12-1-2016

Covert Channel using Man-In-The-Middle over HTTPS

Matthew Johnson

Peter Lutz

Daryl Johnson

Recommended Citation
M. Johnson, P. Lutz and D. Johnson, "Covert Channel Using Man-in-the-Middle over HTTPS," 2016 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, 2016, pp. 917-922. doi: 10.1109/CSCI.2016.0177

Covert Channel using Man-In-The-Middle over HTTPS

Matthew Johnson, Peter Lutz, and Daryl Johnson
Rochester Institute of Technology
152 Lomb Memorial Drive
Rochester, NY 14623
mrj9096@rit.edu, peter.lutz@rit.edu, Daryl.johnson@rit.edu

Full/Regular Research Paper submitted to CSCI-ISMC

***Abstract*—The goal of this covert channel is to prove the feasibility of using encrypted HTTPS traffic to carry a covert channel. The encryption key is not needed because the original HTTPS payload is not decrypted. The covert message will be appended to the HTTPS data field. The receiver will extract the covert channel and restore the original HTTPS traffic for forwarding. Only legitimate HTTPS connections will be used as the overt channel. A Man-in-the-Middle (MITM) attack at the sending and receiving ends will give access to modify the traffic streams. The HTTPS return traffic from the server can carry a covert channel. Without the original HTTPS traffic for comparison or the original encryption keys, this covert channel is undetectable.**

Key Terms: Covert Channel, Man-In-The-Middle, MITM, HTTPS

I. Introduction

Secret forms of communication have been used throughout history. Examples such as tattooed slave scalps and wax tablets are but a few. Covert channels are a special clas of secret communications that seek to deny that a conversation is even taking place.

The goal of this project was to create a covert channel that exhibited the following characteristics:

- Routable, being able to traverse wide area networks such as the internet without difficulty
- Difficult to detect even with existing monitoring tools such as SNORT and BRO
- Utilizing a commonly used protocol, in this case HTTPS
- Taking advantage of encryption capabilities to further protect the message stream, while not generating any additional or unexpected traffic

II. Background and Related Work

Butler W. Lampson was the first individual to give a definition to the term "covert channel" in 1973. He defined covert channels as "(channels) not intended for information transfer at all, such as the service program's effect on system load."[1] Since then, the definition of a covert channel has been evolving. A current definition for covert channel is "A mechanism for sending information without the knowledge of the network administrator or other users." as stated by Erik Curture in his paper *Covert Channels*[2].

There have been many research papers exploring the varied mechanisms and implementations of covert channels[3]. HTTPS and HTTP protocols have been the vehicle for several covert channel applications[4][5][6]. Numerous other studies have been done using other aspects of the TCP/IP protocol[7][8][9][10][11][12][13]. There has even been a paper on using a covert channel to detect Man-In-The-Middle attacks[14] The authors were unable to find any other covert channel implementations that successfully utilized the SSL encrypted payload of a HTTPS connection in which the decryption key for the SSL encryption was not needed.

III. Concept

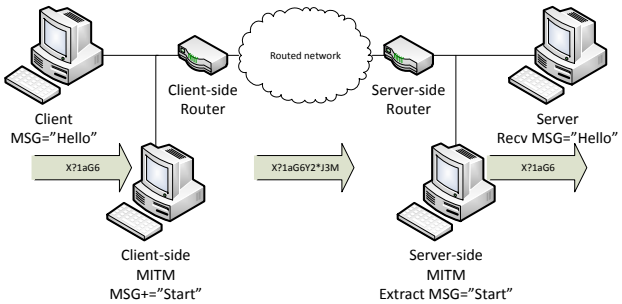

Figure 1. Concept Overview

The covert channel described in this paper (see Figure 1) builds upon the realization that if one appends an encrypted message to the end of the data packet that is part of SSL encrypted traffic crossing the internet, it is virtually impossible to detect that it has been tampered with. In other

words, as long as it looks sufficiently random. The format of the encrypted portion of an SSL packet includes the length of the data encapsulated as well as the length of the padding used to obscure the message context. These length fields are also encrypted to deny that information to an observer or adversary. It is also then unavailable to detect any addition to the encrypted field and cannot be viewed unless one has the correct key.

Many IDSs have the capability to decrypt SSL traffic. This could allow the detection of our covert channel. However, it is unlikely that the traffic being utilized will pass through an IDS that has the correct SSL keys. This makes the detection of our covert channel nearly impossible.

When the intended recipient receives the modified SSL traffic containing the covert communication, they will have the correct SSL key. Upon decryption, the recipient will be able to determine that the traffic has been tampered with. This would raise a red flag. In order to prevent this there are two approaches:

- We can own the client and prevent them from complaining
- Place a system in the way that can perform a man-in-the-middle attack against the receiver, intercept the traffic, and remove the covert message which will restore it to its original form.

If one owns the receiver, it is straightforward to handle the modified traffic and prevent raising an alarm. Likewise on the sending end of the connection, if one owns the sender, it is easier to create the covert channel when the traffic is originally sent. Our approach is more involved by utilizing a man-in- the-middle attack to capture the traffic after being sent and modify it to carry the covert channel. We then extract the covert content before delivering it to the receiver. This way, we can deploy our channel without owning any hosts other than our own.

HTTPS was the encrypted protocol chosen for this project because of it is widely used and its presence is not unexpected anywhere.

IV. Why use MITM?

The use of a Man-in-the-Middle(MITM) approach does complicate the implementation but it allows this covert channel to piggy back on existing traffic. Thus this covert channel does not create any potentially unexpected traffic over the routed network. The MITM approach allows us to utilize traffic from actual clients going to the expected servers. the core of this covert channel is that the covert data is piggy backed on real SSL traffic and is undetectable by a an observer between the source and destination. An additional benefit of the MITM on the client side is that it allows the covert channel capture all clients on the local network and distribute the covert channel over many conversations. Thus an observer would have to aggregate all of the connections to analyse the channel. This enhances the bandwidth of the covert channel as well by aggregating the natural traffic from multiple users.

V. ARP Spoofing

This covert channel requires an HTTPS connection to travel over. When an HTTPS connection is established, a Man-In-The-Middle (MITM) attack will be performed on both ends of the connection. The MITM attack will begin by performing ARP spoofing. ARP spoofing is accomplished by sending fake ARP (Address Resolution Protocol) reply messages to the targeted host. The ARP messages "informs" the targeted host that its default gateway is the attacker who is performing the MITM attack. The default gateway is then updated in the host's ARP table with the MAC address of the attacker and the IP address of the actual default gateway. While it appears (at the IP level) that all of the host's traffic is going directly to the default gateway, in fact host's traffic is going to the attacker. The attacker then chooses to either forward the traffic to the default gateway or drop the packets. In the case or our covert channel, we modify SSL packets and then send them to the default gateway.

The default gateway may send out its own ARP messages to keep the host's ARP table updated with the correct information. This can be a problem. To prevent the host's ARP table from being updated by the default gateway, the attacker will send out its own ARP reply messages either immediately after the default gateway sends out its ARP messages or repeat the ARP message periodically. For a brief window of time, the host may have the correct MAC address for the default gateway. It is possible that some packets may be forwarded directly to the default gateway. This problem cannot be entirely prevented but it will not stop the covert channel from performing its purpose. The gateway is also ARP spoofed so that it registers the attacker's MAC address for the client IP. This causes the gateway to send all of its traffic for the client through the attacker.

While the client and the server can be on the same network, this protocol is also routable. This means that the covert channel can work across any network, including the internet because the underlying protocol (HTTPS) is routable.

The program Ettercap was selected for implementing this covert channel. It natively supports ARP spoofing but also supports a plugin API that allows for the manipulation of forwarded frames. This plugin architecture was one of the primary reasons Ettercap was chosen. In addition, Ettercap is one of the best known MITM platforms in existence. While not well documented, there was a large user community to help with development. For our test, we used 512 byte (or fewer) covert fragments. This is just data, not including the sequence number, CRCs or length field.

VI. Sending a Covert Message

When examining the HTTPS protocol process, there is an initial flurry of messages that are exchanged to instantiate the connection, negotiate the encryption used, and start the data exchange process. These initial packets are not useful to the covert channel as there is not any encrypted data content on which to piggyback covert content. The initial exchange must be observed, watching for the beginning of the data transfer to start inserting the covert fragments. The SSL data packets are identified by their SSL data type of 23 [15].

Simply appending a piece of the covert message to the end of each encrypted HTTPS data field would create problems. The first problem would be that any observation of the encrypted traffic containing the covert message might reveal clues that there had been a modification. ASCII strings or identifiable patterns would be visible and could expose the covert channel.

To prevent this, the appended covert message would be encrypted using a key that has been pre-shared between the sender and receiver. Additionally, a mechanism is necessary for the receiver to determine if a covert message has been appended to the HTTPS data field. The covert receiver does not have the decryption key and thus cannot decrypt the HTTPS data field to determine that there is a covert fragment attached. The mechanism implemented below provides a multi-level check to verify that a covert message fragment exists. The data appended to the HTTPS data field will use the format as shown in Figure 2.

The length of the appended covert message will be in the last two bytes of the HTTPS data field. This will be a binary value hence it will not need to be encrypted itself. The last two bytes are used because that will always be a guaranteed known location. The length field alone is insufficient to guarantee the existence of a covert message. The overt message could possibly share the same ending two bytes as the covert message. To solve this problem, immediately before the length field will be two bytes containing the checksum of the encrypted covert message fragment. The checksum will be a binary value as well and hence does not require encryption. As a double check, since it's possible that an overt message could end in the same four bytes as the covert checksum and length as in Figure 2, the checksum will be repeated in the two bytes preceding the covert message fragment.

VII. Justification for Covert Message Complexity

The reason for the complexity of the appended data is to ensure that the receiver can accurately determine if a covert fragment exists. Without a reliable mechanism for detection, false positive identification could occur and cause two problems: loss of integrity of the covert message and damage to the overt message. If a non-covert message is identified as containing a covert fragment, it will be decoded and included in the covert message. This misidentification as a covert fragment will also cause the receiver to remove that portion identified as a covert fragment from the part that is passed on to the overt receiver. The overt receiver would then register an error as the SSL data packet would not be the correct length and would not decrypt properly. The following analysis assumes an encryption with uniform distribution. Assume that packet damage will be detected at a higher level first.

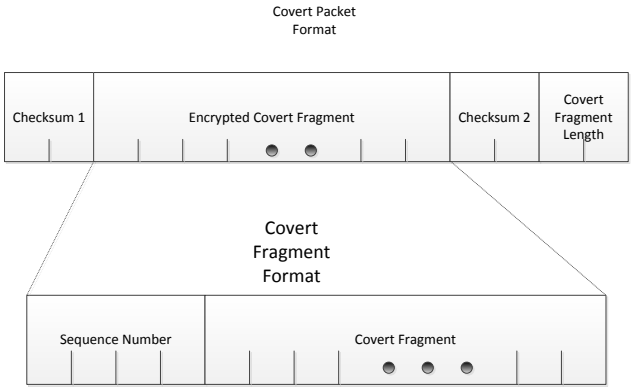

Figure 2. Format of appended data to HTTPS encrypted data

For this implementation, the

The Covert Fragment(CF) Length field is a 2 byte unsigned integer. The range of legal values for the CF length value is

$$10 <= CFLength <= maxpacketlength(< 1450)$$

where 1450=1500-20-20-10 represents the max length of an Ethernet packet less the IP and TCP header and the minimum space needed for the covert fragment. The probability of false identification = legitimate values/ possible values

$$1440/65536 = 0.0220 = 2.20\%$$

The check sums, CRC1 & CRC2, are unsigned integers

$$0 <= CRC1 <= 65536; 0 <= CRC2 <= 65536$$

The probability of the CRCs matching and false identification is the probability of crc1 and crc2 being the same value even if there is no covert data. If we fix the value of CRC1, the probability of CRC1=CRC2

$$1/65536 = 1.52e - 5 = 0.0000152 = 0.00152\%$$

Finally, the CRC used in our implementation is a simple 16 bit addition discarding the overflow. Calculated CRC(CF data bytes) = CRC1 = CRC2 Probability of false identification

where the calculated CRC of the CF data equals the stored CRC1 or CRC2 is based on:

$$0 <= eachpairofCCdatabytes <= 65536$$

Regardless of how many CF data byte there are the result will be some 2 byte value. Therefore the comparison is similar to colliding two random 2 byte values.

$$1/65536 = 1.52e-5 = .0000152 = 0.00152\%$$

Overall probability of a false positive is the product of the three cases or

$$0.0220 * 0.0000152 * 0.0000152 = 5.08\text{e}{-12}\%$$

This likelihood is small enough to be irrelevant to the covert channel success.

VIII. Receiving a Covert Message

To verify the existence of a covert message fragment (see Figure 3), the receiver will use several checks to validate that the incoming HTTPS data packet contains a covert fragment . The last two bytes of the HTTPS data field are used as the length of the covert fragment. The HTTPS length field is not useable because it is encrypted. The TCP frame length cannot be used because it is a calculated value and not an actual data field. Therefore, the length field for the IP frame will be used instead. If the covert fragment length indicated by the last two bytes is greater than the IP length from the IP header, then there can be no covert fragment.

The preceding two bytes are the second covert fragment checksum. Locate and retrieve the two bytes that are the first covert channel checksum (end of the packet less 6 bytes for the width of the length and both checksum fields less the value in the CF length field). These bytes are for the first checksum value. If these two checksums are not equal, then there is no covert message.

Next, the covert message is retrieved from the bytes between the first and second checksum. The checksum of the covert message is computed and compared to the retrieved checksum. If they don't match, then there is no covert message. If all of these checks succeed, then we have found the covert fragment. Remove the covert fragment from the HTTPS data field. Subtract the length of the covert fragment from the IP frame length. The TCP packet is ready to forward.

To insert a covert message into the HTTPS data packet, a TCP packet must be captured. Only HTTPS data packets will be used to transport the covert message. Other HTTPS traffic, such as protocol negotiation and key exchange, will be bypassed as they do not contain an encrypted payload to hide the covert communication. The HTTPS data field contains both HTTPS communication and a random number of padding bytes to make traffic analysis more difficult. This does not affect the covert channel. It is not necessary to

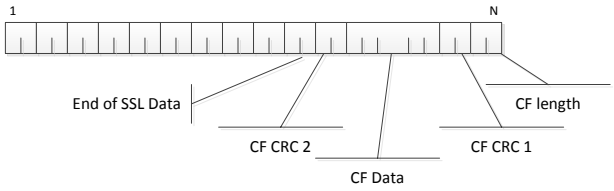

Figure 3. Structure of SSL packet with Covert Fragment

decrypt the HTTPS data field to embed our message. Once the correct TCP/HTTPS packet is captured, the HTTPS data field will be extracted. Next, the covert packet (consisting of the first checksum, encrypted covert data, second checksum, and covert length) will be appended to the end of the HTTPS data field. The new HTTPS data field will be placed back into the TCP packet. Even though the HTTPS data field has increased in length with the addition of the covert packet, the HTTPS length field is not changed for two reasons. First, the HTTPS length field is encrypted. Without the key or certificate, this would be all but impossible. Second, the covert packet will be removed before it arrives at the HTTPS client or server and then the encrypted length will be correct again. Instead, the IP frame length will be extracted and placed into a value. The length of the covert fragment will be added to the length of the IP frame length. The new IP frame length will be placed back in the TCP frame. The TCP packet is ready to be forward to the default gateway.

IX. Reassembly of Covert Channel

The reassembly of the covert message from the piece received over the covert channel in the HTTPS traffic must be managed. In order to detect lost fragments and reorder the pieces to reconstruct the original message some additional overhead is necessary. We designed a data structure for the message portion of the covert channel to provide the sequencing information needed to identify where the fragment goes in the final message. This information also allows the detection of missing fragments. The structure is a four byte sequence number followed by the data. The length of the covert fragment is known from the covet packet data and we can calculate the data field length from that information. Reassembly is done by allocating a buffer at the receiving end and placing the fragments in the buffer at the indicated offset. A second buffer would be used to mark bytes that have been loaded thus indicating bytes that have not been received yet.

X. TCP Sequence Number Adjustment

When implementing the covert channel, we needed to adjust the TCP sequence and acknowledgement numbers.

This is because these numbers are actually byte counts of the TCP payload [16][17]. Since adding covert data added TCP payload bytes, if we had not adjusted these numbers, an observer (indeed, with only a packet analyzer such as wireshark[18]) would note the discrepancy and know that something was awry.

To this end, once a covert flow is established, our software keeps track of outstanding (unacknowledged) packets and both their adjusted (because of covert content) and their original (without covert content) sequence numbers. As replies arrive, sequence numbers being acknowledged are removed from this list, and adjusted sequence numbers are replaced with original sequence numbers.

A difficult problem that arose when designing this was that the sender of a flow might not receive an acknowledgement before it timed out, causing a retransmission of one or more TCP segments. Since our covert software only keeps track of sequence numbers, and not entire packets, we could not resend the covert packet that corresponded to the retransmission. In this rare case, we quarantine the flow, dropping all packets from that IP address and port number. Eventually the sender gives up on the flow and treats it as a lost connection. Again, these events are rare, and would not likely give rise to suspicions on the internet.

XI. The Use of Multiple Clients

Due to the design of the client end of the covert channel, the sending Man-in-the-Middle agent can utilize any number of clients on the local network for carrying portions of the covert message. This increases the throughput of the covert channel considerably.

In fact, our sender will piggyback its covert message on any HTTPS traffic coming from any client in the client network as long as it is destined to the target server. In addition, replies from the server to the client may also carry covert content from the server end of the channel. Our implementation is an N-to-1 full duplex channel.

XII. Implementation and Testing

As is the norm with new covert channels, we have produced a proof of concept implementation and executed it on a testbed of our design. In this section we will describe the idea behind our implementation approach and the specifics of the development platform we used.

Our test platform (shown in Figure 4) was implemented on a VMWare ESXi 6.0.0 server. This platform hosts three networks, connected in series by two routers.

The network on the left (10.1.1.0/24) is attacked by a man in the middle (MITM) attack from the MITM Left machine. This attack is configured to send all traffic addressed to the router to the MITM Left machine as well as all traffic destined to the HTTPS Client. The network on the right (10.2.2.0/24) is similarly attacked by the MITM Right machine which captures all traffic to the router or to the HTTPS Server.

The covert channel software runs on, and is integral with, the MITM software on both networks. An observer on either the left or the right network could observe that a MITM attack is underway. Our goal is to use normal HTTPS traffic between the client on the left, and the server on the right, to hide covert content as it passes through the internet. The center network (10.3.3.0/24) above simulates the internet, and is where we attempt to observe traffic to detect the covert channel.

The software used to promote the MITM attacks is Ettercap. Ettercap can be customized by writing a plugin, in the C language, that manipulates packets in any way the programmer wishes. In our case, the plugin only captures TCP traffic (the rest being sent on unmolested). Even most TCP packets are sent on. We only focus on HTTPS packets coming from anyplace on the 10.1.1.0/24 network and destined to the HTTPS server at 10.2.2.1. In addition, we track reverse traffic from the server to any host on the 10.1.1.0/24 network.

HTTPS traffic captured, if carrying an SSL data payload, is modified to appear to be SSL data traffic, but with covert content added, as described above. On the receiving end, this SSL content is stripped off and used by the MITM machine. The sender of the original HTTPS packet and the receiver see only that packet. The MITM machines use these packets to piggyback their data. An observer in the internet (the 10.3.3.0/24 net) will not be able to detect the presence of the covert content.

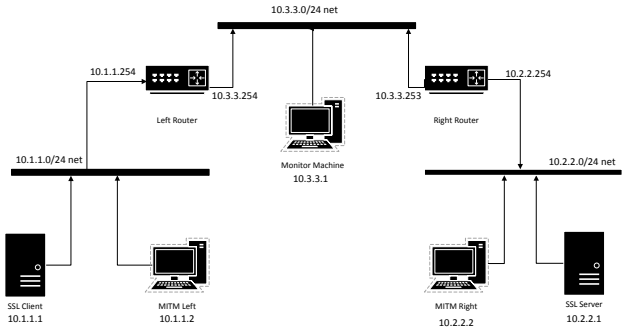

Figure 4. Testbed Topology

XIII. Future Work & Questions

Our proof of concept implementation does not address packet fragmentation over the internet. If the sending end of the channel sends a TCP packet that is larger than the MTU of some network along the way, a router will fragment the packet and it will arrive at the receiving end of the channel

in pieces. We are uncertain whether Ettercap will receive the reassembled TCP packet or the individual fragments. In the former case, all will be well. In the latter case, we would need to add reassembly code to our implementation.

We have left exploration of the fragmentation issue to other researchers and claim that, while important for production use of this covert channel, it is not important to its proof of concept.

To further conceal the covert fragment, the entire covert fragment should be encrypted not just the data section. This change would make the length and the two CRC fields more difficult to detect. In the event that this covert channel is known to an observer, without the decryption key it could still not be detected.

XIV. Conclusion

The goals of the project were to demonstrate that a covert channel could be developed that would exhibit the characteristics of routability, stealthy-ness, utilization of a ubiquitous protocol, and not generating any unexpected traffic across the routed networks. The first and third goals were achieved by employing the HTTPS protocol as the carrier. HTTPS is built on the TCP/IP protocol which makes it routable across a corporate network as well as the internet. This provides the greatest reach for this covert channel. HTTPS is a very common protocol observed on almost any network. HTTPS is the protocol that supports encrypted web browsing[19].

The covert channel was made stealthy by two factors. The use of MITM agents at both the sender and receiver makes the participants unaware of the covert channel. Their activity patterns were not affected and thus provided a legitimate and expected cover for the covert channel.

By piggy backing on the SSL payload and encrypting the covert fragment, the covert message was virtually impossible to detect after leaving the local network. Detection could only be accomplished if an observer had the decryption key or a copy of the original overt traffic for comparison.

We ran a tcpdump[20] capture containing the covert channel traffic through the intrusion detection programs, SNORT[21] and BRO[22][23]. There were no alerts triggered by the covert channel traffic using all available rulesets.

We believe that this covert channel has achieved all of the goals set forth.

XV. References

References

[1] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973. [Online]. Available: http://doi.acm.org/10.1145/362375.362389

[2] E. Couture, "Covert channels," *Commun. ACM*, Aug. 2010. [Online]. Available: https://www.giac.org/paper/gcia/5603/covert-channels/120171

[3] K. Anjan and J. Abraham, *Recent Trends in Network Security and Applications: Third International Conference, CNSA 2010, Chennai, India, July 23-25, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. Behavioral Analysis of Transport Layer Based Hybrid Covert Channel, pp. 83–92.

[4] M. Bauer, "New covert channels in http: Adding unwitting web browsers to anonymity sets," in *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '03. New York, NY, USA: ACM, 2003, pp. 72–78.

[5] L. Ji, W. Jiang, B. Dai, and X. Niu, "A novel covert channel based on length of messages," *Information Engineering and Electronic Commerce, International Symposium on*, vol. 0, pp. 551–554, 2009.

[6] W. Huba, B. Yuan, D. Johnson, and P. Lutz, "A http cookie covert channel," in *Proceedings of the 4th International Conference on Security of Information and Networks*, ser. SIN '11. New York, NY, USA: ACM, 2011, pp. 133–136.

[7] S. J. Murdoch and S. Lewis, "Embedding covert channels into tcp/ip," in *Proceedings of the 7th International Conference on Information Hiding*, ser. IH'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 247–261.

[8] M. Wolf, "Covert channels in lan protocols," in *Proceedings on the Workshop for European Institute for System Security on Local Area Network Security*, ser. LANSEC '89. London, UK, UK: Springer-Verlag, 1989, pp. 91–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=646904.710349

[9] L. Frikha, Z. Trabelsi, and W. El-Hajj, "Implementation of a covert channel in the 802.11 header," in *2008 International Wireless Communications and Mobile Computing Conference*, Aug 2008, pp. 594–599.

[10] K. Ahsan, "Covert channel analysis and data hiding in tcp/ip," Master's thesis, University of Toronto, Toronto, Canada, 2002.

[11] Z. Trabelsi, H. El Sayed, L. Frikha, and T. Rabie, "A novel covert channel based on the ip header record route option," *Int. J. Adv. Media Commun.*, vol. 1, no. 4, pp. 328–350, Aug. 2007. [Online]. Available: http://dx.doi.org/10.1504/IJAMC.2007.014811

[12] L. Ji, H. Liang, Y. Song, and X. Niu, "A normal-traffic network covert channel," in *Computational Intelligence and Security, 2009. CIS '09. International Conference on*, vol. 1, Dec 2009, pp. 499–503.

[13] C. H. Rowland, "Covert channels in the tcp/ip protocol suite," *First Monday*, vol. 2, no. 5, 1997. [Online]. Available: http://firstmonday.org/ojs/index.php/fm/article/view/528

[14] D. J. E. Brown, B. Yuan and P. Lutz, "Covert channels in the http network protocol: Channel characterization and detecting man-in-the-middle attacks," in *Proceedings of the 5th International Conference on Information Warfare and Security: ICIW2010*, April 2010, pp. 56–64.

[15] F. et al, "The secure sockets layer (ssl) protocol version 3.0," Internet Requests for Comments, RFC Editor, RFC 6101, August 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6101.txt

[16] P. et al, "Transmission control protocol," Internet Requests for Comments, RFC Editor, RFC 793, Septenber 1981. [Online]. Available: http://www.rfc-base.org/rfc/rfc793.txt

[17] S. et al, "A tcp/ip tutorial," Internet Requests for Comments, RFC Editor, RFC 1180, January 1991. [Online]. Available: https://tools.ietf.org/html/rfc1180

[18] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

[19] "Usage of http/2 for websites." [Online]. Available: http://w3techs.com/technologies/details/ce-http2/all/all

[20] S. McCanne, C. Leres, and V. Jacobson, "Tcpdump," *Lawrence Berkeley National Laboratory, fipV/ftp. ee. lbl. gov/tcpdump. tar. Z*, 2001.

[21] R. Alder, A. Baker, E. Carter, J. Esler, J. Foster, M. Jonkman, C. Keefer, R. Marty, and E. Seagren, "Snort: Ids and ips toolkit," 2007.

[22] I. Bro, "Homepage: http://www. bro-ids. org," 2008.

[23] P. Mehra, "A brief study and comparison of snort and bro open source network intrusion detection systems," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 6, pp. 383–386, 2012.