
# She sells root shells by the C(++) shore

building a safe execution environment

**Costin Ionescu**, Distinguished Engineer, Symantec Enterprise Division

# whoami

# pwd

# hostname

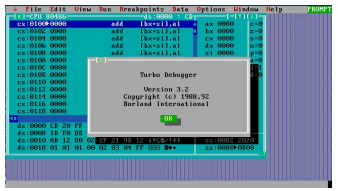
# cat $HISTFILE

costin.ionescu@broadcom.com

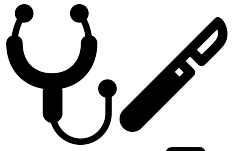
Symantec

BROADCOM®

- 19+ years



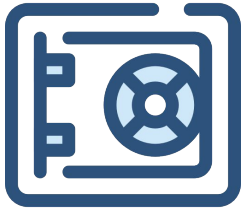

- threat analyst



- security tech dev

# crontab -e

- security solutions
- hardening
- CP3



# Security solutions

*Goals and challenges*

# Security Solutions

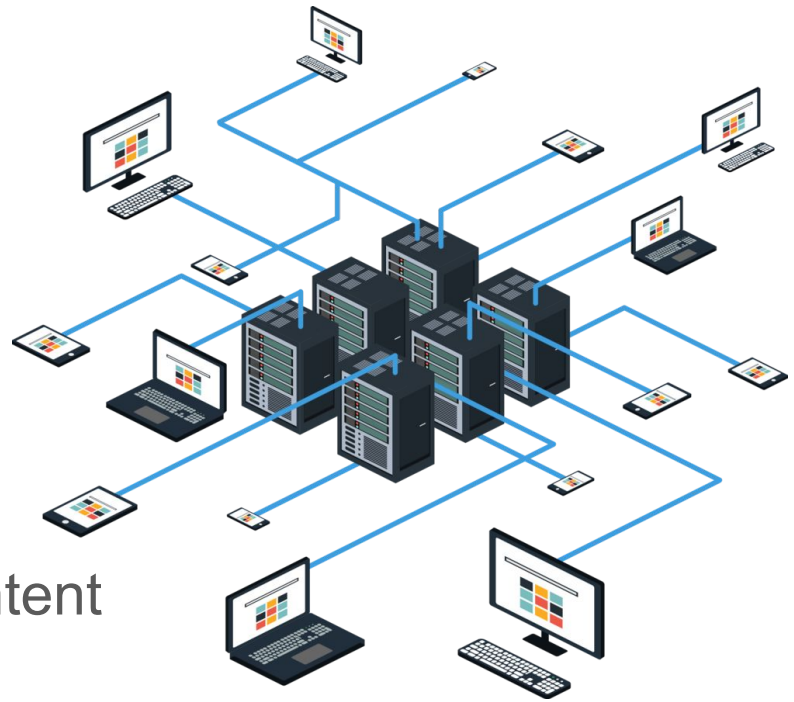
## Goals

- protect

- secure

- efficient

# Security Solutions



- wide spectrum
- rapidly update content
  - `[ [1.2 …] [1.1 …] … ]`
  - `c95f1c68c4...` *evil.bin*
  - `"rm\s+-rf /"` *evil.sh*
  - `new ActiveXObject ("Scripting.CloudSystemObject")` *evil.js*
  - `cmpxchg1024b kernel32!IsTheBuggerPresent` *evil.exe*
- hardened runtime environment

# Operating Environments

# Hardening

*Robust, reliable, resilient software*

# Hardening

## Process

- separate address space
- privilege control

```
  1  [||        2.6%]    4  [||        6.6%]   7  [          0.0%]    10 [|         0.7%]
  2  [|         0.7%]    5  [||        3.3%]   8  [||        1.3%]    11 [||        2.6%]
  3  [||        2.6%]    6  [||        1.3%]   9  [|         2.6%]    12 [|         0.7%]
  Mem[||||| ||                   4.34G/31.1G]   Tasks: 154, 999 thr; 1 running
  Swp[                                 0K/0K]   Load average: 0.33 0.30 0.35
                                                Uptime: 01:40:57

    PID  VIRT   RES   SHR S CPU% MEM% Command
   3747 3590M  410M  221M S  5.9  1.3                 ├─ firefox-esr -p --no-remote
   4413 3590M  410M  221M S  0.0  1.3                    ├─ firefox-esr -p --no-remote
   4314 3590M  410M  221M S  0.0  1.3                    ├─ firefox-esr -p --no-remote
   4268 3590M  410M  221M S  0.0  1.3                    ├─ firefox-esr -p --no-remote
   4254 2440M  146M 98424 S  0.7  0.5                    ├─ firefox-esr -contentproc -childID
   4304 2440M  146M 98424 S  0.0  0.5                       ├─ firefox-esr -contentproc -child
   4303 2440M  146M 98424 S  0.0  0.5                       ├─ firefox-esr -contentproc -child
   4302 2440M  146M 98424 S  0.0  0.5                       ├─ firefox-esr -contentproc -child
   4301 2440M  146M 98424 S  0.0  0.5                       ├─ firefox-esr -contentproc -child
   4300 2440M  146M 98424 S  0.0  0.5                       ├─ firefox-esr -contentproc -child
F1Help  F2Setup F3SearchF4FilterF5SortedF6CollapF7Nice -F8Nice +F9Kill  F10Quit
```

# Hardening

*Multi-process boundary*

# Multi-process Hardening

- process group boundaries
  - sessions
  - effective user ID, cgroups

- virtualization - stricter isolation


- containers - isolation with less overhead

# Hardening

*Process-level hardening technologies*

# Process-level Hardening

von Neumann architectures

- code + data = 

```
(gdb) run < aaa.txt
Starting program: /tmp/a.out < aaa.txt
read 32 items!
buffer: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x00000000004011d9 in bufover ()
(gdb) info frame
Stack level 0, frame at 0x7fffffffedfe8:
 rip = 0x4011d9 in bufover; saved rip = 0xa41414141414141
 called by frame at 0x7fffffffedff8
 Arglist at 0x4141414141414141, args:
 Locals at 0x4141414141414141, Previous frame's sp is 0x7fffffffedff0
 Saved registers:
  rip at 0x7fffffffedfe8
```

# Process-level Hardening

## No-Execute

- NX, XD, XN, FoE
- ro, rw, rx, rwx
- W^X policy

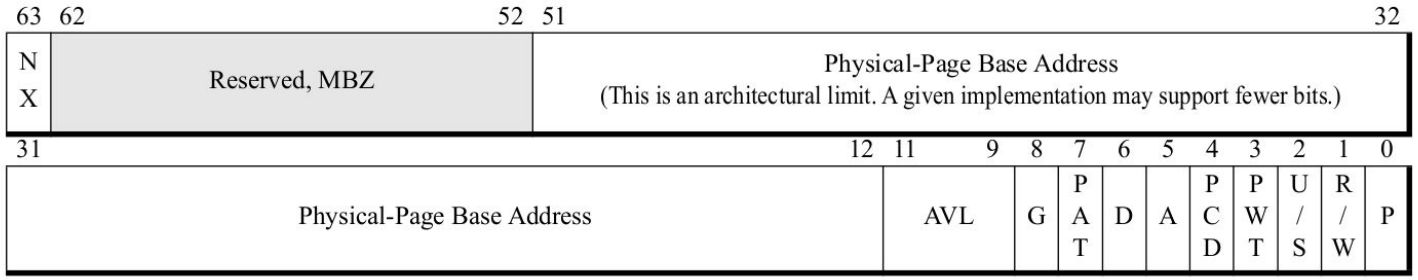



**Figure 5-12. 4-Kbyte PTE—PAE Paging Legacy-Mode**

# Process-level Hardening

## ASLR

- variable memory layout
- user mode:
  - module addresses
  - process control structures (PEB/TEB)
  - stacks, heaps
- kernel mode (KASLR):
  - kernel address
  - driver module addresses
  - page table mappings / hyperspace

# Process-level Hardening

## Stack Cookies

- compiler inserted checks
- mitigates stack buffer overflows
- weakness: guessing the value

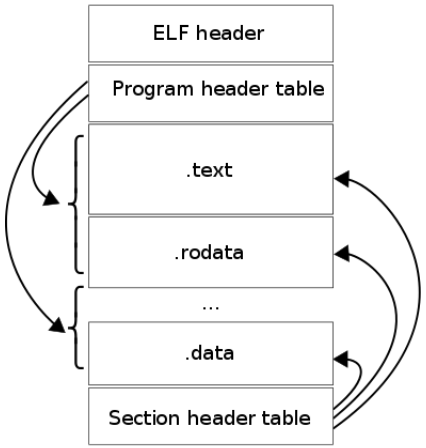
# Process-level Hardening

## Structured Exception Handling

- SEH
- VEH
- SafeSEH
- SEHOP

# Process-level Hardening

## GNU RELRO

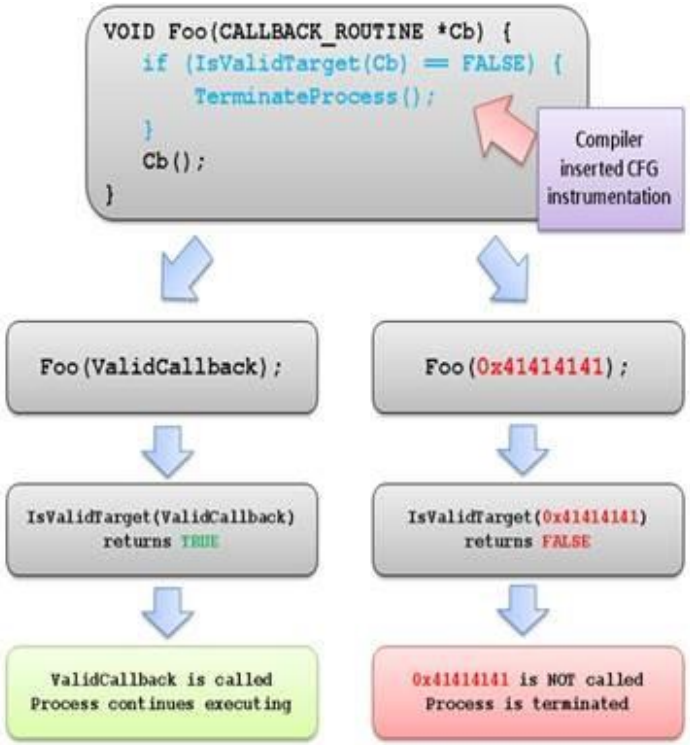

- ELF read-only sections have no relocations
- PLT/GOT easy targets to hijack execution
- RELRO turns pages read-only after processing relocations

# Process-level Hardening

## CFG

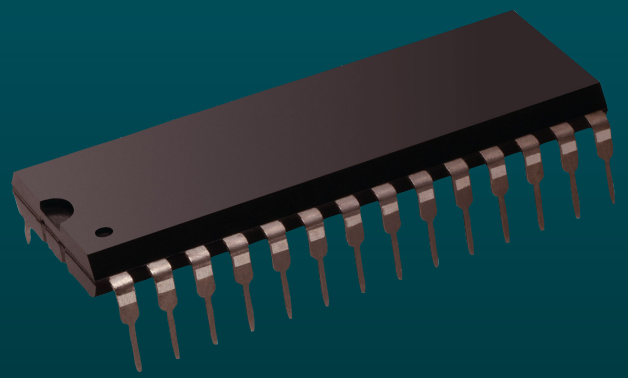
Microsoft - Control Flow Guard

- /guard:cf
- checks sparse bitmap before indirect call
- limitation: non CFG-enabled modules
- limitation: unaligned function addresses



# Process-level Hardening

## MPX



Intel MPX - Memory Protection Extensions

COMPILER EXPLORER | Add... | More | Sponsors PC-lint PVS-Studio Solid Sands | Share | Other

C++ source #1

```
1
2  int (*fn[5]) (int);
3
4  int caller (int a) {
5      return fn[a](a); }
```

x86-64 gcc 6.4 (Editor #1, Compiler #1) C++

x86-64 gcc 6.4 | -fcheck-pointer-bounds -mmpx -Ofast

Output... | Filter... | Libraries | Add new... | Add tool...

```
1   caller(int):
2           bndmov  bnd0, XMMWORD PTR __chkp_bounds_of_fn[rip]
3           movsx   rdx, edi
4           lea     rax, fn[0+rdx*8]
5           bndcl   bnd0, [rax]
6           bndcu   bnd0, [rax+7]
7           jmp     [QWORD PTR fn[0+rdx*8]]
8   __chkp_bounds_of_fn:
9           .quad   fn
10          .zero   8
11  __chkp_none_bounds:
12          .quad   -1
13          .quad   -1
14  __chkp_zero_bounds:
15          .zero   16
16  fn:
17          .zero   40
18  _GLOBAL__sub_B_00102_0_fn:
```

https://godbolt.org

# Process-level Hardening

## MPK

Intel MPK - Memory Protection Keys

### 2.7 PROTECTION-KEY RIGHTS REGISTERS (PKRU AND IA32_PKRS)

Processors may support either or both of two protection-key rights registers: PKRU for user-mode pages and the IA32_PKRS MSR (MSR index 6E1H) for supervisor-mode pages. 4-level paging and 5-level paging associate a 4-bit **protection key** with each page. The protection-key rights registers determine accessibility based on a page's protection key.

If CPUID.(EAX=07H,ECX=0H):ECX.PKU [bit 3] = 1, the processor supports the protection-key feature for user-mode pages. When CR4.PKE = 1, software can use the **protection-key rights register for user pages** (PKRU) to specify the access rights for user-mode pages for each protection key.

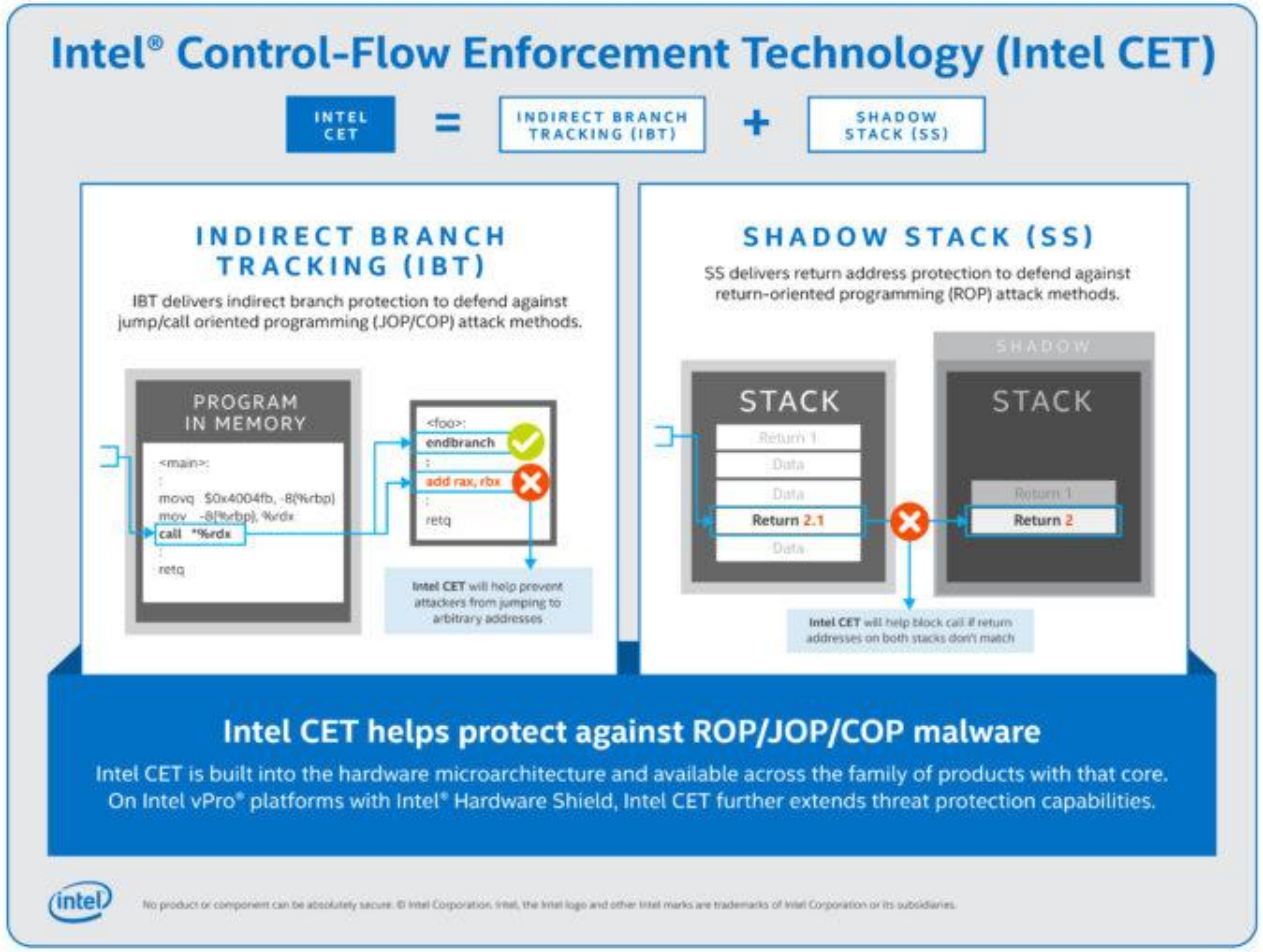
If CPUID.(EAX=07H,ECX=0H):ECX.PKS [bit 31] = 1, the processor supports the protection-key feature for supervisor-mode pages. When CR4.PKS = 1, software can use the **protection-key rights register for supervisor pages** (the IA32_PKRS MSR) to specify the access rights for supervisor-mode pages for each protection key.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Bit Position |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| WD15 | AD15 | WD14 | AD14 | WD13 | AD13 | WD12 | AD12 | WD11 | AD11 | WD10 | AD10 | WD9 | AD9 | WD8 | AD8 | WD7 | AD7 | WD6 | AD6 | WD5 | AD5 | WD4 | AD4 | WD3 | AD3 | WD2 | AD2 | WD1 | AD1 | WD0 | AD0 | |

Figure 2-9. Format of Protection-Key Rights Registers

# Process-level Hardening

## CET






BROADCOM

# Hardening

*In-process security boundaries*

# In-process hardening

# [P]NaCl

NaCl

- statically typed languages
- modified GCC toolchain
- single forward pass verification
- near native speed
- con: native code + web = no adoption

PNaCl

- 1 bitcode for all hosts
- clang/LLVM toolchain
- near native speed
- not adopted by Chrome competitors

# In-process hardening


**asm.js**

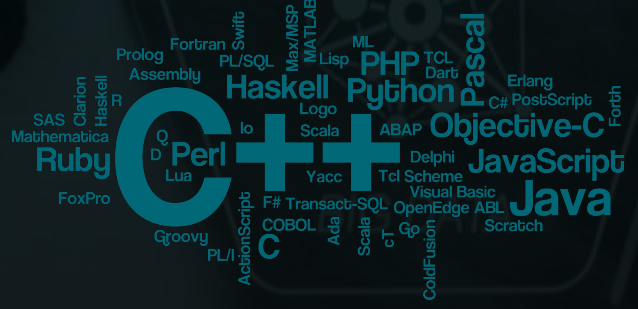
- better compatibility
- browser-specific optimisations
- LLVM derived toolchain - emscripten

**WebAssembly**

- C/C++/Rust -> LLVM
- virtual architecture
- supported by major browsers

# Language features for safety

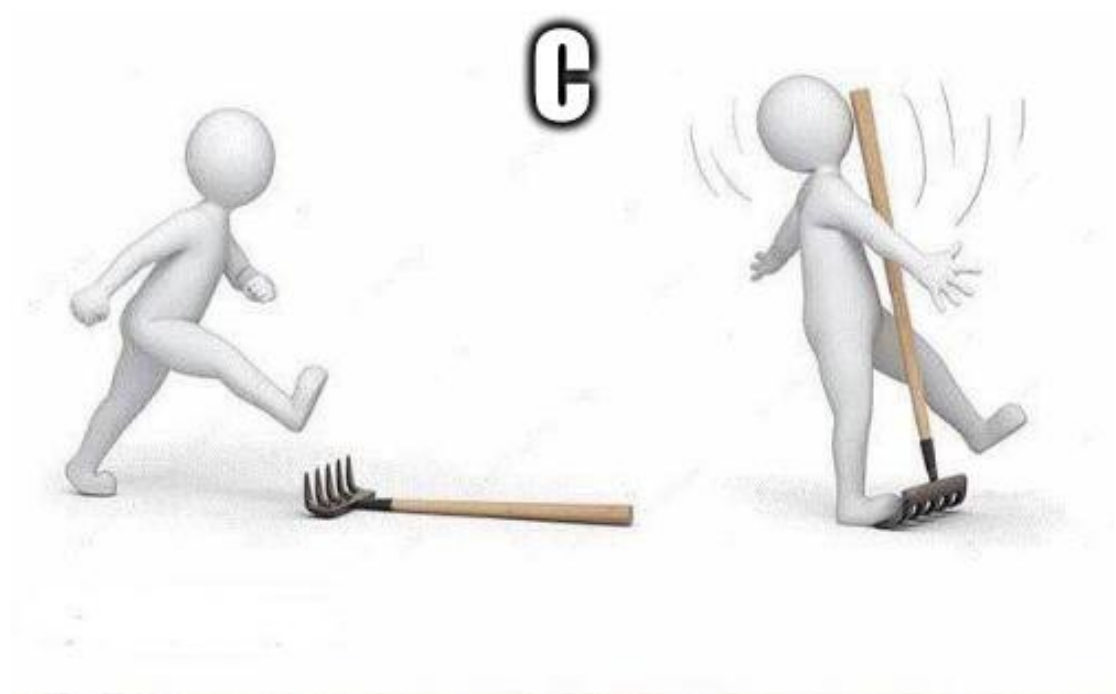
*Safe Execution Environment*

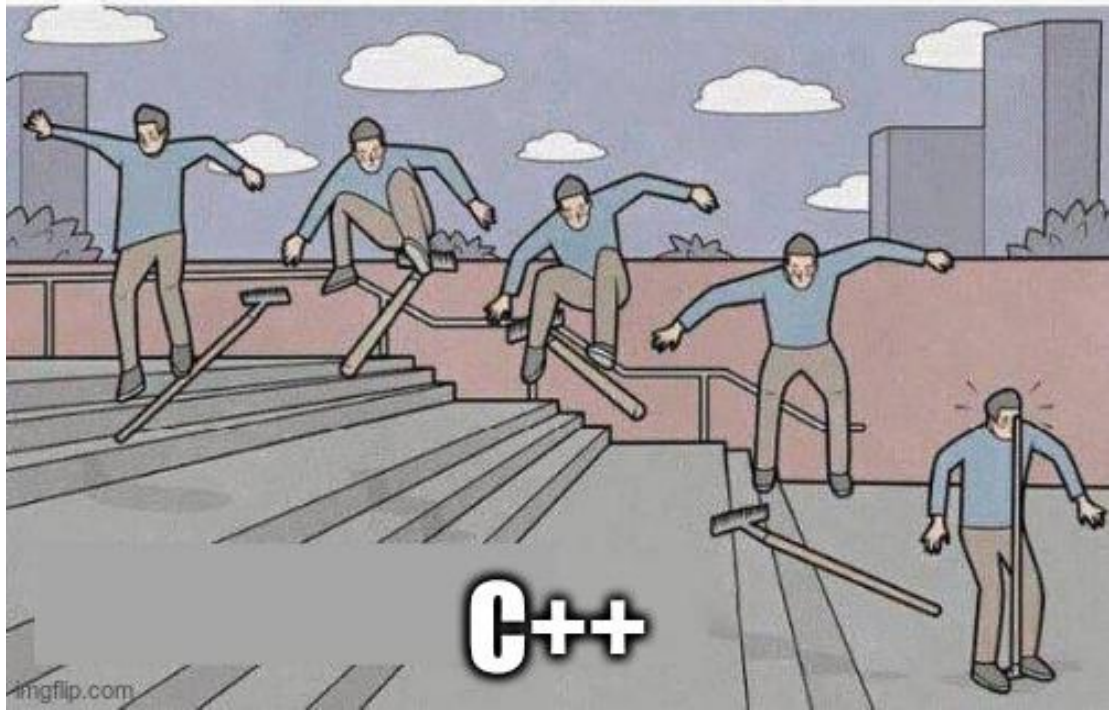


# Language features for safety

THIS PAGE INTENTIONALLY LEFT BLANK

# Language features for safety



# Language features for safety


**Pros**

- RAII
- smart pointers
- new standard every 3 years
- zero-cost abstraction

**Cons**

- compatibility
- lifetime
- concurrency
- undefined behaviour

# Language features for safety

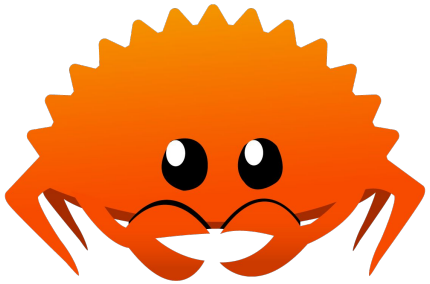
## Good practices


- warnings-as-errors
- high warning level
- static code analyzers
- code coverage
- ASan (address sanitizer)
- refactoring tools
- fuzzing

# Language features for safety

## Rust

- prevents memory corruption
- no data races
- aliasing ^ mutability
- explicit lifetimes
- no undefined behaviour
- FFI to interact with other languages
- runtime checks
- panic: controlled shutdown
- catch_unwind



# CP3

*Safe Execution Environment*

# CP3

# Safe execution environment

Quick recap:

- complex logic, rapidly updatable
- interpreters
  - domain-specific languages
  - limited performance

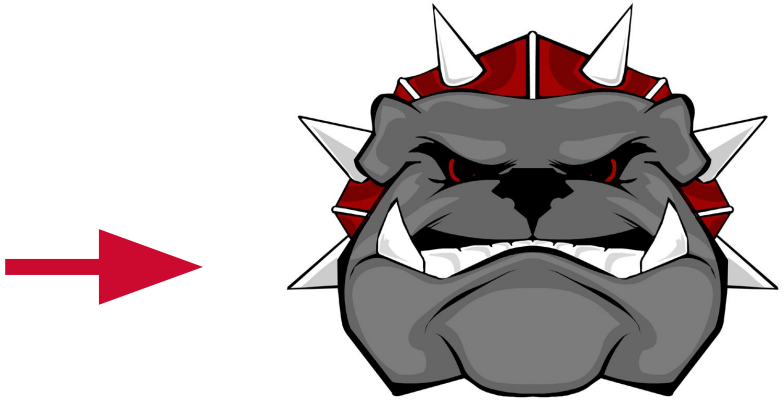
LLVM

- modular design
- intermediate representation
- language-focused frontend
- SSA form for optimisation
- ABI-focused backends

# CP3

# Safe execution environment - recipe -

instrumentation

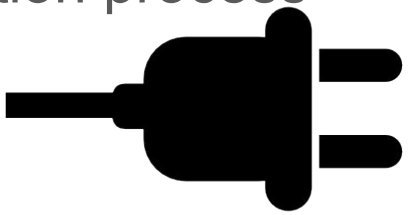


hosting library

# CP3

# Instrumentation

- pluggable compilation process

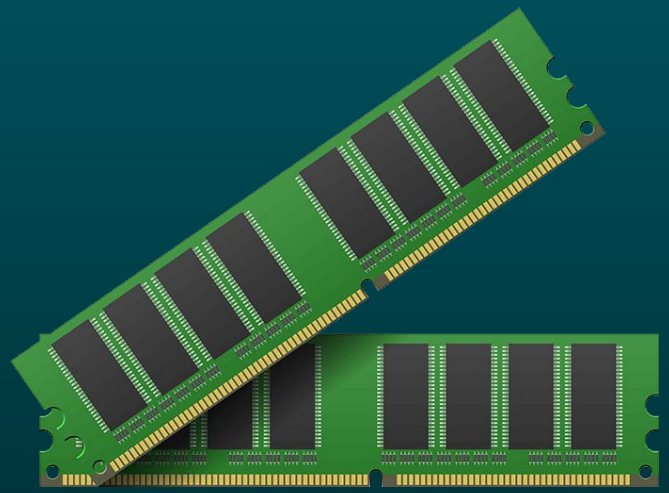


- instrumentation at IR layer

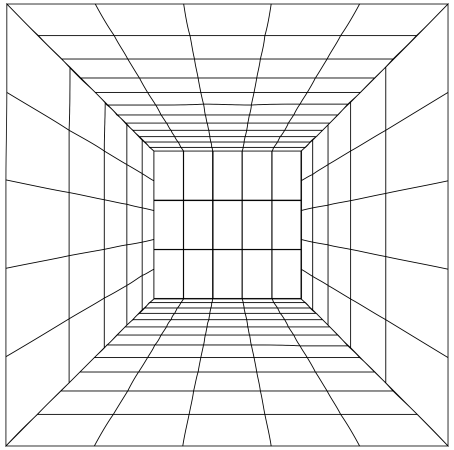
- small IR opcode set

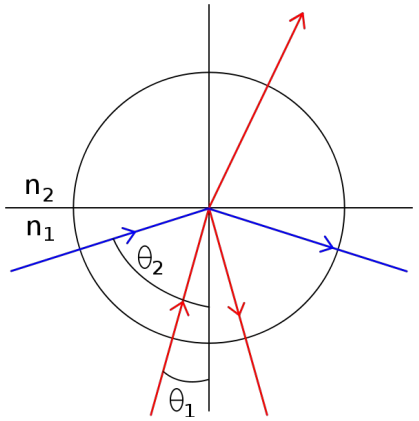
# CP3

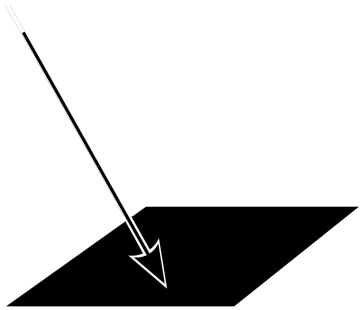
# Memory safety

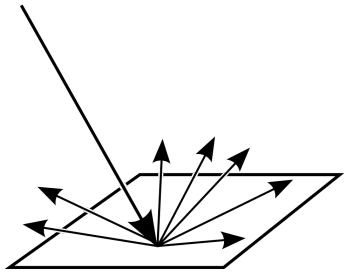


- load
- store
- fence
- cmpxchg
- atomicrmw





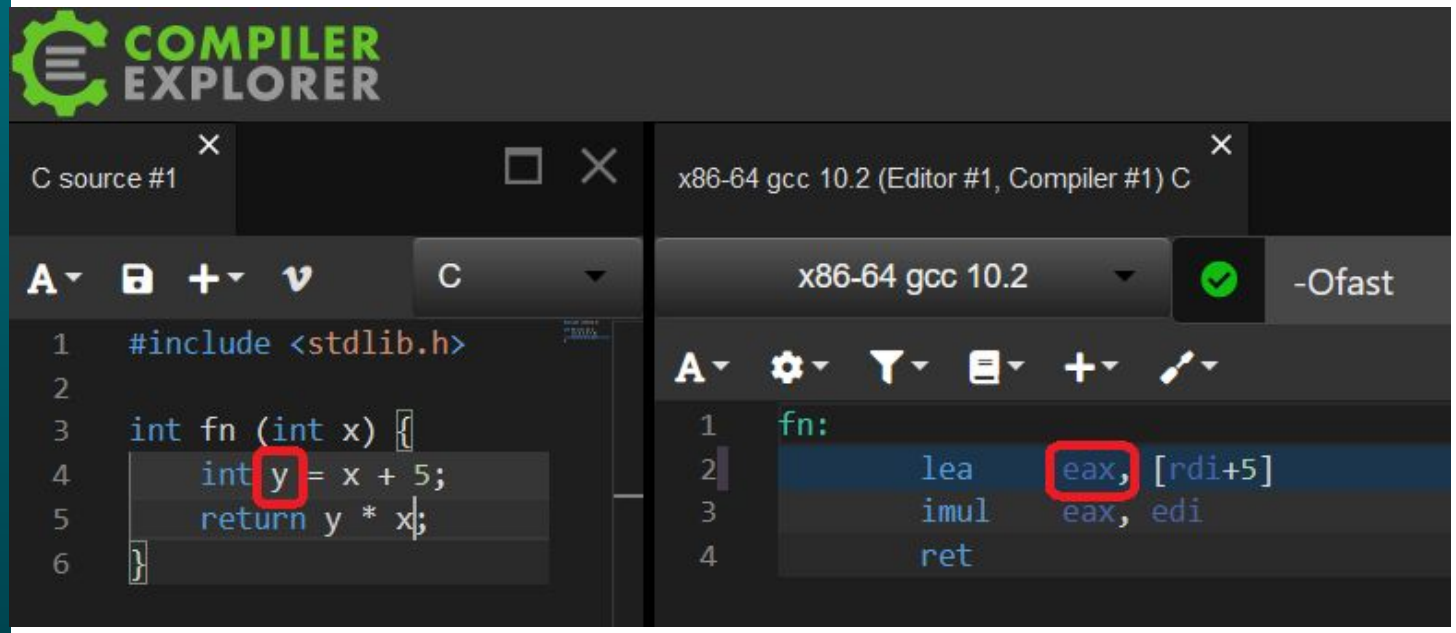





# CP3

## Stack safety

- follow ABI rules
- sensitive call stack information must be kept safe



# CP3

# Stack safety

COMPILER EXPLORER

C source #1

```c
include <stdlib.h>

nt fn (int x) {
    int y = x;
    return rand_r(&y)

```

x86-64 gcc 10.2 (Editor #1, Compiler #1) C

x86-64 gcc 10.2 -Ofast

```asm
fn:
        sub     rsp, 24
        mov     DWORD PTR [rsp+12], edi
        lea     rdi, [rsp+12]
        call    rand_r
        add     rsp, 24
        ret
```

## ‘alloca’ Instruction

### Syntax:

```
<result> = alloca [inalloca] <type> [, <ty> <
<
```

### Overview:

The ‘alloca’ instruction allocates memory on the

be automatically released when this function retu

# CP3

## Stack safety

```c
#include <stdio.h>
#include <string.h>

int bufover (FILE * f) {
    char buf[0x20];
    size_t n = fread(buf, 2, sizeof(buf), f);
    fprintf(stderr, "read %zu items!\n", n);
    buf[sizeof(buf) - 1] = 0;
    fprintf(stderr, "buffer: %s\n", buf);
    return (int) n;
}

int main (void) {
    char dashes[0x20];
    fread(dashes, 1, 1, stdin);
    memset(dashes, '-', sizeof(dashes));
    int n = bufover(stdin);
    dashes[sizeof(dashes) - 1] = 0;
    fprintf(stderr, "dashes: %s\n", dashes);
    return n;
}
```

# CP3

# Stack safety

```
(gdb) run < aaa.txt
Starting program: /tmp/a.out < aaa.txt
read 32 items!
buffer: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x00000000004011d9 in bufover ()
(gdb) info frame
Stack level 0, frame at 0x7fffffffedfe8:
 rip = 0x4011d9 in bufover; saved rip = 0xa41414141414141
 called by frame at 0x7fffffffedff8
 Arglist at 0x4141414141414141, args:
 Locals at 0x4141414141414141, Previous frame's sp is 0x7fffffffedff0
 Saved registers:
  rip at 0x7fffffffedfe8
```

```
read 32 items!
buffer: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
dashes: AAAAAAAAAAAAAAAA---------------
```

# CP3

# Function calling

```c
#include <stddef.h>

// source 1
int add (int a, int b)
{
    return a + b;
}

// source 2
void add (int * r, int const * v, size_t n);

int fn (int const * v)
{
    int x;
    add(&x, v, 2);
    return x;
}
```

# CP3

# Function calling

x86-ms-abi:

- stdcall: _symbol@N
- fastcall: @symbol@N

```c
int __fastcall square (double num)
{
    return num * num;
}

int __fastcall mul (int a, int b)
{
    return a * b;
}
```

```asm
_num$ = 8
@square@8 PROC
        movsd   xmm0, QWORD PTR _num$[esp-4]
        mulsd   xmm0, xmm0
        cvttsd2si eax, xmm0
        ret     8
@square@8 ENDP

@mul@8  PROC
        imul    ecx, edx
        mov     eax, ecx
        ret     0
@mul@8  ENDP
```

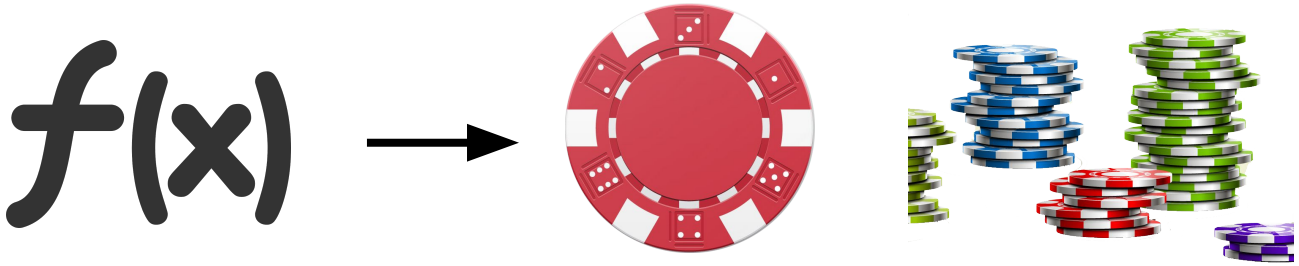
Solutions:

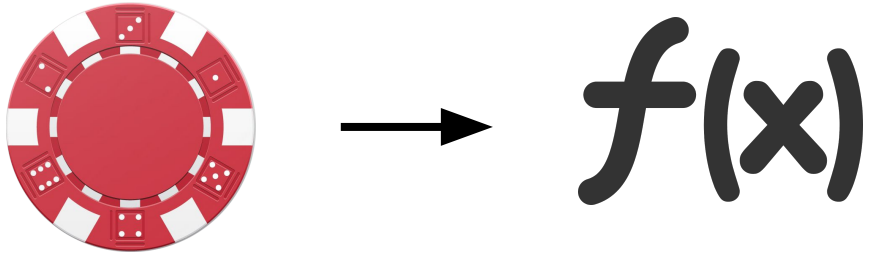
- decorate symbols
- cross-check definitions and calls

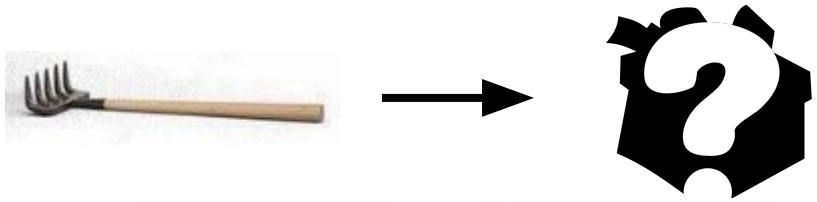
# CP3

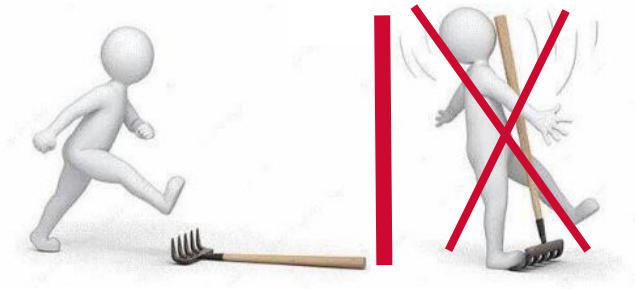
# Indirect function calls

```
void (* fn_ptr) (...);
class GotVTable { ~GotVTable () {} };
```





# CP3

# Indirect function calls

```c
#include <stdint.h>
#include <stdio.h>

void greet1 (short x) {
    fprintf(stderr, "Greetings mortals!\n");
}

void greet2 (short x) {
    fprintf(stderr, "Hello world!\n");
}

void (*greeters[]) (short) = {
    greet2,
    greet2,
    greet1
};

int main (void) {
    char x;
    void (*fn) (short);

    fn = greeters[fread(&x, 1, 1, stdin)];
    /* fn = greeters[0 or 1] = greet2 */
    fn(42);

    *(uintptr_t *) &fn += 4; // huh?!
    fn(42);

    return 0;
}
```

# CP3

# Indirect function calls

## Native run

```
(gdb) run </dev/null
Starting program: /tmp/a.out </dev/null
Hello world!
Hello world!

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff79ba00 in _IO_2_1_stdin_ () from /lib/x86_64-linux-gnu/libc.so.6
(gdb) bt
#0  0x00007ffff79ba00 in _IO_2_1_stdin_ () from /lib/x86_64-linux-gnu/libc.so.6
#1  0x0000000000401050 in fprintf@plt ()
#2  0x0000000000401184 in greet2 ()
#3  0x0000000000401230 in ?? ()
#4  0x00007ffff60409b in __libc_start_main (main=0x4011c0 <main>, argc=1, argv=0
    at ../csu/libc-start.c:308
#5  0x000000000040107a in _start ()
```

## CP3 run

```
Hello world!
Greetings mortals!
```

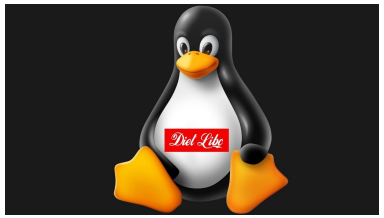
## External calls

# CP3

# Support libraries

- must compile all libs
- libc & libc++
- replace syscalls
- LLVM libc++
- libc

uclibc




dietlibc

musl

newlib


bionic

glibc

# CP3

# Hosting library

# CP3

# Trapping Exceptions

- **SEH**
- **VEH**



**signal**

- SIGSEGV
- SIGBUS
- SIGABRT
- SIGTRAP
- SIGFPE
- SIGILL

# CP3

# Timeout protection

```
for (;;);
```

SuspendThread
GetThreadContext
SetThreadContext

pthread_kill
sigsetjmp
siglongjmp

# CP3

# Stack exhaustion

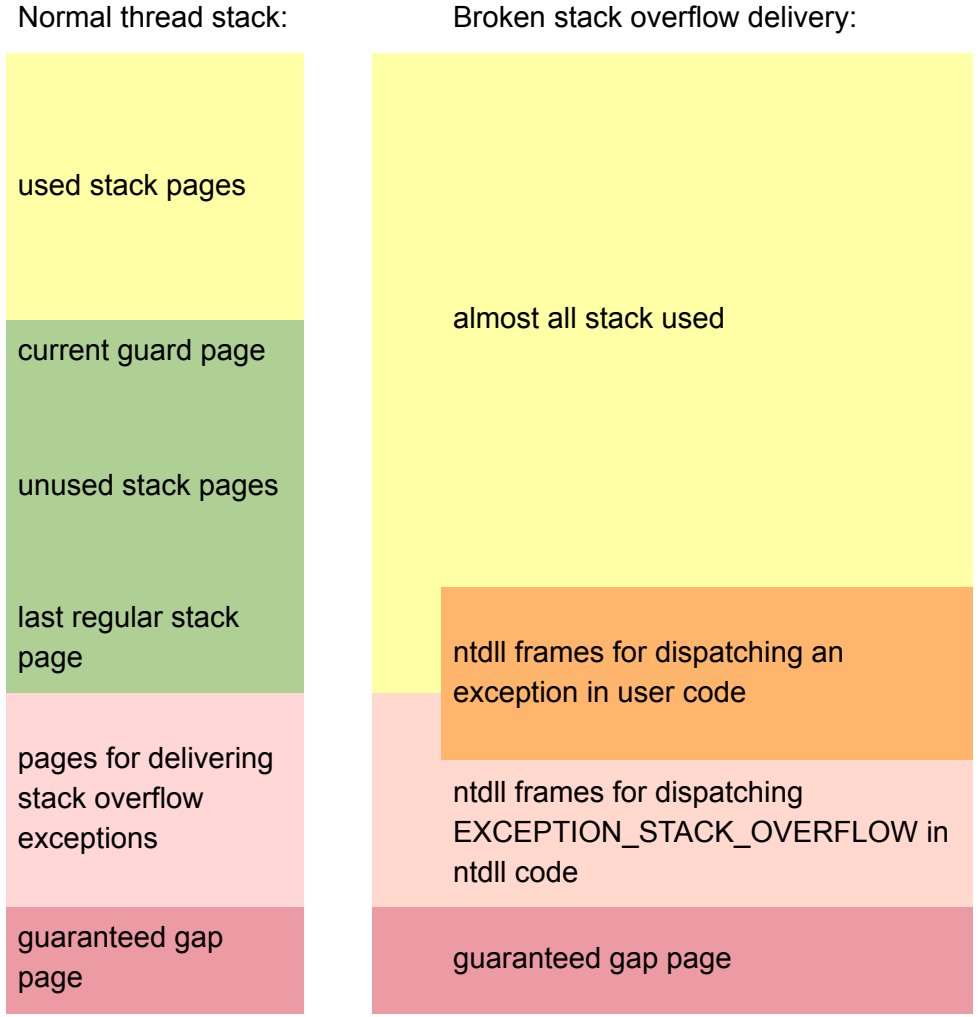
```
unsigned long long fibonacci (unsigned int n)
{
    return
        n < 2
        ? n
        : fibonacci(n - 1) + fibonacci(n - 2);
}


void in_order_traverse (node_t const * node)
{
    if (node->left) traverse(node->left);
    print_node(node);
    if (node->right) traverse(node->right);
}
```

# CP3

# Stack exhaustion

EXCEPTION_STACK_OVERFLOW



# CP3

# Stack exhaustion

`clang -fstack-check` ✗

"probe-stack"

This attribute indicates that the function will trigger
stack must be no further apart than the size of the
string value, the name of the stack probing functio

✓

# CP3

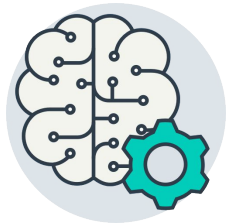
# Tech comparison

| | | |
|---|---|---|
| | NaCl/PNaCl | 5-15% |
| | PNaCl for AMD64 | 25% |
| WA WEBASSEMBLY | | 55% |
| | | 45% |
| CP3 | | 15% (10-35%) |

# CP3

# Uses

Static scanning technologies





Scan events

AMSI

cmd line

WMI

.reg

# Conclusion

BROADCOM®

# Thank You

costin.ionescu@broadcom.com