Tyler Jackson

3/23/15

Mid Term

Software Security

1. Buffer overflows are a problem in c/c++ for several reasons. First, c/c++ allow you to overwrite places in memory that weren’t specifically allocated for an application. In other words some of the libraries will write whatever the input is regardless of the length. This can be dangerous, because a user can gain access to a system, get information or resources he/she is not authorized to see, or use an application in a way it was not intended to be used.

2. There are 2 possible spots for buffer overflow. First gets will read any input until the return character is read. This gets stored in string, so if we input 30 characters, the variable string would not be null terminated and would overflow into other memory spaces. This can be fixed by using fgets and specifying a length to read in, or by just reading in a character at a time until 15 characters have been read in or return is read in using getchar().

fgets(string,size\_of(‘0’)\*15)

Second is when string is passed to fund. It will try to copy string into buffer using strcpy which doesn’t check the sizes of the variables it is interacting with, so if string has 10 characters, and is copied into buffer, it will overwrite other values in memory that weren’t intended by the program. This can be fixed by using strncpy or strlcpy and specifying a length to copy over.

strncpy(buffer,str,size\_of(‘0’)\*4);

3. a. 1

b. 5

c. 2

d. 4

e. 3

4.

One difference is what type of data you are overwriting. So on the stack is statically allocated local variables, return pointers, stack frame pointers, etc. While on the heap is dynamically allocated local variables, meaning the size of the variables is allocated at run time. Another difference is the direction you are writing in memory. The stack and the heap write towards each other in memory, vs. in the same direction.

5.

a. 0xbffff538

b. 0xbffff4fc

c. 0x08048567

6.

Address Space Layout Randomization because the location of where the system calls are stored in memory is random each time an application is run, so an attacker would have to guess and hope to get lucky. This means no script kiddies could just copy an attack either, and try to open a shell without having to know anything. People also relate non executable stacks to ASLR under the umbrella of DEP.

7.

a.

Integrity, because anyone receiving something from the Macro can decrypt the signature of the macro and compare to the data to make sure they are the same. Also non-repudiation, because only the macro could sign with their private key. Non-repudiation falls into authentication as well, because you are verifying that the macro is actually the macro with their signature.

b. The encryption meets confidentiality, because if someone is sniffing the network, they will only see encrypted data.

c. The biggest security risk is that once a user has access to the mainframe she is never monitored by the server again. This means that none of her input is being validated by the server, and she is much more likely to have some serious access to stuff she shouldn’t on the mainframe.

d. To mitigate this risk I would recommend that they don’t remove the server from the process, and that they limit the type of commands she is allowed to make on the server side.

e. An authenticated user could become an attacker and could brute force or dictionary attack the mainframe application to try to generate LoginTickets. Now that she has direct access, she just needs to simulate a similar mainframe id and host application id to her own and could probably make some educated brute force attacks. Might even be able to get access to whatever table is storing the combination of these 2 and then an attacker could generate any login ticket they wanted.

8.

segment 1

-stack-based buffer overflow

segment 2

-none

segment 3

-heap-based buffer overflow

9.

Principle of least privilege means only giving someone authorization to resources that they need in order to complete an assigned task. So for example at work right now we are in the process of building out access control models for our users for an internal product we are building, but we have a lot of contract workers and seasonal workers (interns). We also have a lot of confidential information that we don’t want them to be able to see. So we are using a user role system where we will give all the interns the bare minimum access to the application so that they can do what they need to do, but can’t access any of the confidential information. The access levels we are considering for each group are create, read, update, delete, and execute.

10.

a. healthcare.gov - I don’t know what this site specifically does, but I am guessing you can sign up for health care and get information on healthcare on this site.

**defacer**- a government website wouldn’t want their homepage to have some gross picture on it.

**pharmer** - through pharming - it would be pretty dangerous if every time users tried to access healthcare.gov they were unknowingly redirected to a different site where they would be prompted for information or be given dangerously incorrect health advice.

**phisher** -through phishing maybe if a new healthcare act was passed attackers could send out a slew of emails to people saying “click here”, route them to their own copycat website and say “give your SSN and DOB and get free health care!”

b. usbank.com

**thief** -sql injection - could maybe access credit card information for all users through a user input field

**infiltrator** - if a user got access to their systems they could wreak havoc deleting money from accounts

**exfiltrator** - could get credit card information if they got admin access somehow

c. smu.edu

-**student** could try to gain admin access to my.smu.edu to view everyone’s financial information, grades, etc.

-hopefully there is least privilege set up so that no 1 user could access everything

-**student** could try to delete all the classes from the courses list using sql injection when searching to register

-**freshman student** could try to gain access early to register by falsely authenticating as a senior or something

-**defacer** could deface homepage with an incriminating photo of a professor or staff member

-**teacher** could try to authenticate as the head of a department and change a student’s grades who he hated to all Fs

-**teacher** could try to use phishing on the head of the department to get his login credentials to send himself an email (as the head of the department) saying that he had received tenure and would be getting a $100,000 grant money guaranteed.

All of them are vulnerable to a “DDoSer” (person who attacks using DDoS).