VIEW MODULE CONTENT

Buffer Overflow –" Don't go out of Bounds" – CS2

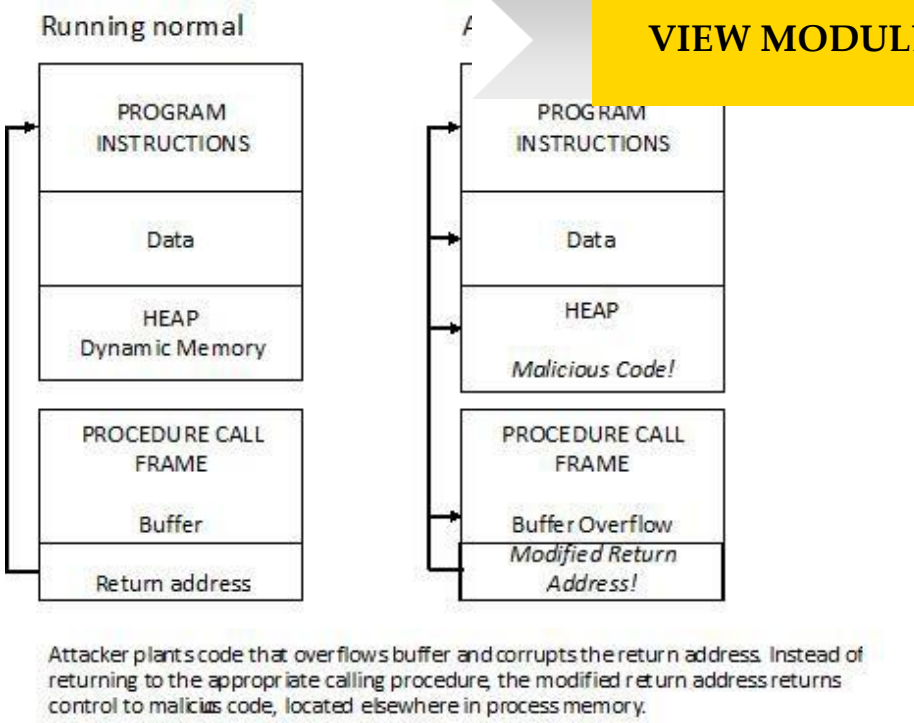
Background

Summary:

Buffer overflow occurs when data is input or written beyond the allocated bounds of an buffer, array, or other object causing a program crash or a vulnerability that hackers might exploit.

Description:

A **buffer overflow** occurs when data is written beyond the boundaries of a fixed length buffer overwriting adjacent memory locations which may include other buffers, variables and program flow data. Considered the "nuclear bomb" of the software industry, the buffer overflow is one of the most persistent security vulnerabilities and frequently used attacks. A malicious user can overwrite a buffer and change important data nearby or, as in the example below, change the runtime stack and force control to pass to malicious code.



Buffer overflow in action

Example in Code:

```
#include
using namespace std;
int main()
{
  int vals[10];
  for (size_t i=0; i<20;i++)
      vals[i]=i;
  return 0;
}
```

This program increments the loop counter past the last element of the array. When the assignment statement tries to store a value in vals[10], a buffer overflow occurs. The results are unpredictable, depending on the compiler and the specific nature of the overflow. Some overflows will not cause any

apparent problems, while others will cause the program to crash. However, all l
languages have tools that will catch buffer overflows before they can become vu
ArrayIndexOutOfBoundsException when a buffer overflow occurs.

VIEW MODULE CONTENT

Because buffer overflow is a common problem in C/C++, it is recommended to declare any variable that is used to represent the size of an object, including integer values used as sizes, indices, loop counters, and lengths, as *size_t*. The *size_t* type is an unsigned integer type.

Code Responsibly-- How can I avoid buffer overflow?

1. *Mind your indices*!
    1. Validate your input. Always check values that are input as an array index.
    2. Check your loops! Especially watch the limit, beware of off-by-one errors.
    3. Check any methods that may modify an array index.

2. *Make sure you have enough space:* Before copying data to a fixed size block, make sure it is large enough to hold the new data. Do not copy more data than your available space can hold.
3. *Validate indices:* If you have an integer variable, declare it as an unsigned int and verify that it is within the proper bounds before you use it as an index to an array. This validation is particularly important for any values that might have come from untrusted sources such as user input, network data, or untrusted files.
4. *When possible, use buffer-size accessors:* Some languages—such as Java—provide operators that can be used to retrieve the size of an array. Using these operators can help you avoid buffer overflow.
5. *Use alternative data structures that reduce the risk of overflows:* When possible, use vectors and iterators instead of arrays and integer-indexed loops. These tools will not eliminate the problem, but will greatly reduce the risk of buffer overflow.
6. *Try to avoid allocating storage until you know how much you need:* When possible, wait to allocate memory until after you know how much space you need. In some cases, this may mean allocating a new buffer instead of reusing an old one.
7. *Send the size of the array along with the array:* If you're using an array as an argument to a function, be sure to send the size of the array to the function as well. This value can be used as an upper limit on array indices.
8. *Avoid risky functions:* Some languages have a variety of library functions that may lead to buffer overflow vulnerabilities. If you are using any library functions for reading user data, copying data, or allocating/freeing blocks of data, understand the appropriate uses of these functions. In many cases, more secure versions of risky functions are available—use these instead.
9. *Use your tools:* Many compilers provide warnings in cases of potential buffer overflows. Use high warning settings, and fix your code to avoid these warnings. Use static analysis tools to analyze your source coda or use dynamic analysis tools to examine and report on the state of your program while running.
10. *Handle exceptions with care:* Checking for and responding to potential overflows in your code, instead of relying on the exception-handling mechanism, will make your code more robust and secure.

Program 1

```
#include
#include
```

VIEW MODULE CONTENT

Video by Lydia Spurrier and Miya Dubler.

Risk – How can it happen?

Buffer oveflow problems are most prevalent in languages such as C/C++ which do not have inherent bounds checking and pointer control, but can happen in other languages as well.

Example of occurrence:

In July 2012, multiple buffer overflow vulnerabilities were found in the firmware used for VeriFone point-of-sale devices. By exploiting these vulnerabilities, hackers were able to control the terminal and log information input by customers, such as PIN numbers and the magnetic stripe data of a bank card. Lucian Constantin. "Artema Hybrid Point-of-sale Devices Can Be Hacked Remotely, Researchers Say", PCWorld, July 12, 2012 http://www.pcworld.com/businesscenter/article/259152/artema_hybrid_pointofsale_devices_can_be_hacked_remotely_researchers_say.html

```
using namespace std;
const int INPUT_SIZE=10;
string getString();
void copyVals(string, char[] );
void getSubstring(char[],char[]);
void getChars(int,int,char[],char[]);

int main()
{
  char vals[INPUT_SIZE];
  char sub[INPUT_SIZE];
  string s1 = getString();

  copyVals(s1,vals);
  getSubstring(vals,sub);
  cout << "sub string: " << sub << endl;

  return 0;
}

string getString()
{
  cout << "Please type a string: ";
  string s;
  getline(cin,s);
  return s;
}

void copyVals(string s, char vals[])
{
  for (size_t i = 0; i < s.length(); i++)
    vals[i] = s.at(i);
  vals[i] ='\0';
}

void getSubstring(char vals[],char newChars[])
{
```

```
    size_t start;
    size_t end;
    cout << "Starting point(integer index): ";
    cin >> start;

    cout << "Ending point(integer index): ";
    cin >> end;

    getChars(start,end,vals,newChars);
}

void  getChars(size_t start,size_t end,char vals[],char newChars[])
{
    size_t sz = end-start+1;
    // must do only sz-1 characters, to leave room for temrinator
    for (size_t i=0; i < sz - 1; i++)
       newChars[i] = vals[start + i];
    newChars[i] = '\0';
}
```

Lab Questions:

1. Type*, Compile, and Run the above program.
2. Complete the security checklist for this program (print the checklist).
3. List the potential buffer overflow errors.
4. Provide example inputs that might cause buffer overflow problems.
5. What strategies might you use to remove potential buffer overflow vulnerabilities from this program?
6. Revise the program to eliminate potential buffer overflow problems. You should be able to do this without adding any exception handling code.

**Copying and pasting programs may result in syntax errors and other inconsistencies. It is recommended you type each program.*

Program 2 (optional - check with your instructor if you need to complete this program)

Write a function that will copy an arbitrary range of one array of integers into another array. Your procedure will take seven arguments:

1. The source array of integers
2. The length of the source array
3. A starting point in the source array
4. An ending point in the source array
5. The destination array of integers – the array that you will be copying numbers to
6. The length of the destination array

7. An integer indicating the index in the destination array where copying sh ou

To avoid overflow, your routine should copy only those values that are available there are not enough values from the source array in the given range, use only those valu es that are available. If there is not enough space in the destination array, copy values only until the available space is full. Your function should return the number of values copied.

Security Checklist	
Vulnerability: Buffer Overflow **Course:** CS2	
Task - Check each line of code	Completed
1. Finding Arrays:	
1.1 Underline each array declaration	
1.2 For each array, underline all subsequent references	
2. Index Variables – *legal range for an array of size n is 0 <= i < n*	
2.1 For each underlined access that uses a variable as an index, write the legal range next to it.	
2.2 For each index marked in 2.1, underline all occurrences of that variable.	
2.3. Circle any assignments, inputs or operations that may modify these index variables.	
2.4. Mark with a V any array that is indexed by a circled index variable.	
3. Loops that modify index variables	
3.1Find loops that modify variables used to index arrays. For any index that occurs as part of a loop conditional, underline the loop limit. For example, if *i < max* is the conditional in a for *loop*, underline *max*	
3.2. Write the legal range of the array index next to the loop limit as you did in step 2.1. Mark with a V if the loop limit could exceed the legal range of the array index. Watch out for loop that go until *i <=max* , as the largest valid index is *max-1*	
3.3 If the upper or lower loop limit is a variable, it must be checked just as indices are checked in Step 2	
4. Sending array indices or loop limits into functions	
4.1 Underline any function arguments that are used to send array indices in.	
4.2 Underline any function arguments that are used to send loop limits in.	
For each underlined function argument:	
4.3 Write the legal range next to each underlined argument. Mark with a V any argument that is not verified to be within these limits.	
4.4 Mark with a V any calls to the function. If the function does not verify that the arguments are within the required range, the code calling the function should do so.	
Highlighted areas indicate vulnerabilities!	

1. Buffer overflows are more troublesome for some programming languagest facilities that Java provides. Some people have argued that this is a good re think this is a good idea? Why or why not?
2. Countless currently running programs were built using C and C++. Buffer overflow vulnerabilities are often found in these programs, often after they have been in use for many years. Why should it be so difficult to find and fix buffer overflow flaws in software?
3. Buffer overflows can be troublesome if they are used by hackers to run their own code. What sort of things might an attacker try to do if he or she were able to run arbitrary code on a computer?

VIEW MODULE CONTENT

G o To To p »