ELEC5616 : Assignment 3
Project 3: Common Vulnerabilities
V. Grotte 309193168

Savegames
Set the characters gold or health to a number greater than 9000 by utilising a buffer overflow. How did you achieve this? Explain using reference to bytes and ASCII as to what the exact value was that you achieved.

Name: "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz" gives Gold: 2054847098.

How could this exploit be prevented?

By limiting and ideally sanitising the input.

Could this exploit be useful for more than just the game? Could it be used to gain access to a system? If not, why not? If so, where might it be used?

Yes. This is a standard buffer overflow. A malicious user could in theory inject malicious code into a system using this attack. This is used wherever there is user input, particularly text.

iCubeKinext
Why does the iCubeKinect system use an asymmetric cipher to verify their DVD games? Would it be possible to use a symmetric cipher instead?

The security requirements for this system are :
R1 - Users can play the game
R2 - Users cannot copy the game
R3 - The developers can copy the game

A symmetric cipher system requires that both parties have the same private key. This would violate R2. An asymmetric cipher is what we used in our project, so that only the master could sign updates but the bots could verify the masters signature. The asymmetric cipher allows for all Rs.

What problem exists in the iCubeKinect verification code? How could you make the machine execute any arbitrary DVD?

The verification process relies on a string compare between *cert_hash* (an MD5 hash of cert.metadata and cert.content_hash) and *sig_hash* (an "rsa_decrypt" of cert.rsa_signature and cert.key_id).
We assume that the *content hash* is the 20 bit output of some hash function with the *content of the dvd* as the input to that function.
We assume that *metadata* and the *content* that produces the *content_hash* can be tampered with.
We assume that a hacker has access to plenty of valid certificates via other legitimate games.
cert_hash is an MD5 hash and only 32 bits in size. It is compared to the first 32 bits of *sig_hash.*
This system is vulnerable to a birthday/collision attack where a hacker could find any random *metadata* such that *cert_hash* matches *sig_hash.* This is a trivial problem in 32 bits and has been since the 90s.

How would you fix it? Would the security vulnerability be made less serious by using either a stronger hashing scheme (such as SHA-512) or a different asymmetric cipher?

A stronger hashing scheme could be a big improvement, as it would allow the string compare operation to compare much larger strings. This would "raise the bar" with regards to birthday attacks. A better scheme might also rearrange the parameters of the hash function.

General Questions

Why is it necessary for us to provide the flag -fno-stack-protector to GCC? What is a canary in terms of a buffer overflow and how can a canary prevent a buffer overflow exploit?

fno-stack-protector disables gcc's built-in stack protection mechanisms (some of which are canaries) which would defend against many examples of the overflow attack (but not all).
A *buffer overflow stack canary* is a little piece of data that is placed by a compiler on input buffers (before the next return address). The canary provides an integrity check, the idea being that if the canary is different, there may have been a buffer overflow.

If the game above was written in Java instead of C, would the savegame still be exploitable?

Java has built-in stack protection mechanisms, some of which automatically check the bounds of arrays. This checking prevents this type of buffer overflow attack but Java has its own exploits.

Imagine you were exploiting a program that was running with escalated privileges (i.e. could read sensitive files, modify other users settings and so on) is it possible to obtain a BASH shell using buffer overflows? Be sure to explain what shellcode is and how the shellcode is executed.

Yes. A *bash shell* is an invocation of the "bash" executable found in the "/bin" directory in every Linux distribution. Bash is a text-based command processor and allows a user to take full control of a system. *Shellcode* is any string of commands (payload) that are injected in a buffer overflow attack to perform malicious actions outside a program's jurisdiction. Shellcode can include commands to invoke a shell, which may or may not be "bash" specifically. Shellcode is often written in machine code.

SQL Exploits

Show how it is possible to log in as any user by performing an SQL injection attack on the username/password login page.

username = <press enter>
password = a' OR 1==1 – – <insert_username gets you into username account otherwise your username is empty>

The website has been clued in on their major security problem and prevented the previous attack. Is it possible to use the status query to work out the password of one of the administrators Bobby?

For the **status** input : ' UNION SELECT password FROM Users WHERE username='Bobby'; -- returns: User ' UNION SELECT password FROM Users WHERE username='Bobby'; -- is lolcats where "lolcats" is the password for Bobby

How can these attacks be prevented? Is it a difficult security problem to fix? Why is it so common?

These attacks can be prevented by *sanitizing* all user inputs. This essentially means ensuring that no *escape* characters or otherwise "dangerous" inputs are allowed to reach the database in any form that could cause harm. Though the implementation varies across languages and platforms it is generally easy and simple. This problem is common due to ignorance, laziness and bad development practices.

Is an SQL injection vulnerability more or less sever than a buffer overflow exploit? Justify.

It depends on the context of the vulnerability. The reality is either could be more or less severe in different circumstances, depending on your priorities and system configuration. Bank? SQL.