Covert Channel Techniques

- FIV, Nov 2021
- KM, Oct 2022

Introduction

Each available technique to inject and/or extract covert channels must have its individual python script within the `~/CCgen.v2/ccgen/techniques` folder. Within ccgen, techniques are identified by the name of the python file; for instance, the covert timing channel technique that follows the principles published by Berk et al. in [1] is always addressed as `timing_ber.py` and is implemented in the `~/CCgen.v2/ccgen/techniques/timing_ber.py` file.

Configurations

Due to multiple options for covert channel this generator has to be configured to run adequately and successfully. A set of configurations are preinstalled in the default database.
In general, ccgen configurations can be separated in three parts: General, Filters, Channel & Mapping.
In the section **General**, general data like config name, input/output file paths are configured.
In the section **Filters**, data about the connection, where the covert channel shall be implemented, are defined.
In the section **Channel & Mappings** the technique of the covert channel is defined.

Mappings

To be usable by ccgen, each technique requires a defined mapping. This mapping contains the name (only for db distinction), the technique name, the layer (IP, TLS, PCAP), bits of covert channel per packet and a set of parameters and value mappings.
You will find a set of mapping files in the default database. Examples how they are used, you can find in the section **Implemented techniques in CCgen.v2** below.

PROF

Parameters

Inside the mapping section of the *CCgen.v2 Configurator* one can define parameters, which are used in the given technique file.

Value mappings

Additionally, in the *CCgen.v2 Configurator* one can define specific value mappings. Normally, bits are converted directly to integer (e.g. "b'00101" -> 5). With a defined value mapping inside the mapping section of the configurator, one can map the bit value "b'00101" to a different value (e.g. 23).

Implemented techniques in CCgen.v2

CCgen.v2 comes with a varied set of possible techniques to inject covert channels. We arrange them in subgroups according to the classification given in [2]:

- Value to symbol (v2s)
- Ranges as symbols (r2s)
- Containers (con)
- Derivative (dev)
- Covert timing channels (ctc)

Value to symbol (v2s)

In a value to symbol correspondence either one or multiple field values can correspond to the same symbol of a covert message. For example, if the size of IP packets is used to contain a binary covert message, a packet with size 40-byte size could mean '0', whereas a packet with size 60-byte could stand for '1'.

IP Flags (#1)

The cc technique implemented in **techniques/ipflags.py** uses the *Reserved* bit and the *Don't Fragment* bit of the *IP Flags* field to hide a binary covert channel. This technique is inspired in the one published by Ahsan and Kundur in [7].

- Config: *ipflags_v2s*
- Mapping: *mapping_bin*
- Bits: *1*

IP Identification (#2)

The cc technique implemented in **techniques/ipid.py** uses the 8-highest bits of the *Identification* field to hide a covert channel. It additionally clears the *Don't Fragment* bit of the *IP Flags* field. This technique has been proposed in several publications, e.g. [6, 4].

- Config: *ipid_v2s*
- Mapping: *mapping_8bits*
- Bits: 8

PROF

IP Length (#3)

The cc technique implemented in **techniques/iplen.py** uses the size of the IP *Total Length* field to hide a covert channel. The mapping requires a "poff" parameter containing the offset (in bytes) to take as minimum length (it must be higher than 20). The implementation follows the method published in [8].

- Config: *iplen_v2s*
- Mapping: *mapping_8bits_off50*
- Bits: 1-8
- Parameters:
    - *poff* (value offset)

IP Protocol (#4)

The cc technique implemented in **techniques/ipproto.py** appears in [10]. It uses different values of the *Protocol* field to hide covert values. Note that for this technique to work properly the configured flowkey must be 2tuple (*srcIP, dstIP*).

- Config: *ipproto_v2s*
- Mapping: *mapping_ipproto_2bit*
- Bits: 1-2
- Valuemappings:
    - '00' -> 1
    - '01' -> 4
    - '10' -> 6
    - '11' -> 17

IP Type of Service (#5)

In [11] Postel proposes using the unused bits of the *Type of Service* field of the IP datagram to convey hidden information. This cc technique is implemented in **techniques/iptos.py** and conveys six bites per packet, since the less significant bits are reserved.

- Config: *iptos_v2s*
- Mapping: *mapping_6bits*
- Bits: 6

TCP/UDP source port (#6)

In [12] Gimbi et al. present different ways of using the *Source Port* field of the TCP or UDO datagrams to convey ASCII symbols. The method implemented in **techniques/srcport.py** is a simplification that directly maps port numbers with binary encoding of ASCII symbols. The mapping requires a "poff" parameter that accounts for an offset to avoid low-range source ports. Note that, to ensure a correct cc communication, the flowkey must be ideally a 3tuple (*srcIP, dstIP, protocol*), where the protocol is TCP or UDP. The original, base method proposed in [12] appears among the derivative. Note that for this technique to work properly the configured flowkey must be 3tuple (*srcIP, dstIP, Protocol*), ensuring that the protocol used is TCP or UDP (*tcp*, *udp*, or *tcp/udp* options in the *const* field when using the *ccGen-wrapper*).

PROF

- Config: *srcport_v2s*
- Mapping: *mapping_8bits_off1k*
- Bits: 8
- Parameters:
    - *poff* (value offset)

TTL (#7)

The cc technique implemented in **techniques/ttl_v2s.py** uses the *TTL* field of the IP datagram to hide a binary covert channel as introduced in [15].

- Config: *ttl_v2s*
- Mapping: *mapping_ttl_v2s*

- Bits: 1
- Valuemappings:
  - '0' -> 100
  - '1' -> 200

IP Destination Address (#8)

Among other options, Girling et al. propose using the *Destination IP Address* field to hide a covert channel [8]. Note that, in this case, the receiver of the cover communication can't be in a destination device, but in a location able to sniff traffic in the range of the destination addresses used for the covert channel. The cc technique in
techniques/ipaddr.py uses a "pdst" parameter that consists of a string formed by 3 octets and 4 dots, e.g., "123.103.24.", the missing octet being reserved for the value to be covertly sent. It also uses a "pmin" parameter, an offset value for the last octet to be summed to the covert value (if *Bits=8*, then "pmin" must be 0) Note that this technique must be adjusted with a 1tuple (*srcIP*) flowkey to be correctly grabbed during extraction.

- Config: *ipaddr_v2s*
- Mapping: *mapping_ipaddr_4bits*, *mapping_ipaddr_8bits*
- Bits: 1-8
- Parameters:
  - *pdst* (base for the range of allowed destination addresses
  - *pmin* (minimum value for the last octet).

Ranges as symbols (r2s)

This type of covert channels maps a range of values with a covert symbol. For instance, taking again the example with the size of IP packets, any packet with a size below or equal 50-bytes could stand for `0`, and then packets above 60 bytes would be `1`.

TTL (#9)

PROF

The cc technique in **techniques/ttl_r2s.py** uses the *TTL* field of the IP datagram to hide a binary covert channel where both 0s and 1s are represented by different values. This implementation is inspired in the methods introduced in [14]. Three parameters are required: "p0" and "p1", which are the base values for "0" and "1" symbols, and "pvar" which is the maximum number of hops allowed above or below base values. Take care when selecting parameter values and ensure that possible overlaps are avoided.

- Config: *ttl_r2s*
- Mapping: *mapping_ttl_r2s*
- Bits: 1
- Parameters:
  - *pvar* (maximum hops allowed over or under base values)
- Valuemappings:
  - '0' -> 60
  - '1' -> 200

TCP/UDP source port (#10)

The cc technique in **techniques/srcport_r2s.py** uses the *Source Port* field of the IP datagram to hide a binary covert channel where both 0s and 1s are represented by different values. This implementation is inspired in the methods introduced in [14]. Three parameters are required: "p0" and "p1", which are the base values for "0" and "1" symbols, and "pvar" which is the maximum variation above or below base values. Take care when selecting parameter values and ensure that possible overlaps are avoided. Note that for this technique to work properly the configured flowkey must be 3tuple (*srcIP, dstIP, Protocol*), ensuring that the protocol used is TCP or UDP (*tcp*, *udp*, or *tcp/udp* options in the *const* field when using the *ccGen-wrapper*).

- Config: *srcport_r2s*
- Mapping: *mapping_srcport_r2s*
- Bits: 1
- Parameters:
    - *pvar* (maximum hops allowed over or under base values)
- Valuemappings:
    - '0' -> 1000
    - '1' -> 2000

IP Length (#11)

The cc technique implemented in **techniques/iplen_r2s.py** uses the size of the IP *Total Length* field to hide a covert channel. In this implementation, four parameters are required: "p0" and "p1", which are the base lengths for "0" and "1" symbols, "pinc" is a base length increment, and "pvar" is the maximum times that "pinc" can be added or substracted to the base lengths. Take care when adjusting parameters values, otherwise a wrong selection of parameters can make the extraction not possible. The implementation is inspired the discussion in [8], but allowing different lengths to be mapped to the same symbol in order to hamper the detection of the covert channel.

- Config: *ttl_r2s*
- Mapping: *mapping_len_r2s*
- Bits: 1
- Parameters:
    - *pinc* (base increment)
    - *pvar* (maximum number of times that *pinc* is allowed over or under base length values)
- Valuemappings:
    - '0' -> 60,
    - '1' -> 120

PROF

Containers (con)

we consider that a covert channel is hidden in *container* fields when the amount of covert information sent per packet is greater than 1 byte. Container fields are usually (but not always) accompanied by *marker* fields, which inform the receiver about the existence of covert information in the current packet.

IP fragment (#12)

The cc technique implemented in **techniques/ipfragment.py** uses the *Fragment offset* field of the IP frame to hide a covert channel. Here each packet can contain up to 13 bits of clandestine information. Additionally, in each modified packet, it clears the *Don't Fragment* bit and sets the *More Fragment* of the *IP Flags*. Note that this covert channel breaks protocol rules and can be easily detected or noticed by common traffic visualization tools. This option for hiding covert channels is discussed by Goher et al. in [5].

- Config: *ipfragement_con*
- Mapping: *mapping_13bits*
- Bits: 13

URG bit-pointer (#13)

The cc technique implemented in **techniques/urgent.py** uses the *URG* bit of the *TCP Flags* field and the *Urgent Pointer* field of the TCP frame to hide a covert channel. Here the URG bit acts as marker: when it is set to '0', the Urgent Pointer contains up to 16 bits of information. This technique has been proposed by Fisk et al. in [17]. The implementation here developed requires one parameter: "b2n" indicates that a *binary-to-integer* function is used instead of direct mapping. *b2n* takes 16 as value, which stands for the length of the bit-word. Note that for this technique to work properly the configured flowkey must be 3tuple (*srcIP, dstIP, Protocol*) or 5tuple (*srcIP, dstIP, Protocol, srcPort, dstPort*), ensuring that the protocol used is TCP (*tcp* option in the *const* field when using the *ccGen-wrapper*).

- Config: *urg-pointer_con*
- Mapping: *mapping_16bits*
- Bits: 16

Derivative channels (dev)

Derivative channels occur when covert symbols are not directly hidden in the value of the field but in how this value changes throughout successive packets.

PROF

TTL (#14)

The cc technique implemented in **techniques/ttl_dev.py** uses the *TTL* field of the IP datagram to hide a binary derivative covert channel as explained by Zander et al. in [13].

- Config: *ttl_dev*
- Mapping: *mapping_ttl_dev*
- Bits: 1
- Parameters:
    - *ph* (upper TTL value)
    - *pl* (lowest TTL value)

TCP/UDP source port (#15)

Gimbi et al. present different ways of using the *Source Port* field of the TCP or UDP datagrams to convey ASCII symbols in [12]. The method in **techniques/srcport_dev.py** follows their proposal and encapsulates ASCII symbols in value increments. This implementation uses two parameters: "pmin" sets a minimum value for the source port, and "pthr" an upper threshold to start again from low values once it is surpassed. Note that for this technique to work properly the configured flowkey must be 3tuple (*srcIP, dstIP, Protocol*), ensuring that the protocol used is TCP or UDP (*tcp*, *udp*, or *tcp/udp* options in the *const* field when using the *ccGen-wrapper*).

- Config: *srcport_dev*
- Mapping: *mapping_srcport_dev*
- Bits: 8
- Parameters:
  - *pmin* (minimum value)
  - *pthr* (upper threshold)

Covert timing channels (ctc)

Covert timing channels use time properties --mainly IAT (Inter-Arrival Times between packets)-- to convey hidden communication. Further descriptions of the methods implemented in ccGen.v2 con be consulted in [3]. Note that most techniques manipulate IDT (Inter-Departure Times) in origin, which transform into IATs in destination (in short, IAT = IDT + tx_delays). All techniques that uses IATs to hide covert channels must define a "pIAT" feature in the corresponding mapping file.

BER (#16)

Presented in [1] by Berk et al., the cc technique implemented in **techniques/timing_ber.py** agrees on two different IDTs to mask binary symbols. This version requires a "pmask" parameter (only useful when *offline*), which sets a number of decimal places below a second to maintain the original value and simulate a residual transmission delay.

- Config: *timing_ber*
- Mapping: *mapping_timing_ber*
- Bits: 1
- Parameters:
  - *pIAT* (use of IATs)
  - *pmask* (scale-mask to keep residual original time)
- Valuemappings:
  - '0' -> 0.10
  - '1' -> 0.20

PROF

GAS (#17)

Gasior et al. presents in [9] a timing technique that sets a time-threshold to discriminate 0s and 1s. If a given IAT is above the threshold, it will mark 1, 0 if below. This technique is implemented in **techniques/timing_gas.py**, and requires a "pthr" for the threshold and an additional "pmin" parameter (only useful when *offline*) to set a minimal transmission delay.

- Config: *timing_gas*

- Mapping: *mapping_timing_gas*
- Bits: 1
- Parameters:
  - *pIAT* (use of IATs)
  - *pthr* (time threshold)
  - *pmin* (minimal residual transmission delay)

SHA (#18)

The technique proposed by Shah et al. [16] is designed to interfere legitimate communications. It uses a base sample interval and adds some delay to IDTs. A covert 1 or a 0 is interpreted depending on if a given IAT is divisible by the interval or only half of it. The implementation in **techniques/timing_sha.py** uses "pw2" to define the half-interval (in seconds), "prdx" establishes a maximum for the times that two-times-"pw2" can happen between two consecutive packets, and "pmask" parameter (only useful when *offline*), which sets a number of decimal places below a second to maintain the original value and simulate a residual transmission delay.

- Config: *timing_sha*
- Mapping: *mapping_timing_sha*
- Bits: 1
- Parameters:
  - *pIAT* (use of IATs)
  - *pw2* (half time interval)
  - *pmask* (scale-mask to keep residual original time)
  - *prdx* (integer determining the randomness of IDT values, the greater the number the greater the randomness)

ZAN (#19)

PROF

This is a technique originally described for the Time-to-Live (TTL) field by Zander et al. in [13], yet easily applicable to IATs. This technique makes use of two parameters, tb and tinc.tb refers to the base IDT at which packets may be sent and tinc to the time that may be added or substracted from tb. If the covert symbol 0 is to be transmitted, the IDT is set to tb; if the covert symbol 1 is to be transmitted, however, the IDT is set to tb ± tinc. Addition and substraction are applied alternately on subsequently occurring 1 symbols.

- Config: *timing_zan*
- Mapping: *mapping_timing_zan*
- Bits: 1
- Parameters:
  - *pIAT* (use of IATs)
  - *pmask* (scale-mask to keep residual original time)
  - *pTb* (float giving the base time between packets)
  - *pTinc* (incrementing value beeing subtracted or added to base time)

ZAN2 (#20)

This implementation of covert timing channel is not a dedicated technique in literature but its base idea is after a figure in [13], where three time constants are needed to cover each possible transition with the symbol '0' and '1'. Let the transition from 0 to 1 be time constant A, and 1 to 0 the time constant B, then is the transition from 0 to 0 time constant C - A and the transition from 1 to 1 C - B.

- Config: *timing_zan2*
- Mapping: *mapping_timing_zan2*
- Bits: 1
- Parameters:
    - *pIAT* (use of IATs)
    - *pmask* (scale-mask to keep residual original time)
    - *pA* (time constant A)
    - *pB* (time constant B)
    - *pC* (time constant C)

ED1 (#21)

This technique [18] encodes the covert symbol 0 with a waiting time t0 (authors suggest 300 ms), whereas the covert symbol 1 triggers the immediate sending of a packet. Note that t0 is not defining any IDT; therefore, sending 0s does not imply sending any packet. For example, if ‘A’, which corresponds to the ASCII encoding ‘01000001’, is to be covertly send, the sender will follow this sequence: (1) wait 300 ms, (2) send one packet, (3) wait 1.5 s (300 ms×5), (4) send one packet. Messages are forced to finish transmitting a ‘1’.

- Config: *timing_ed1*
- Mapping: *mapping_timing_ed1*
- Bits: 1
- Parameters:
    - *pIAT* (use of IATs)
    - *pSB* (start bit added to the beginning of message)
    - *pTw* (waiting time if 0 occurs)

PROF

References

[1] Berk, V., Giani, A., & Cybenko, G. (2005). Detection of covert channel encoding in network packet delays. Link

[2] Iglesias, F., Annessi, R., & Zseby, T. (2016). DAT detectors: uncovering TCP/IP covert channels by descriptive analytics. Security and Communication Networks, 9(15), 3011-3029. Link

[3] Iglesias, F., Annessi, R., & Zseby, T. (2017). Analytic Study of Features for the Detection of Covert Timing Channels in Network Traffic. Journal of Cyber Security and Mobility, 245-270. Link

[4] Xu, B., Wang, J. Z., & Peng, D. Y. (2007, March). Practical protocol steganography: Hiding data in IP header. In First Asia International Conference on Modelling & Simulation (AMS'07) (pp. 584-588). IEEE. Link

[5] Goher, S. Z., Javed, B., & Saqib, N. A. (2012, December). Covert channel detection: A survey based analysis. In High Capacity Optical Networks and Emerging/Enabling Technologies (pp. 057-065). IEEE. Link

[6] Rowland, C. H. (1997). Covert channels in the TCP/IP protocol suite. Link

[7] Ahsan, K., & Kundur, D. (2002, December). Practical data hiding in TCP/IP. In Proc. Workshop on Multimedia Security at ACM Multimedia (Vol. 2, No. 7, pp. 1-8). Link

[8] Girling, C. G. (1987). Covert Channels in LAN's. IEEE Transactions on software engineering, 13(2), 292. Link

[9] Gasior, W., & Yang, L. (2011, October). Network covert channels on the android platform. In Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research (pp. 1-1). Link

[10] Wendzel, S., & Zander, S. (2012, October). Detecting protocol switching covert channels. In 37th Annual IEEE Conference on Local Computer Networks (pp. 280-283). IEEE. Link

[11] Postel, J. Internet Protocol. RFC 0791, IETF, Sept. 1981. Link

[12] Gimbi, J., Johnson, D., Lutz, P., & Yuan, B. (2012). A Covert Channel Over Transport Layer Source Ports. Link

[13] Zander, S., Armitage, G., & Branch, P. (2007, November). An empirical evaluation of IP Time To Live covert channels. In 2007 15th IEEE International Conference on Networks (pp. 42-47). IEEE. Link

[14] Lucena, N. B., Lewandowski, G., & Chapin, S. J. (2005, May). Covert channels in IPv6. In International Workshop on Privacy Enhancing Technologies (pp. 147-166). Springer, Berlin, Heidelberg. Link

[15] Qu, H., Su, P., & Feng, D. (2004, October). A typical noisy covert channel in the IP protocol. In 38th Annual 2004 International Carnahan Conference on Security Technology, 2004. (pp. 189-192). IEEE. Link

[16] Shah, G., Molina, A., & Blaze, M. (2006, July). Keyboards and Covert Channels. In USENIX Security Symposium (Vol. 15, p. 64). Link

PROF

[17] Fisk, G., Fisk, M., Papadopoulos, C., & Neil, J. (2002, October). Eliminating steganography in Internet traffic with active wardens. In International workshop on information hiding (pp. 18-35). Springer, Berlin, Heidelberg. Link

[18] Castillo, Eduardo J., Xenia Mountrouidou, and Xiangyang Li. “Time Lord: Covert Timing Channel Implementation and Realistic Experimentation.” In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 755–56. SIGCSE ’17. New York, NY, USA: Association for Computing Machinery, 2017. Link