shellforge

Philippe Biondi

<biondi@cartel-securite.fr>

—

Libre Software Meeting
July 9-12, 2003

Outline

- What is shellforge ?
  - Needs
  - Presentation
  - Exemples
- Internals
  - C to asm
  - asm manipulations
  - asm to binary
  - loaders
- Future
  - More loaders
  - More platforms

- What is shellforge ?
  - Needs
  - Presentation
  - Exemples
- Internals
  - C to asm
  - asm manipulations
  - asm to binary
  - loaders
- Future
  - More loaders
  - More platforms

- Needs
  - need to design very specific shellcodes
  - need to do very complex operations
  - need to have your shellcode working now

- What is shellforge
  - shellforge is a set of C headers and a python program
  - shellforge is aimed at enabling you write your shellcode in C
  - shellforge is inspired from Stealth's Hellkit
  - shellforge only works on linux/x86 for the moment

- Hello world example

```
#include "include/sfsyscall.h"

int main(void)
{
        char buf[] = "Hello world!\n";
        write(1, buf, sizeof(buf));
        exit(0);
}
```

- Hello world example : one line output

```
$ ./shellforge.py hello.c
** Compiling hello.c
** Tuning original assembler code
** Assembling modified asm
** Retrieving machine code
** Shellcode forged!
\x55\x89\xe5\x83\xec\x24\x53\xe8\x00\x00\x00\x00\x5b\x83\xc3\xf4\x8b\x83\x67\x00
\x00\x00\x89\x45\xf0\x8b\x83\x6b\x00\x00\x00\x89\x45\xf4\x8b\x83\x6f\x00\x00\x00
\x89\x45\xf8\x0f\xb7\x83\x73\x00\x00\x00\x66\x89\x45\xfc\x8d\x4d\xf0\xba\x0e\x00
\x00\x00\xb8\x04\x00\x00\x00\xc7\x45\xec\x01\x00\x00\x00\x53\x8b\x59\xfc\xcd\x80
\x5b\xb8\x01\x00\x00\x00\xc7\x45\xec\x00\x00\x00\x00\x53\x8b\x59\xfc\xcd\x80\x5b
\x5b\xc9\xc3\x48\x65\x6c\x6c\x6f\x20\x77\x6f\x72\x6c\x64\x21\x0a\x00
```

- Hello world example : C output

```
$ ./shellforge.py -C hello.c
** Compiling hello.c
** Tuning original assembler code
** Assembling modified asm
** Retrieving machine code
** Shellcode forged!
unsigned char shellcode[] =
"\x55\x89\xe5\x83\xec\x24\x53\xe8\x00\x00\x00\x00\x5b\x83\xc3\xf4\x8b\x83\x67"
"\x00\x00\x00\x89\x45\xf0\x8b\x83\x6b\x00\x00\x00\x89\x45\xf4\x8b\x83\x6f\x00"
"\x00\x00\x89\x45\xf8\x0f\xb7\x83\x73\x00\x00\x00\x66\x89\x45\xfc\x8d\x4d\xf0"
"\xba\x0e\x00\x00\x00\xb8\x04\x00\x00\x00\xc7\x45\xec\x01\x00\x00\x00\x53\x8b"
"\x59\xfc\xcd\x80\x5b\xb8\x01\x00\x00\x00\xc7\x45\xec\x00\x00\x00\x00\x53\x8b"
"\x59\xfc\xcd\x80\x5b\x5b\xc9\xc3\x48\x65\x6c\x6c\x6f\x20\x77\x6f\x72\x6c\x64"
"\x21\x0a\x00";
int main(void)  ((void (*)())shellcode)();
```

- Hello world example : test shellcode

```
$ ./shellforge.py -tt hello.c
** Compiling hello.c
** Tuning original assembler code
** Assembling modified asm
** Retrieving machine code
** Shellcode forged!
** Compiling test program
** Running test program
Hello world!

** Test done! Returned status=0
```

- Hello world example : without zero bytes

```
$ ./shellforge.py -x hello.c
** Compiling hello.c
** Tuning original assembler code
** Assembling modified asm
** Retrieving machine code
** Computing xor encryption key
** Shellcode forged!
\xeb\x0d\x5e\x31\xc9\xb1\x75\x80\x36\x02\x46\xe2\xfa\xeb\x05\xe8\xee\xff\xff\xff
\x57\x8b\xe7\x81\xee\x26\x51\xea\x02\x02\x02\x02\x59\x81\xc1\xf6\x89\x81\x65\x02
\x02\x02\x8b\x47\xf2\x89\x81\x69\x02\x02\x02\x8b\x47\xf6\x89\x81\x6d\x02\x02\x02
\x8b\x47\xfa\x0d\xb5\x81\x71\x02\x02\x02\x64\x8b\x47\xfe\x8f\x4f\xf2\xb8\x0c\x02
\x02\x02\xba\x06\x02\x02\x02\xc5\x47\xee\x03\x02\x02\x02\x51\x89\x5b\xfe\xcf\x82
\x59\xba\x03\x02\x02\x02\xc5\x47\xee\x02\x02\x02\x02\x51\x89\x5b\xfe\xcf\x82\x59
\x59\xcb\xc1\x4a\x67\x6e\x6e\x6d\x22\x75\x6d\x70\x6e\x66\x23\x08\x02
```

- Hello world example : almost alphanumeric

```
$ ./shellforge.py --loader=alpha -R hello.c
** Compiling hello.c
** Tuning original assembler code
** Assembling modified asm
** Retrieving machine code
** Encoding with alphanumeric characters
** Shellcode forged!
hAAAAX5AAAAHPPPPPPPPafhA0fXf5A0fPDfhA0fXf5AFfPDfhA0fXf5ABPTX18XfPDfhA0fXf5A0fPDf
hA2fXf5A8fPDfhABfXf5AcfPDfhA0fXf5ATfPDfhA4fXf5AXfPDfhA0fXf5ABfPDfhA5fXf5AZfPDfhA
0fXf5AGfPDfhAAfXf5AafPDfhA5fXf5AZfPDfhA4fXf5AXfPDfhA4fXf5AXfPDfhA0fXf5AUfPDfhA0f
Xf5AxfPDfhADfXf5AxPTX18XfPDfhAAfXf5AwPTX18XfPDfhA0fXf5AkfPDfhA0fXf5AkfPDfhA0fXf5
AOPTX18XfPDfhAAfXf5AsPTX18XfPDfhA0fXf5A3PTX18XfPDfhA0fXf5AifPDfhA0fXf5ADPTX18XfP
DfhA0fXf5AcfPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhAAfXf5ARPTX1
8XfPDfhA0fXf5AufPDfhAAfXf5AyPTX18XfPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0
fXf5A1fPDfhA0fXf5AwPTX18XfPDfhA0fXf5AkfPDfhA0fXf5AOPTX18XfPDfhAAfXf5AsPTX18XfPDf
hA0fXf5A3PTX18XfPDfhA0fXf5AifPDfhA0fXf5ADPTX18XfPDfhA0fXf5AcfPDfhA0fXf5A0fPDfhA0
fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A1fPDfhAAfXf5ARPTX18XfPDfhA0fXf5AufPDfhAAfXf5AyPTX
18XfPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A4fPDfhA0fXf5AwPTX18XfPDfhA
0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA6fXf5A8fPDfhA0fXf5AuPTX18XfPDfhA6fXf5A9PT
X18XfPDfhA4fXf5AyfPDfhA0fXf5ABPTX18XfPDfhA0fXf5A3PTX18XfPDfhA0fXf5AufPDfhA0fXf5A
FPTX18XfPDfhA0fXf5AVfPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5ACfPDfhA0f
```

```
Xf5ALPTX18XfPDfhA0fXf5AxPTX18XfPDfhA6fXf5A9fPDfhA0fXf5A7PTX18XfPDfhA0fXf5AufPDfh
A0fXf5AFPTX18XfPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA5fXf5AZfPDfhA0fXf5ALP
TX18XfPDfhA0fXf5ADPTX18XfPDfhA2fXf5A9PTX18XfPDfhA0fXf5AufPDfhA0fXf5AFPTX18XfPDfh
A0fXf5A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhA1fXf5AZfPDfhA0fXf5ALPTX18XfPDfhA0fXf5ADP
TX18XfPDfhA6fXf5A9PTX18XfPDfhA0fXf5AufPDfhA0fXf5AFPTX18XfPDfhA0fXf5A0fPDfhA0fXf5
A0fPDfhA0fXf5A0fPDfhA0fXf5AWfPDfhA0fXf5ALPTX18XfPDfhA0fXf5ADPTX18XfPDfhA2fXf5A9P
TX18XfPDfhADfXf5AxPTX18XfPDfhA0fXf5ALPTX18XfPDfhA0fXf5AkfPDfhA0fXf5A0fPDfhA0fXf5
A0fPDfhA0fXf5A0fPDfhA0fXf5A0fPDfhAAfXf5AVPTX18XfPDfhA0fXf5AcfPDfhAAfXf5AefPDfhAA
fXf5ARPTX18XfPDfhA0fXf5ALPTX18XfPDfhABfXf5AXPTX18XfPDfhA0fXf5AFPTX18XfPDfhA0fXf5
AefPDÿä
```

To make a shellcode that scans ports :

```
#include "include/sfsyscall.h"
#include "include/sfsocket.h"
int main(void) {
        struct sockaddr_in sa;
        int s,i; int FIRST=1; int LAST=1024; unsigned int HOSTIP=0x0100007f;
        char buf[1024];

        sa.sin_family = PF_INET;
        sa.sin_addr.s_addr = HOSTIP;

        i=FIRST-1;
reopen: if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) write(1,"erreur\n",7);
        while(++i < LAST) {
                sa.sin_port = htons(i);
                if (connect(s, (struct sockaddr *)&sa, sizeof(struct sockaddr)) ==  0
                    write(1, &i, sizeof(i));
                    close(s);
                    goto reopen;
        }       }
        close(1); exit(0);
}
```

- Hello world example : almost alphanumeric

```
$ ./shellforge.py scanport.c
** Compiling scanport.c
** Tuning original assembler code
** Assembling modified asm
** Retrieving machine code
** Shellcode forged!
\x55\x89\xe5\x81\xec\x4c\x04\x00\x00\x57\x56\x53\xe8\x00\x00\x00\x00\x5b\x83\xc3
\xef\x66\xc7\x45\xf0\x02\x00\xc7\x45\xf4\x7f\x00\x00\x01\xc7\x85\xd8\xfb\xff\xff
\x00\x00\x00\x00\x8d\x8b\x10\x02\x00\x00\xb8\x04\x00\x00\x00\xc7\x85\xec\xfb\xff
\xff\x01\x00\x00\x00\x89\xc2\x53\x8b\x9d\xec\xfb\xff\xff\xcd\x80\x5b\x8d\x8d\xe0
\xfb\xff\xff\x89\x8d\xc4\xfb\xff\xff\x8d\xb4\x26\x00\x00\x00\x00\xc7\x85\xe0\xfb
\xff\xff\x02\x00\x00\x00\xc7\x85\xe4\xfb\xff\xff\x01\x00\x00\x00\xc7\x85\xe8\xfb
\xff\xff\x00\x00\x00\x00\xb8\x66\x00\x00\x00\xc7\x85\xdc\xfb\xff\xff\x01\x00\x00
\x00\x8b\x8d\xc4\xfb\xff\xff\x53\x8b\x59\xfc\xcd\x80\x5b\x89\xc6\x85\xf6\x7d\x27
\x8d\x8b\x15\x02\x00\x00\xba\x07\x00\x00\x00\xb8\x04\x00\x00\x00\xc7\x85\xdc\xfb
\xff\xff\x01\x00\x00\x00\x8b\xbd\xc4\xfb\xff\xff\x53\x8b\x5f\xfc\xcd\x80\x5b\x8b
\x85\xd8\xfb\xff\xff\x40\x89\x85\xd8\xfb\xff\xff\x3d\x88\x13\x00\x00\x0f\x8f\xcf
\x00\x00\x00\x8b\x95\xc4\xfb\xff\xff\x89\x95\xcc\xfb\xff\xff\x8d\x8d\xd8\xfb\xff
\xff\x89\x8d\xd4\xfb\xff\xff\x8d\xbd\xe0\xfb\xff\xff\x89\xbd\xd0\xfb\xff\xff\x66
\x0f\xb6\x95\xd8\xfb\xff\xff\xc1\xe2\x08\xc1\xf8\x08\xb4\x00\x09\xc2\x66\x89\x55
\xf2\x89\xb5\xe0\xfb\xff\xff\x8d\x45\xf0\x89\x85\xe4\xfb\xff\xff\xc7\x85\xe8\xfb
```

```
\xff\xff\x10\x00\x00\x00\xc7\x85\xdc\xfb\xff\xff\x03\x00\x00\x00\x8b\x8d\xcc\xfb
\xff\xff\xb8\x66\x00\x00\x00\x53\x8b\x59\xfc\xcd\x80\x5b\x85\xc0\x7c\x44\xc7\x85
\xdc\xfb\xff\xff\x01\x00\x00\x00\xb8\x04\x00\x00\x00\x8b\x8d\xd4\xfb\xff\xff\xba
\x04\x00\x00\x00\x8b\xbd\xcc\xfb\xff\xff\x53\x8b\x5f\xfc\xcd\x80\x5b\x89\xb5\xdc
\xfb\xff\xff\xb8\x06\x00\x00\x00\x8b\x95\xd0\xfb\xff\xff\x53\x8b\x5a\xfc\xcd\x80
\x5b\xe9\xca\xfe\xff\xff\x8b\x85\xd8\xfb\xff\xff\x40\x89\x85\xd8\xfb\xff\xff\x3d
\x88\x13\x00\x00\x0f\x8e\x55\xff\xff\xff\x8d\x8b\x1d\x02\x00\x00\xba\x05\x00\x00
\x00\xb8\x04\x00\x00\x00\xc7\x85\xdc\xfb\xff\xff\x01\x00\x00\x00\x8b\xb5\xc4\xfb
\xff\xff\x53\x8b\x5e\xfc\xcd\x80\x5b\x8d\x8d\xe0\xfb\xff\xff\xba\x06\x00\x00\x00
\xc7\x85\xdc\xfb\xff\xff\x01\x00\x00\x00\x89\xd0\x53\x8b\x59\xfc\xcd\x80\x5b\xba
\x01\x00\x00\x00\xc7\x85\xdc\xfb\xff\xff\x00\x00\x00\x00\x89\xd0\x53\x8b\x59\xfc
\xcd\x80\x5b\x5b\x5e\x5f\xc9\xc3\x67\x6f\x5b\x0a\x00\x65\x72\x72\x65\x75\x72\x0a
\x00\x5d\x6f\x67\x0a\x00\x8d\xb4\x26\x00\x00\x00\x00\x8d\xbc\x27\x00\x00\x00\x00
```

- What is shellforge ?
  - Needs
  - Presentation
  - Exemples
- Internals
  - C to asm
  - asm manipulations
  - asm to binary
  - loaders
- Future
  - More loaders
  - More platforms

- We must only use system calls. No libc (`printf()`,...)
- Usually, system calls have wrapper functions in libc that do the system call
- We redefine every syscall name to an inline function. No library is used

```
static inline _sfsyscall0( pid_t, fork )
static inline _sfsyscall1( int, close, int, fd )
static inline _sfsyscall1( time_t, time, time_t *, t )
static inline _sfsyscall1( int, nice, int, inc )
static inline _sfsyscall0( uid_t, geteuid )
```

- Almost every system calls have their calling function defined (thanks to a man page parser!)

- Macro used for syscall function definition :

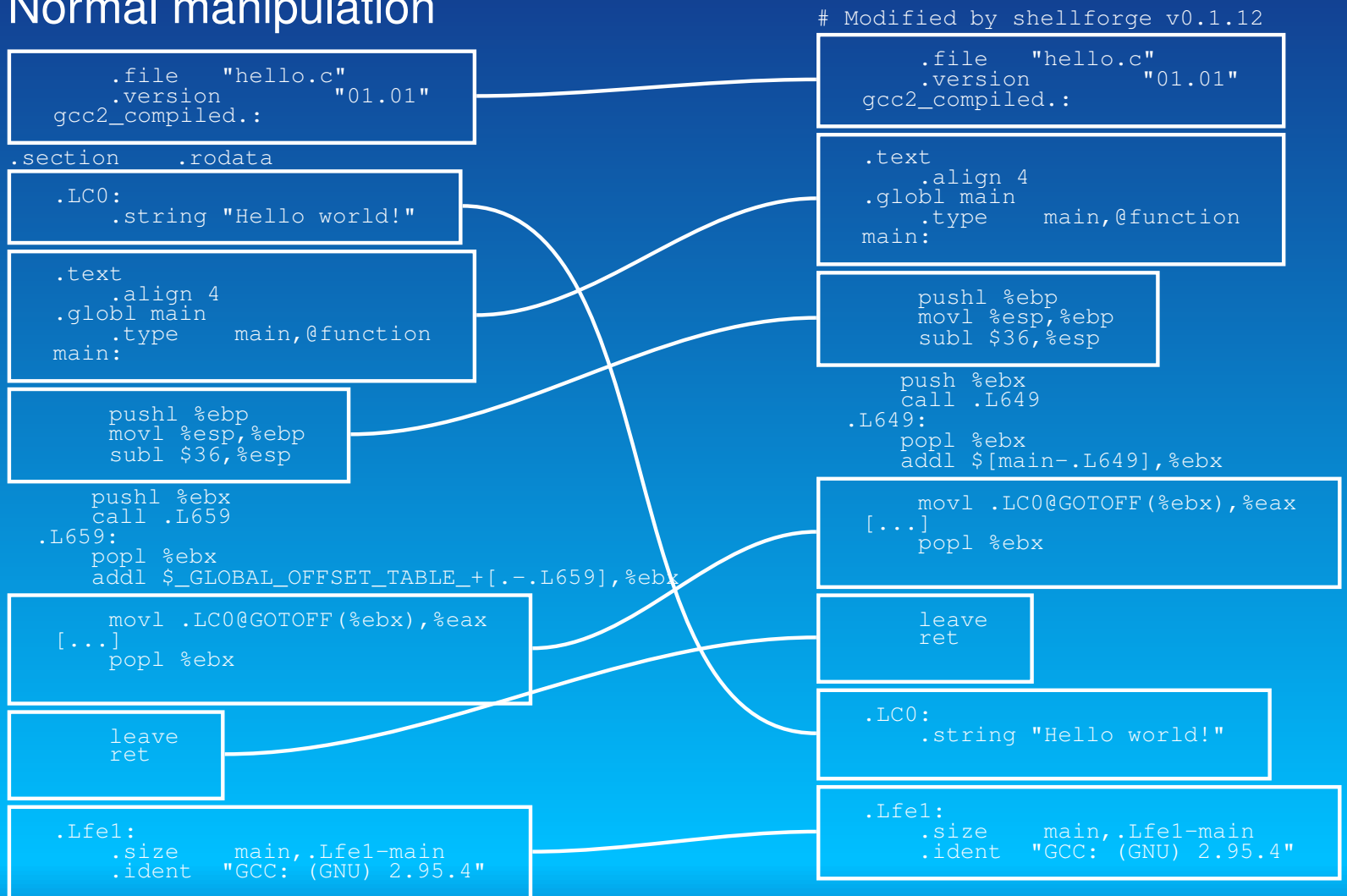
```
#define _sfsyscall1(type,name,type1,arg1) \
type name(type1 arg1) \
{ \
long __res; \
__asm__ volatile ("pushl %%ebx\n\t"   \
                  "mov %2,%%ebx\n\t" \
                  "int $0x80\n\t"     \
                  "popl %%ebx"        \
        : "=a" (__res) \
        : "0" (__NR_##name),"m" ((long)(arg1)) : "ebx"); \
__sfsyscall_return(type,__res); \
}
```

- Compilation
  - gcc is used to compile the C source
  - generate position independant code ($\Longrightarrow$ use `%ebx` register)
  - force function inlining to have the shellcode in one block
  - do not use built-in functions (`exit()`, `memcpy()`, ...)
  - output assembly code, so that we can manipulate it

```
gcc -O3 -S -fPIC -Winline -finline-functions -ffreestanding
```

- Assembler manipulations. Assembly output is parsed with an automaton
  - `.rodata` section is deleted. Its content is moved at the end of `.text`
  - if stack must be relocated (`%esp` may point into the shellcode) some code is added (option `-s`)
  - if all registers must be restored, code is also added (option `-S`)

Normal manipulation

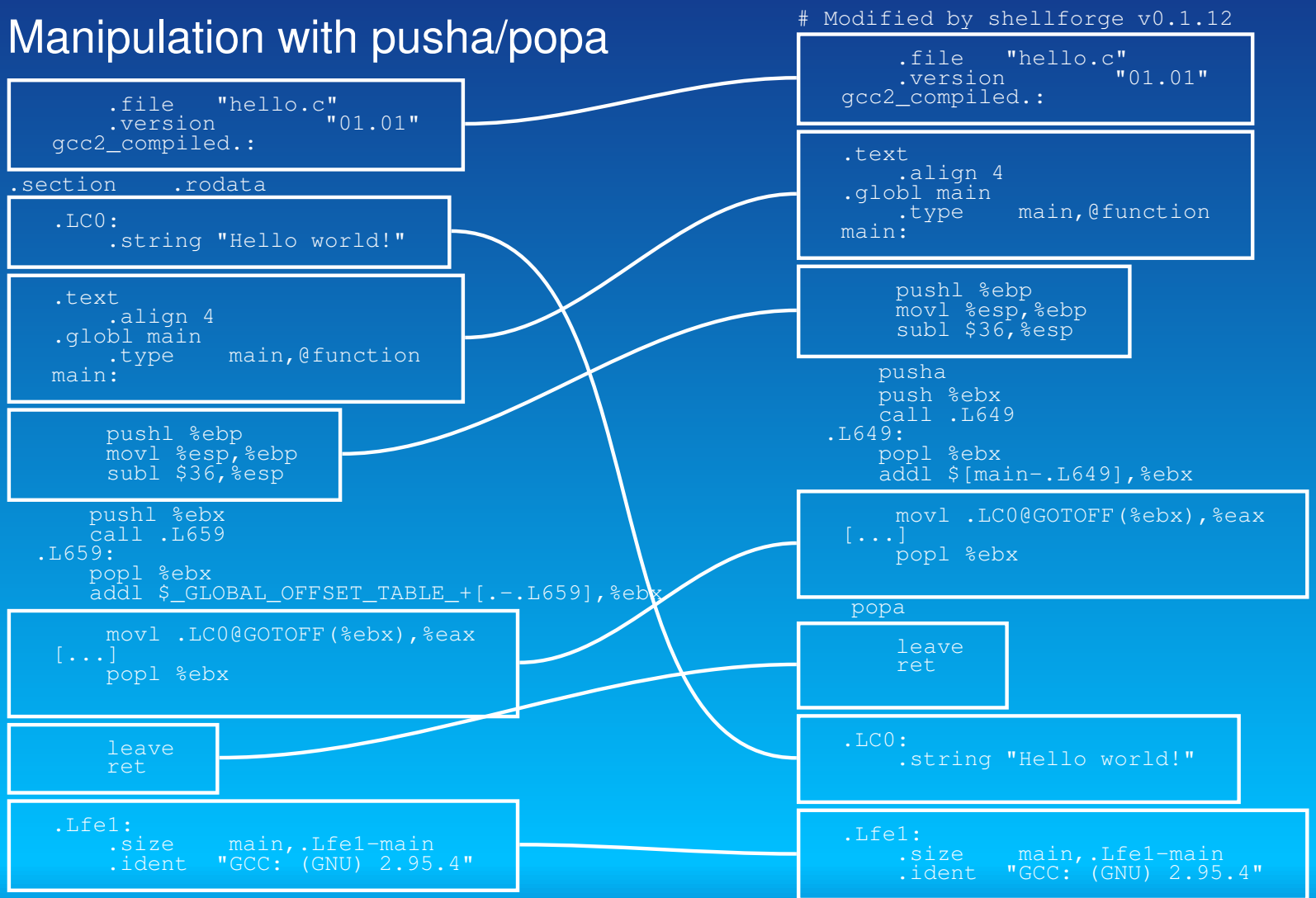


Manipulation with stack relocation

```
        .file   "hello.c"
        .version        "01.01"
gcc2_compiled.:
.section        .rodata
.LC0:
        .string "Hello world!"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %ebp
        movl %esp,%ebp
        subl $36,%esp
        pushl %ebx
        call .L659
.L659:
        popl %ebx
        addl $_GLOBAL_OFFSET_TABLE_+[.-.L659],%ebx
        movl .LC0@GOTOFF(%ebx),%eax
[...]
        popl %ebx
        leave
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.95.4"
```

```
# Modified by shellforge v0.1.12
        .file   "hello.c"
        .version        "01.01"
gcc2_compiled.:
.text
        .align 4
.globl main
        .type    main,@function
main:
        popl %eax
        call .L649
.L649:
        popl %ebx
        pushl %eax
        addl $[main-.L649],%ebx
        movl %ebx, %eax
        xorl %esp, %eax
        shrl $16, %eax
        test %eax, %eax
        jnz .Lnotinstack
        movl %ebx,%esp
.Lnotinstack:
        pushl %ebp
        movl %esp,%ebp
        subl $36,%esp
        movl .LC0@GOTOFF(%ebx),%eax
[...]
        popl %ebx
        leave
        ret
.LC0:
        .string "Hello world!"
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.95.4"
```

Manipulation with pusha/popa



- ASM to binary
  - assembly source is then assembled into an executable
  - code is extracted with objdump

- Use of loaders
  - no loader
    - do nothing
  - `xor` loader
    - xor the binary to avoid "`\x00`" byte
    - prepend the loader that will decode the binary
  - `alpha` loader
    - make a shellcode that will reconstruct the original one on the stack
    - the new shellcode will be alphanumeric (only A→Z, a→z, 0→9), except the 2 last bytes :(
  - other loaders may be implemented

- What is shellforge ?
  - Needs
  - Presentation
  - Exemples
- Internals
  - C to asm
  - asm manipulations
  - asm to binary
  - loaders
- Future
  - More loaders
  - More platforms

- Loaders
  - True alphanumeric only shellcode
  - Loaders like ADM mutate

- Other architectures
  - cross compilation
- Other OS
  - new includes

That's all folks. Thanks for your attention.

You can reach me at **<phil@secdev.org>**

These slides and shellforge are available at
http://www.cartel-securite.fr/pbiondi/