for the considered protocol of the network object. For ease of evaluation, the threat model where the receiver is able to obtain outgoing traffic logs from the sender was chosen (see Fig. 1(a)). The decoder thus expects a traffic log file of the sender's generated traffic, which can be delivered offline without loss of generality, since the channel is unidirectional. Both encoder and decoder implementations are able to call and retrieve output from the command-line version of Wireshark, Tshark, to perform traffic analysis. The encoder is also able to call Matlab routines such as k-means and data fitting functions through the *C++ Matlab Engine*[2]. Therefore, it can execute the mimicking algorithm described in Sect. 4 automatically. The network objects utilized were taken from

[2] `http://www.mathworks.nl/help/techdoc/matlab_external/f29148.html`

publicly available sources. Http servers were taken from Alexa[3]. Ftp servers were taken from several public lists of Linux distro mirror servers. A number of accounts from email services supporting IMAP, POP and SMTP were used.

5.1 Capacity

To determine the real world capacity of both schemes, the average rate at which network objects instances can be sent and acknowledged over the wire was identified experimentally. Varying rates were obtained through a number of repeated tests, from which average values of approximately 60 packet instances/second and 35 session instances/second were obtained. This results in the real world capacity for both encoding schemes depicted in Fig. 5 for varying number of network objects n. Again, the length l from permutations in the power encoding scheme were chosen equal to n, to simplify comparison of both schemes. We see that a capacity of 399 bits/second is reached by the power encoding scheme utilizing packet network object instances, when using one hundred network objects.



(a) Empirical capacity of *ProtoLeaks* for packet network object instances

(b) Empirical capacity of *ProtoLeaks* for session network object instances

Fig. 5. Comparison of capacity of presented encoding schemes

5.2 Stealth

To evaluate the stealthiness of *ProtoLeaks*, the mimicking algorithm was deployed to mimic legitimate traffic taken from the ISCX 2012 Intrusion Detection Evaluation Dataset [12]. This dataset contains complete and non-anonymized network logs of exactly one week and was made specifically to evaluate the effectiveness of intrusion detection systems. An extract of this dataset containing traffic of the first day generated by one of the workstations in the network was

[3] `http://www.alexa.com`

taken and handed to the encoder to mimic traffic from. Only four out of five supported protocols were present in the extract, no IMAP traffic was observed.

The algorithm was executed for varying values of γ, the legitimate traffic limit parameter. The results are depicted in Table 1 for packet instances only, due to page limitations. One can see that ratios present in original traffic are maintained in generated traffic. It's also clear that γ is taken into account. Figure 6 depicts the CDFs of protocol-specific legitimate and generated packet instance traffic. One can conclude from these graphs that protocol-specific legitimate traffic is mimicked successfully. Finally, the popular KS-test [13] and Regularity Test [14] were performed on legitimate and generated instance transmit times. They were both negative.

γ	#clusters	#bits	#total instances	#http	#ftp	#smtp	#pop
10	21	20969	3584	3441	30	83	30
20	24	44068	7166	6865	54	171	76
30	25	66623	10739	10296	77	254	112
40	25	89720	14313	13717	101	341	154
50	28	112745	17893	17144	125	417	207
60	28	135411	21455	20565	143	510	237
70	28	158421	25013	23991	167	583	272
80	28	180620	28588	27406	190	679	313
90	28	203540	32140	30826	214	753	347
100	29	226699	34421	32950	234	833	404
Total Legitimate Data			34439	32892	228	864	455

Table 1. Results of the transformClusters algorithm on packet instances for varying γ

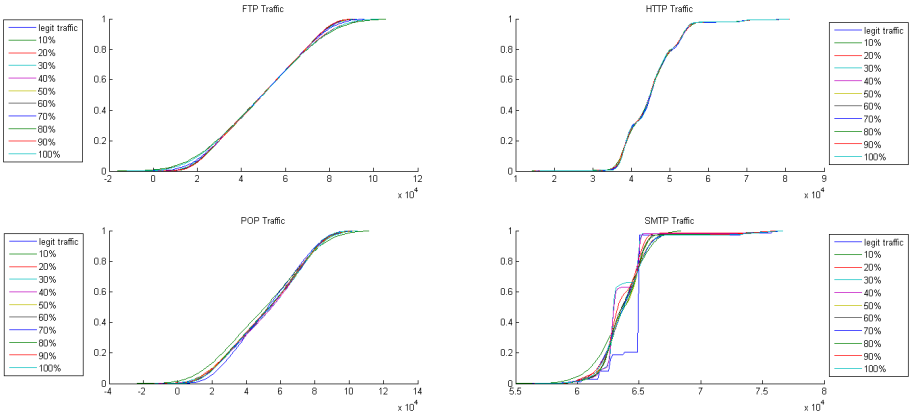


Fig. 6. Protocol-specific packet instance CDFs of legitimate and generated traffic

6 Related Work

Most of the research in the field of covert channels has been done on storage channels [2]. Only recently several practical ordered channels emerged. Cloak encodes covert data in unique distributions of packets over several TCP flows [15], which is comparable to the approach taken in [16]. It was the first channel that took advantage of the reliability service that TCP offers to construct a robust ordered channel. The packet reordering phenomenon in the internet protocol is misused to hide covert data in [7]. The dependency on the natural phenomenon and dynamic network conditions severely constrain the capacity of this covert channel. However, by mimicking the original phenomenon, it becomes undetectable. *WebLeaks* is an active ordered channel which encodes covert data in permutations of web page requests without repetitions in [6], which corresponds to the factorial encoding scheme *ProtoLeaks* also deploys. It utilizes the payload of HTTP requests as well as distinct websites to obtain a high number of arrangements and thus capacity. It is also very hard to detect, since it mimics legitimate IPDs. This makes it a perfect candidate to be embedded in *ProtoLeaks*.

7 Conclusion

This paper introduced a theoretical framework for a network covert channel based on enumerative combinatorics, called *ProtoLeaks*. It features a novel design that offers a pluggable protocol interface, essentially making the design fully protocol independent. Any protocol that offers reliability guarantees can be plugged into the framework. This includes any protocol that is built on top of the TCP protocol. Furthermore it exhibits a number of desirable properties such as reliability, better capacity than existing channels, and the possibility to embed other protocol-dependent covert channels.

In addition, a mimicking algorithm was presented which models legitimate traffic with cluster-based machine learning techniques. Covert traffic is then generated based on these clusters, which successfully evades detection by modern intrusion detection systems.

References

1. National Computer Security Center, US DoD. Trusted Computer System Evaluation Criteria. *Tech. Rep. DOD 5200.28-STD*, 1985.
2. S. Zander, G. Armitage, P. Branch. A Survey of Covert Channels and Countermeasures in Computer Network Protocols. *IEEE Communications Surveys and Tutorials*, 9(3):44–57, 2007.
3. G. Fisk et al. Eliminating Steganography in Internet Traffic with Active Wardens. *Proc. 5th Int'l. Wksp. Information Hiding*, 2002.
4. H. Eßer, F. Freiling, "Kapazitätsmessung eines verdeckten Zeitkanals über HTTP," *Tech. Rep. TR-2005-10*, 2005.

5. G. Shah, A. Molina, M. Blaze, "Keyboards and covert channels," *Proc. 15th Conf. USENIX Security Symposium*, 2006.
6. X. Luo, P. Zhou, E. W. W. Chan, R. K. C. Chang, W. Lee. A Combinatorial Approach to Network Covert Communications with Applications in Web Leaks. *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2011.
7. A. El-Atawy, E. Al-Shaer. Building Covert Channels over the Packet Reordering Phenomenon. *IEEE INFOCOM 2009*, 2009.
8. W. Myrvold and F. Ruskey, "Ranking and unranking permutations in linear time," *Information Processing Letters*, vol. 79, pp. 281–284, 2000.
9. J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1979.
10. P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, 1987.
11. P. B. Danzig and S. Jamin, "tcplib: A library of internetwork traffic characteristics," tech. rep., 1991.
12. A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection.," 2012.
13. S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia, "Model-based covert timing channels: Automated modeling and evasion," 2008.
14. S. Cabuk, C. E. Brodley, C. Shields. IP Covert Timing Channels: Design and Detection. *Proc. 11th ACM Conf. Computer and Communications Security (CCS)*, pages 178–187, 2004.
15. X. Luo, Ee W. W. Chan, R. K. C. Chang. Cloak: A Ten-fold Way for Reliable Covert Communications. *Proceedings of European Symposium on Research in Computer Security (ESORICS)*, 2007.
16. Hassan Khan, Yousra Javed, Fauzan Mirza, Syed Ali Khayam. Embedding a Covert Channel in Active Network Connections. *IEEE Global Telecommunications Conference*, 2009.