Introduction to Information Security

Vulnerabilities
Vulnerabilities

- DoS
- ID
- RCE
- PE
- ???
int check(char* password)
{
    int invalid = 1;
    char buff[10];

    if (strlen(password) > 10)
        return -1;

    strcpy(buff, password);
    ...
}

Reorder!
int receive(int sock) {
    int size;
    char buff[100];

    if (recv(sock, &size, 1, 0) < 0)
        return -1;

    if (recv(sock, buff, size, 0) < 0)
        return -1;

    ...
}
SEH Overflow

```c
int receive(int sock)
{
    int size;
    char buff[100];
    if (recv(sock, &size, 1, 0) < 0)
        return -1;
    if (recv(sock, buff, size, 0) < 0)
        return -1;
    ...
}
```

Also, brute force

Data Execution Prevention!
int receive(int sock)
{
    int size;
    char buff[100];

    if (recv(sock, &size, 1, 0) < 0)
        return -1;

    if (recv(sock, buff, size, 0) < 0)
        return -1;

    ...
}
Tab* openTab(char* URL)
{
    Tab tab();
    tab.load(URL);
    return &tab;
}

Tab* tab = openTab("www.google.com");
tab->body->show();

var shellcode = "...";
var chunks = new Array();
for (i = 0; i < 1000; i++)
    chunks[i] = shellcode + ";";
Double Fetch

```c
int f(char* input, int size) {
    char* buff[1000];
    if (size > 1000) {
        return -1;
    }
    memcpy(buff, input, size);
    ...}
```

```c
int* EBP = (int*) 0xbffff100;
while (1) {
    *(EBP+12) = 0;
    *(EBP+12) = 1000;
}
```
def main(path):
    verify_signature(path)
    execute_file(path)

def verify_signature(path):
    data = open(path, 'rb').read()
    return sha1(data) == '...' 

def execute_file(path):
    data = open(path, 'rb').read()
    exec data

main('file.py')
def dirlist(path):
    cmd = 'ls %s' % path
    out = subprocess.check_output(cmd)
    return out.splitlines()

>>> dirlist('/tmp')
['dir/', 'file.txt']

>>> dirlist('/tmp; sh')
$
def getfile(filename):
    path = '/tmp/' + filename
    return open(path, 'rb').read()

>>> getfile('file.txt')
'Hello, world!
'

>>> getfile('.. '/../etc/passwd')
'root:'
Putting it all Together

• /tmp/server.py  – receives a name and a text and logs it in /tmp/messages/<name>.txt
• /tmp/archive.py  – compresses /tmp/messages/* into /tmp/archives/<date>.gz once a day
• This runs under user
  • Which is a sudoer, but we don't know his password
Putting it all Together

• Send a message with the name ../archive.py and some malicious code

• This code will run later that day, and:
  • Create the /tmp/evil/ directory
  • Put a fake sudo in it, which saves the provided password and calls the real sudo with it
  • Edit the user's ~/.bashrc:
    • PATH="/tmp/evil:$PATH"

• The next time the users logs in, his PATH will be compromised
• The next times he runs sudo apt-get install ..., we'll get his password
Addendum
Vulnerabilities IRL
The hash is never completed, and the SSL/TLS encryption is compromised.
Stuxnet CPL/LNK

- Windows supports shortcuts (.LNK files)
- An .LNK file references:
  - A path
  - An icon
    - Which can come from a .CPL file
    - Which can bundle resources
    - ... But is similar to a .DLL file
      - Which can have initialization routines
WinRAR File Extension Spoofing

- The GUI developer needed a place for extra settings
- The ZIP developer provided the "extra" field
- The GUI developer put the displayed file name there
- But there was already a field for that...
Master Key

• The Java developer validated the APK
  • He loaded its files to a LinkedHashTable and checked their signatures

• The C developer installed the APK
  • He loaded its file to a LinearHashTable and wrote their data to disk

• An APK is a ZIP file
  • Which can have multiple entries with the same name...

```python
files = LinkedHashTable()
for file in APK:
    files[file.name] = file.data

for name in files:
    validate(files[file])
```

```python
files = LinearHashTable()
for file in APK:
    files[file.name] = file.data

for name in files:
    write(files[name])
```