NETWORK INFORMATION HIDING

CH. 2: INTRODUCTION TO LOCAL COVERT CHANNELS

Prof. Dr. Steffen Wendzel
Worms University of Applied Sciences

https://www.wendzel.de (EN) | https://www.hs-worms.de/wendzel/ (DE)

Online Class: https://github.com/cdpxe/Network-Covert-Channels-A-University-level-Course/

Sample Covert Channel

- Consider two processes, P_1 and P_2, running within the same environment. Several possible covert channels between these processes are imaginable:

  1. P_1 performs intensive computations to influence the system load (measured by P_2).
  2. P_1 stops its operation at a given time t_1 or t_2 to signal a `0' or `1' bit (while P_2 monitors the process table).
  3. P_1 either creates or does not create an entry in the file system known by P_2 (existence of the file signals the hidden information)

- These simple examples reveal that covert channels are usually not noise-free, need a protocol (when does a transmission start/end?) and need to detect errors in transmissions (e.g. using parity bits).
  – I will discuss this at least briefly in a **later chapter on sophisticated network covert channels**.

Sample Docker Covert Channels [1]

- Docker and other container technology has been proven not to be resistant against covert channels.
- Several covert channels possible, e.g. globally used memory (GUM), Inode (exhaustion), …, cf. [1]

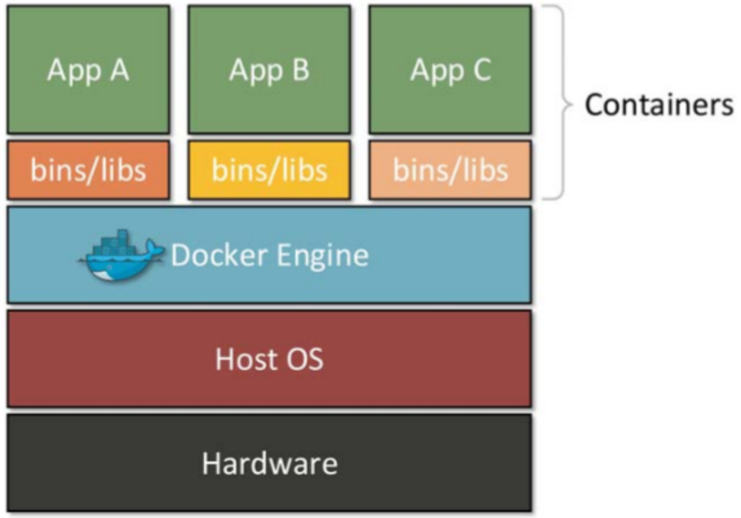


Fig. 1. Architecture of Docker containers

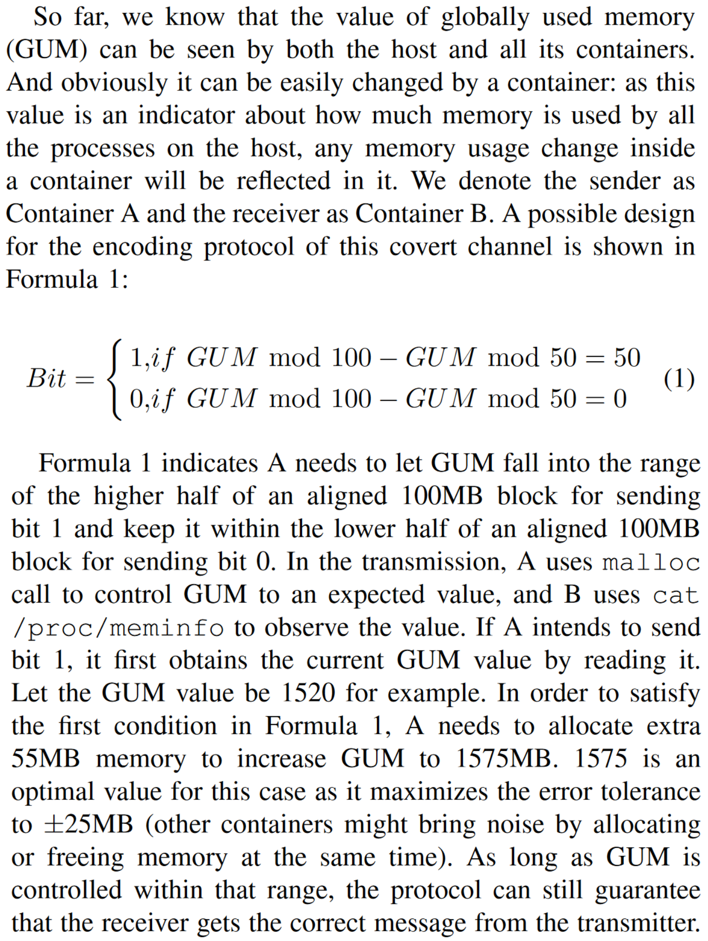
TABLE I
NAMESPACE ISOLATION IN DOCKER

Namespace	Isolates
IPC	System V IPC, POSIX message queues
Network	Network devices, stacks, ports, etc.
Mount	Mount points
PID	Process IDs
User	User and group IDs
UTS	Hostname and NIS domain name

[1] Luo, Y., Luo, W., Sun, X., Shen, Q., Ruan, A., Wu, Z.: Whispers Between the Containers: High-capacity Covert Channel Attacks in Docker, in Proc. TrustCom-BigDataSE-ISPA, pp. 630-637, IEEE, 2016.

Sample Docker Covert Channels [1]

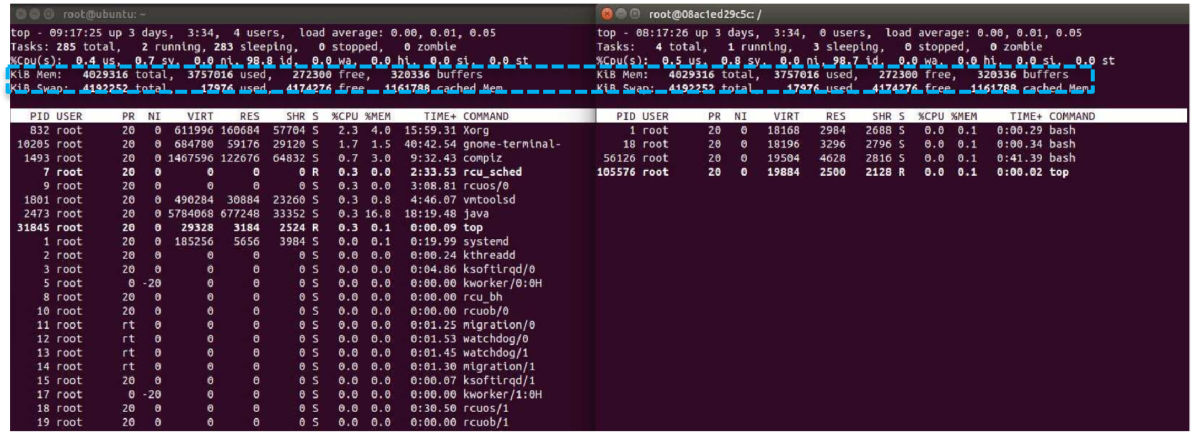
- Example of a GUM channel:



So far, we know that the value of globally used memory (GUM) can be seen by both the host and all its containers. And obviously it can be easily changed by a container: as this value is an indicator about how much memory is used by all the processes on the host, any memory usage change inside a container will be reflected in it. We denote the sender as Container A and the receiver as Container B. A possible design for the encoding protocol of this covert channel is shown in Formula 1:

$$Bit = \begin{cases} 1, if\ GUM \bmod 100 - GUM \bmod 50 = 50 \\ 0, if\ GUM \bmod 100 - GUM \bmod 50 = 0 \end{cases} \qquad (1)$$

Formula 1 indicates A needs to let GUM fall into the range of the higher half of an aligned 100MB block for sending bit 1 and keep it within the lower half of an aligned 100MB block for sending bit 0. In the transmission, A uses `malloc` call to control GUM to an expected value, and B uses `cat /proc/meminfo` to observe the value. If A intends to send bit 1, it first obtains the current GUM value by reading it. Let the GUM value be 1520 for example. In order to satisfy the first condition in Formula 1, A needs to allocate extra 55MB memory to increase GUM to 1575MB. 1575 is an optimal value for this case as it maximizes the error tolerance to ±25MB (other containers might bring noise by allocating or freeing memory at the same time). As long as GUM is controlled within that range, the protocol can still guarantee that the receiver gets the correct message from the transmitter.



```
top - 09:17:25 up 3 days,  3:34,  4 users,  load average: 0.00, 0.01, 0.05
Tasks: 285 total,   2 running, 283 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.7 sy,  0.0 ni, 98.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   4029316 total,  3757016 used,   272300 free,   320336 buffers
KiB Swap:  4192252 total,    17976 used,  4174276 free.  1161788 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  832 root      20   0  611996 160684  57704 S   2.3  4.0  15:59.31 Xorg
10205 root      20   0  684780  59176  29120 S   1.7  1.5  40:42.54 gnome-terminal-
 1493 root      20   0 1467596 122676  64832 S   0.7  3.0   9:32.43 compiz
    7 root      20   0       0      0      0 R   0.3  0.0   2:33.53 rcu_sched
    9 root      20   0       0      0      0 S   0.3  0.0   3:08.81 rcuos/0
 1801 root      20   0  490284  30884  23260 S   0.3  0.8   4:46.07 vmtoolsd
 2473 root      20   0 5784068 677248  33352 S   0.3 16.8  18:19.48 java
31845 root      20   0   29328   3184   2524 R   0.3  0.1   0:00.09 top
    1 root      20   0  185256   5656   3984 S   0.0  0.1   0:19.99 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.24 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:04.86 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:01.25 migration/0
   12 root      rt   0       0      0      0 S   0.0  0.0   0:01.53 watchdog/0
   13 root      rt   0       0      0      0 S   0.0  0.0   0:01.45 watchdog/1
   14 root      rt   0       0      0      0 S   0.0  0.0   0:01.30 migration/1
   15 root      20   0       0      0      0 S   0.0  0.0   0:00.07 ksoftirqd/1
   17 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   18 root      20   0       0      0      0 S   0.0  0.0   0:30.50 rcuos/1
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/1
```

```
top - 08:17:26 up 3 days,  3:34,  0 users,  load average: 0.00, 0.01, 0.05
Tasks:   4 total,   1 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us,  0.8 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   4029316 total,  3757016 used,   272300 free,   320336 buffers
KiB Swap:  4192252 total,    17976 used,  4174276 free.  1161788 cached Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
     1 root      20   0   18168   2984   2688 S   0.0  0.1   0:00.29 bash
    18 root      20   0   18196   3296   2796 S   0.0  0.1   0:00.34 bash
 56126 root      20   0   19504   4628   2816 S   0.0  0.1   0:41.39 bash
105576 root      20   0   19884   2500   2128 R   0.0  0.1   0:00.02 top
```

(a) The host (b) The container

Fig. 3. The memory usage observed from the host and one of its containers

[1] Luo, Y., Luo, W., Sun, X., Shen, Q., Ruan, A., Wu, Z.: Whispers Between the Containers: High-capacity Covert Channel Attacks in Docker, in Proc. TrustCom-BigDataSE-ISPA, pp. 630-637, IEEE, 2016.

Covert Channels in Android

- Plethora of research was conducted in recent years on covert channels in mobile phone environments.

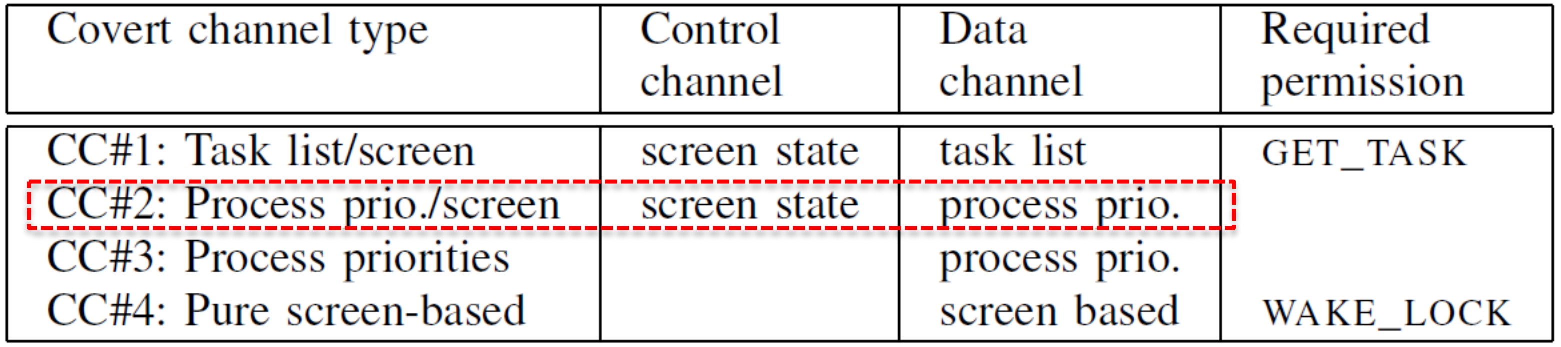
- The goal is usually to establish a policy-breaking communication within two sandboxed apps.

- In Android, apps have permissions, e.g. the permission to access the contacts [or to use the Internet connection] ← slightly out-dated.

Covert Channels in Android

- Many covert channels possible, here are just four we published in 2013 [1]:

Table II
CONTROL AND DATA CHANNELS OF OUR COVERT CHANNELS.



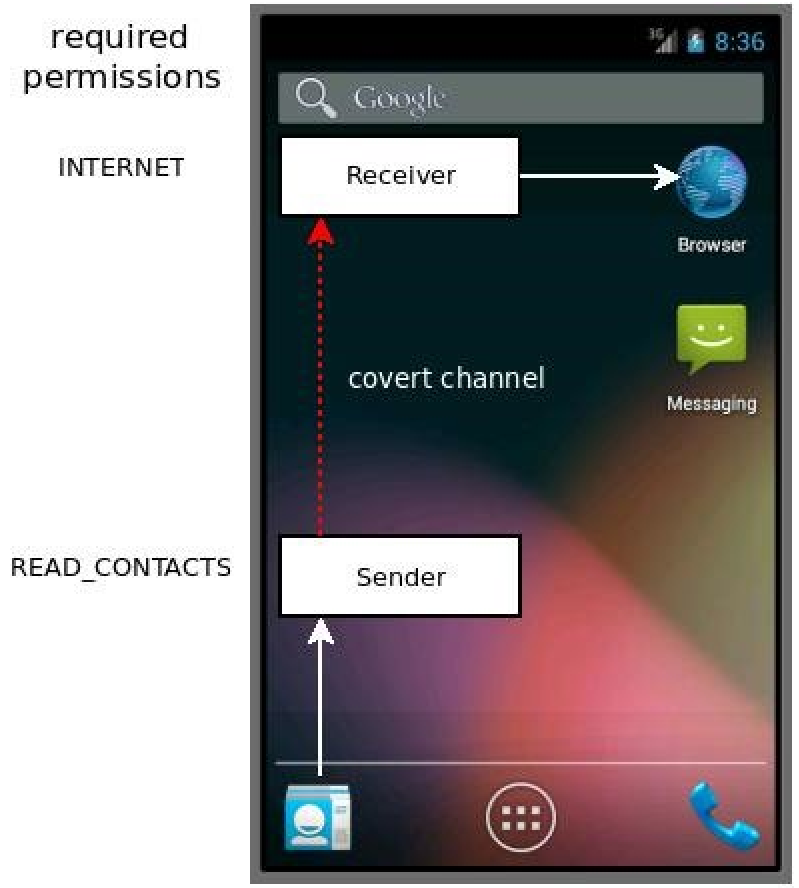
Covert channel type	Control channel	Data channel	Required permission
CC#1: Task list/screen	screen state	task list	GET_TASK
CC#2: Process prio./screen	screen state	process prio.	
CC#3: Process priorities		process prio.	
CC#4: Pure screen-based		screen based	WAKE_LOCK

[1] J.-F. Lalande, S. Wendzel: Hiding Privacy Leaks in Android Applications Using Low-Attention Raising Covert Channels, in Proc. ARES 2013, pp. 701-710, Regensburg, 2013.

Covert Channels in Android

- Example scenario using two apps (e.g. two smart home apps, one for monitoring energy consumption; one app is an energy advisor).

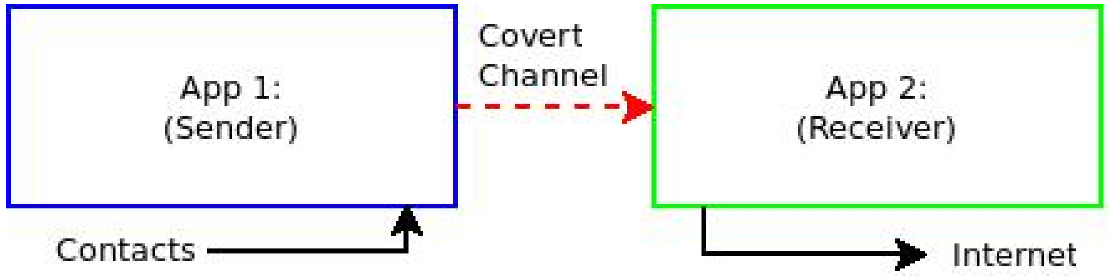
- Requirements for covert transmission:
  - Sender and receiver must run simultaneously
  - Transmission via process priority of ‚Sender'
  - Transfer process starts when user turns off the screen

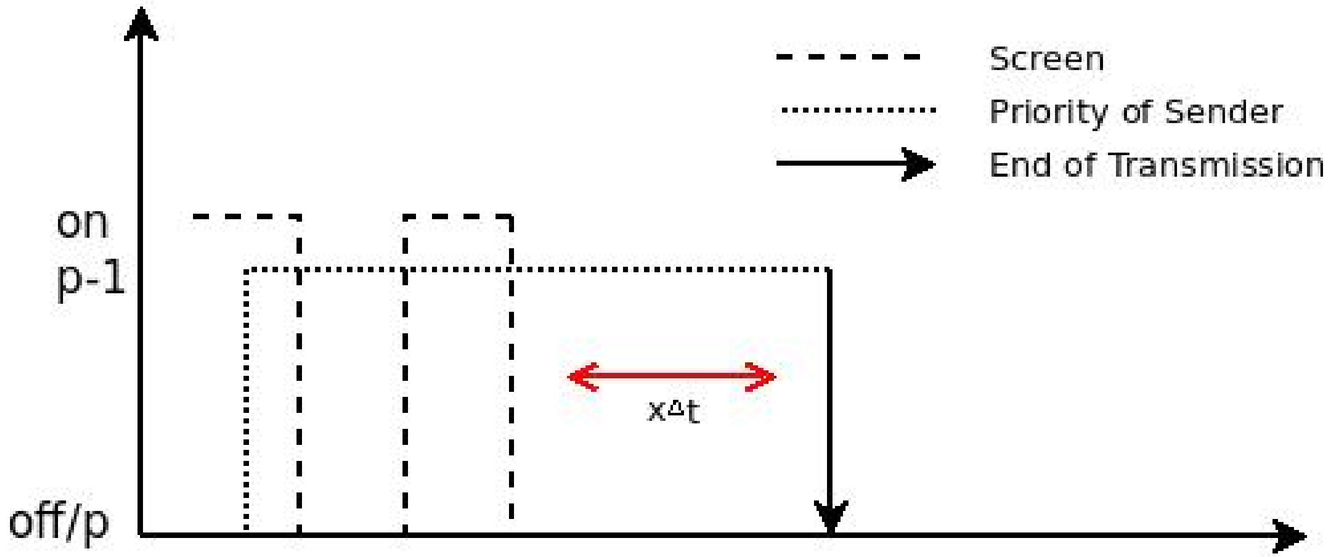


Hochschule Worms
University of Applied Sciences

Covert Channels in Android

- How bits are transmitted:





J.-F. Lalande, S. Wendzel: Hiding Privacy Leaks in Android Applications Using Low-Attention Raising Covert Channels, in Proc. ARES 2013, pp. 701-710, Regensburg, 2013.

- Video:
  - http://www.dailymotion.com/video/x10lcyq_ectcm-2013-hiding-privacy-leaks-in-android-applications_tech

- Original slides:
  - http://www.wendzel.de/dr.org/files/Papers/ares13_slides.pdf