Access Control and Privacy Policies (2)

Email:	christian.urban at kcl.ac.uk
Office:	S1.27 (1st floor Strand Building)
Slides:	KEATS (also home work is there)

Homework

. . . I have a question about the homework.

Is it required to submit the homework before the next lecture?

Thank you!
Anonymous

SmartWater

THIEVES
BEWARE
SmartWater Trap Devices In Use
SMARTWATER
Cable & metal is traceable
Thieves can be tracked
Spot checks at scrap yards
Police and Recyclers equipped to check for SmartWater
100% conviction rate
www.smartwater.com
© SmartWater Technology Ltd 2010

- seems helpful for preventing cable theft
- wouldn't be helpful to make your property safe, because of possible abuse
- security is always a tradeoff

Plaintext Passwords from IEEE

On 25 September 2012, a report on a data breach at IEEE:

- IEEE is a standards organisation (not for profit)
- many standards in CS are by IEEE

- 100k plain-text passwords were recorded in logs
- the logs were openly accessible on their FTP server

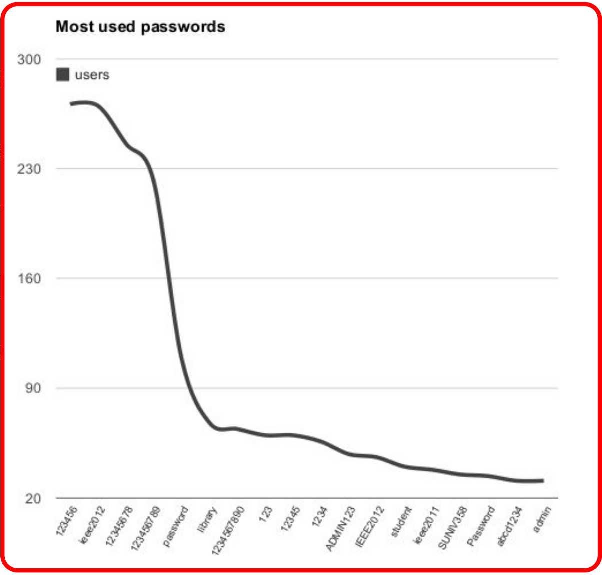
`http://ieeelog.com`

Plaintext Passwords from IEEE

On 25 S IEEE:

- IEEE is
- many s
- 100k pl
- the log

og.com



Virgin Mobile (USA)

http://arstechnica.com/security/2012/09/virgin-mobile-password-crack-risk/

- for online accounts passwords must be 6 digits
- you must cycle through 1M combinations (online)

Virgin Mobile (USA)

http://arstechnica.com/security/2012/09/
virgin-mobile-password-crack-risk/

- for online accounts passwords must be 6 digits
- you must cycle through 1M combinations (online)

- he limited the attack on his own account to 1 guess per second, **and**
- wrote a script that cleared the cookies set after each guess

Smash the Stack for Fun…

- "smashing the stack attacks" or "buffer overflow attacks"
- one of the most popular attacks
  ($>$ 50% of security incidents reported at CERT are related to buffer overflows)
- made popular in an article by Elias Levy
  (also known as Aleph One):

  "Smashing The Stack For Fun and Profit"

  `http://www.phrack.org`, Issue 49, Article 14

The Problem

- The basic problem is that library routines look as follows:

```
void strcpy(char *src, char *dst) {
  int i = 0;
  while (src[i] != "\0") {
    dst[i] = src[i];
    i = i + 1;
  }
}
```

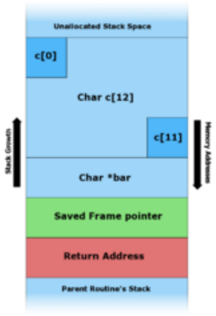
- the resulting problems are often remotely exploitable
- can be used to circumvents all access control

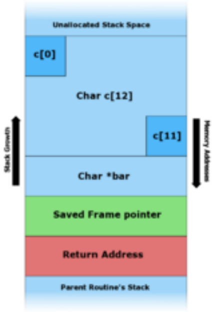
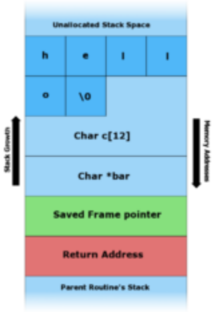
my_float is printed twice:

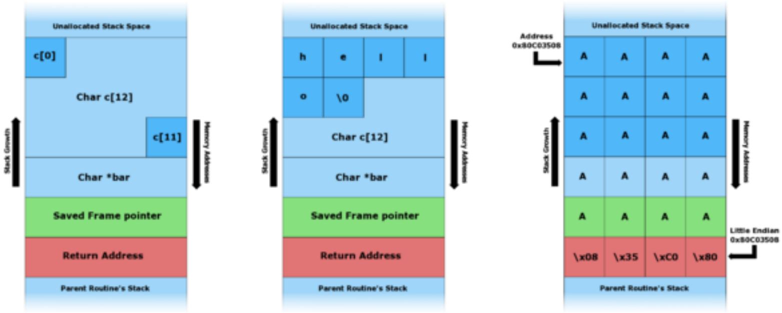
```
void foo (char *bar)
{
  float my_float = 10.5;     // in hex: \x41\x28\x00\x00
  char  buffer[28];

  printf("my float value = %f\n", my_float);
  strcpy(buffer, bar);
  printf("my float value = %f\n", my_float);
}

int main (int argc, char **argv)
{
  foo("my string is too long !!!!! ");
  return 0;
}
```


Unallocated Stack Space
c[0]
Char c[12]
c[11]
Char *bar
Saved Frame pointer
Return Address
Parent Routine's Stack
Stack Growth
Memory Addresses


Unallocated Stack Space
c[0]
Char c[12]
c[11]
Char *bar
Saved Frame pointer
Return Address
Parent Routine's Stack
Stack Growth
Memory Addresses

Unallocated Stack Space
h
e
l
l
o
\0
Char c[12]
Char *bar
Saved Frame pointer
Return Address
Parent Routine's Stack
Stack Growth
Memory Addresses


Unallocated Stack Space
c[0]
Char c[12]
c[11]
Char *bar
Saved Frame pointer
Return Address
Parent Routine's Stack
Stack Growth
Memory Addresses
h
e
l
l
o
\0
Address 0x80C03508
A
Little Endian 0x80C03508
\x08
\x35
\xC0
\x80

```
int match(char *s1, char *s2) {
  while( *s1 != '\0' && *s2 != '\0' && *s1 == *s2 ){
    s1++; s2++;
  }
  return( *s1 - *s2 );
}

void welcome() { printf("Welcome to the Machine!\n"); exit(0); }
void goodbye() { printf("Invalid identity, exiting!\n"); exit(1); }

main(){
  char name[8];
  char pw[8];

  printf("login: ");
  get_line(name);
  printf("password: ");
  get_line(pw);

  if(match(name, pw) == 0)
    welcome();
  else
    goodbye();
}
```

A programmer might be careful, but still introducing vulnerabilities:

```
// Since gets() is insecure and produces lots of warnings,
// I use my own input function instead.
char ch;
int i;

void get_line(char *dst) {
  char buffer[8];
  i = 0;
  while ((ch = getchar()) != '\n') {
    buffer[i++] = ch;
  }
  buffer[i] = '\0';
  strcpy(dst, buffer);
}
```

Payloads

- the idea is you store some code as part to the buffer
- you then override the return address to execute this payload
- normally you start a root-shell

Payloads

- the idea is you store some code as part to the buffer
- you then override the return address to execute this payload
- normally you start a root-shell
- difficulty is to guess the place where to “jump”

Payloads (2)

- another difficulty is that the code is not allowed to contain `\x00`:

```
xorl %eax, %eax
```

```
void strcpy(char *src, char *dst) {
  int i = 0;
  while (src[i] != "\0") {
    dst[i] = src[i];
    i = i + 1;
  }
}
```

Format String Vulnerability

string is nowhere used:

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

main(int argc, char **argv)
{
  char *string = "This is a secret string\n";

  printf(argv[1]);
}
```

this vulnerability can be used to read out the stack

Protections against BO Attacks

- use safe library functions
- ensure stack data is not executable (can be defeated)
- address space randomisation (makes one-size-fits-all more difficult)
- choice of programming language (one of the selling points of Java)