Race for Root
Analysis of the Linux Kernel Race Condition Exploit

Alexander Popov

Positive Technologies

SHA2017

About Me

- Alexander Popov
- Linux kernel developer
- Security researcher at Positive Technologies

Agenda

- CVE-2017-2636 overview
- Exploit demo
- Exploit steps:
  - Achieve double-free with a race condition
  - Turn double-free into use-after-free and exploit it
  - Bypass SMEP (without ROP)
- Defense

About CVE-2017-2636

- LPE in Linux kernel
- Bug type: race condition
- In `drivers/tty/n_hdlc.c`
- All major distros were affected (`CONFIG_N_HDLC=m`)

What Is HDLC?

- Stands for `High-Level Data Link Control`
- Is a data link layer protocol
- Its frames can be transmitted over serial links
- Now used mainly for device-to-device connection

Timeline (1)

The bug is introduced	2009-06-22
...	...
Suspicious crash by syzkaller (cool project!)	2017-02-01
Have a stable race condition repro	2017-02-03
Almost no sleep... :)	...
Have the exploit PoC and a fixing patch	2017-02-28

Timeline (2)

Inform security@kernel.org	2017-02-28
Linux distros are informed	2017-03-02
End of embargo, announce at oss-security	2017-03-07
Publish a write-up	2017-03-24
Patch the mainline to block similar exploits	In progress

'N_HDLC' Race Condition (1)

The original driver used:

- Self-made singly linked lists for data buffers
- `n_hdlc.tbuf` pointer for buffer retransmitting after tx error in `n_hdlc_send_frames()`

'N_HDLC' Race Condition (2)

The commit `be10eb75893` added buffer flushing:

- `flush_tx_queue()` can put `n_hdlc.tbuf` to `tx_free_buf_list` too
- Insanely wrong locking
- Possible double-free in `n_hdlc_release()`

'N_HDLC' Race Condition

Yes, it's dangerous!

http://www.foxnews.com/sports/slideshow/2013/02/23/crash-during-final-lap-2013-nascar-nationwide-series-race-at-daytona.html

CVE-2017-2636 Exploit Demo

Demo!

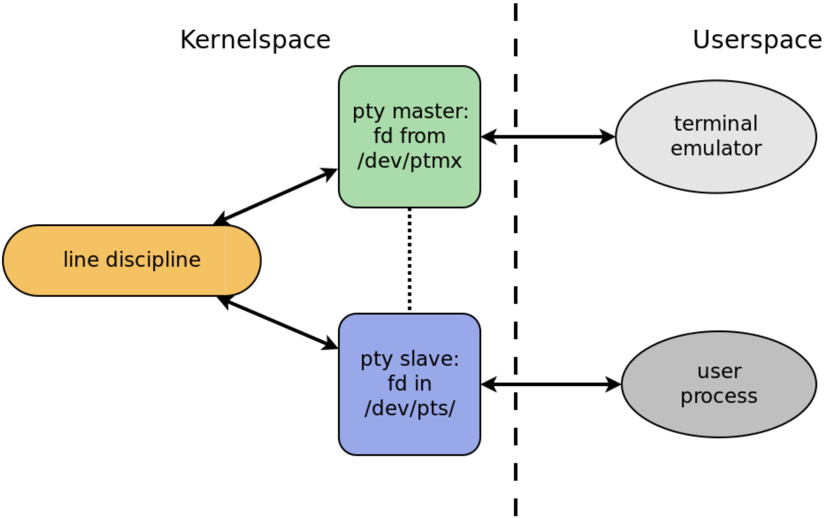
CVE-2017-2636 Exploit Overview

1. Preparing N_HDLC line discipline
2. Hitting the race condition to get double-free
3. Heap spraying for turning double-free into use-after-free
4. Another heap spraying to exploit use-after-free
5. Heap stabilization
6. SMEP bypass (without ROP)

Preparing for the Race

http://www.superstreetonline.com/features/1601-daigo-saito-garage-visit/photo-gallery/#photo-09

Who Is Who: PTY Components



Preparing for the Race (1)

- Stick to one CPU core with sched_setaffinity()
- Create a pseudoterminal master and slave pair:

```
ptmd = open("/dev/ptmx", O_RDWR);
```

- Set N_HDLC ldisc (n_hdlc.ko is loaded automatically):

```
const int ldisc = N_HDLC;
ioctl(ptmd, TIOCSETD, &ldisc);
```

Preparing for the Race (2)

- Suspend the pty output:

```
ioctl(ptmd, TCXONC, TCOOFF);
```

- Write one data buffer (saved in `n_hdlc.tbuf`):

```
write(ptmd, buf, size);
```

- Allow to run on all available CPU cores

Now Go Racing!

http://findwallpaper.info/street+racing+cars/page/7/

Racing

Start two threads:

- Thread 1, flush the data:

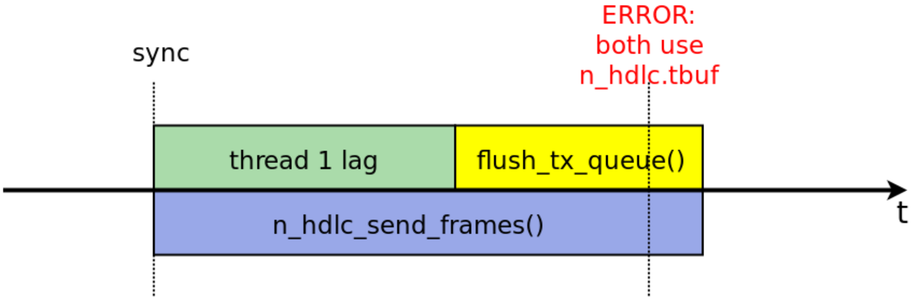
```
ioctl(ptmd, TCFLSH, TCIOFLUSH);
```

- Thread 2, start the suspended output:

```
ioctl(ptmd, TCXONC, TCOON);
```

Lags Make It... Faster (1)

1. Synchronize at `pthread_barrier`
2. Spin the lag in a busy loop
3. Interact with `n_hdlc`



Lags Make It... Faster (2)

Calculate the lags (in microseconds) for the racing threads:

```
if (loop % 2 == 0)
  tmo1 = loop % MAX_RACE_LAG_USEC;
else
  tmo2 = loop % MAX_RACE_LAG_USEC;
```

Triggering Double-Free

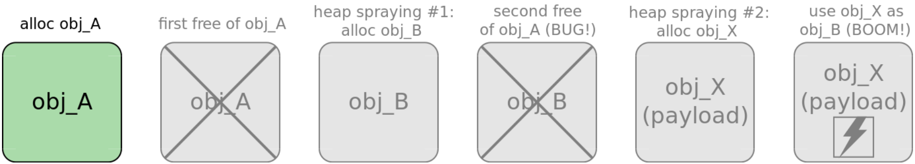
- Stick to a single CPU core again
- Close the pseudoterminal master fd:
  - `n_hdlc_release()` frees `n_hdlc_buf` items
  - The possible double-free error happens here
  - KASAN detects it as use-after-free

Exploiting Double-Free

- Now disable KASAN and try to exploit it!
- If successful (`uid` has become `0`), run shell
- Otherwise, go racing again

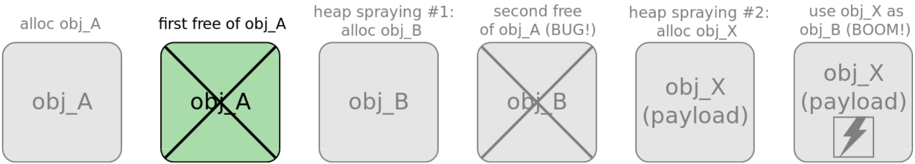
A Usual Double-Free Exploit (1)

All these objects reside at the same address



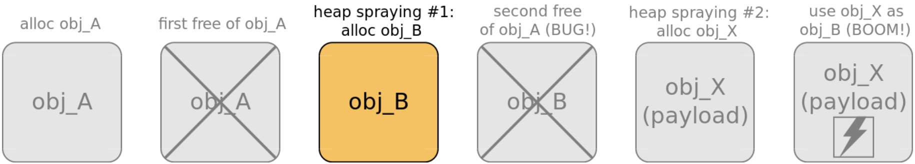
A Usual Double-Free Exploit (2)

All these objects reside at the same address



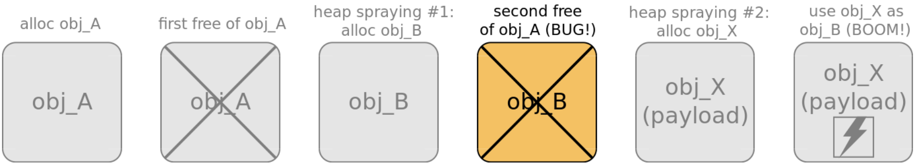
A Usual Double-Free Exploit (3)

All these objects reside at the same address



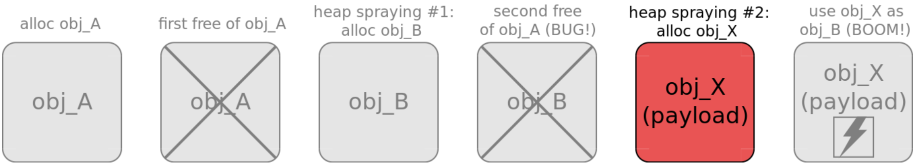
A Usual Double-Free Exploit (4)

All these objects reside at the same address



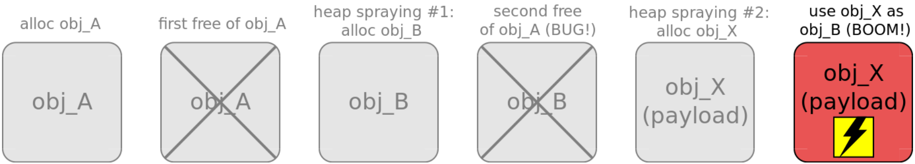
A Usual Double-Free Exploit (5)

All these objects reside at the same address



A Usual Double-Free Exploit (6)

All these objects reside at the same address



CVE-2017-2636 Exploit Requirements

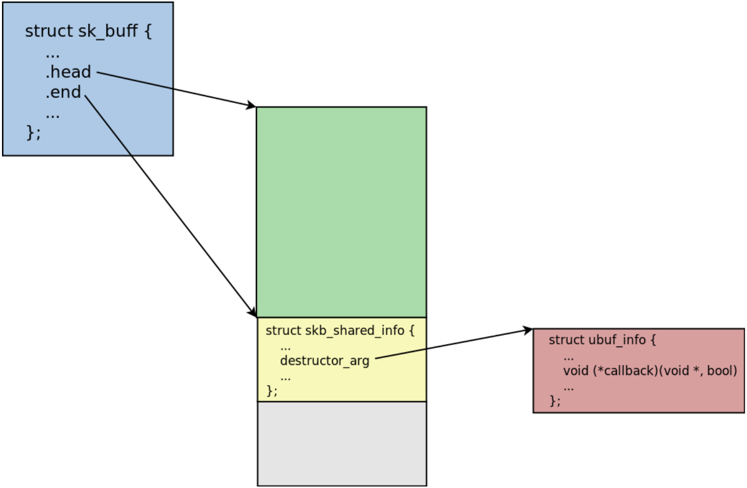
`n_hdlc_buf` is allocated in the `kmalloc-8192` slab cache

=> need 2 types of kernel objects from that cache:

1. With a function pointer
2. With the controllable payload to overwrite it

'sk_buff' Fits Well

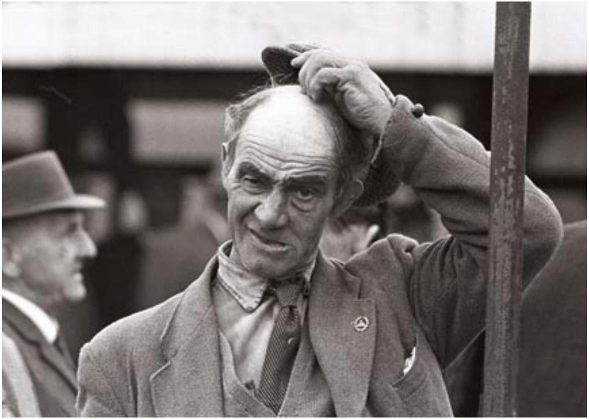
It can provide a function pointer at the `kmalloc-8192` slab



Heap Spraying #1: Not So Easy

- `n_hdlc_release()` frees 13 `n_hdlc_buf` items **straight away** without any pause
- Doubly freed item is somewhere at the beginning
- I can't allocate `sk_buff` data between double `free()`
- So the usual technique doesn't work here...

Still Puzzled Anyway #1...



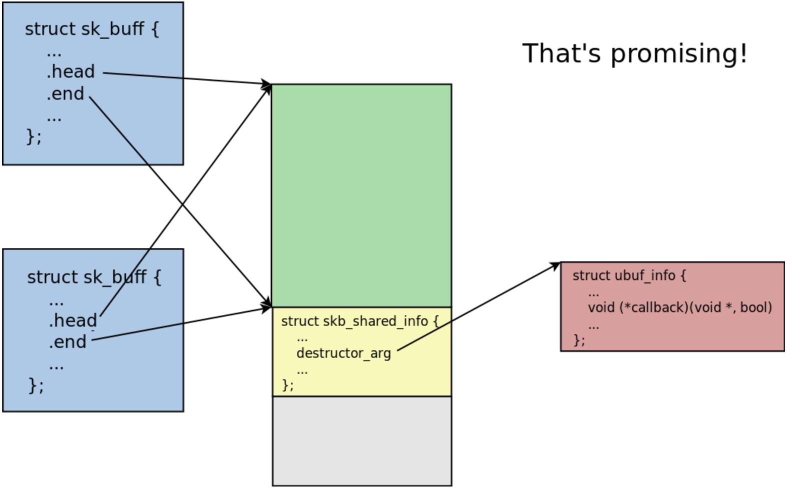
https://www.flickr.com/photos/philipdunn/3041924216

Just Look Carefully And...

- Wait, `n_hdlc_release()` doesn't crash the kernel =>
- SLUB allocator accepts consecutive `free()` of the same address =>
- I can spray after `n_hdlc_release()` and...

… Abuse SLUB's Naivety!

… get two `sk_buff`'s pointing to the same memory! :)



Hence, Heap Spraying #1

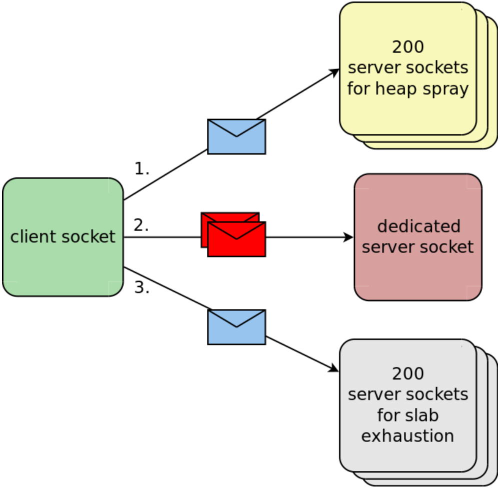
For turning double-free into use-after-free:

- Spawn a lot of 8 KB UDP packets **after** the race
- Keep them allocated to avoid a mess in SLUB freelist
- Receive one of the twin `sk_buff`'s
- Using the other one is a **use-after-free** error!

N.B. Socket queues are limited in size

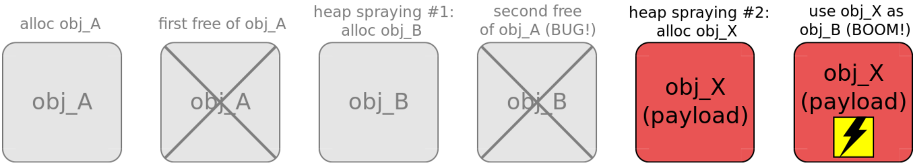
Spraying Implementation (1)

Debugged with `ftrace` (nice technology!)



A Usual Double-Free Exploit (5, 6)

All these objects reside at the same address



Now Exploiting Use-After-Free

- Heap spraying #2 for overwriting `destructor_arg`
- Another `sk_buff` can't do it
  - `skb_shared_info` is at the same offset from `head`
  - We don't control its contents
- But the `add_key` syscall can:
  - Allocate controllable data
  - Allocate in `kmalloc-8192`

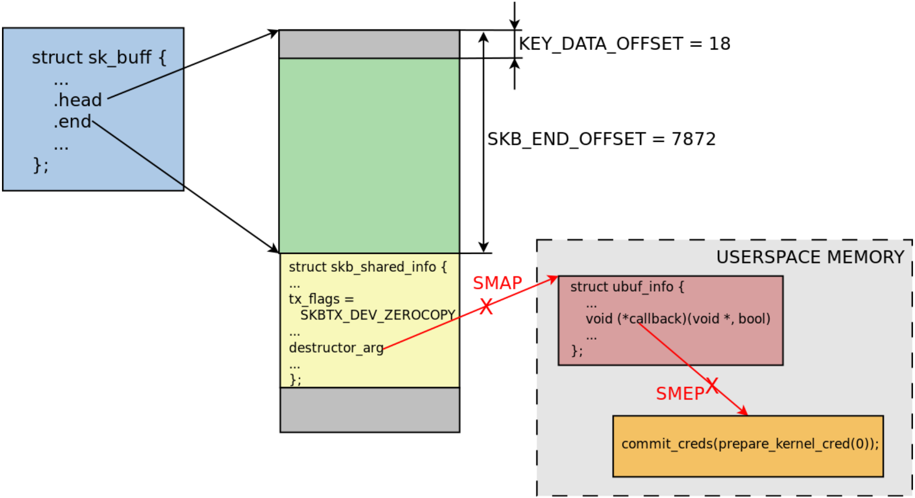
'destructor_arg' Usage

At linux/net/core/skbuff.c in skb_release_data():

```
if (shinfo->tx_flags &
                SKBTX_DEV_ZEROCOPY) {
    struct ubuf_info *uarg;

    uarg = shinfo->destructor_arg;
    if (uarg->callback)
        uarg->callback(uarg, true);
}
```

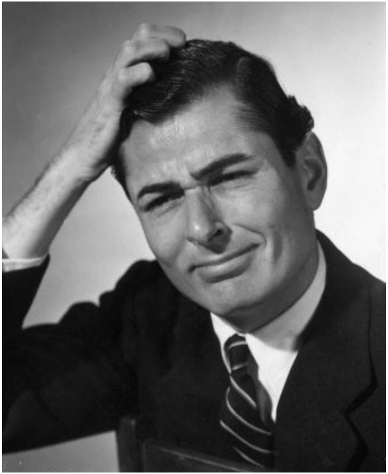
‘add_key‘ VS ‘destructor_arg‘



Nice, But Key Data Quotas...

- Controlled with /proc/sys/kernel/keys/
- Owned by root
- Default value of maxbytes is 20000 =>
- Only 2 add_key syscalls can concurrently store our 8 KB payload in the kernel memory
- Doesn't seem enough for heap spraying

Still Puzzled Anyway #2...



http://www.ideachampions.com/weblogs/puzzled.jpg

The Bright Idea

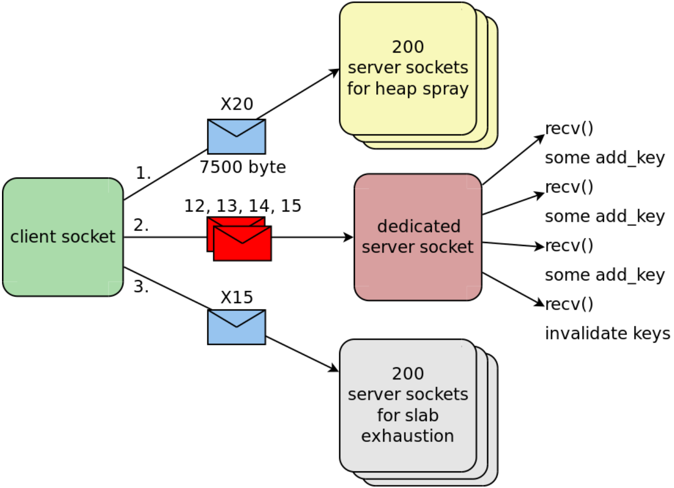
Inspired by the slides of Di Shen from Keen Security Lab:

Heap spraying can be successful even when `add_key` fails!

Kudos to him!

Spraying Implementation (2)

The number of packets and `add_key` calls is determined empirically



Working with Keys

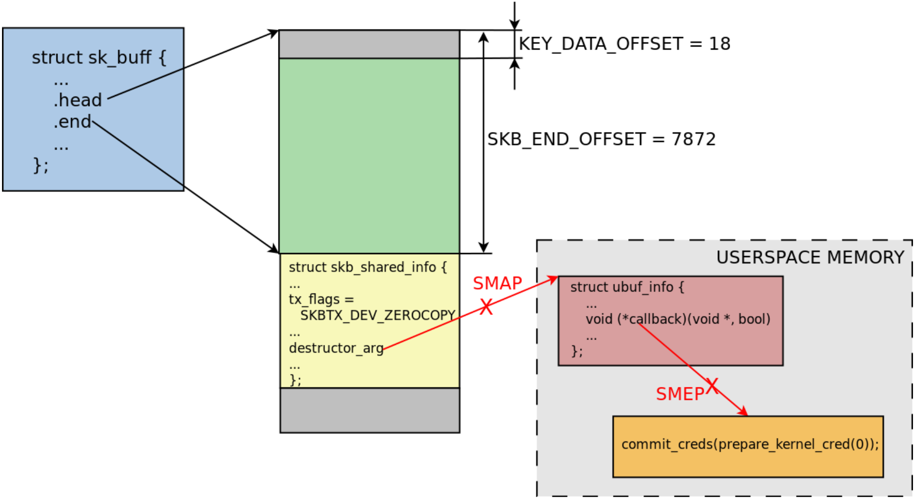
add_key usage example:

```
k[i] = syscall(__NR_add_key, "user",
    "payload0", payload, payload_size,
        KEY_SPEC_PROCESS_KEYRING);
```

Key invalidation:

```
if (k[i] > 0)
    syscall(__NR_keyctl, KEYCTL_INVALIDATE, k[i]);
```

'add_key' VS 'destructor_arg' (Again)



SMEP

- Supervisor Mode Execution Prevention
- The x86 feature controlled by bit 20 of the `CR4` register
- Fault on fetching an instruction from a user-mode address in the supervisor-mode

Known SMEP Bypass Techniques (Linux Kernel)

- Vitaly Nikolenko at Syscan360 (2016):
  - Overwrite `CR4` with stack pivoting + ROP
  - Bypass SMEP+SMAP by abusing vDSO (need an arbitrary write)
- Philip Pettersson exploit for CVE-2016-8655:
  - `set_memory_rw()` for vDSO and overwrite it
- Gonna show another **easy** way!

Nice Find!

In arch/x86/include/asm/special_insns.h:

```
static inline void native_write_cr4(unsigned long val)
{
    asm volatile("mov %0,%%cr4"
                 : : "r" (val), "m" (__force_order));
}
```

‘destructor_arg‘ Usage (Again)

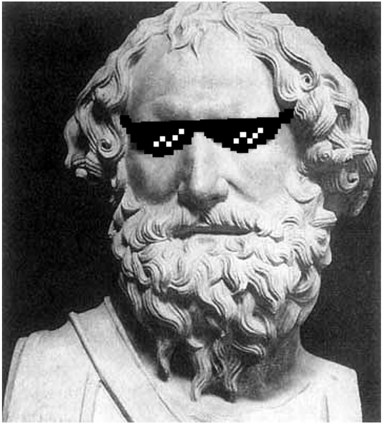
At linux/net/core/skbuff.c in skb_release_data():

```
if (shinfo->tx_flags &
                SKBTX_DEV_ZEROCOPY) {
    struct ubuf_info *uarg;

    uarg = shinfo->destructor_arg;
    if (uarg->callback)
        uarg->callback(uarg, true);
}
```

Eureka!

- Use `native_write_cr4()` as `ubuf_info.callback`
- Put `ubuf_info` item at the mmap'ed address `0x406e0`



Modified from http://www.timesofsicily.com/wp-content/uploads/2014/01/archimedes.bmp

SMEP Is Disabled

- `uarg->callback(uarg, true)` works as

  `native_write_cr4(0x406e0)`

- `0x406e0` is the right value of `CR4` with disabled SMEP (on my machine)
- => SMEP is disabled without ROP
- Now win the race again to run the shellcode!

CVE-2017-2636 Fix

- My patch: 82f2341c94d
- Use standard kernel linked list and proper locking
- Get rid of racy `n_hdlc.tbuf`
- In case of tx error, put current data buffer after the head of `tx_buf_list`

SLUB Fix

- SLUB assertion similar to `fasttop` check in GNU libc
- Is currently discussed at LKML
- Will hopefully come behind `SLAB_FREELIST_HARDENED`

Thanks. Questions?

alex.popov@linux.com
@a13xp0p0v

http://blog.ptsecurity.com/
@ptsecurity_UK
https://www.linkedin.com/company/positive-technologies