Project 1 :: Defeating SkyNet
(Security Essentials)

Dean Pisani 311210775
Kristy Hughes 310186293

1 Low Level Exploits

1.1 Savegames

1. A buffer overflow can be executed by typing a name longer than 10 bytes.
   E.g.

```
klaptop-2:common_vulns k$ ./a.out
putsomethingreallyreallylonghere
Name: putsomethingreallyreallylonghere
HP: 30 | Gold: 1818322290
```

2. For C code in general, the advice to avoid buffer overflows is:

   - Making sure that memory auditing is done properly with valgrind
   - using fgets instead of gets
   - use strncmp() instead of strcmp() or strncpy() instead of strcpy() etc

   In the case of savegame.c, the line:

   ```
   scanf("%s", h.name);
   ```

   Should be replaced with

   ```
   scanf("%10s", h.name);
   ```

   So that the program knows how big the input string should be and ignores anything longer than what is expected.

3. Buffer overflows can be used to inject malicious instructions inside of a running program. This can be done in various ways, but a typical case is where an attacker corrupts the stack by overwriting the return address of the current stack frame. The program will then jump to a section of user-supplied instructions, also given as input when overflowing the buffer. With knowledge of the OS architecture that the program runs on an attacker can invoke system commands, spawn a shell or run malicious code from within the corrupted program. Once an attacker recognises the potential of a buffer overflow attack, they can use methods such as the NOP-sled technique to find stack offsets and return addresses. This technique finds the exact address of the buffer by incrementally adding no-operation op-codes onto the overflowed buffer and widening the target area. An executing program will 'slide' down these operations until it hits a relative jump instruction placed at the end which will move the execution to the section of malicious code.

1.2 iCubeKinect

1. An asymmetric cipher is used for signing DVD data so that users (and attackers) do not have access to the key used to create signatures. The public key can be used for verification, but if someone was to modify the DVD then it is infeasible to generate a matching signature without the corresponding private key, which is held only by the DVD manufacturer. Thus if an attempt is made to change the DVD it will no longer work when loaded.
2. The function used to compare cert_hash and sig_hash is strncmp, which returns 0 if the two strings are equal or if their first MD5_LENGTH characters are equal. If one of the hash strings contains the NULL byte then strncmp will interpret this as the end of the string and return the result of the comparison so far. So instead of brute force looking for hash collisions using the entire hash, one can brute force only part of the hash and have a null byte early.
3. This error can be fixed by first checking the length of cert_hash and sig_hash to check that they are not shorter than MD5_LENGTH. If one of them is shorter than MD5_length then something is wrong and RSA_CERT_BAD should be returned. This will work because the length function works by linearly searching though the string to find the first instance of the null byte

1.3 General Questions

1. The GCC complier implements the "Stack Guard" security mechanism to prevent buffer overflows. The stack protector in GCC is off by default, so implementing a buffer overflow will return "stack smashing detected" from gcc's stack protector. In our case we would like to disable this so that we can demonstrate a buffer overflow.
   Canaries are known values put on the stack between a buffer and control data to monitor buffer overflows. This means that the canary will be the first data to be corrupted when a buffer overflow is executed so a program just has to verify if the canary data is valid to check if an overflow has occurred. Once the overflow is detected it can be handled accordingly, by invalidating the corrupted data etc.

2. Java automatically checks the bounds on arrays which prevents any buffer overflow from occurring. This doesn't mean that savegame could not be exploited, however, as Java has it's own problems. But it does rule out the possibility of a buffer overflow.

3. As mentioned earlier, it is possible to execute pre-written scripts using a buffer overflow attack. This pre-written code could simply start a command shell which the attacker can control. This kind of script is called shellcode. This command shell is opened with the privileges of the program that called it, so the attacker has access to a shell with escalated privileges.

2 SQL Exploits

1. The following lines used in login are susceptible to SQL injection

```
cur.execute("""SELECT 1 FROM Users
    WHERE username = '%s'
    AND password = '%s'"""
    % (username, password))
```

Type any value for the user name as no injection happens here. In the password field, type anything followed by " ' OR 1=1/* ". The apostrophe ends the password string, the OR 1=1 provides an alternative to having a correct user name password pair as it is always true. The /* begins multi-line comments so the rest of the query is ignored.

```
klaptop-2:common_vulns k$ python injection.py hulk
>>> Enter your username... Hulk
>>> Enter your password... Smash' OR 1=1/*
You've successfully logged in as Hulk
```

2. Again, the status query is also susceptible to SQL injection:

```
cur.execute("SELECT status FROM Users
    WHERE username = '%s'"
    % username)
```

Injection can be performed as follows:

```
>>> Find out the status for which user? ' UNION SELECT password FROM Users WHERE username='Bobby';/*
User ' UNION SELECT password FROM Users WHERE username='Bobby';/* is lolcats
```

3. SQL injections can be prevented by using prepared statements, which create a sort of templated query. Input values are substituted separately using a separate protocol which does not require escaping characters like in a raw SQL query. This makes it resiliant against malicious user input, since that input is never interpreted as part of the query and used only as a parameter. SQL injections attacks remain pervasive because developers remain lax with their input sanitation. However methods to prevent malicious or malformed input from disrupting SQL queries do exist and when used provide a reliable safeguard against such injections.

In python, never use:

```
cur.execute("""SELECT 1 FROM Users
    WHERE username = '%s'
    AND password = '%s'"""
    % (username, password))
```

Instead, use:

```
cur.execute("""SELECT 1 FROM Users
    WHERE username = ?
    AND password = ?""",
    (username, password))
```

And similarly for the user status code

This makes the SQL injection incur an error:

```
klaptop-2:common_vulns k$ python injection.py
>>> Enter your username... Hulk
>>> Enter your password... Smash' OR 1=1/*
Your username or password was incorrect
```

```
>>> Find out the status for which user? ' UNION SELECT password FROM Users WHERE username='Bobby';/*
No such user...
```