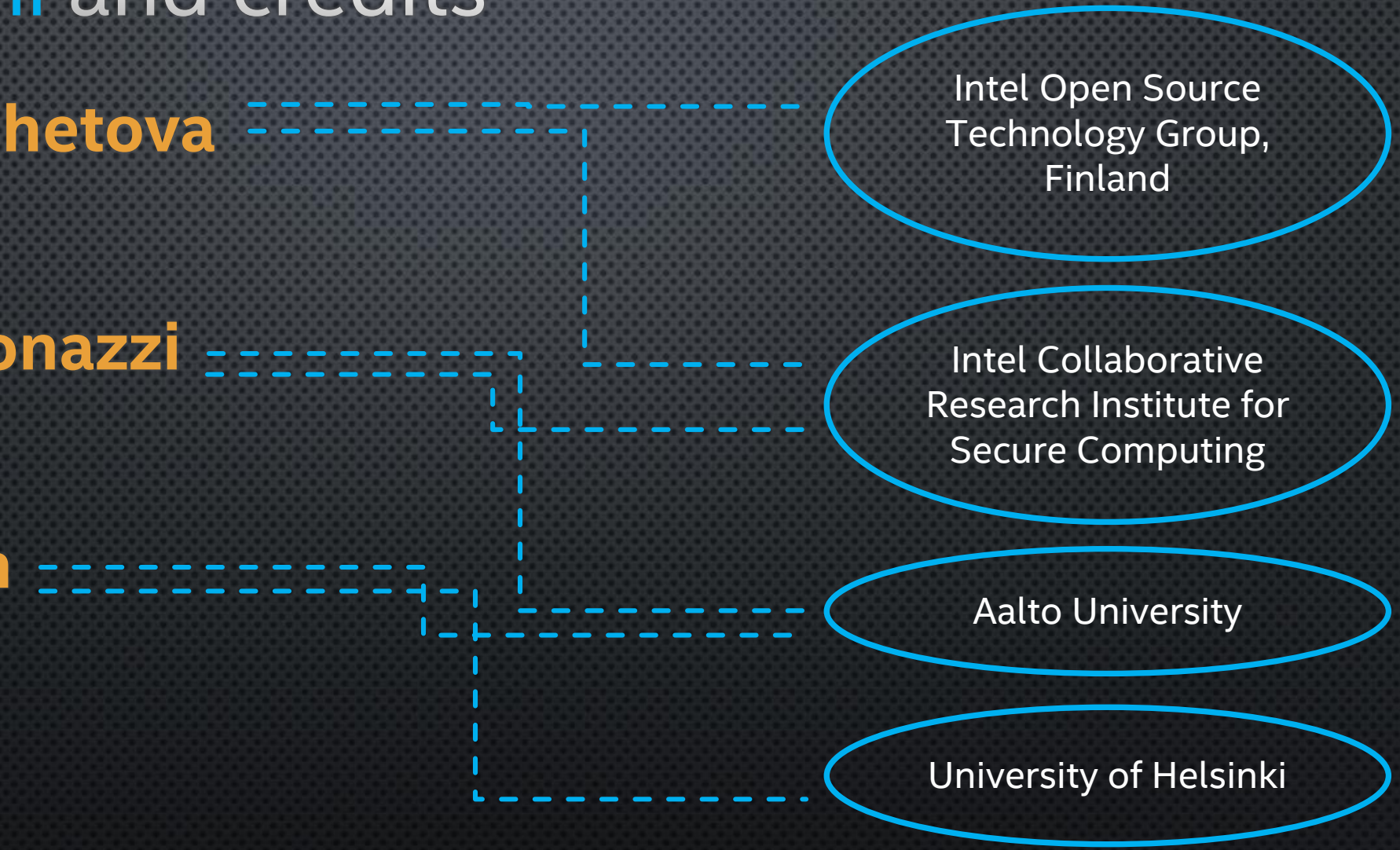
RANDOMIZATION CAN'T STOP BPF JIT SPRAY

Elena Reshetova

Filippo Bonazzi

N.Asokan

#whoami and credits



Special credits to **Daniel Borkmann** for really great discussions on BPF and JIT!

What you are about to hear...

- Overview of BPF
- JIT compiler for BPF
- Original JIT spray attack by Keegan McAllister
- Community response
- Our attack: making it real
- Demo
- Implemented mitigations

This work has been done within the upstream Kernel Self Protection Project

> The Berkeley Packet Filter (BPF) provides a raw interface to data link layers, permitting raw link-layer packets to be sent and received.
>
> BPF supports filtering packets, allowing a userspace process to supply a filter program that specifies which packets it wants to receive.

--Wikipedia

= A kernel component
allowing a userspace process
to supply a program and
get it executed in kernel context!

Overview of Berkeley Packet Filter

Where is it used?

Packet filtering, various tracepoints, seccomp...

Filter programs are written in machine language for BPF virtual machine

Operations allowed:

fetch data from the packet

arithmetic operations with constants and packet data

compare the results against constants or against data

BPF verifier - sanity checks on supplied BPF program

length, correct header and end, BPF instruction codes, etc.

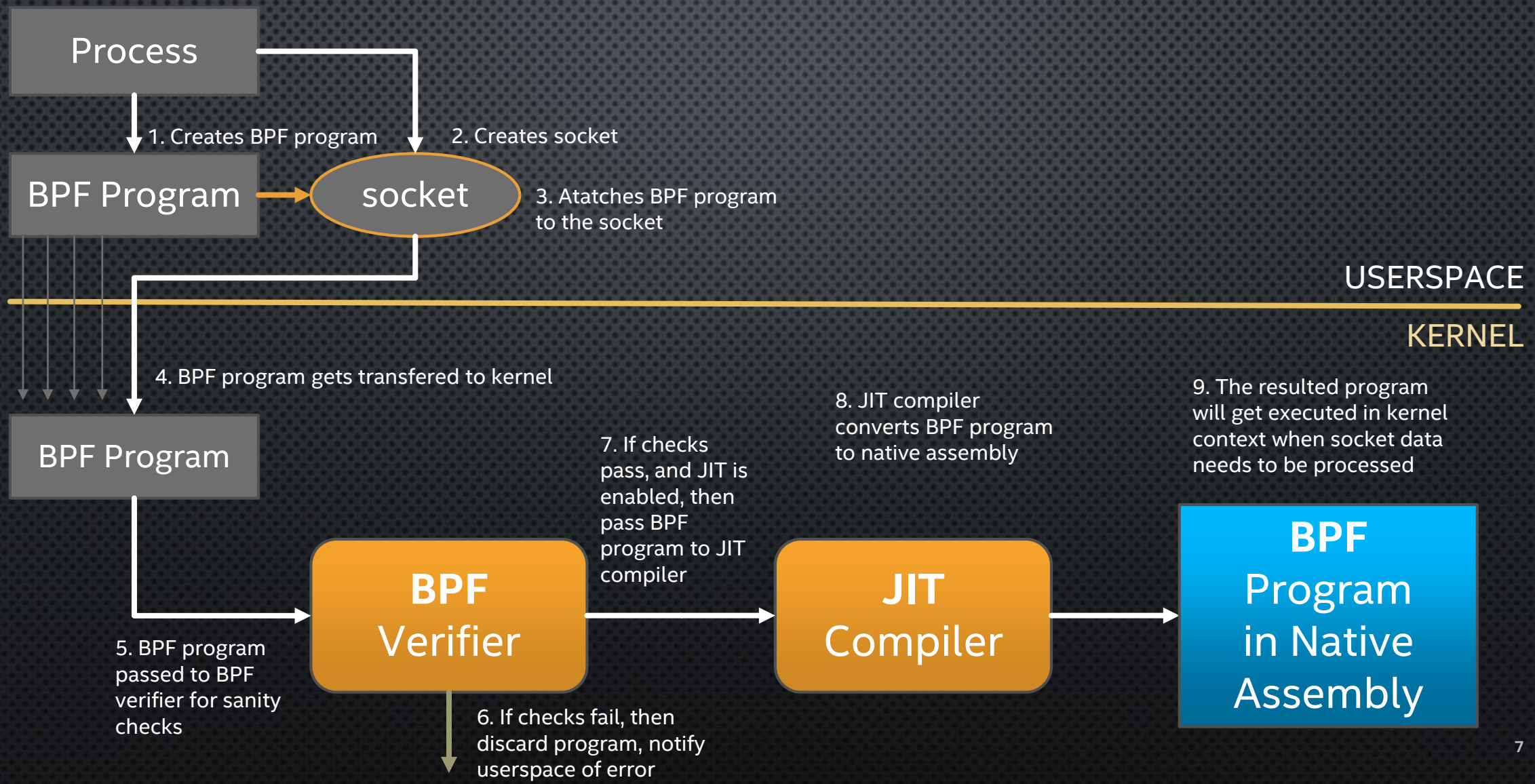
JIT compiler for BPF

Packet filtering needs to be SUPER FAST in order to be useful

Solution: Just-In-Time compiler for BPF

- Convert BPF instructions into native instructions
- Support for x86, ARM and others.
- Disabled by default on typical desktop machine
- Enabled on networking equipment such as routers ☺

What do we have so far?

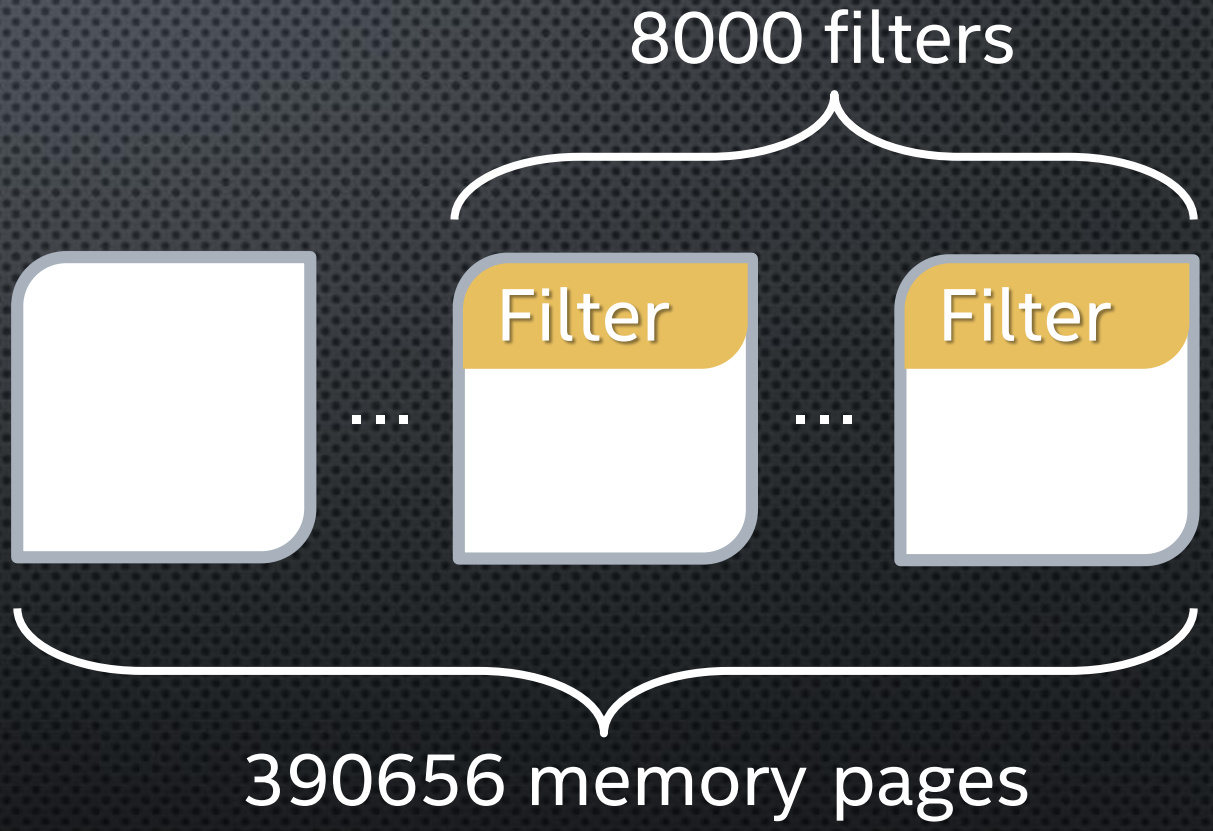


ORIGINAL JIT SPRAY ATTACK

by Keegan McAllister

2012

- Pass payload instructions as constants in different BPF instructions
- Populate address space with many filters
  - Use FD passing as a trick
- Randomly guess filter start page and jump to it



Passing payload instructions as constants

Pseudocode

x = 0xa8XXYYZZ
x = 0xa8PPQQRR
x = ...

Machine code

b8 ZZ YY XX a8
b8 RR QQ PP a8
...

Assembly (AT&T syntax)

mov $0xa8XXYYZZ, %eax
mov $0xa8PPQQRR, %eax
...

Using unaligned instruction execution, start executing from second byte

Machine code

ZZ YY XX
a8 b8
RR QQ PP
a8 b8
...

Assembly (AT&T syntax)

(payload instruction)
test $0xb8, %al
(payload instruction)
test $0xb8, %al

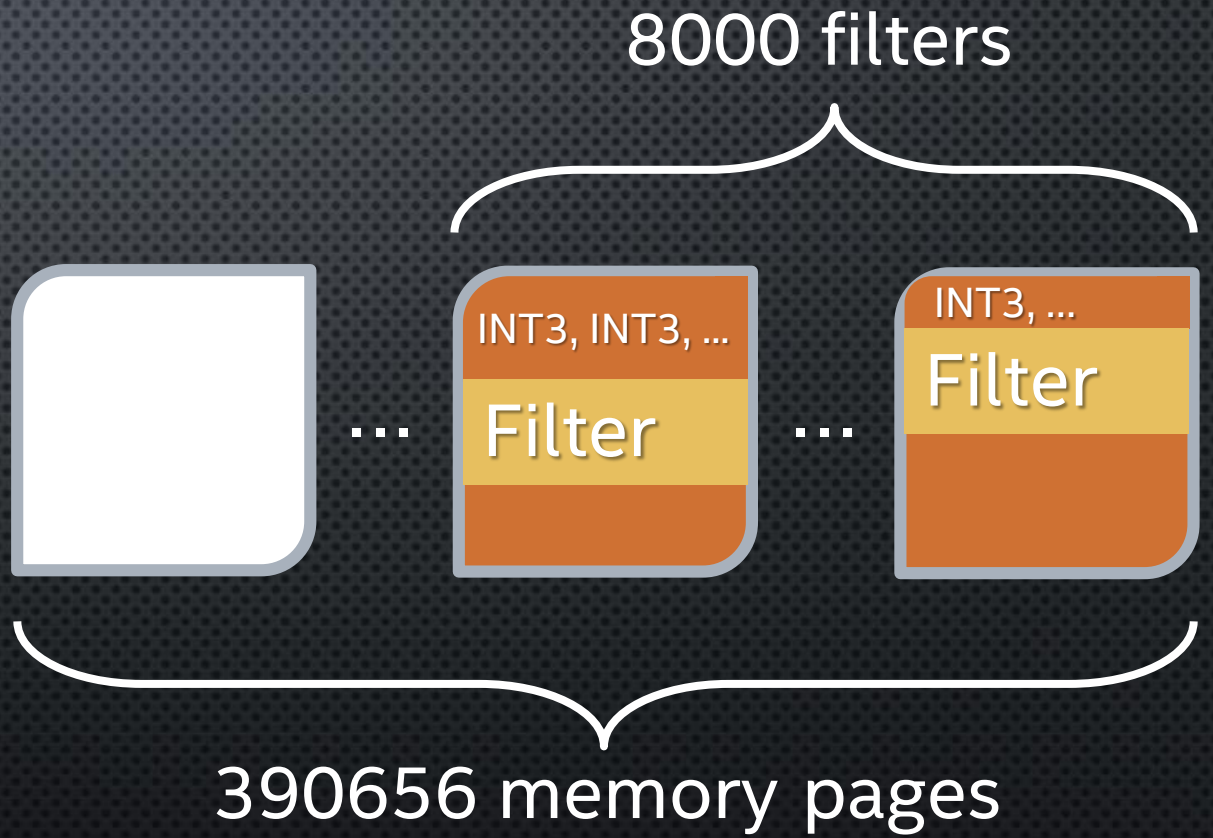
Example taken from http://mainisusuallyafunction.blogspot.de/2012/11/attacking-hardened-linux-systems-with.html

Community response

Grsecurity: blind constants in BPF instructions

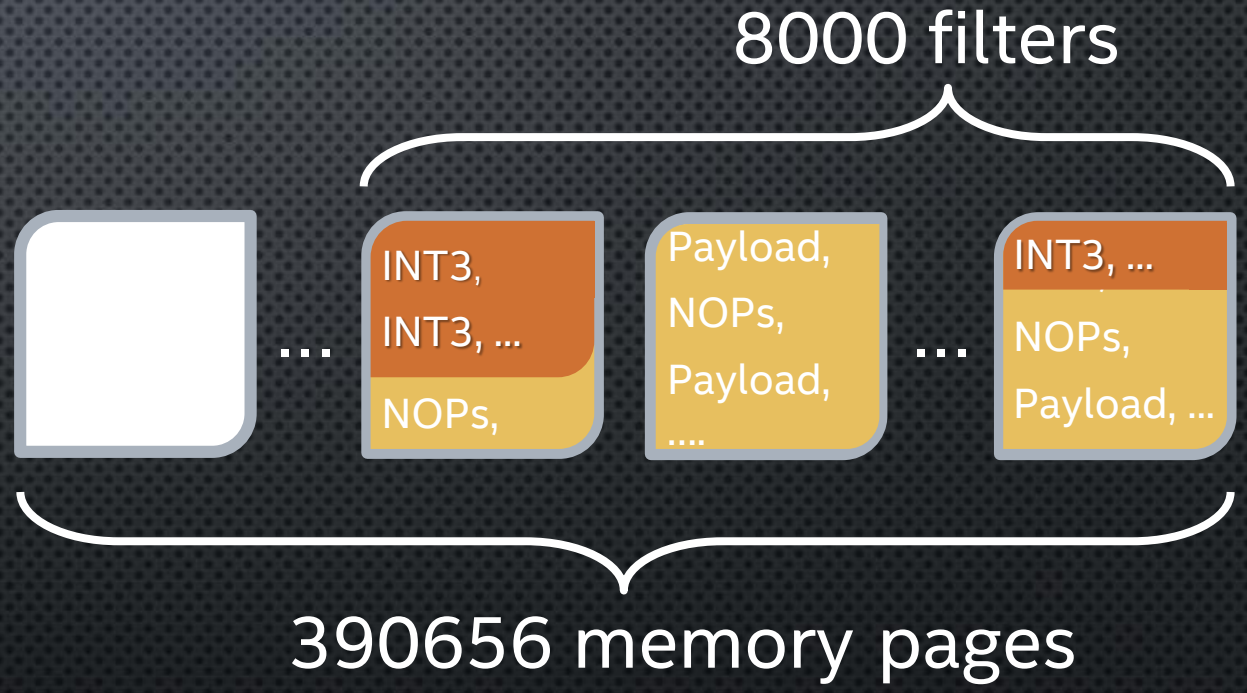
Upstream kernel: randomize BPF start address and fill the space with illegal instructions

No Attack Against Upstream Fix Was Presented



Our Attack: Approach #1

- Repeat payload enough times for filter to grow beyond one page
- Guess random page but try executing 10 consecutive offsets at page start to find payload
- Downside: we still jump to the beginning of the page and execute INT3 instructions in some cases



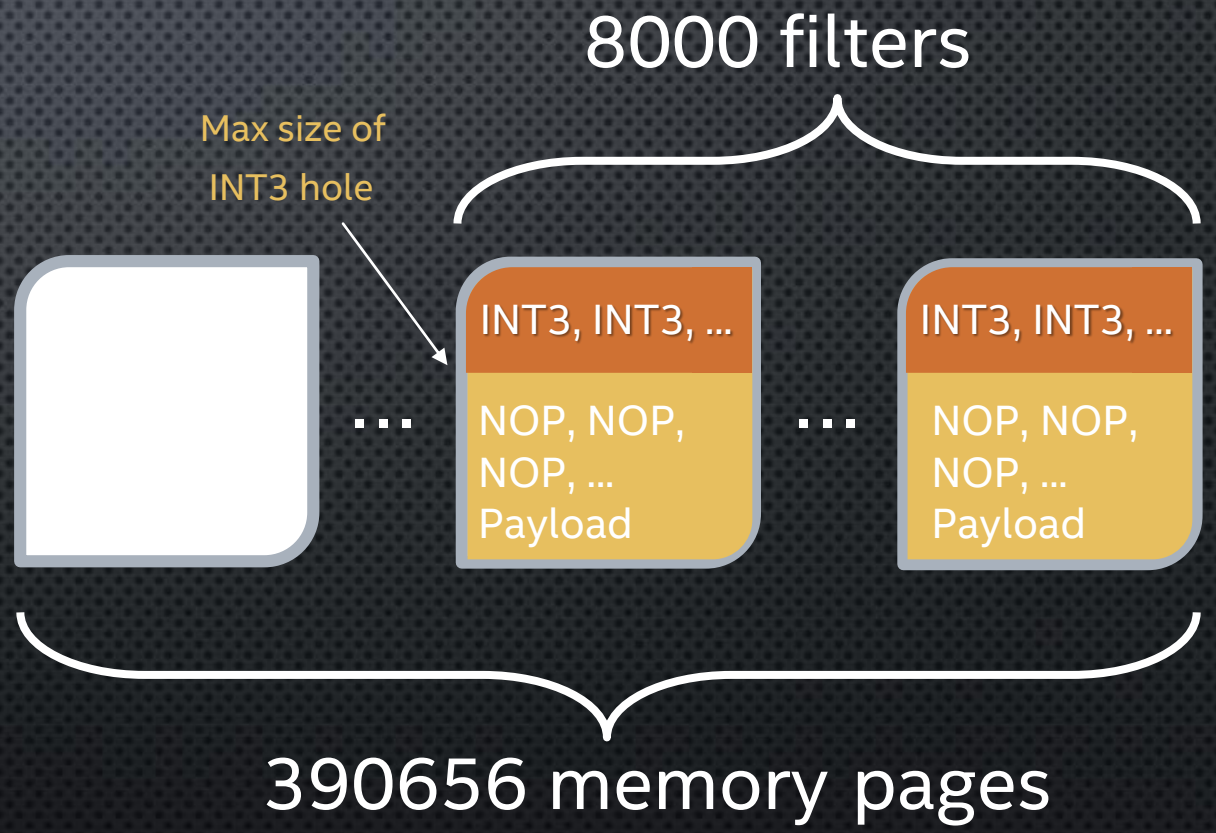
Our Attack: Approach #2

- Adjust filter size to fill exactly PAGE_SIZE – 128 – 4

  This forces the INT3 section to be max 132 bytes

- Make filter program many NOPs + payload at the end

- Guess random page, but jump past first 132 bytes to safely land on filter



DEMONSTRATION

Implemented Mitigations

BPF: add generic constant blinding for use in jits

Daniel Borkmann

Upstream Linux kernel commit 4f3446b and related

- No more payload instruction passing using constants

UNIX: properly account for FDs passed over UNIX sockets

Willy Tarreau

Upstream Linux kernel commit 712f4aa

- No more process limit bypass on number of UNIX sockets using FDs passing

KALSR feature for x86_64 in 4.8

GET INVOLVED!

Upstream Kernel-Self Protection Project (KSPP)

https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project

http://www.openwall.com/lists/kernel-hardening/

Exploits, proof of concepts, patches, reviews,... all needed!!

http://ssg.aalto.fi/projects/kernel-hardening

RANDOMIZATION

CAN'T STOP

BPF JIT SPRAY