Minemu

Protecting buggy software from memory corruption attacks
Traditional Stack Smashing

buf[16]

GET / HTTP/1.1 00
baseretnarg1arg2

SHELLCODE!@#$%^&*()_&buf
Address Space Layout Randomisation

buf[16]
↓
GET / HTTP/1.1 00baseretnarg1arg2

SHELLCODE!@#$%^&*()_????
GET / HTTP/1.1

buf[16]

SHELLCODE!@#$%&^*()_&buf
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

buf[16]

GET / HTTP/1.1 0 0 base ret n arg1 arg2

sh;STACKSMASHER.....ROP1ROP2var1

pointer to useful code ← ←
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   d8 4b 06 08 a0 2e 05 08                      |                |
0805bea0   02 00 00 00 00 00 00 00                         |                |
0805beb0   00 00 00 00 00 00 00 00 ef be ad de a4 be 05 08 |..&.............|
0805bec0   a4 be 05 08 2f 62 69 6e 2f 73 68 00 a4 be 05 08 |..../bin/sh.....|
0805bed0   00 00 00 00 00 00 00 00 45 49 4e 44             |......EINDBAZENEIN|
0805bee0   45 49 4e 44 4e 44 45 49 4e 45 4e 45 4e 45 4e 45 |...DBAZENEINDBAZENE|
0805bef0   45 49 4e 44 45 49 4e 45 4e 45 4e 45 4e 45 4e 45 |...AZEN.......|
0805bff0   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

useful, but slow as hell
Is this slowness fundamental?

memory layout
use SSE registers to hold taint
Memory layout (linux)

- USER
- Linux kernel
Memory layout (minemu)

- **linux kernel**
- **USER**
- **minemu**
- **TAINT**
Memory layout (minemu)

- Linux kernel
- User
- Taint

Write to x
Memory layout (minemu)

- **linux kernel**
- **minemu**
- **USER**
- **TAINT**

write to x

x+const
Memory layout (minemu)

- **USER**
- **TAINT**
- Linux kernel
- minemu

Data flows from taint memory to user memory.
Addressing shadow memory

mov EAX, (EDX)
Addressing shadow memory

```assembly
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?

memory layout
use SSE registers to hold taint
Taint propagation in SSE registers

xmm5

xmm6

xmm7

scratch register

T(eax) T(ecx) T(edx) T(ebx)

T(esp) T(ebp) T(esi) T(edi)

128-bit
Taint propagation in SSE registers

add EDX, x

128-bit

tax registers

xmm5

xmm6

xmm7

scratch register

T(eax)

T(ecx)

T(edx)

T(ebx)

T(esp)

T(ebp)

T(esi)

T(edi)
Taint propagation in SSE registers

add EDX, x

xmm5

xmm6

xmm7

T(eax)  T(ecx)  T(edx)  T(ebx)
Taint propagation in SSE registers

\[
\text{add EDX, } x
\]

vector insert
Taint propagation in SSE registers

```plaintext
add EDX, x
```

or
## Effectiveness

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

SPECINT 2006

Normalized runtime

overall

2.4x overall

SPECINT 2006

2.4x overall
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:

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

any questions?