Low-Attention Forwarding for Mobile Network Covert Channels

Steffen Wendzel, Jörg Keller

Low-attention Forwarding for Mobile Network Covert Channels

Steffen Wendzel, Jörg Keller

University of Hagen
Faculty of Mathematics and Computer Science
58084 Hagen, Germany

Abstract. In a real-world network, different hosts involved in covert channel communication run different covert channel software as well as different versions of such software, i.e. these systems use different network protocols for a covert channel. A program that implements a network covert channel for mobile usage thus must be capable of utilizing multiple network protocols to deal with a number of different covert networks and hosts. We present calculation methods for utilizable header areas in network protocols, calculations for channel optimization, an algorithm to minimize a covert channel's overhead traffic, as well as implementation-related solutions for such a mobile environment. By minimizing the channel's overhead depending on the set of supported protocols between mobile hosts, we also minimize the attention raised through the channel's traffic. We also show how existing covert network channel infrastructure can be modified without replacing all existing infrastructure elements by proposing the handling of backward-compatible software versions.

Keywords: network covert channel, covert channel protocols, covert proxy, mobile security

1 Introduction

Covert channels are hidden communication channels which are *not intended for information transfer at all* [5]. The intention of a covert channel is to hide the existence of an information flow that possibly violates a system's security policy [6, 7]. Covert channels contribute to the free expression of opinion since they are useful to bypass censorship [28]. Network covert storage channels are covert channels that transfer information through a network by altering an attribute within the channel, while network timing channels communicate via modifications of packet timings and packet ordering [14]. The remainder of this paper focuses on covert storage channels.

For optimal auto-configuration of a program implementing a covert storage channel, it is necessary to provide means to minimize the raised attention and to select network protocols depending on the situation. Such an auto-configuration of a program is a requirement for mobile implementations which need to be able to add new storage channel types on demand (e.g. in form of an extension). If

such a program is able to deal with different network protocols, it can communicate with a number of different covert channel systems, i.e. different covert systems utilizing different network protocols. Fig. 1 depicts the scenario of a mobile covert channel user Bob communicating with Alice by utilizing different covert channels from different locations. Our proposed extendable implementation is able to provide such scenarios.

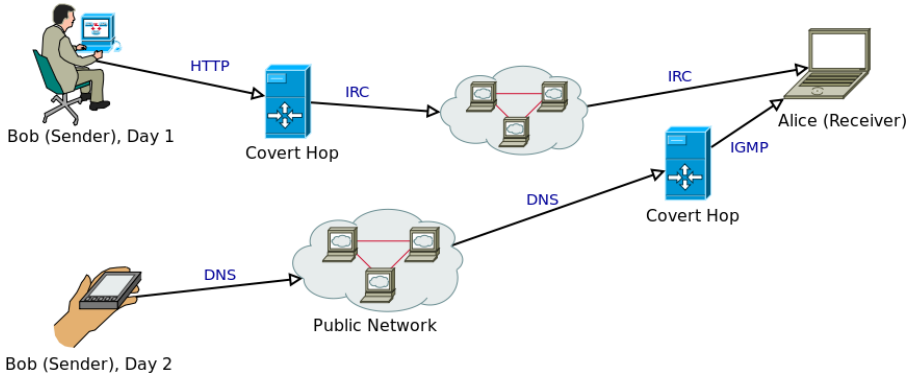

Fig. 1. Mobile communication from Bob to Alice using a single covert channel software able to deal with multiple networks with a number of covert channel systems utilizing different protocols.

In a large number of network protocols such as in TCP, IPv4, IPv6, or ICMP, header areas usable for covert channels are known [14–17]. At the same time, methods to prevent, limit and detect covert channels have been developed as well, such as the shared resource matrix methodology, the pump, covert flow trees, fuzzy timings and others [18–26].

Implementations capable of covert channel protocol switching or proxy chaining exist, e.g. LOKI2 [4], phcct [11, 13], and cctt [1]. Yet, they lack auto-configuration and dynamic extensibility. The idea of interoperability between covert channel hosts using different network protocols was discussed in [9], where a two-phase protocol was presented which contained a so called "network environment learning phase" (NEL, where the protocols used between two covert hosts where determined), as well as a "communication phase".

This paper describes a solution for the missing aspects of such covert network channels and improves on the related work: We discuss the handling of micro protocol implementations in mobile covert channels. Furthermore, we motivate their forensic usefulness in the context of multiprotocol channels, and we also focus on optimization of case-dependent covert channels (e.g. optimized performance and optimized packet sizes). This optimization is necessary as the available covert space per packet and the packet or header sizes may vary considerably between protocols. Our approach use a linear optimization. Additionally, we present a

simple but effective forwarding algorithm for covert proxies, able to reduce a channel's raised attention in such mobile environments.

The remainder of this paper is organized as follows. Section 2 serves as an introduction to the topic of utilizable header areas in network protocols while section 3 extends the topic of the previous section for multiprotocol channels with different probabilities, and additionally motivates multiprotocol channels from a forensic's point of view. Section 4 deepens the knowledge on utilizable areas in network packets, and section 5 focusses on the attribute-dependent optimization of protocol usage. Section 6 discuses implementation related topics and deals with the question, how a smart sharing of utilizable space for both, covert channel internal protocols as well as payload, is feasible. The calculations of sections 2 and 3 are extended for the context of covert channel proxy chains in section 7, where a forwarding algorithm for covert data is presented. The paper closes with a conclusion in section 8.

2 Calculating Basic Space Requirements

It is well-known that many hiding options exist in popular network protocols. A classification of these network protocols from a covert channel user's point of view enables the user to choose protocols according to his requirements. E.g. one classification attribute would be the detectability of a specific hiding method in a given protocol, another classification attribute is the amount of space to be utilized for covert data in a specific protocol. In case a covert channel user needs to transfer a large amount of data, it is required to place as much data as possible within a single network packet, since too many unexpected or abnormal network packets can alert a monitoring authority.

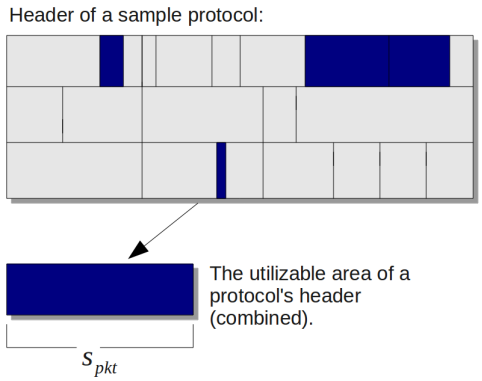

Fig. 2. Sum of utilizable areas of a network protocol's header.

To calculate the available space s_{pkt} within a network packet's header, it is required to sum up the sizes of all utilizable areas of a header, which is illustrated

in Fig. 2. Given an unidirectional covert network channel and assuming that the amount of covert data $s_{overall}$ required to be sent is known a priori, the number of packets N needed to transfer all data is

$$N = \left\lceil \frac{s_{overall}}{s_{pkt}} \right\rceil \tag{1}$$

since $s_{pkt} \cdot N \geq s_{overall}$. If the channel is able to let a receiver acknowledge a network packet with another network packet containing only an ACK flag and the ID of the acknowledged packet, the number of transmitted packets is $2N$. Here, we do not count re-transmitted packets. We further assume that the ACK flag and the received packet's ID fit into a single acknowledgement packet and that each such packet acknowledges only one received packet.

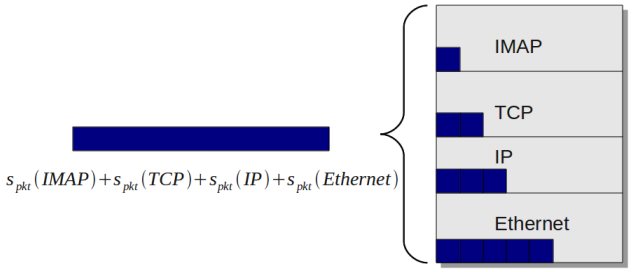

Fig. 3. s_{pkt} value for a multi-layer covert channel.

When a multi-layered network environment is used (as in case of the TCP/IP model), more than one layer can be used to place hidden data (as shown in Fig. 3). In such cases, the value of s_{pkt} is the sum of all utilizable header areas in every header. If a data transport over routers is mandatory, the utilizable areas of non-routeable network headers (e.g. Ethernet frames) cannot be used for the covert channel since they are replaced on every hop of the routing path, thus s_{pkt} may not contain these areas.

When a plaintext protocol is utilized, it is usually possible to dynamically extend a protocol's header by a number of attributes (e.g. by adding different HTTP attributes as "User-Agent"). Also, it is possible to dynamically extend the size of a single attribute to a limited size (e.g. the HTTP URL parameter can usually grow up to 1024 bytes length[1]). Such dynamic values can be considered in our calculation when they are used with a static length or if the average length is used.

[1] According to [27], HTTP servers should be able to handle an unlimited number of bytes within the URL parameter, but in practice many servers' URLs, specially in embedded systems as CISCO VoIP telephones, are limited to a length of 1024 bytes or less.

3 Multiprotocol Channels

Multiprotocol network covert channels were introduced by a proof of concept tool called "LOKI2"[4] and enhanced through transparent protocol switching in [11]. Such channels utilize a number of protocols to increase the effort to detect and analyze the traffic they carry. Multiprotocol channels operate as follows: Before a new covert network packet is sent, the sender chooses one of the available protocol hiding methods (e.g. in HTTP [2] or in DNS [3]), embeds the covert data into the new network packet, and sends it.

In case more than one network protocol is used by a covert channel, the amount of available space per packet s_{pkt} can vary. Assume a set of network protocols $P = \{P_1, P_2, \ldots, P_n\}$ used by the covert channel with the available spaces $s_1, s_2, ..., s_n$, where $s_i = s_{pkt}(P_i)$, the average amount of available space $\bar{s}_{pkt}$ is

$$s_{pkt} := \sum_{i=1}^{n} p_i s_i \tag{2}$$

if protocol P_i is chosen with probability p_i.

In the simplest case, a network protocol header, e.g. the DNS header, contains a combined sum of all spaces, s_{pkt}, as mentioned above. Due to defined rules (e.g. protocol standards) for such protocols, it is apparent that it is not always possible to combine all utilizable parts of a header at the same time, e.g. a HTTP POST request is not possible together with a HTTP OPTIONS request. To achieve a correct calculation of s_{pkt}, all non-combinable header parts must be treated as different protocols. If we assume that a network covert channel uses HTTP with the three request types GET, POST, and OPTIONS with different utilizable areas, the set of network protocols will be $P = \{HTTP_{GET}, HTTP_{POST}, HTTP_{OPTIONS}\}$. In case ICMP type 0 and 8 (echo response and echo request) would be used as well, both ICMP types can be represented by *one* element $p_{icmp-echo}$ in P since the utilizable header areas are identical [8]. If both protocol types are represented as one element, the probability of occurrence of both protocol types $p_{icmp-echo}$ must be the sum of p_{icmp-8} and p_{icmp-0}.

The utilizable area of a network packet's header is not always used completely by a covert channel, since attributes such as the raised attention per bit of a packet can be high for some bits. For example, if only the 3 least significant bits of the IPv4 TTL value are used for a covert channel, their raised attention is much lower compared to changing the 3 least significant bits of the IPv4 source address, since it is usual for network packets to take different routes, but it is (at least for internet servers outside of a LAN) not usual to have connections to the same host from a small range of 2^3 different IP addresses. Thus, protocol utilizations with different attributes should be separated in calculations by representing distinguishable elements of the set P.

From a forensic point of view, such multiprotocol covert channels result in a positive side-effect: In case one of the protocols used could be detected as used for a covert channel and was successfully analyzed, i.e. an attacker was able to

understand the hidden content, the non-analyzed data packets of a communication are still hidden since they are transferred within other protocols using different hiding techniques, as described in [12].

Additionally, it is feasible to send highliy sensitive data in fragments to make a forensic analysis harder: As shown in [29], TCP/IP normalizers face a problem called *inconsistent TCP retransmissions*, i.e. by sending packets that have identical TCP sequence numbers but different content and different TTL values, a normalizer cannot easily determine a connection's real content. By fragmenting covert channel payload using a multiprotocol channel, a forensic analysis of covert channel traffic faces a similar problem: For a forensic analysis, it is hard to determine the real covert channel's content, as well as the relevance of the content. This can be achieved by sending a key information in at least two pieces and different protocols: ($Part_1$, P_1) and ($Part_2$, P_2). For example: A covert channel user sends "secret" using ("se", P_1), ("cr", P_2), ("et", P_1). If an attacker is capable of detecting and analyzing covert traffic of protocol P_1, the attacker will only see the traffic "seet" instead of "secret". Thus, hiding of critical keywords can be improved and a forensic analysis can result in presumably useless data.

4 Utilizable Areas in Well-known Protocols

This section exemplifies s_{pkt} values for known covert channel utilizations of network protocols. Each implementation of a covert channel can cover different areas of a header, thus s_{pkt} values are implementation-dependent. Therefore, it is necessary for the covert communication between two hosts to modify exactly the same bits of a header. To achieve this goal, covert channel software versions should be backward-compatible, i.e. when a transaction begins, one or two version bits should be exchanged and the lowest available covert channel software version available on both hosts may be used to prevent errors.

Taking the ICMPv4 address mask request as an example to determine s_{pkt}, one can find the information to utilize the 32 bit address mask in [14], i.e. $s_{pkt}(ICMP_{AddrMaskReq}) = 32$ bits. The same source shows a number of IPv4 header hiding options; in case one chooses the 3 lowest bits of the TTL as well as the "Identification" value (16 bits) for the implementation of a covert channel, then s_{pkt} is 19 bits, but it can also have lower and higher values if a different number of header bits is used.

If the ICMP address mask request as well as the IP header are utilized together (as shown in Fig. 3) and assuming that they are used as described before, then s_{pkt} will be the sum of both s_{pkt} values: $s_{pkt} = 19 + 32 = 51$ bits.

Different bits of a header correspond to different detection probabilities, e.g. the modification of the least significant bit (LSB) of the IPv4 TTL value is harder to detect than a modification of the most significant bit (MSB) of the TTL. We do not believe that it is possible to define concrete values for the detectability of each bit, since such values always depend on the given monitoring situation of a network. E.g. the probability of detecting a covert channel that uses the ISN

bits varies depending on whether the TCP ISN flag is checked for covert data for all connections in a network or not [26].

To quantify such detectability values we propose to link a protocol P_i to an element of a small classification set (e.g. low, average and high detectability, $C = \{Low, Medium, High\}$). This may require an additional separation of elements of the set P (Sect. 3 already separated incompatible protocol header utilizations like $HTTP_{OPTIONS}$ and $HTTP_{GET}$).

We assume as an example that a HTTP covert channel is used to tunnel data through a requested URL as well as through the "Accept-Language" value via the GET method (cf. [27]). Requested URLs are usually written to a logfile and displayed in monitoring software like "Webalizer"[2]. The value of the field Accept-Language is unlikely to be monitored. Most small websites do not even handle the language settings sent by a client. Thus, the detection of covert data within the URL is usually more likely than the detection within the Accept-Language field. Therefore, a developer should define one element in P containing only the flag with the low (or average) detectability ($HTTP_{GET_{AccLang}}$) as well as one element that contains the flag with the higher detectability as well, i.e. $P = \{HTTP_{GET_{AccLang}}, HTTP_{GET_{All-Fields}}\}$, which enables a covert channel to be configured as desired (more space per packet vs. keeping a lower profile).

Additionally, it must be kept in mind that a protocol's classification should also depend on the protocol's probability of occurence, since exotic protocols can result in raising higher attention.

5 Optimizing Channels on Demand

A covert channel might be used in different situations and with different intentions, e.g.:

- When an automatic password cracking program needs to transfer short password information out of a network (e.g. one password per hour), the throughput of the covert channel is not important, while it is highly important to keep a low profile.
- When a blogger wants to upload a set of pictures of harmed protesters as soon as possible, a good covert channel throughput is required since the upload is to be done fast because of the relevance of that data in spite of its large size. Still a certain level of coverage from detection is necessary.

Being able to develop a covert channel program capable to deal with such dynamic situations, it is required to make its behavior dependent on the attributes of the protocols used. First, we introduce a value q_i that indicates how many bits are transferred to send a single covert bit using network protocol P_i with an (average) size sizeof(P_i) of that protocol's complete header:

[2] Webalizer is an open source logfile monitoring software available on *http://www.mrunix.net/webalizer/*.

$$q_i := \frac{sizeof(P_i)}{s_{pkt}(P_i)} \tag{3}$$

Depending on the given situation, we use linear optimization to calculate optimal probabilities $p_1, ..., p_n$ for all protocols $P_1, ..., P_n$ utilized between two systems. The set of constraints is that $\sum_i p_i = 1$ and that $0 < m \leq p_i \leq 1$ for all protocols P_i, where m is a minimum threshold to ensure that each protocol is used at all, even if it is not really desirable in the current situation. A possible value for m is c/n, where $c < 1$ is a suitable constant. For example, with $n = 20$ and $c = 0.2$, each protocol will be chosen with at least 1% probability. Using such a threshold complicates forensic reverse engineering because it prevents concentration on a small number of protocols.

If a high throughput of covert data for a fixed number of packets is required, then the target function to be maximized can be chosen as

$$f_1 = \sum_{i=1}^{n} p_i \cdot s_i \ .$$

If the goal is a small amount of header information sent for a given number of covert data bits, then the target function to be minimized is

$$f_2 = \sum_{i=1}^{n} p_i \cdot q_i \ .$$

The latter function can also be maximized by subtracting it from a large fixed value such as $\sum_i q_i$.

In a similar manner, the number of packets used can be minimized. By using a weighted sum of those target functions, one can optimize for any desired compromise. Furthermore, other optimization criteria may be possible which can be used to construct further target functions.

The choice of the target function, i.e. the intended situation, can be communicated between sender and receiver in a manner similar to the choice of the software version.

6 Implementations Based on Micro Protocols

Micro protocols are small protocols placed within the hidden data of a covert channel and used to control the channel [12].

The usage of micro protocols is mandatory for the implementation of the features described in earlier sections.[3] Within a micro protocol's data, information about supported protocols of a client, as well as covert channel version

[3] While the set of deployed protocols between two hosts is *static*, a micro protocol is not mandatory. To define a sequence of protocols to use, identical procedures and pseudo-random number generators with identical seeds for random choices can be used on both hosts. Other approaches leading to equal protocol sequences on both hosts are possible as well.

information can be exchanged between covert channel hops. E.g., one can define two bits representing a version information as described in Sect. 4. Other bits can determine supported elements of P. As mentioned in Sect. 1, Yarochkin et. al. presented a simple technique for the determination of available protocols between covert channel hosts in [9], but did not focus on optimization, forwarding, protocol classification or covert channel versioning, as we do. An important point mentioned by Yarochkin et. al. is of course the need to automatically filter administrativly blocked protocols.

Extensibility of existing covert channel software requires protocol information to be version dependent, i.e., in a new software version, a new protocol P_i could be supported which requires a new representation of P_i within the microprotocol. As mentioned in Sect. 1, the dynamic extendability of a software requires on-demand support for new protocols. Given the explained techniques as well as a modular covert channel software design, all these features can be implemented. Such modular designs are well known from the Apache[4] webserver module API as well as from Linux[5] kernel modules.

In case a micro protocol is used, it should be unified for all utilized network protocols to increase the stability of a covert channel software implementation. To find the maximum size such a micro protocol can use, it is required to find the minimal available size over all protocols used, i.e. $s_{min} = \min_i s_{pkt}(P_i)$. In case a constant or minimum required payload size is used, s_{min} must also contain this payload, which decreases the header size: $s_{payload} + s_{header} = s_{min}$, as presented in the left half of Fig. 4. If the payload is not of a constant size, e.g. if it requires only a defined minimum of space, the remaining space $s_{i,remain} = s_i - s_{min}$ can be used for additional payload.

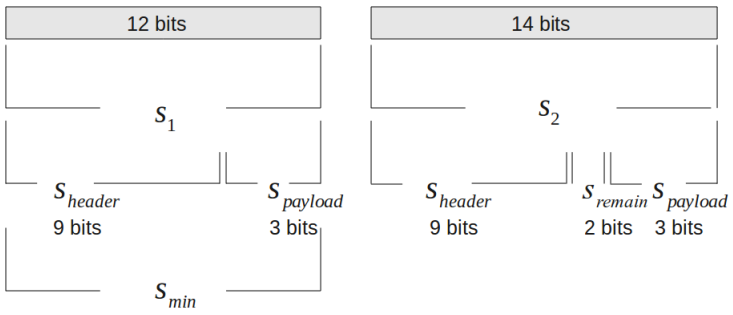

Fig. 4. Minimum space required for a minimum payload size and a constant micro protocol header size visualized for a set of two network protocols.

[4] www.apache.org
[5] www.kernel.org

7 Network Covert Channels with Proxy Chains

Covert communication paths based on covert proxy servers are a means to implement anti-traceability into a network covert channel (see Fig. 5). The value of s_{pkt} becomes useful in this context too, if multiple protocols as well as multiple covert proxy hosts are used.

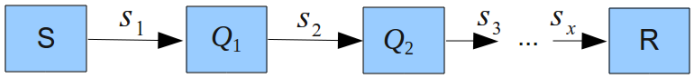

Fig. 5. A sample proxy network containing links with different s_{pkt} values ($S_i = s_{pkt}(P_i)$, where $i = 1, \ldots, x$), S=sender, R=receiver, Q_1 ... Q_n represent covert proxies.

Each host Q_i of a given covert proxy chain from the sender (S) to the receiver (R) chooses one network protocol to transfer data to the next host Q_{i+1} on the proxy chain. The intersection of the protocol sets on Q_i and the next hop Q_{i+1} can be used for the communication between both hosts as described in [9].

In many cases, it is not necessary for a network covert channel to be fast, as only small amounts of data such as passwords may be transferred. On the other hand, it is important to keep the raised attention of a channel as small as possible. To achieve this goal, the number of network packets per transaction must be limited (e.g. fragmentation must be prevented). A solution for this problem is possible as follows:

Let SP_i be the intersection of the set $P(Q_i)$ of protocols available on host Q_i and the set $P(Q_{i+1})$ of protocols available on host Q_{i+1}, i.e. $SP_i = P(Q_i) \cap P(Q_{i+1})$. The set S_i represents the sizes s_{pkt} for all elements P_j of SP_i. Now let $s_{max}(i)$ be the maximum over all elements of S_i, i.e. $s_{max}(i) = \max S_i$. To prevent fragmentation and to send as few packets as possible, host Q_i sends network packets with the maximum data size $s_{max}(i)$ to host Q_{i+1}. When Q_{i+1} receives the network packet from Q_i that needs to get forwarded to Q_{i+2}, it acts as follows:

- If $s_{max}(i) = s_{max}(i+1)$, then forward the data.
- If the transaction ended (i.e. if there is no remaining data), then forward the data, too. Whether a transaction ends (or begins) can be determined by covert channel-internal protocols as presented in [10].
- Otherwise: Send as many complete data packets of size $s_{max}(i+1)$ as possible. To avoid bursts in case that $s_{max}(i+1) \ll s_{max}(i)$, one may use a leaky bucket approach to regulate packet frequency. If the data left to send is smaller than $s_{max}(i+1)$, then wait for a time t for further data to arrive from Q_i and then send as many complete data packets as possible. Repeat this step as long as no data is left or no new data arrived for time t.

Also here, a linear optimization might be useful that tries to balance the number of packets (by preferring protocols in S_i with large s_{pkt}) and detectability

(by using a multitude of protocols). Also, one might desire in some scenarios to combine this approach with the one from Sect. 5.

If more than one covert channel software version is in use, the newest software version between two hosts must be defined before the intersecting set of all protocols is calculated, since each local set P depends on the software version, as described in section 4. Such functionality can be implemented in a micro protocol.

8 Conclusions

We presented techniques for the dynamic implementation of network covert storage channels, i.e. it is possible to build a program that is able to dynamically load extension information for the utilization of new network protocol headers. Such dynamic covert channel implementations enable their users to act in mobile covert channel networks.

Additionally, our concept enables implementers to continue the development of existing covert channel network infrastructure without dealing with the need to replace all distributed existing software versions. This paper also introduced calculation methods as well as an forwarding algorithm which are able to deal with different protocol types, micro protocols, and the utilization of covert proxy systems, while minimizing the channel's raised attention to keep a low profile.

A drawback of the described techniques is that they – in comparison to existing implementations – result in more complex algorithms, since additional control capabilities must be implemented using micro protocols. A proof of concept implementation is under development but, since it is part of a larger project, unfinished at the moment. Yet previous work, such as [9, 11, 12] indicates that protocol switching using a (micro) protocol is feasible.

Future work will include the design of a description structure for the dynamic extension of covert channel programs.

References

1. Castro, S.: cctt (covert channel testing tool) v0.1.8 *http://gray-world.net/it/pr_cctt.shtml*, 2003.
2. Castro, S.: Covert Channel and tunneling over the HTTP protocol detection: GW implementation theoretical design, *http://www.gray-world.net*, November 2003.
3. Born, K.: Browser-Based Covert Data Exfiltration, In Proc. 9th Annual Security Conference, Las Vegas, NV, April 7–8, 2010.
4. Daemon9: LOKI2 (the implementation), Phrack Magazine, Vol. 7(5), *http://www.phrack.com/issues.html?issue=51&id=6&mode=txt*, September 1997.
5. Lampson, B.W.: A Note on the Confinement Problem, Commun. ACM, Vol. 16(10), pp. 613-615, 1973.
6. Murdoch, S.J.: Covert channel vulnerabilities in anonymity systems, PhD thesis, University of Cambridge (Computer Laboratory), 2007.
7. Wonnemann, C., Accorsi. R., Müller, G.: On Information Flow Forensics in Business Application Scenarios, In Proc. IEEE COMPSAC Workshop on Security, Trust, and Privacy for Software Applications, 2009.

8. Postel, J.: Internet Control Message Protocol, DARPA Internet Program Protocol Specification, RFC 793, September 1983.
9. Yarochkin, F.V., Dai, S.-Y., Lin, C.-H., et. al.: Towards Adaptive Covert Communication System, 14th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'08), pp. 153–159, 2008.
10. Ray, B. and Mishra, S.: A Protocol for Building Secure and Reliable Covert Channel, Sixth Annual Conference on Privacy, Security and Trust (PST), pp. 246–253, 2008.
11. Wendzel, S.: Protocol Hopping Covert Channel Tool v.0.1, *http://packetstormsecurity.org/files/60882/phcct-0.1.tgz.html*, 2007.
12. Wendzel, S.: Protocol Hopping Covert Channels, Hakin9 Magazine 1/08, pp. 20–21, 2008. (*in German*)
13. Bejtlich, R.: Analyzing Protocol Hopping Covert Channel Tool, *http://taosecurity.blogspot.com/2007/11/analyzing-protocol-hopping-covert.html*, November 2007.
14. Ahsan, K.: Covert Channel Analysis and Data Hiding in TCP/IP, M.Sc. thesis, University of Toronto, 2002.
15. Rowland, C.H.: Covert Channels in the TCP/IP Protocol Suite, First Monday, Vol. 2, Nu. 5, May 1997.
16. Scott, C.: Network Covert Channels: Review of Current State and Analysis of Viability of the use of X.509 Certificates for Covert Communications, Technical Report RHUL-MA-2008-11, Department of Mathematics, Roal Holloway, University of London, January 2008.
17. Hintz, D.: Covert Channels in TCP and IP Headers, Presentation slides of the DEFCON 10 conference, *http://www.thedarktangent.com/images/defcon-10/dc-10-presentations/dc10-hintz-covert.pdf*, 2002.
18. Berk, V., Giani, A., Cybenko, G.: Detection of Covert Channel Encoding in Network Packet Delays, Technical Report TR536, Rev. 1, Dep. of Computer Science, Dartmouth College, November 2005.
19. Cabuk, S., Brodley, C.E., Shields, C.: IP Covert Timing Channels: Design and Detection, In Proc. 11th ACM Conference on Computer and Communications Security (CCS '04), pp. 178–187, 2004.
20. Fadlalla, Y.A.H.: Approaches to Resolving Covert Storage Channels in Multilevel Secure Systems, Ph.D. Thesis, University of New Brunswick, 1996.
21. Fisk, G., Fisk, M., Papadopoulos, C. and Neil, J.: Eliminating Steganography in Internet Traffic with Active Wardens, In Proc. Information Hiding Conference 2003, Lecture Notes in Computer Science, Volume 2578, pp. 18–35, 2003.
22. Hu, W.-M.: Reducing Timing Charmers with Fuzzy Time, 1991 Symposium on Security and Privacy, IEEE Computer Society, pp. 8–20, 1991.
23. Kang, M.H., Moskowitz, I.S.: A Pump for Rapid, Reliable, Secure Communication, Proceedings of the 1st ACM Conference on Computer and Communication Security, pp. 119–129, November 1993.
24. Kemmerer, R.A.: Shared resource matrix methodology: an approach to identifying storage and timing channels, ACM Transactions on Computer Systems (TOCS), ACM, Volume 1, Issue 3, pp. 256–277, 1983.
25. Kemmerer, R.A., Porras, P.A.: Covert Flow Trees: A Visual Approach to Analyzing Covert Storage Channels, IEEE Transactions on Software Engineering, Volume 17. No. II, pp. 1166–1185, November 1991.
26. Murdoch, S. J., Lewis, S.: Embedding Covert Channels into TCP/IP, In Proc. Information Hiding 2005, pp. 247–261, 2005.

27. Fielding, R., Gettys, J., Mogul, J., et. al.: Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, June 1999.
28. Zander, S., Armitage, G., Branch, P.: Covert Channels and Countermeasures in Computer Networks, IEEE Communications Magazine, pp. 136–142, December 2007.
29. Handley, M., Paxson, V., Kreibich, C.: Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics, In Proc. 10th USENIX Security Symposium, Volume 10, pp. 115–131, 2001.