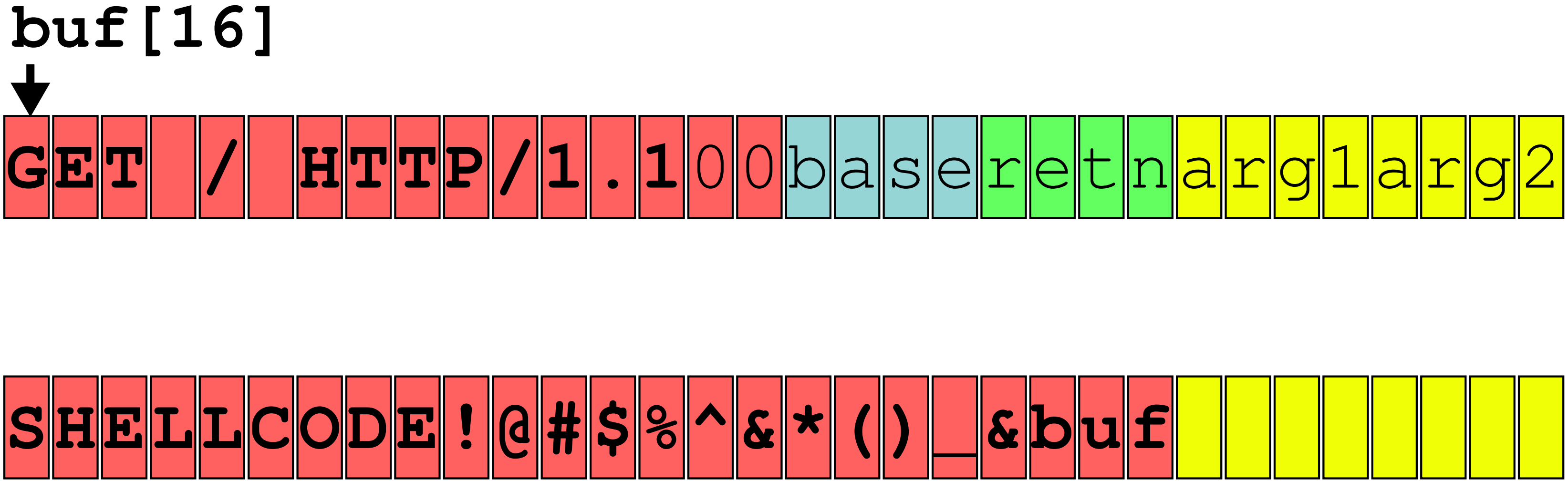
Minemu

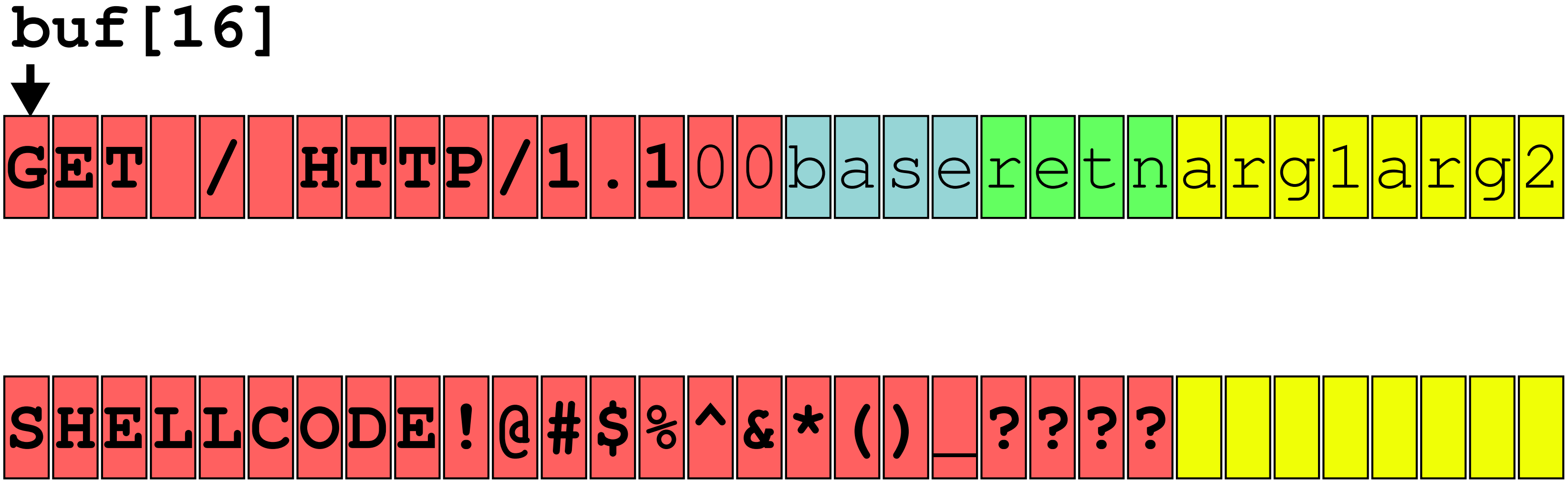
Protecting buggy software from memory corruption attacks

VU VRIJE UNIVERSITEIT AMSTERDAM

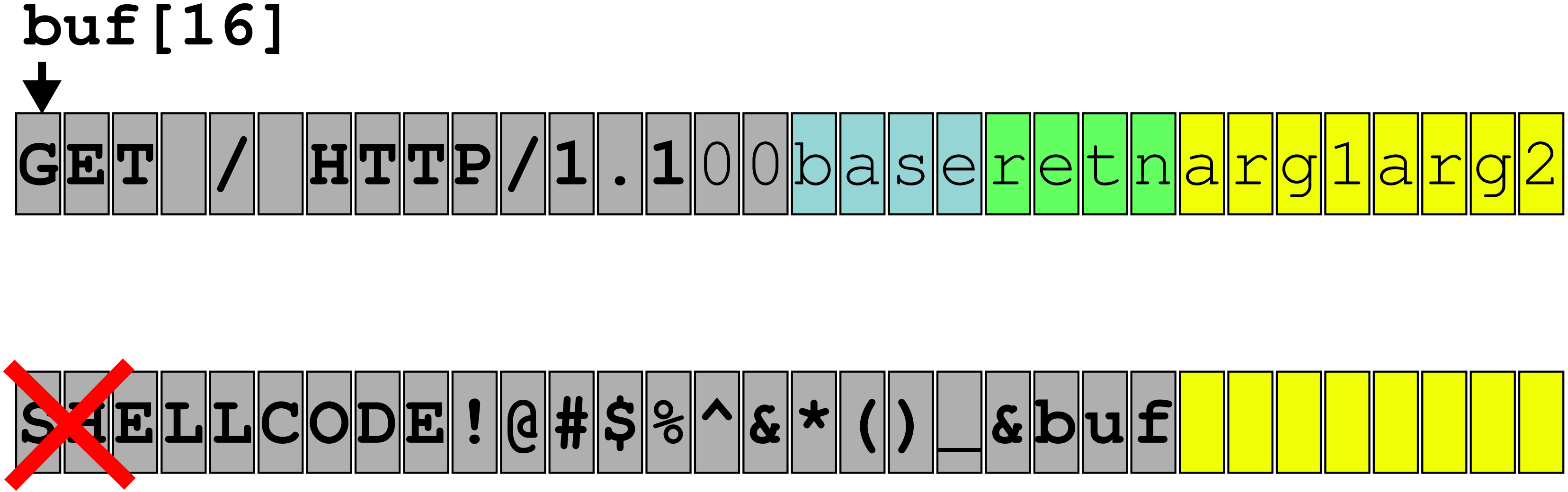
Traditional Stack Smashing



Address Space Layout Randomisation



DEP / NX



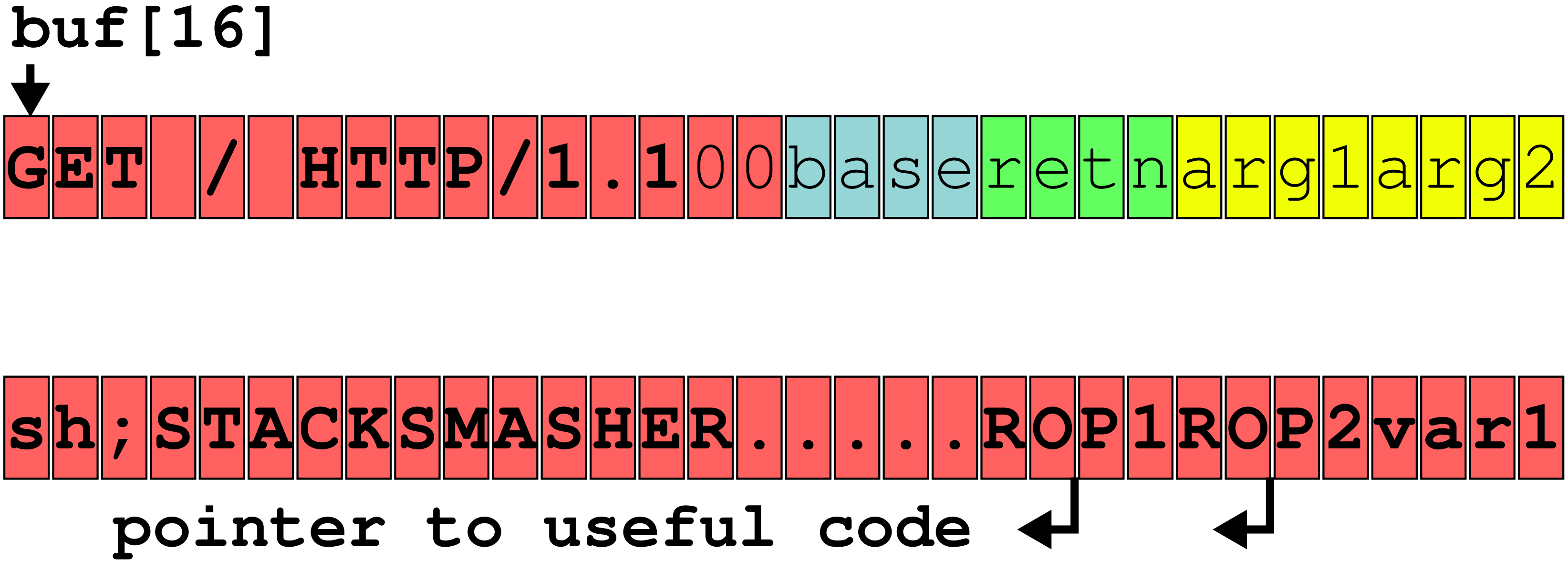

FORTIFY ALL THE THINGS!

This is still not enough

- ASLR can be brute forced

- Protecting against heap overflows is much harder than against stack overflows.

Return Oriented Programming



But the situation is even worse

But the situation is even worse

- needs to be enabled at compile time, and
  there is a lot of old code out there

But the situation is even worse

- needs to be enabled at compile time, and there is a lot of old code out there

- many packages do not apply these defence mechanisms even today

But the situation is even worse

- needs to be enabled at compile time, and there is a lot of old code out there

- many packages do not apply these defence mechanisms even today

- flaws in how ASLR/stack cookies are implemented

Can we do more?

Can we do more?

>> DEP prevents untrusted data from being run as code

Can we do more?

>> DEP prevents untrusted data from being run as code

<< ROP replaces untrusted code with pointers
   to original code.

Can we do more?

>> DEP prevents untrusted data from being run as code

<< ROP replaces untrusted code with pointers
   to original code.

>> Can we prevent untrusted pointers from being used
   as jump addresses?

Taint analysis

```
0805be60   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
0805be70   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
0805be80   00 00 00 00 02 00 00 00  d8 4b 06 08 a0 2e 05 08   |.........K......|
0805be90   94 be 05 08 78 a0 04 08  ef be ad de a4 be 05 08   |....x...........|
0805bea0   ac be 05 08 2f 62 69 6e  2f 73 68 00 a4 be 05 08   |..../bin/sh.....|
0805beb0   00 00 00 00 45 49 4e 44  42 41 5a 45 4e 45 49 4e   |....EINDBAZENEIN|
0805bec0   44 42 41 5a 45 4e 45 49  4e 44 42 41 5a 45 4e 45   |DBAZENEINDBAZENE|
0805bed0   00 00 00 00 41 5a 45 4e  90 be 05 08 ef 1f 05 08   |....AZEN........|
0805bee0   ff fa 26 08 ff f0 00 00  00 00 00 00 00 00 00 00   |..&.............|
0805bef0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
0805bf00   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
```

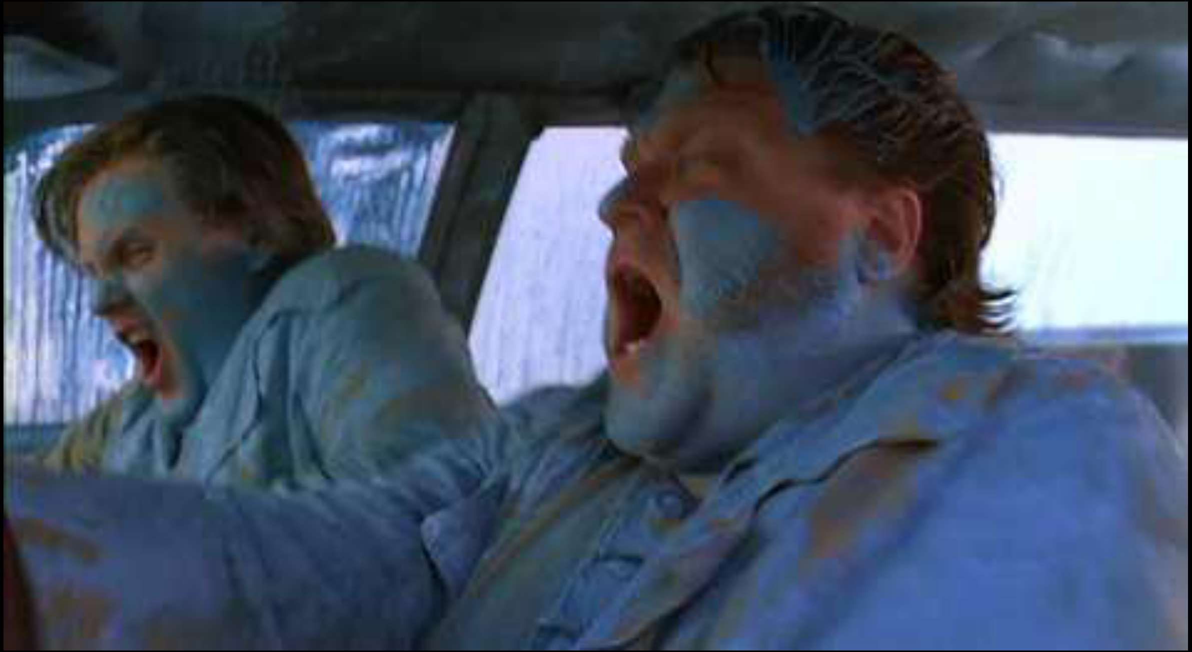
Taint tracking (1/2):

- remember whether data is trusted or not
- untrusted data is 'tainted'
- when data is copied, its taint is copied along
- taint is ORed for arithmetic operations, except
  when the result is always 0

Taint tracking (2/2):

When the code jumps to an address in memory,
the source of this address is checked for taint.

eg.:
- RET
- CALL *%eax
- JMP *0x1c(%ebx)



Taint tracking

photo: sammydavisdog@flickr

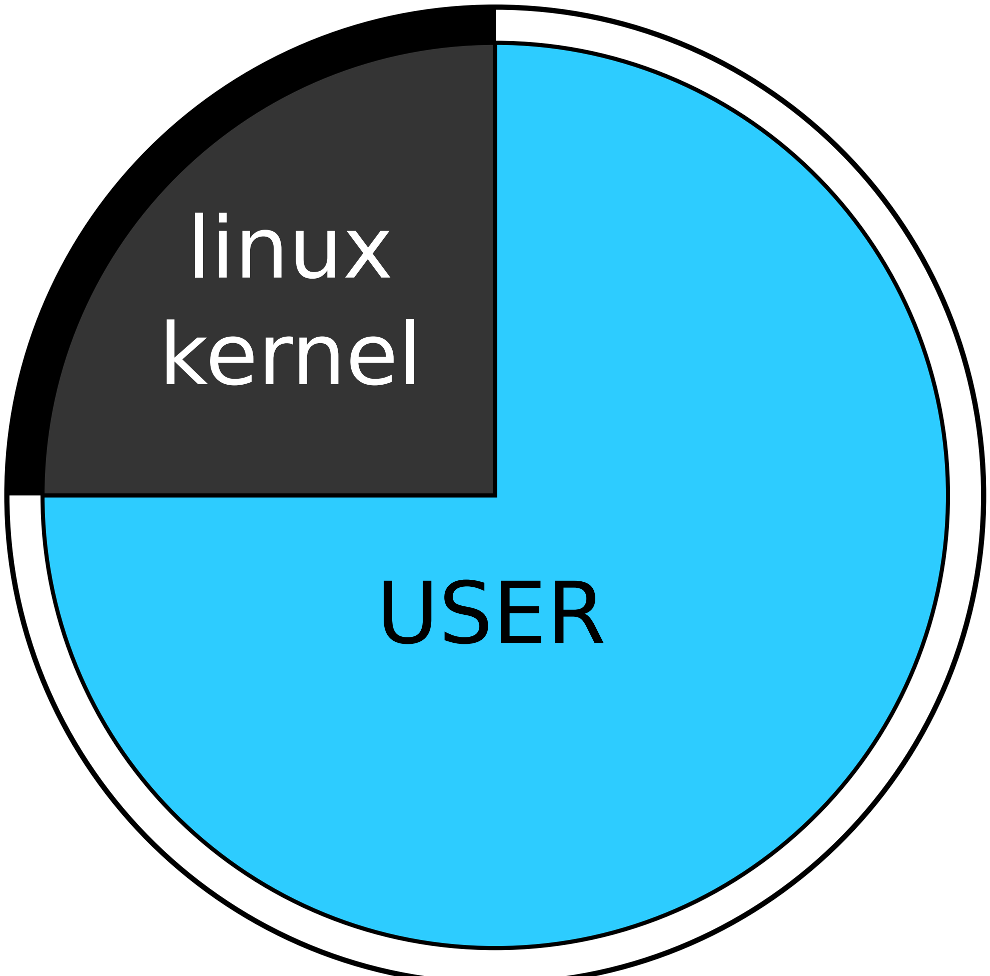
useful, but slow as hell

Is this slowness fundamental?

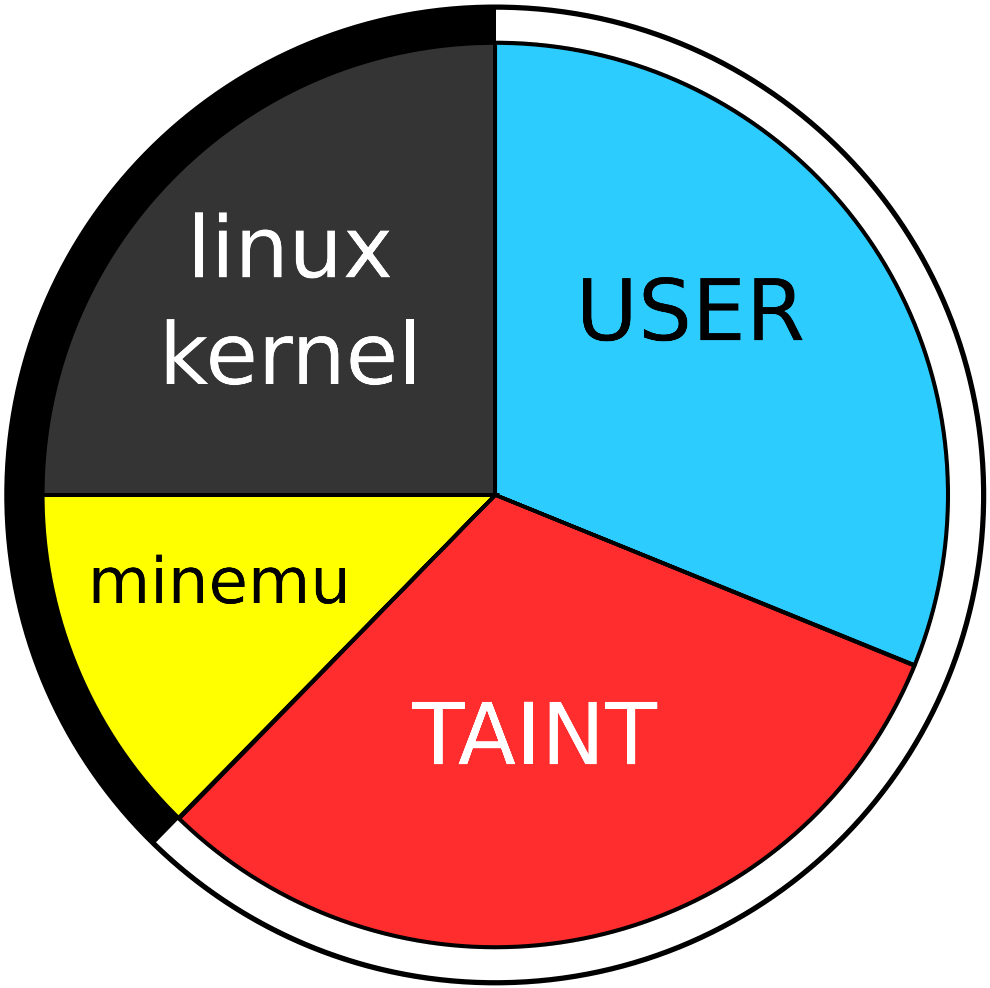
minemu

- memory layout
  use SSE registers to hold taint

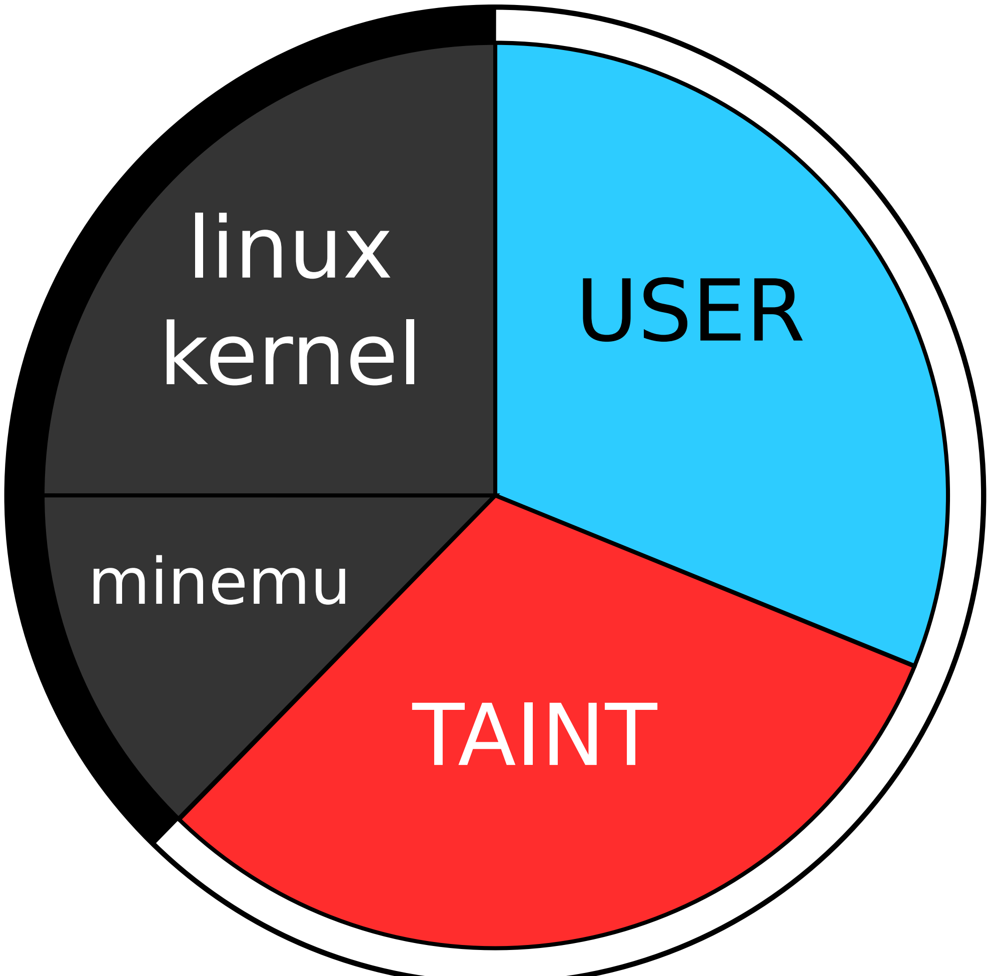
Memory layout (linux)



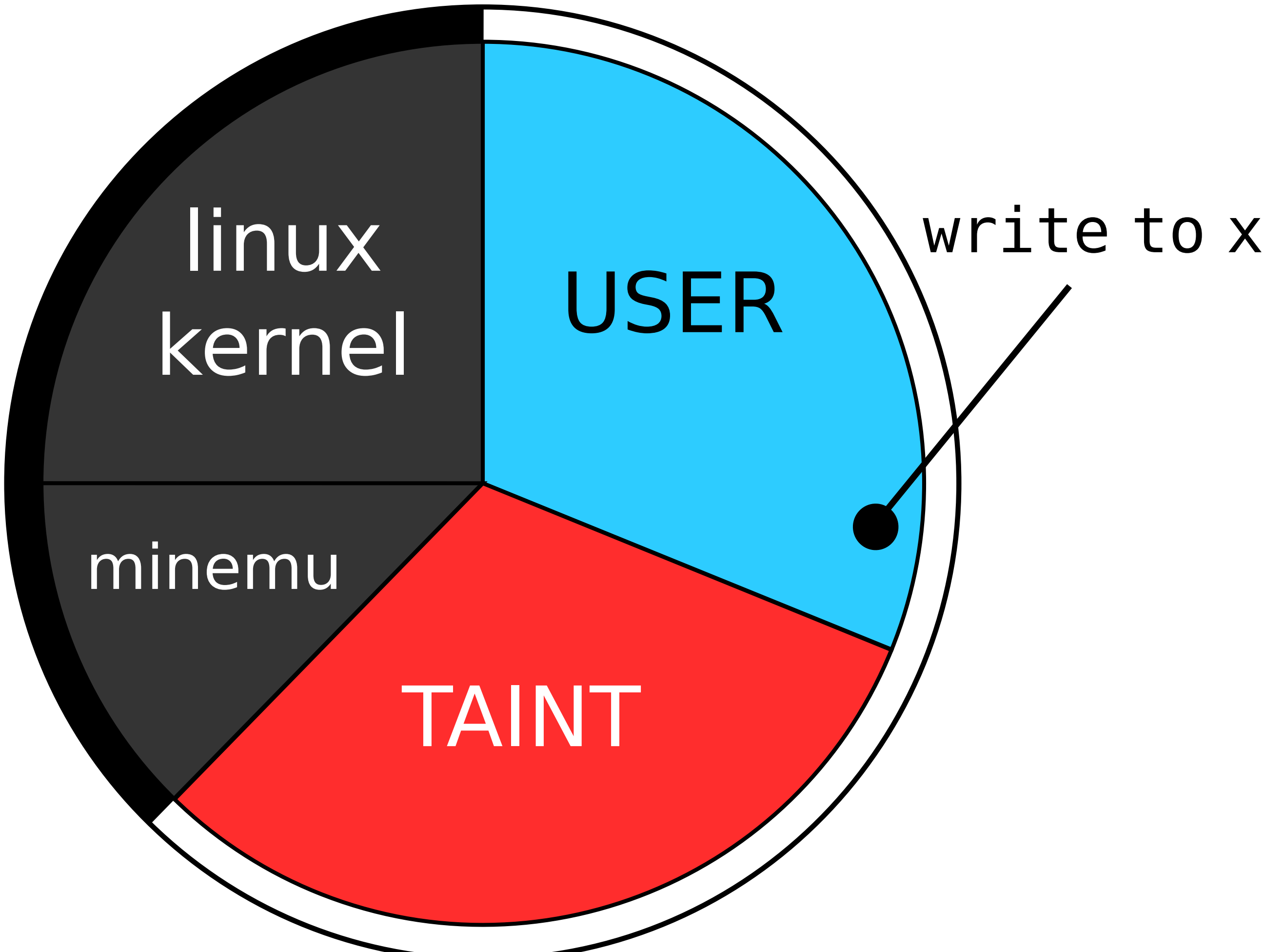
Memory layout (minemu)



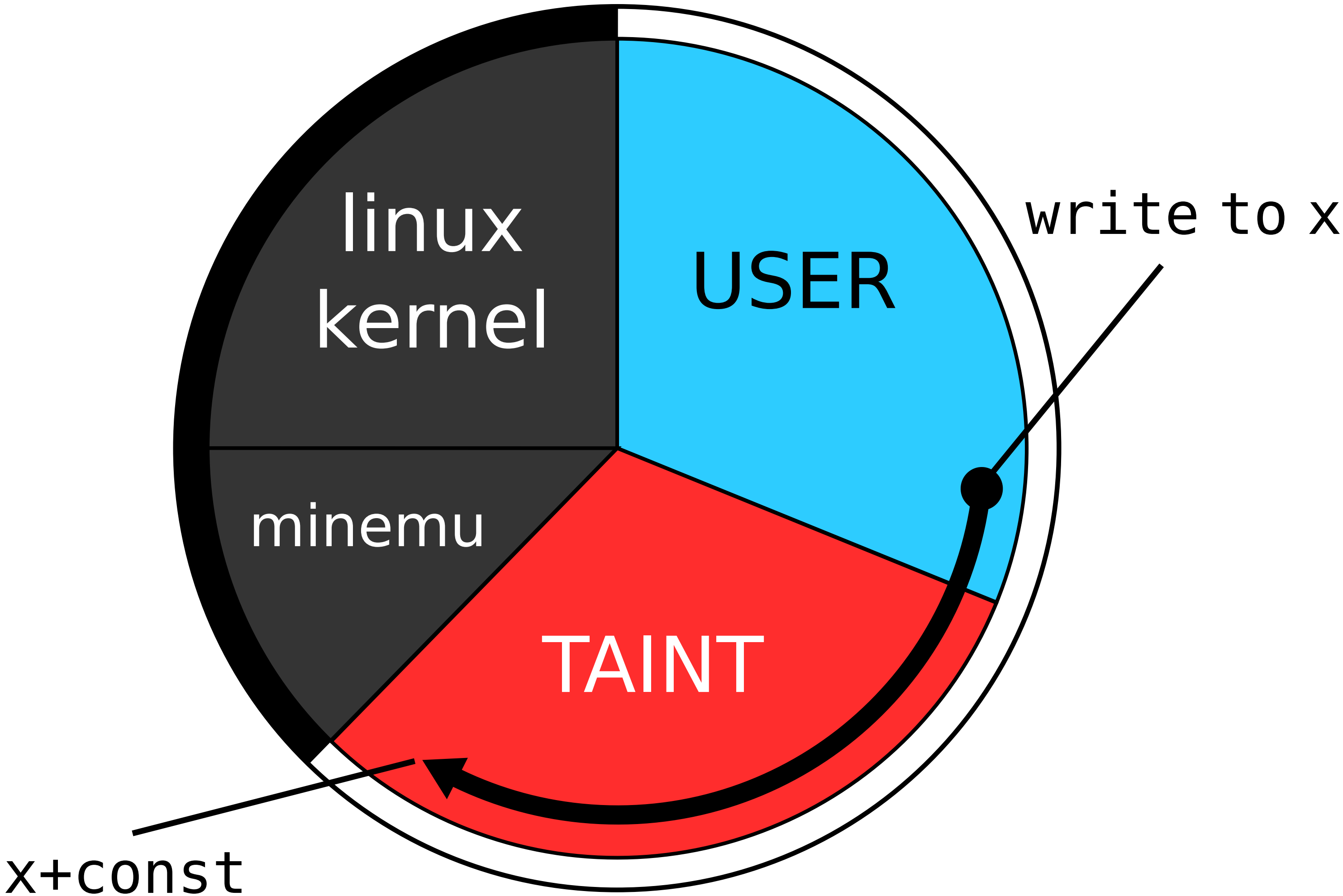
Memory layout (minemu)



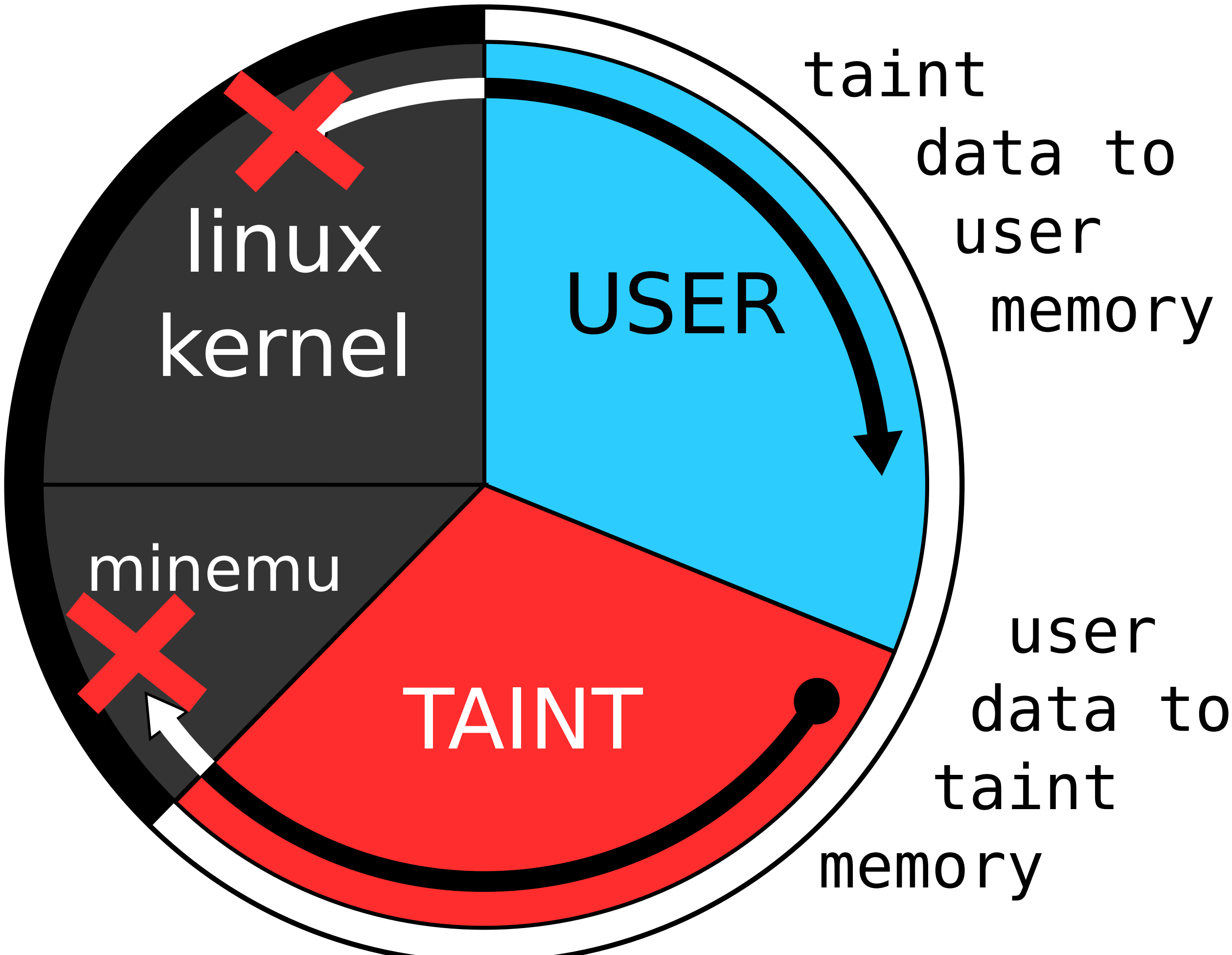
Memory layout (minemu)



Memory layout (minemu)



Memory layout (minemu)



Addressing shadow memory

```
mov EAX, (EDX)
```

Addressing shadow memory

```
mov EAX, (EDX)

address:

    EDX
```

Addressing shadow memory

```
mov EAX, (EDX)

address:

    EDX

taint:

    EDX+const
```

Addressing shadow memory

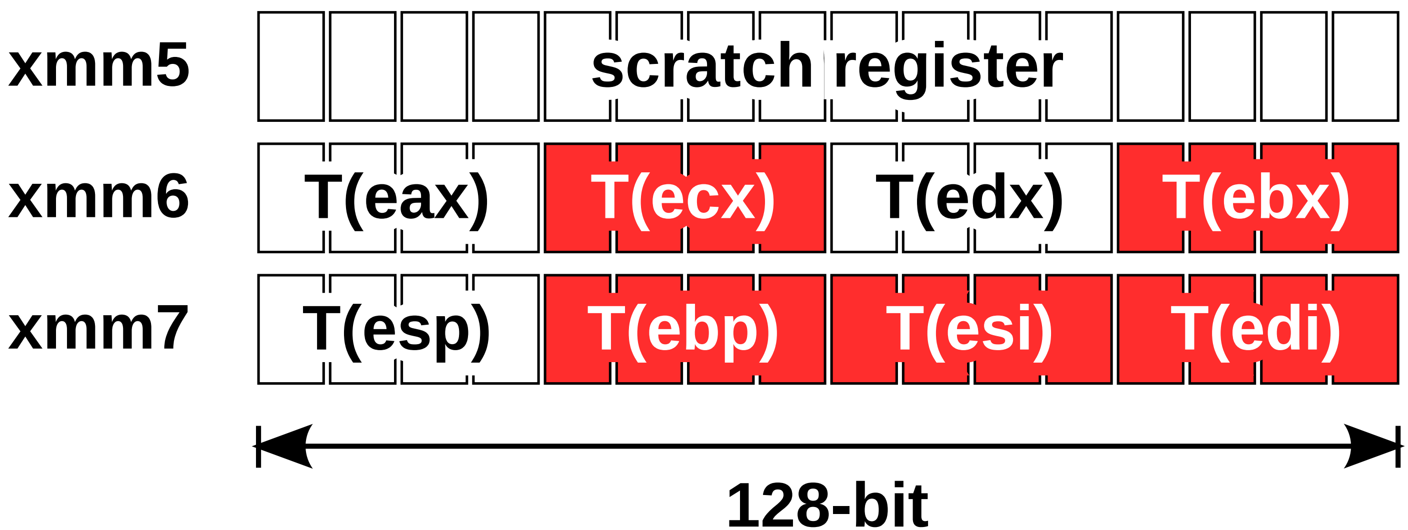
```
mov EAX, (EDX+EBX*4)

address:

    EDX+EBX*4

taint:

    EDX+EBX*4+const
```

Is this slowness fundamental?

minemu

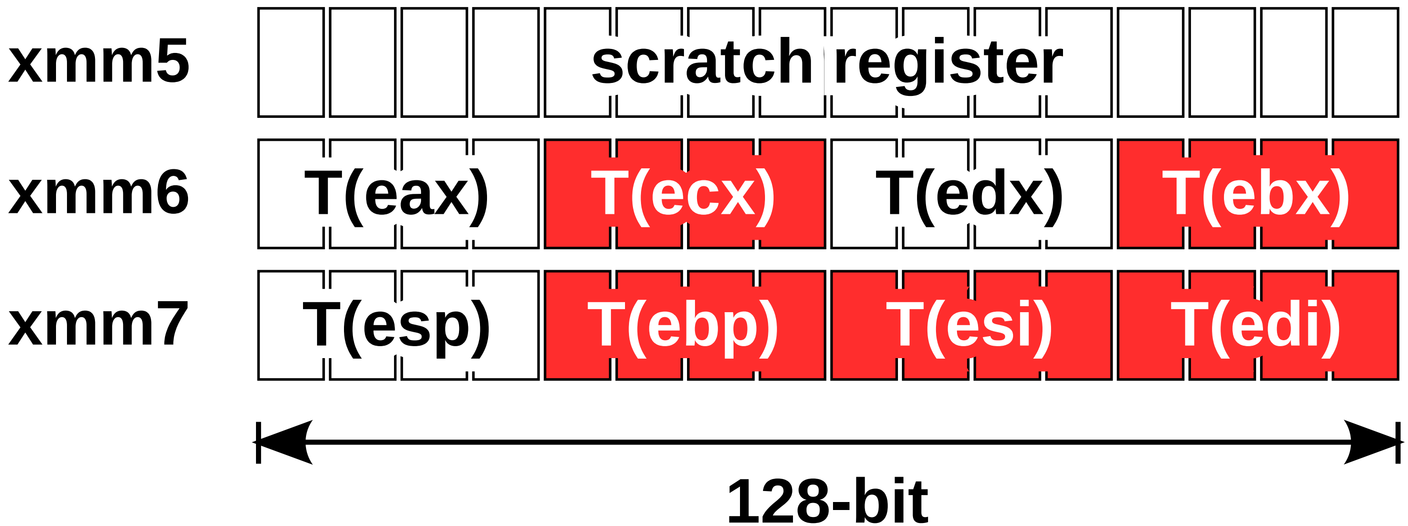
- memory layout
- use SSE registers to hold taint

Taint propagation in SSE registers



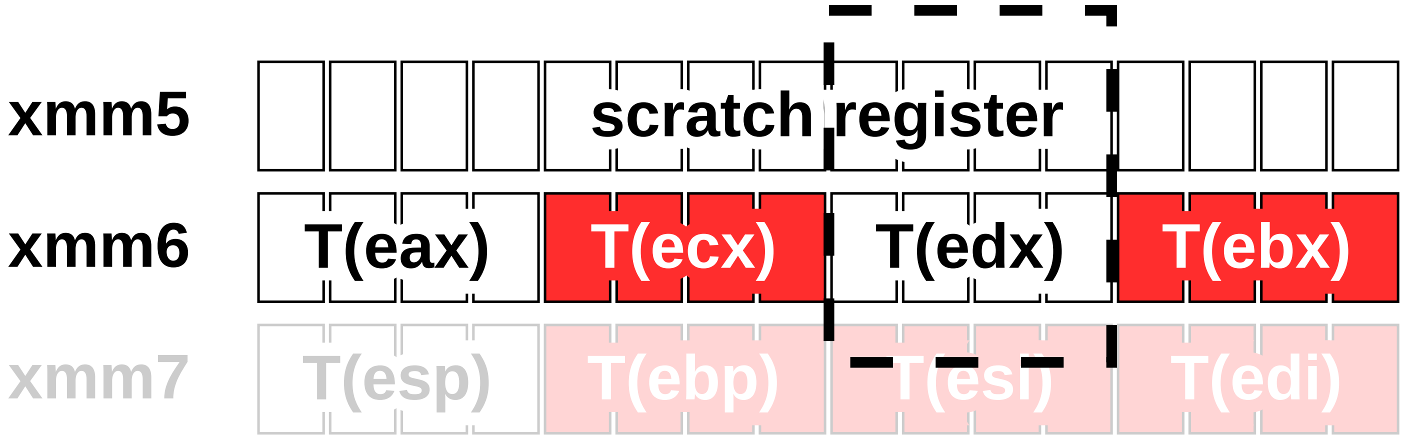
Taint propagation in SSE registers

```
add EDX, x
```



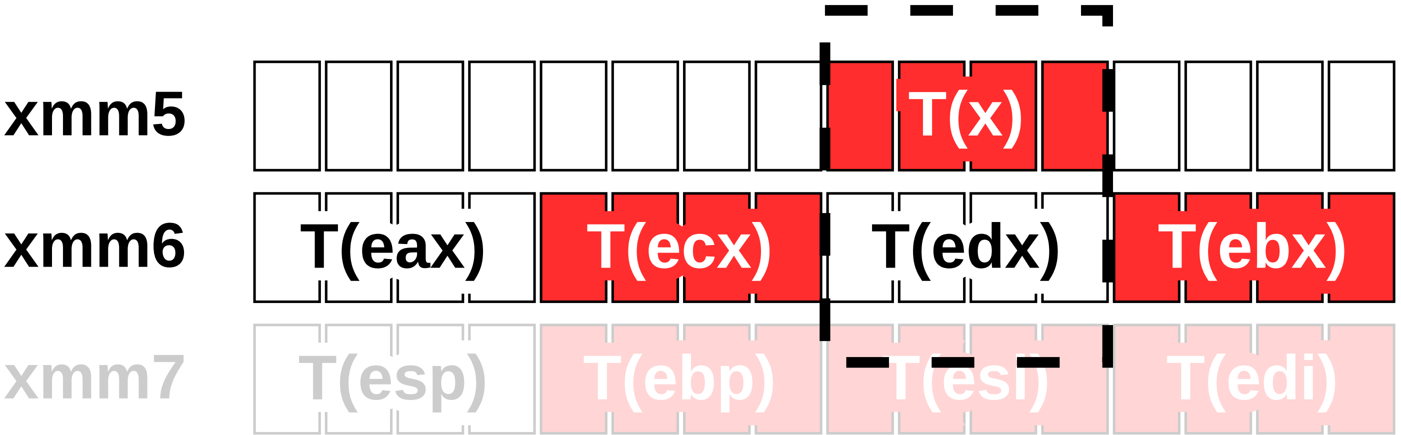
Taint propagation in SSE registers

`add EDX, x`



Taint propagation in SSE registers

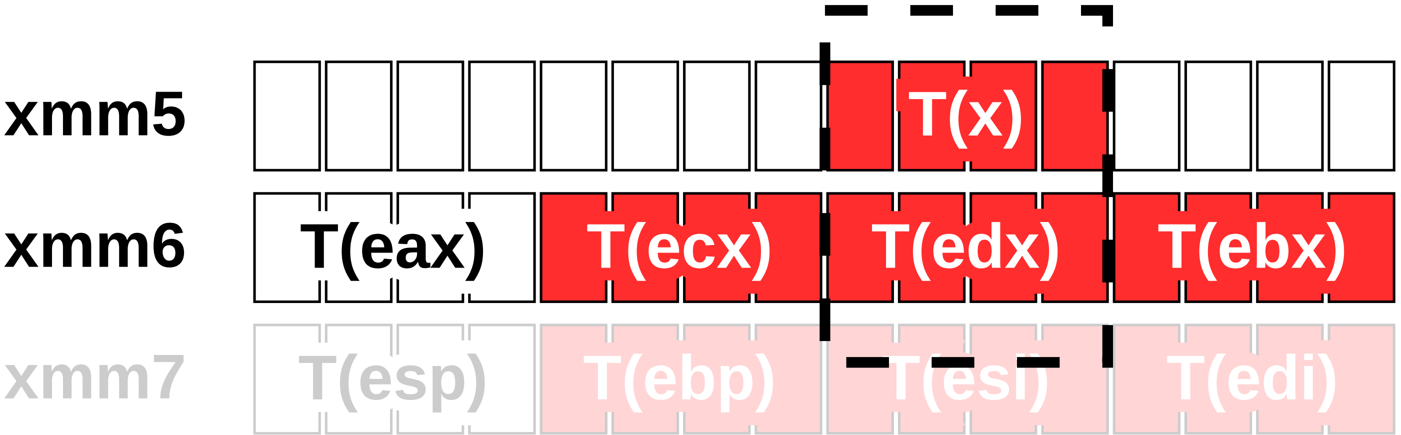
`add EDX, x`



`vector insert`

Taint propagation in SSE registers

`add EDX, x`

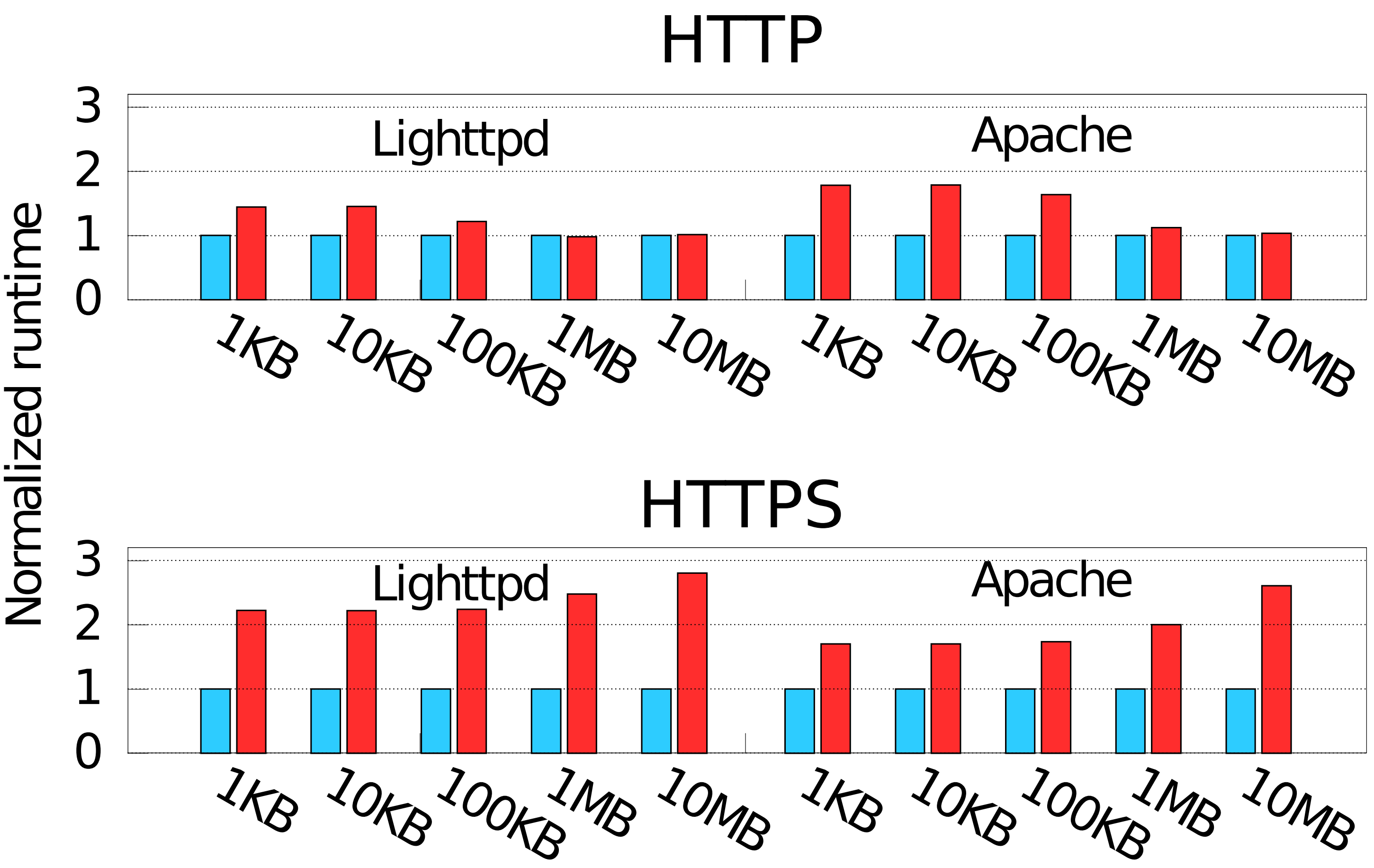


or

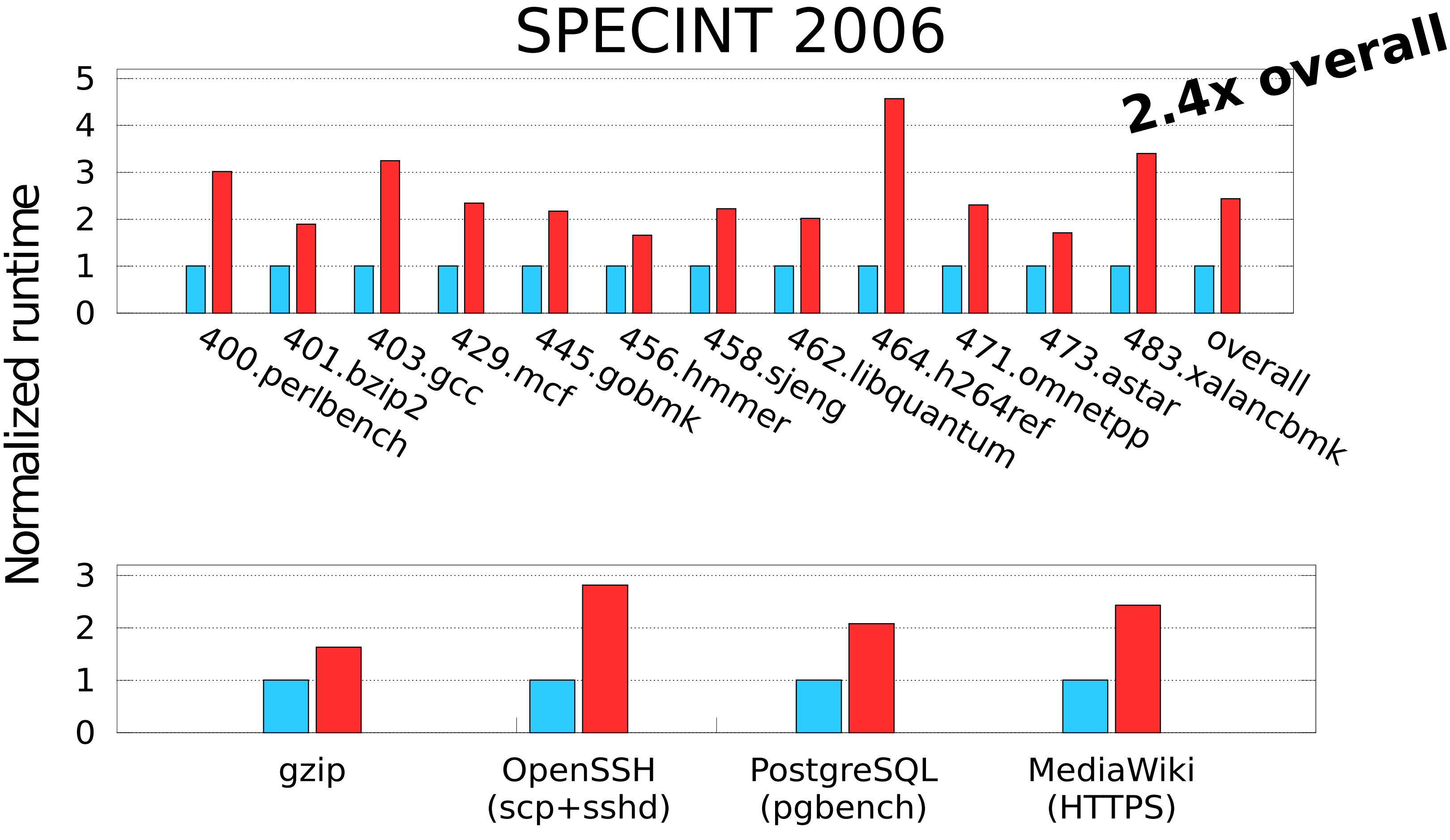
Effectiveness

Application	Type of vulnerability	Security advisory
Snort 2.4.0	Stack overflow	CVE-2005-3252
Cyrus imapd 2.3.2	Stack overflow	CVE-2006-2502
Samba 3.0.22	Heap overflow	CVE-2007-2446
Memcached 1.1.12	Heap overflow	CVE-2009-2415
Nginx 0.6.32	Buffer underrun	CVE-2009-2629
Proftpd 1.3.3a	Stack overflow	CVE-2010-4221
Samba 3.2.5	Heap overflow	CVE-2010-2063
Telnetd 1.6	Heap overflow	CVE-2011-4862
Ncompress 4.2.4	Stack overflow	CVE-2001-1413
Iwconfig V.26	Stack overflow	CVE-2003-0947
Aspell 0.50.5	Stack overflow	CVE-2004-0548
Htget 0.93	Stack overflow	CVE-2004-0852
Socat 1.4	Format string	CVE-2004-1484
Aeon 0.2a	Stack overflow	CVE-2005-1019
Exim 4.41	Stack overflow	EDB-ID#796
Htget 0.93	Stack overflow	
Tipxd 1.1.1	Format string	OSVDB-ID#12346

Performance



Performance



Limitations

Limitations

Doesn't prevent memory corruption, only acts when the untrusted data is used for arbitrary code execution.

Limitations

Tainted pointer dereferences

```
tainted_pointer->some_field = useful_untainted_value;
```

Limitations

Does not protect against non-control-flow exploits:

```
void try_system(char *username, char *cmd)
{
    int user_rights = get_credentials(username);
    char buf[16] = strcpy(buf, username);
    if (user_rights & ALLOW_SYSTEM)
        system(cmd);
    else
        log_error("user %s attempted login", buf);
}
```

in some cases we can add validation hooks.

_IO_vfprintf() in glibc can be hooked to check format strings for taint.

mysql_query() can be hooked to check for taint outside of literals in SQL queries.

Minemu

`git clone https://minemu.org/code/minemu.git`

any questions?

VRIJE UNIVERSITEIT AMSTERDAM