Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2022 EU - Selected Papers of the Ninth Annual DFRWS Europe Conference

Knock, knock, log: Threat analysis, detection & mitigation of covert channels in syslog using port scans as cover

Kevin Lamshöft [a, *], Tom Neubert [b], Jonas Hielscher [a], Claus Vielhauer [b], Jana Dittmann [a]

[a] Otto-von-Guericke-University Magdeburg, Germany
[b] Brandenburg University of Applied Sciences, Germany

ARTICLE INFO

Article history:

ABSTRACT

In this paper we perform a threat analysis for a covert Command and Control (C2) channel using port scans as cover and syslog as carrier for data infiltration. We describe a theoretical threat scenario in which an adversary makes use of known covert channels in TCP and DNS, and propose a novel method for hiding information in TCP ports scans and the resulting (sys)logs as a carrier for hidden messages. For forensic purposes, we provide details on Indicators-of-Compromise (IoC) as well as mitigating measures aiming at preventing the covert channel apriori. Moreover, we propose a novel detection scheme in order to identify and prevent such threats hidden in port scans and evaluate its effectiveness using datasets generated by a proof of concept implementation of the proposed covert channel based threat scenario.

1. Introduction

In recent years, network covert channels attracted more attention as novel malware generations increasingly use information hiding techniques, e.g. for stealthy infections, loading of further malware contents and data exfiltration (Jazi, 2021). Besides common techniques of network information hiding, e.g. modifications to the payloads or header of network packets or modulation of timing aspects, threat actors reportedly hide advanced attacks within seemingly harmless attacks or infections, e.g. by obfuscating the actual attack by using crypto currency mining malware (Microsoft Threat Intelligence Center (MSTIC), 2020) or ransomware (Microsoft Threat Intelligence Center (MSTIC), 2022). What might look like a common (and mostly harmless) malware infection turns out to be a highly targeted attack, penetrating deep into the organizations network. Based on those principles, we describe and evaluate a threat scenario using an indirect covert channel leveraging port scans in combination with syslog to infiltrate information into a firewall segregated network, e.g. to establish a Command and Control Channel (C2). The proposed covert channel is based on different carriers for embedding and retrieval, port scans for embedding and syslog for retrieval. However, syslog is a promising candidate for covert channels in many regards. Together with SNMP, syslog is one of most common ways to store, process and transfer log messages and is present in many IT-Systems. A fact which is leveraged by the proposed covert channel to persistently store information in the target network. As syslog network packets contain logging messages, the payload of these is mainly text, and therefore potentially subject to a subset of text steganography techniques. While this specific type of information hiding seems promising for some scenarios, we focus on other aspects of syslog, which make it interesting for covert channels: Logging messages are commonly created and stored on host systems, transferred over the network, and finally processed and stored at central points in a network – hidden messages therefore might be embedded at different points in this process as well as retrieved asynchronously at different points in time. The proposed covert channels make use of this circumstance to create an asynchronous hidden channel by encoding hidden information within the sequence of consecutive port scans, which eventually get logged and stored in the target network, accessible to other infected systems. The embedding of a hidden message is performed within a port scan. As a consequence of the port scan, the targeted firewall logs the knocked ports and forwards this information using syslog to a central logging server, e.g. to be then aggregated and correlated with other information in a Security Information and Event Management System (SIEM). Such syslog communication is then used as carrier for retrieval, for example in the case of a compromised log server or workstation. In this paper, we perform a threat analysis in order to evaluate plausibility of such scenarios, highlight indicators of compromise (IoC)

* Corresponding author.
E-mail address: kevin.lamshoeft@ovgu.de (K. Lamshöft).

https://doi.org/10.1016/j.fsidi.2022.301335

and discuss countermeasures. For the detection of the proposed covert channel, we propose a novel technique leveraging methods of transfer learning and deep convolutional neural networks (DCNN).

Following in Section 2 we provide background information regarding syslog and related work. In Section 3 we discuss how covert channels can be established using port scans and describe our chosen approach. In Section 4 we present and evaluate novel detection approaches for such covert channels. In Section 5 we discuss potential countermeasures, put the results in perspective and reflect on further covert channels regarding syslog. Section 6 concludes this works and gives a perspective on potential future work.

2. State of the art

In this section we provide background information regarding syslog and related work.

2.1. Syslog

The term *syslog* refers on the one hand to a logging service found on unixoid systems and on the other hand to the RFC3164 standard for storing and processing log messages and network protocol to transport them. In this work, we focus on the latter. Syslog was firstly specified in 2001 (Lonvick). The current standard (Gerhards) states that it should be send over UDP Port 514 (Okmianski) and that it should be encrypted with DTLS (Salowey et al., Feng). The transport over TCP is possible as well (Gerhards and Lonvick). Syslog can be cryptographically signed (Kelsey et al., Clemm). Together with SNMP, syslog is the most widely used logging protocol and a mapping mechanism between both protocols does exist (Marinov and Schoenwaelder). Syslog works in a strict client-server mode, where a client sends its syslog datagrams to a server without any response. Hence, no mechanisms for the detection of packet loss exist. The server can store and further process received datagrams. One syslog packet fits into one UDP packet. The size is practically only limited by what the implementations can handle and the size of one UDP datagram (max. 65,535 bytes). A packet has a header with a fixed size of one byte (five bits *facility* and three bits *message level*) and variable header with eight fields: *priority* (in the form $< 0..191 >$), *version* (max. two digits), *timestamp*, *hostname*, *app-name*, *procid* and *msgid*. The header is followed by a payload with *structured data* and a *message* with variable size. While the message can contain any form of text-information, the structured data is an array of possible well defined data-elements, each containing an *element*, an *id* and *params*. The header elements must be from ASCII charset, while the structured data and message can be from unicode charset. All header fields and even the payload are optional and not required to be present in a packet. Therefore, syslog is a highly variable protocol and implementation depended. Furthermore, the complete content can be seen as a raw character string send on-top of an UDP packet. Syslog timestamps need to be in accordance with RFC 3339 (Klyne and Newman) but can at maximum resolve to microseconds (six digits). RFC 5848 (Kelsey et al., Clemm) explicitly states that no countermeasures against covert channels have been taken in the syslog signing mechanism and that a variety of covert channels might be possible.

2.2. Related work

In 2016 Mohamed et al. (2016) published a work on covert channels realized by port scans, in which the sender opens network ports in a certain sequence and a receiver performs a port scan on those ports in order to retrieve the information based on ideas introduced in (Krzywinski, 2003). Forte et al. (2005) implemented a covert channel where syslog messages are send over DNS. Schmidbauer et al. used another common network logging protocol, SNMP, together with ARP to establish persistent covert channels (Schmidbauer et al., 2019), reflecting the concept of dead drops or secret steganographic storage (Wendzel et al., 2017) (Lamshöft et al., 2021). Alongside general considerations regarding covert channels in syslog and port scans, we describe and evaluate one selected covert channel in more detail which is using the order of port numbers to infiltrate a hidden message into a firewall segregated network.

3. Proposal for a covert channel using port scans and syslog

In this section we illustrate the covert channel design framed by a selected attack scenario and present a first proof-of-concept.

3.1. Covert channels in port scans

While a variant of techniques are summarized under the term *port scan*, this paper focuses on common TCP scans, which use TCP-SYN packets sent to different destination ports in order to identify potentially open ports of a target system. In the following, we describe a port scan as ordered list of ports p_i, which are tested one after another. As port scans induce a significant amount of additional traffic to a network, they provide several options to hide information. The trick here is, that port scans are very common in real-world scenarios and usually would not raise any suspicion to a systems owner. However, this fact let port scans seem like a viable option for hiding information in such. Multiple approaches to hide information seem likely, e.g. following timing channels T_i and storage channels S_i seem plausible:

T_1 Inter-Packet Timing: Modulation of the time delta d_t between two consecutive packets (p_i, p_{i+1})
T_2 Timing: Port scan at specific times (e.g. at midnight, every hour etc.)
S_1 Source Address: Modulation of the source address(es), e.g. encoding *within* the address or encoding by the *change* of address
S_2 Ports: Amount/order of source and destination ports or direct encoding within port number
S_3 Flags: Varying TCP flags
S_4 Timestamps: Modulation of TCP timestamp

However, besides these examples other known TCP covert channels might be applicable in a plausible way as well. Table 1 shows a comparison which known TCP covert channels would be applicable in the context of two scenarios with open source firewall implementations. By conducting preliminary experiments, we found the sequence of destination ports the most promising candidate due to its potential randomness and resulting entropy. Therefore, we chose to focus in this work on the destination port, worthwhile noting that in some cases other fields might yield comparable results. In the following, we describe the selected attack scenario leveraging the sequence of destination ports for establishing a covert channel for infiltration.

3.2. Threat scenario

The selected threat scenario assumes an adversary whose aim is to covertly infiltrate information into a target network, e.g. to load further malware components or to give instructions to an infected system. As common in the description of covert channels we use Alice to describe the covert sender and Bob as covert receiver,

Table 1
Comparison of information stored in syslogs of OPNSense and UFW in regards to TCP properties of known Covert Channels. 'S' indicates a storage channel, 'T' a timing channel. Most of the cited papers were summarized by (Mileva and Panajotov, 2014).

Paper	Channel	Type	UFW	OPNSense
Qu et al. (2004)	TTL	S	✓	✓
Mazurczyk and Szczypiorski (2009); Zander et al. (2006); Rowland	IP Fragments	S	✗	✗
Rowland	Sequence Number	S	✗	✓
Rutkowska (2004); Murdoch et al. (2005)	ISN	S	✗	✓
Giffin et al. (2002)	TCP Timestamps	S	✗	✗
Luo et al. (2009)	TCP ACK Number	S	✗	✗
Hintz (2003)	Urgent Pointer	S	✓	✓
Luo et al. (2008)	TCP loss & reordering	T	✗	✗
Mazurczyk et al. (2010); Mazurczyk et al. (2013)	TCP Retransmission	T	✓	✓
Luo et al. (2007)	TCP Timing	T	(✗)	(✗)

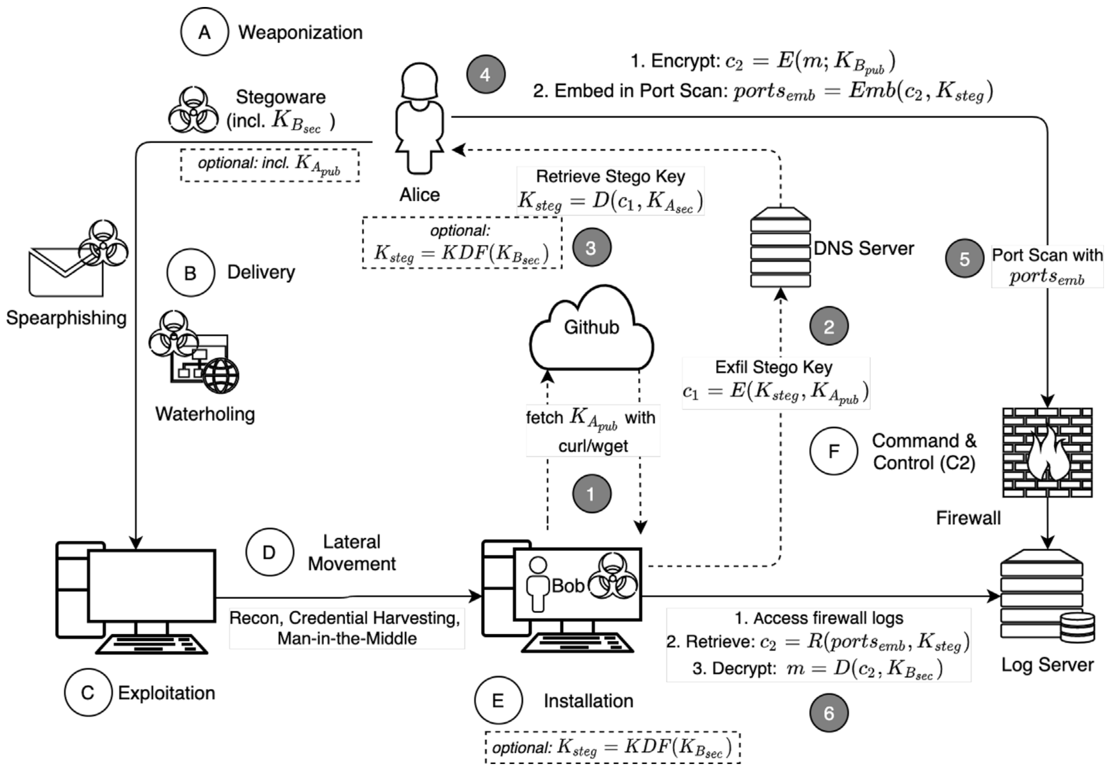
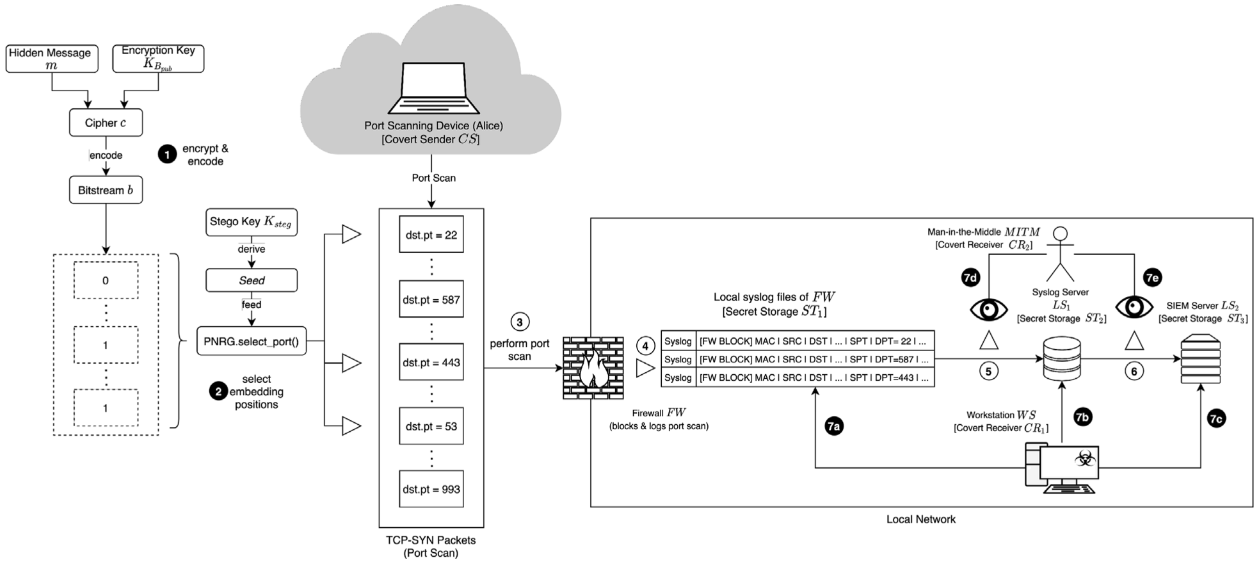

A Weaponization
Stegoware (incl. $K_{B_{sec}}$)
optional: incl. $K_{A_{pub}}$
4
1. Encrypt: $c_2 = E(m; K_{B_{pub}})$
2. Embed in Port Scan: $ports_{emb} = Emb(c_2, K_{steg})$
Alice
Retrieve Stego Key $K_{steg} = D(c_1, K_{A_{sec}})$
optional: $K_{steg} = KDF(K_{B_{sec}})$
3
DNS Server
5
Port Scan with $ports_{emb}$
Spearphishing
B Delivery
Waterholing
Github
2
Exfil Stego Key $c_1 = E(K_{steg}, K_{A_{pub}})$
fetch $K_{A_{pub}}$ with curl/wget
1
F Command & Control (C2)
Firewall
D Lateral Movement
Recon, Credential Harvesting, Man-in-the-Middle
Bob
1. Access firewall logs
2. Retrieve: $c_2 = R(ports_{emb}, K_{steg})$
3. Decrypt: $m = D(c_2, K_{B_{sec}})$
Log Server
C Exploitation
E Installation
6
optional: $K_{steg} = KDF(K_{B_{sec}})$

Fig. 1. Selected threat scenario following a shortened version of the Cyber Kill Chain (Hutchins et al., 2011) with steps (A)–(D) and the proposed covert C2 channel with steps (1)–(6). Dashed lines indicate optional or alternative methods.

respectively (see Fig. 1). We assume the adversary has the capability to compromise a system in the target network and is able to laterally move to be able to intercept the resulting syslog messages for receiving the covert message. As shown in Fig. 1, we use a shortened version of the cyber kill chain (Hutchins et al., 2011) with steps (A) Weaponization, (B) Delivery, (C) Exploitation, (D) Lateral Movement, (E) Installation and (F) Command and Control (C2) to describe the bigger picture of the threat scenario. The selected threat scenario is based on real world incidents and imitates techniques, tactics and procedures (TTPs) of actual threat actors. In this case, a basis for the proposed threat scenario can be seen in the recently discovered TinyTurla malware by the Turla APT group, which also includes a https backchannel (Unterbrink, 2021). The proposed covert channel is used in step (F) Command & Control and described by the numbered steps (1)–(6). However, Steps (A) to (E) from the cyber kill chain are required to be performed priory to be able to establish the C2 in the first place. It is worth mentioning that prior to these steps, thorough reconnaissance is required, e.g. performing port scans to test if any ports are ISP filtered or open and therefore would not be logged. In the following we will describe each step in more detail. In the weaponization phase (A) the malware is packaged, required asymmetric keys are generated and in the next step (B) delivered to the target. For the threat scenario we assume a first compromise of a system within the corporate network using common methods like spear phishing or water holing. After the initial exploitation (C), the stegoware is laterally moving using common reconnaissance techniques, credential harvesting or man-in-the-middle (MitM) attacks (see details in (Strom et al., Thomas)). It is worth noting, that the lateral movement might incorporate covert channels as well. Once the stegoware reached a target in step (D) with access to the (sys)logs (see Fig. 2 for more details), it installs on the target and initiates the procedures (1)–(6) to establish the covert C2 channel. In this threat scenario we model the malware with different capabilities, marked by the optional steps (1) and (2) as illustrated in 1. The first optional Step (1) (if internet access is available) is to fetch the asymmetric


Hidden Message m
Encryption Key $K_{B_{pub}}$
Cipher c
encode
1 encrypt & encode
Bitstream b
0
1
1
Stego Key K_{steg}
derive
Seed
feed
PNRG.select_port()
2 select embedding positions
Port Scanning Device (Alice) [Covert Sender CS]
Port Scan
dst.pt = 22
dst.pt = 587
dst.pt = 443
dst.pt = 53
dst.pt = 993
TCP-SYN Packets (Port Scan)
3 perform port scan
4
Firewall FW (blocks & logs port scan)
Local syslog files of FW [Secret Storage ST_1]
Man-in-the-Middle $MITM$ [Covert Receiver CR_2]
7d
7e
Syslog Server LS_1 [Secret Storage ST_2]
SIEM Server LS_2 [Secret Storage ST_3]
5
6
7a
7b
7c
Workstation WS [Covert Receiver CR_1]
Local Network

```
Syslog | [FW BLOCK] MAC I SRC I DST I ... I SPT I DPT= 22 I ...
Syslog | [FW BLOCK] MAC I SRC I DST I ... I SPT I DPT=587 I ...
Syslog | [FW BLOCK] MAC I SRC I DST I ... I SPT I DPT=443 I ...
```

Fig. 2. Embedding process with steps (1)–(2), port scan, logging and transfer of logs with steps (3)–(6) and the cover retrieval (7a) - (7d). Dark numbers indicate covert operations, light numbers overt traffic.

public key of Alice (the secret sender) $K_{A_{pub}}$ which is required to encrypt the messages that are sent to Alice (e.g. to secretly exchange shared secretes, like the Stego Key K_{steg}). For this scenario we use github.com as a cover – Github allows users to store their public SSH keys and retrieve them using http(s) under https://github.com/user.keys for example using `wget` or `curl`. In scenarios were such back-channels are not viable the alternative is to deliver the public key $K_{A_{pub}}$ directly within the Stegoware as illustrated in Fig. 1. This might make the initial compromise more obtrusive, but does not require a back-channel to the adversary (Alice) which might render a detection less likely. In the next steps (2) and (3) the stego key $\mathrm{K}_{\mathrm{steg}}$ (which drives the covert port scan channel) is either already delivered with the stegoware or generated on Bobs side, asymmetrically encrypted using $K_{A_{pub}}$ and exfiltrated towards Alice, in this scenario for example using a DNS covert channel (Bromberger, 2011). As said before, this is an optional step if DNS queries can be made. This step can be skipped if the stego key is already delivered with the malware itself. However, choosing one option over the other can be seen as a game-theoretic problem – either the initial compromise and lateral movement are more obtrusive or additional network communication such as the exfiltration of the stegokey is required. Depending on the adversaries knowledge on the target environment and potential countermeasures one option might be preferable to the other in a game theoretic sense. In step (4) Alice encrypts the hidden message (for example further malware modules or C2 instructions) using the public key of Bob $K_{B_{pub}}$ and embeds the message within a port scan that she performs against the corporate firewall (the embedding process is described in section 3.3). The port scan gets logged by the firewall separating the corporate network from the internet and detailed information is sent by the firewall via syslog to a central log server. In Step (6) Bob accesses the syslog files to retrieve the hidden message using the stego key $\mathrm{K}_{\mathrm{steg}}$ and decrypts the message using his secret key $K_{B_{sec}}$. However, due to the use of syslog as carrier, multiple scenarios or options for retrieval (Step (6)) can be expected as illustrated in Fig. 2. How embedding in the port scan works in detail and how the message is retrieved is described in the following section.

3.3. *Covert channel design: Embedding & retrieval*

The embedding and retrieval processes are illustrated in more detail in Fig. 2. In this first scenario we choose to embed in the destination port of a TCP port scan. In scenarios requiring higher bandwidths the covert channel can be expanded using additional sub-carrier (sacrificing stealthiness for bandwidth). In the case of TCP port scans, the TCP timestamp is a promising candidate (e.g. see (Hildebrandt et al., 2020) for reference) as well as the TCP source port, as it appears to be random as it is the case for the destination port (however, in our lab we see inconsistent behavior between different Operating Systems, which require further investigations). This sub-carrier could provide up to additional 16 bit per port. For the proof-of-concept we focus on the destination port though. In the first step, the Covert Sender *CS* (Alice) generates a bit sequence by encoding and encrypting a hidden message *m* with the public key of Bob $K_{B_{pub}}$. As stated before the main driver of the covert channel algorithm is the stego key K_{steg} which is a shared secret between Alice and Bob which is either pre-shared or generated by Bob and exfiltrated towards Alice. As shown in Fig. 2, in Step 2 K_{steg} is used to drive the embedding function. K_{steg} is used as a seed for a pseudo-random-number-generator (PRNG) which is used to select a (pseudo-random) subset of ports in the sequence of ports. This part of the algorithm is vital as it scatters the hidden message seemingly random across the entire spectrum of the port scan, aiming to render detection less likely. In Step 3 this new ordered list of ports to be scanned is used to perform a port scan against the targeted firewall *FW*, which separates the covert sender's network from the receiver network. For the retrieval we see several plausible scenarios. For security-focused corporate and industrial networks it is common to have a Security Incident and Event Management (SIEM) in order to get insights and have alerting for security-relevant events. Therefore, it is common to log firewall events and store them on central logging servers, e.g. using syslog and SNMP. We use this common infrastructure to illustrate five different options for covert retrieval of the injected information. The proposed covert channel is specifically designed for this purpose. The selected retrieval scenarios are illustrated in Fig. 2. In Step 4 the firewall blocks the port scan and creates a local log file. In Step 5, the firewall logs are transmitted via syslog to the central logging server LS_1. In Step 6 logs are aggregated in a SIEM LS_2. Steps 7 a)-e) mark the five different options for retrieval: In case of 7a, a user or computer (WS) accesses the firewall logs and retrieves the hidden message m embedded in the port number sequence. In case of 7b the logs are retrieved from the central logging server LS_1, in scenario 7c from the SIEM server (LS_2). Scenarios 7d and 7e are MitM approaches, where the transmission of the syslogs are intercepted either in a passive way by a compromised network element (e.g. switches, firewalls etc) or in active way using MitM attacks, e.g. ARP spoofing. In the following we provide details on how we implemented and tested this approach.

3.4. Proof of concept

For the proof of concept we use nmap, a common network security tool, to perform the TCP port scans. The covert channel is designed to use the order of destination ports. In default, nmap uses a randomized order of ports, providing potentially enough entropy to allow for secure embedding of hidden messages. The exact understanding of how this randomization works is necessary to reduce the risk of detection. The logging behavior of the utilized firewall is important in order for the covert channel to work. The ability to create syslog logs and send them to a remote server as well as the frequency and verbosity of the logs decides whether the firewall can be abused for the covert channel. Two common open source firewalls are tested for this purpose: Linux UFW[1] (FW_1) and OPNsense[2] (FW_2). UFW is a user-friendly front-end for configuring iptables and therefore can be used as host as well as network firewall. By default FW_1 does not log any activity. In case logging is enabled, different log levels can be distinguished[3]: On *low* and *medium* blocked packets are logged but with rate-limits. Only on *high* and *full* every blocked packet is logged without limitations. In practice only the lower log levels are found in UFW deployments, since the size of logs exceeds acceptable limits otherwise. With default settings, FW_2 creates a log-entry for every blocked packet, without rate limits, including Ports, IP addresses, used protocol, flags and timestamps (see Table 2 for a comparison with UFW). This allows for multiple different covert channels without any information loss due to rate-limiting. The sending to a remote server can easily be enabled as well.

We utilize the default nmap scan of the most common TCP ports without an initial ping to establish the covert channel.[4] In this scan, nmap will send a TCP-SYN packet to 1,000 common TCP ports in a randomized order. To adapt the covert channel to the default behaviour of nmap, we analyze the source code of nmap[5] in order to understand the exact functionality of the scan and the randomization. It turns out that nmap uses an undocumented feature that puts the most common 28 TCP ports in a random order at the begin of the scan. A module called *nbase_rnd.c* is responsible for all randomization in nmap. It relies on UNIX *urandom*[6] function which creates random numbers with high entropy that could even be used for cryptography use-cases. Therefore, it can be expected that the distribution of the 1,000 random ports has an high entropy as well. Since the first 28 ports are always the same but in randomized order the covert channels ignores these first 28 positions and uses the remaining 972 ports to embed a hidden message. For

Table 2
Comparison of stored information in syslog files of OPNSense and UFW firewalls.

Information	UFW	OPNSense
Timestamp	✓ (sec.)	✓ (sec.)
Protocol	✓	✓
Source IP	✓	✓
Destination IP	✓	✓
Source Port	✓	✓
Destination Port	✓	✓
Sequence number	✗	✓
ACK number	✗	✗
Pointer	✗	✗
Window Size	✓	✗
Packet Length	✓	✓
Checksum	✗	✗
Urgent Pointer	✓	✓
Data Offset	✗	✓
Type of Service (TOS)	✓	✓
TTL	✓	✓
IP Flags	✓ (DF)	✓
IP ID	✓	?

sake of simplicity the implemented channel uses an odd/even scheme for destination ports of the scan to encode hidden bits and no error detection or correction is implemented. Therefore, one bit per scanned packet is transferred. However, several more complex encodings are plausible as well, e.g. a binary encoding within the port number (16 bit) or the use of cross sums and similar. We choose the significant lower capacity in order to adapt to nmap's entropy and decrease detection probability. In order to evaluate detection performance in regards to different bandwidths, we generated four datasets with each 1000 port scans (with each 1000 scanned ports) using different bandwidths (amount of altered ports): D_0 with no manipulations (overt traffic), D_{256} with 256 altered ports (equal to 256 hidden bits), D_{512} with 512 bit and D_{768} with 768 bit per scan. The expectation would be, that with increased bandwidth the detection rates would increase, too. This can be seen as a general concept of information hiding, that the amount of modification of a carrier relates to the probability of detection. This principle is often referred to as *steganographic cost*. As stated before we use a PRNG to determine which ports are used to encode the hidden bits. We seed the PRNG with the stegokey K_{steg} which enables Bob to receive and decode the message. For the proof of concept K_{steg} is implemented as 16bit integer, but can be extended to arbitrary long strings (while still using each bit to increase entropy).

Since the nmap source code is not well documented the covert channel logic is implemented in python. The encoding, decoding, encryption and randomization is performed in the program. On the sender side it creates a list of ports as output which is then passed as a parameter list to nmap. nmap has no feature that allows to take a given list of ports and then performing the scan with the given order (the given ports are either fully randomized or sorted in ascending order). We implement such feature in the source code and compile our own nmap version for this purpose. This combination of python and nmap has the advantage that the covert channel logic can be implemented completely independent from nmap, while not changing the nmap specific behaviour (timing, retransmission and other aspects of the single TCP packets). On the receiver side a python program can read and parse syslog log files from FW_1 and FW_2. It extracts all relevant ports from the log files. Regarding the integrity of the generated dataset it can be noted, that by using FW_2 no information loss could be observed. It correctly logged every scanned port. As in the proof-of-concept TCP-SYN scans are used, the proposed covert channel is not affected by potential rate limits of the firewall. In the next section

[1] https://wiki.ubuntu.com/UncomplicatedFirewall, accessed 2021/03/08.
[2] https://opnsense.org/, accessed 2021/03/08.
[3] https://manpages.ubuntu.com/manpages/bionic/man8/ufw.8.html, accessed 2021/03/09.
[4] The command reads *nmap -v -Pn \{ipAddress\}*
[5] written in C, https://github.com/nmap/nmap, accessed 2021/03/09.
[6] https://manpages.ubuntu.com/manpages/hirsute/en/man4/random.4.html, accessed 2021/03/09.

we present our new detection approach for the proposed covert channel.

4. A first DCNN based detection approach

In this section we present an approach based on a deep convolutional neural network (DCNN) and transfer learning to detect the novel covert channel using port scans. We present the concept of the detection approach in Section 4.1 and its evaluation in Section 4.2.

4.1. Concept of detector

In order to detect the novel covert channel using port scans, we perform an extensive analysis on recorded network data. We tried different classic pattern recognition based detection approaches with handcrafted features (in timestamps, time-deltas, port scans) which failed to detect those anomalies, as we were not able to design features with enough discriminatory power to determine relevant patterns to train a lean classifier, for example a decision tree. Thus, we decided to follow the current trend in machine learning to use the power of DCNN architectures as classification engine with its self-learned features, being fully aware of lacking explainability. The rationale for this choice is to try to let the neural network learn a feature space to see in this first step whether the embedding could be detected at all. If the detection would be successful, in a second step we would try to engineer hand-crafted feature spaces to replace the learned features, using the knowledge generated with neural networks to guide this process in future work. We decide to use InceptionV3 (Szegedy et al., 2016) as DCNN architecture to detect anomalies caused by the novel covert channel because of its very promising performance results in *Imagenet Large Scale Visual Recognition Challenge* ILSVRC 2012 (Russakovsky et al., 2015). DCNN are mostly designed for image classification in the wild and their architectures are optimized for this purpose. To use it for our purpose we have to use transfer learning.[7] For transfer learning, we have to pre-process our data and we have to make some small adjustments on the used DCNN architecture. As a feature for classification, we want to use the occurrence of the 1000 ports (where a different number of ports are used for embedding). We represent these ports by occurrence in an 32×32 two dimensional array as a kind of noise pattern. The dimensionality of the two dimensional array means we have 24 slots ($32 \times 32 = 1024$) in the array we do not use, these slots are set to 0. We save this array in a 16-bit .png file format, which so results in grayscale image which represents the distribution of the ports. Since DCNN architectures are primary trained with jpeg files, we convert those arrays in .jpeg file format with maximum quality. Furthermore, we rescale the 32×32 matrix to 299×299 because the DCNN architecture was initially trained with images of this resolution and the architecture of the network is optimized for this resolution. As mentioned before, to use InterceptionV3 for transfer learning, we have to make some adjustments on the networks architecture. Basically, a DCNN has an feature extraction part and a classification part that is build with fully connected layers. For transfer learning we use the existing feature extraction part and replace the classification part of the model (Brownlee), because initially it has 1000 output neurons (which means output classes) to classify images in the wild. Thus, we keep the weights of all convolutional layers and only replace the fully connected layers (classification part of the model after model flattening) that will learn by training to interpret the features extracted. Additionally, we replace the initial output neurons (1000 output neurons to classify images in the wild) by only one output neuron with sigmoid[8] activation function, in order to make a binary decision ($0 =$ non-steganographic or $1 =$ steganographic). We evaluate the described DCNN based detection approach in the section subsection.

Table 3
Classification Results of DCNN based Detector on its training data.

Number of (correct classified) Samples	Accuracy	Training Set
(900) 900	100.0%	D_0 (Class 0)
(806) 900	89.56%	D_{256} (Class 1)
(796) 900	88.44%	D_{512} (Class 1)
(820) 900	91.11%	D_{768} (Class 1)

4.2. Evaluation of detector

For the evaluation of our previously described DCNN based detection approach we create one training and multiple test data sets. One *sample* of a data set contains 1000 port numbers sorted by occurrence and saved in a .jpeg file as described in the previous section. Our detector is trained with 2 classes (Class $0 =$ non-steganographic and Class $1 =$ steganographic). For our training data, we use 900 non-steganographic samples from D_0 (see Section 3.4) and 2700 steganographic samples (900 samples each of D_{256}, D_{512} and D_{768}). The four test data sets contain only completely independent samples from training data and can be summarized as follows:

- Test set TS_{D_0} with 100 test samples from D_0,
- Test set $TS_{D_{256}}$ with 100 test samples from D_{256},
- Test set $TS_{D_{512}}$ with 100 test samples from D_{512} and
- Test set $TS_{D_{768}}$ with 100 test samples from D_{768}.

Thus, the test sets $TS_{D_{256}}$, $TS_{D_{512}}$ and $TS_{D_{768}}$ belonging to class 1 (steganographic data) and test set TS_{D_0} belongs to class 0 (non-steganographic data).

The DCNN based detector was trained in 8 epochs and a batch size of 64. In an initial test, the DCNN was able to classify 92.26% of the training samples correctly. It shows, the detector is able to classify 89.68% of malicious training data correctly (true positive rate) by a false positive (alarm) rate of 0% (100% true negative rate). We visualize the results on training data in Table 3.

The classification results on test data are significantly less accurate and are visualized in Table 4. Overall, the DCNN based detection approach can classify 60.5% of test samples correctly. The approach classifies 53% of authentic data correctly and 63% of malicious data. Thus, the following common success and error rates for the approach can be derived on test data: 63% true positive rate (TPR), 53% true negative rate (TNR), 47% false positive rate (FPR) and 37% false negative rate (FNR).

It can be emphasized that the embedding bandwidth has an impact for the detection approach. The accuracy increased around 10 percent for the bandwidth of 512 and 768 bit in comparison to 256 bit. Furthermore, the results show, that the detector has some shortcomings on independent test data, possibly related to overfitting on training data or there are no reliable detectable patterns in the data especially on data with a low bandwidth. We have visualized the significantly decreasing performance on test data in Fig. 3. In summary, the results show the feasibility of the covert

[7] "Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned." (Torrey and Shavlik).

[8] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

Table 4
Classification Results of DCNN based Detector on test data.

Number of (correct classified) Samples	Accuracy	Test Set
(53) 100	53%	TS_{D_0} (Class 0)
(54) 100	54%	$TS_{D_{256}}$ (Class 1)
(68) 100	68%	$TS_{D_{512}}$ (Class 1)
(67) 100	67%	$TS_{D_{768}}$ (Class 1)

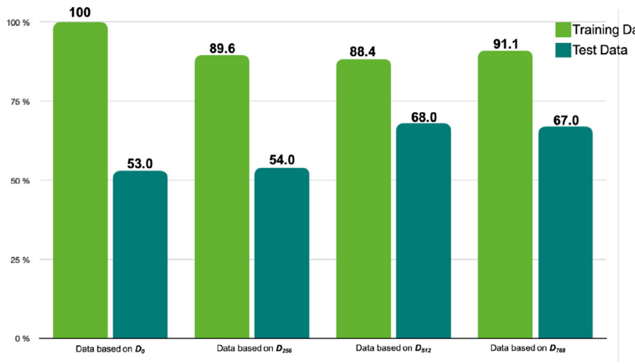


Fig. 3. Visualization of percentage share of correct classified samples on training and test data sets.

channels approach to hide information in the sequence of destination ports. The pseudo-random cover selection using the stego key as a seed for the PRNG seems to provide enough entropy to show no significant patterns in the embedding scheme to be reliably detectable. However, we hope to achieve more accurate results in future work with more available training and test data, e.g. using different setups with other port scanning software and techniques as well as other available DCNN architectures.

5. Discussion

In this section we discuss limitations, potential indicators of compromise, countermeasures and the potential of syslog for other covert channels.

5.1. Limitations

The proof-of-concept implements three options for bandwidth selection (256 bit, 512 bit, 768 bit) per port scan where one bit is represented by one TCP packet. In scenarios where more bandwidth is required, for example to load second stage malware modules, the bandwidth could easily be increased using other encoding schemes or using sub-carriers, such as the source port or timestamps. This increase in bandwidth comes with further modifications to the carrier (steganographic cost) which might result in higher probabilities of detection. In order to transfer hidden information properly in the implemented covert channel, the logging behavior of the firewall must be known. In this work, two different firewalls (UFW and OPNSense) were tested. Other firewalls might show a less verbose logging behavior. In that case the shown covert channel might not yield the same results. The selected threat scenario requires a prior infection of a system within the perimeter and uses lateral movement to be able to retrieve the required logs. However, in the context of recent malware findings such kill chains seem plausible for stealthy operations performed by Advanced Persistent Threats (APT) (Unterbrink, 2021).

5.2. Indicators of compromise

The threat scenario in this paper can be divided into two stages: The first stage is the initial compromise based on current malware scenarios (e.g. (Unterbrink, 2021)) without the proposed covert C2 channel, the second stage then is the actual covert channel for command and control communication. As the first stage of the threat scenario is designed in the context of malware found in the wild, in practice IoCs are dependent on the actual used malware. Therefore, we can only provide general indicators that would apply to most scenarios:

- Delivery: Spearphishing and Waterholing Attacks leave traces on the infected systems (e.g. see (Strom et al., Thomas))
- Exploitation: Traces on OS level (e.g. file structures, process injections etc.)
- Lateral Movement: Traces in network traffic and switch/firewall logs (e.g. port scans, ARP poisoning etc.)
- Installation: https call to `github.com/user.keys.`
- C2: DNS queries to uncommon servers

For the second stage including the proposed covert C2 channel we can see following indicators: (IoC-1) multiple consecutive port scans against a firewall from the same IP address (though an adversary might counter this using different source addresses), (IoC-2) ARP cache poisoning in the case of the MitM scenario based retrieval, (IoC-3) firewall log access or syslog log access from uncommon clients. Due to different retrieval scenarios the IoCs do vary accordingly. However, by employing certain countermeasures the selected threat scenario can be mitigated beforehand.

5.3. Countermeasures

The presented covert channel exploit features of network security themselves, which makes countermeasures contra-intuitive: In case multiple port-scans from suspicious IP-addresses are detected, the logging of the scans and adversary activities is a basic measure for further investigations. To mistrust the created log files is beyond default security policies. Hence, the most effective active countermeasure against the presented covert channel would be to change the logging behavior of the firewall, e.g. just logging that a port scan was detected rather than logging each scanned port individually. However, such measures lead to loss of information, which might be necessary for forensic investigations. In addition to that, it only suppresses certain covert channels. Still the aim remains to reduce the entropy and redundancy in logging to avoid covert channels. Further methods are normalization of logs (e.g. only lower case encoding) and restrictive authenticated and monitored access to logs.

5.4. Other covert channels in syslog

In this paper, we used syslog mainly as a means for retrieval. However, it's structure and broad usage in common networks make syslog a promising candidate for other types of covert channels as well. What makes syslog so different from other protocols is its temporal and locational aspect: Syslogs are partly persistent in the system. We can differentiate three levels of persistence here: (P1) short-term persistence on host (log rotation, e.g. a week), (P2) long-term persistence on logging server (weeks, months, years) and (P3) ephemeral on transmission within the network (no persistence). This is especially reflected in the retrieval options described in the attacker model in Section 3.2: Step 7a is targeting the short-term persistence on the host (the firewall), 7b and 7c the long-term persistence on the logging servers, and Steps 7d and 7e by performing MitM attacks on the ephemeral transmission of logs over Ethernet. This is directly connected to the locational aspect. The same information (in our case the sequence destination ports) is located (at the same time) at different positions within the system (host, server and network), which enables different scenarios for retrieval. Another aspect of syslog is the possibility of indirect as well as direct covert channels. The proposed covert channel in this paper is an indirect channel, in which a device is triggered from the outside, which results in a set of logs — which encode a hidden message, which then is retrieved at later point in time. On the other hand, direct channels e.g. the communication between a logging host and a syslog server can be used as well for channels by encoding a hidden message within a log, e.g. by methods of Text Steganography.

6. Conclusion

In this paper, we performed a threat analysis for a novel indirect and asynchronous covert channel based threat scenario leveraging port scans and syslog as cover. We demonstrated how logging mechanisms can be used to persistently store hidden information in target networks. With a dynamic pseudo-random cover selection scheme using a steganographic key we could show that for carriers with enough entropy (in this case TCP destination ports in port scans) the steganographic cost can be reduced to zero, rendering reliable detection impossible. When doubling the steganographic capacity from 256 bit to 512 bit we see a slight increase in detection accuracy. However, the detection accuracy does not significantly increase when we increase capacity to 768 covert bits. For better understanding of this relationship between capacity, detection accuracy and steganographic cost further investigations are required. For detection, we propose a novel innovative detection approach using DCNNs. The approach uses the order of port scans to generate 16-bit .png image files which represent the port scan order. With the help of transfer learning, we use these files to train a DCNN based detection approach which is able to differentiate between authentic and malicious port scans. A first and brief evaluation of the detection approach shows an accuracy of around 92% on training and around 61% on test data. Port scans and logging mechanisms show several opportunities for information hiding. In this paper we shed light on these opportunities and performed an extensive analysis for one of these. In the future further potential covert channels using similar scenarios should be investigated. However, with small adjustments in logging behaviour most covert channels can be mitigated. Furthermore, we provide a first set of Indicators-of-Compromise. In addition, we provided a first indication for other covert channels in syslog, which shall be investigated in future work. To improve detection, we plan to use the knowledge generated with learned features to generate a set of hand-crafted features to detect the introduced covert channel. This allows us to use the benefits of DCNNs architecture to derive in the next step better explainable feature sets with more accurate results.

Acknowledgements

The presented work is funded by the German Federal Ministry of Economic Affairs and Energy (BMWi, project no. 1501589A and 1501589C) in the framework of the German reactor safety research program. The authors thank Agostino Moosdorf for his help as well as the reviewers for their comprehensive feedback.

References

Bromberger, S., Jan., 2011. DNS as a Covert Channel within Protected Networks. National Electronic Sector Cyber Security Organization (NESCO).

J. Brownlee, Develop a deep convolutional neural network step-by-step to classify photographs of dogs and cats, Deep Learning for Computer Vision. URL https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/.

Forte, D.V., Maruti, C., Vetturi, M.R., Zambelli, M., 2005. Secsyslog: an approach to secure logging based on covert channels. In: First International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'05), pp. 248–263.

R. Gerhards, The Syslog Protocol, Internet Engineering Task Force https://tools.ietf.org/html/rfc5424.

R. Gerhards, C. Lonvick, Transmission of Syslog Messages over TCP, Internet Engineering Task Force https://tools.ietf.org/html/rfc6587.

Giffin, J., Greenstadt, R., Litwack, P., Tibbetts, R., 2002. Covert Messaging through tcp Timestamps, vol. 2482, pp. 194–208.

Hildebrandt, M., Lamshöft, K., Dittmann, J., Neubert, T., Vielhauer, C., 2020. Information hiding in industrial control systems: an OPC UA based supply chain attack and its detection. In: Proceedings of the 2020 ACM Workshop on Information Hiding and Multimedia Security. IH& MMSec '20, Association for Computing Machinery, New York, NY, USA, pp. 115–120. https://doi.org/10.1145/3369412.3395068.

Hintz, A., 2003. Covert channels in tcp and ip headers. In: DefCon 10, Las Vegas. Nevada, August 2 - 4, 2003. https://defcon.org/images/defcon-10/dc-10-presentations/dc10-hintz-covert.pdf.

Hutchins, E.M., Cloppert, M.J., Amin, R.M., 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Lead. Issues Inf. Warf. Secur. Res. 1 (1), 80.

Jazi, H., Apr. 2021. Lazarus APT conceals malicious code within BMP image to drop its RAT. URL https://blog.malwarebytes.com/malwarebytes-news/2021/04/lazarus-apt-conceals-malicious-code-within-bmp-file-to-drop-its-rat/.

J. Kelsey, J. Callas, A. Clemm, Signed Syslog Messages, Internet Engineering Task Force https://tools.ietf.org/html/rfc5848..

G. Klyne, C. Newman, Date and Time on the Internet: Timestamps, Internet Engineering Task Force https://tools.ietf.org/html/rfc3339..

Krzywinski, M., 2003. Port knocking: network authentication across closed ports. SysAdmin Mag. 12, 12–17, 2003.

Lamshöft, K., Neubert, T., Krätzer, C., Vielhauer, C., Dittmann, J., 2021. Information hiding in cyber physical systems: challenges for embedding, retrieval and detection using sensor data of the SWAT dataset. In: Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, IH& MMSec '21. Association for Computing Machinery, New York, NY, USA, pp. 113–124. https://doi.org/10.1145/3437880.3460413.

C. Lonvick, The BSD Syslog Protocol, Internet Engineering Task Force https://tools.ietf.org/html/rfc3164.

Luo, X., Chan, E.W.W., Chang, R.K.C., 2007. Cloak: a ten-fold way for reliable covert communications. In: Biskup, J., López, J. (Eds.), Computer Security — ESORICS 2007. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 283–298.

Luo, Xiapu, Chan, E.W.W., Chang, R.K.C., 2008. Tcp covert timing channels: design and detection. In: 2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN), pp. 420–429.

Luo, X., Chan, E.W.W., Chang, R.K.C., 2009. Clack: A Network Covert Channel Based on Partial Acknowledgment Encoding. IEEE International Conference on Communications, pp. 1–5, 2009.

Microsoft Threat Intelligence Center (MSTIC), Nov. 2020. Threat actor leverages coin miner techniques to stay under the radar — here's how to spot them. URL https://www.microsoft.com/security/blog/2020/11/30/threat-actor-leverages-coin-miner-techniques-to-stay-under-the-radar-heres-how-to-spot-them/.

V. Marinov, J. Schoenwaelder, Mapping simple network management protocol (SNMP) notifications to SYSLOG messages, Internet Engineering Task Force https://tools.ietf.org/html/rfc5675.

Mazurczyk, W., Szczypiorski, K., 2009. Steganography in handling oversized ip packets. In: 2009 International Conference on Multimedia Information Networking and Security, vol. 1, pp. 559–564. https://doi.org/10.1109/MINES.2009.246. URL.

Mazurczyk, W., Smolarczyk, M., Szczypiorski, K., 2010. Retransmission steganography applied. In: 2010 International Conference on Multimedia Information Networking and Security, pp. 846–850.

Mazurczyk, W., Smolarczyk, M., Szczypiorski, K., 2013. On information hiding in retransmissions. Telecommun. Syst. 52 (2), 1113–1121.

Microsoft Threat Intelligence Center (MSTIC), 2022. Destructive malware targeting Ukrainian organizations. https://www.microsoft.com/security/blog/2022/01/15/destructive-malware-targeting-ukrainian-organizations/. (Accessed 19 January 2022).

Mileva, A., Panajotov, B., 2014. Covert channels in tcp/ip protocol stack - extended version-. Cent. Eur. J. Comput. Sci 4, 45–66.

Mohamed, E.E., Mnaouer, A.B., Barka, E., 2016. Pscan: a port scanning network covert channel. In: 2016 IEEE 41st Conference on Local Computer Networks (LCN), pp. 631–634.

Murdoch, S.J., Lewis, S., 2005. Embedding covert channels into tcp/ip. In: Barni, M., Herrera-Joancomartí, J., Katzenbeisser, S., Pérez-González, F. (Eds.), Information Hiding. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 247–261.

A. Okmianski, Transmission of syslog messages over UDP, Internet Engineering Task Force https://tools.ietf.org/html/rfc5426..

Qu, Haipeng, Su, Purui, Feng, Dengguo, 2004. A typical noisy covert channel in the ip protocol. In: 38th Annual 2004 International Carnahan Conference on Security Technology, pp. 189–192, 2004.

C. H. Rowland, Covert Channels in the TCP/IP Protocol Suite, DoIS Documents in Information Science doi:10.5210/fm.v2i5.528.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., 2015. Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. 115 (3), 211–252. https://doi.org/10.1007/s11263-015-0816-y.

Rutkowska, J., 2004. The implementation of passive covert channels in the linux kernel. In: Chaos Communication Congress, 27-29 December, Berlin. https://events.ccc.de/congress/2004/fahrplan/files/319-passive-covert-channels-slides.pdf.

J. Salowey, T. Petch, R. Gerhards, H. Feng, Datagram Transport Layer Security (DTLS) Transport Mapping for Syslog, Internet Engineering Task Force https://tools.ietf.org/html/rfc6012..

Schmidbauer, T., Wendzel, S., Mileva, A., Mazurczyk, W., 2019. Introducing dead drops to network steganography using arp-caches and snmp-walks. In: Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES '19. Association for Computing Machinery, New York, NY, USA.

B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, C. B. Thomas, MITRE ATT&CK^{TM} : Design and Philosophy..

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. URL. http://arxiv.org/abs/1512.00567.

L. Torrey, J. Shavlik, Transfer Learning, University of Wisconsin, Madison WI, USA. URL ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf.

H. Unterbrink, TinyTurla - Turla deploys new malware to keep a secret backdoor on victim machines. URL http://blog.talosintelligence.com/2021/09/tinyturla.html.

Wendzel, S., Mazurczyk, W., Haas, G., 2017. Don't you touch my nuts: information hiding in cyber physical systems. In: 2017 IEEE Security and Privacy Workshops (SPW). IEEE, pp. 29–34.

Zander, S., Armitage, G., Branch, P., 2006. Covert channels in the ip time to live field. In: Australian Telecommunication Networks & Applications Conference (ATNAC). Australia, December 2006, ISBN/ISSN 0977586103.