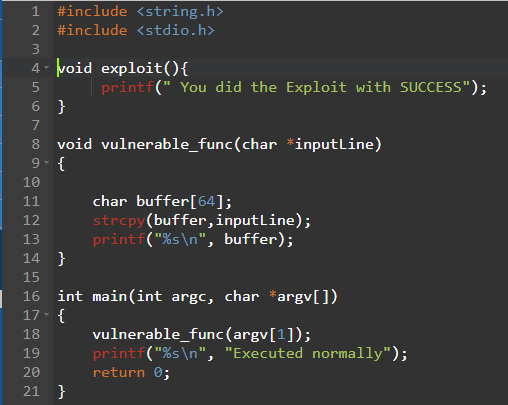
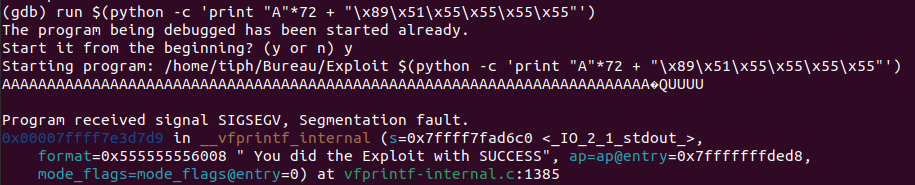
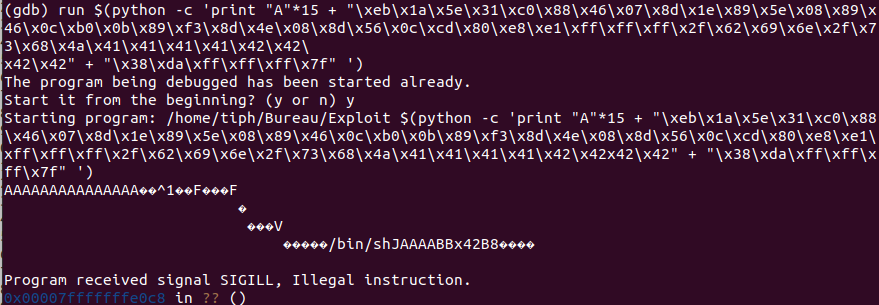
When people ask me what I do, I often tell them that I work in Cybersecurity instead of saying that I work in IT, which would have been much easier for them to understand. But it makes me so happy to see their fascination for this field which is perceived as "Wahouuu, she must know how to hack Facebook". I know, I'm exaggerating... But the reality is quite different when I'm working. I'm lucky enough to be able to touch a little bit of everything and I tell myself that it's essential to broaden one's skills in order to understand what's around us. So I started my career in Governance. I was faced with a dilemma that I can say was a daily one. My main role was to check the compliance of projects related to the PSSI, whereas I was dealing with people who didn't want to be involved in the process and who only saw through the validation of their projects. Which I totally understand... But they didn't understand that it was for the good of the company's security. Today, I integrate a service center where we are several auditors in charge of carrying out a code review through the CheckMarx tool. At the same time, I carry out intrusion tests, which allows me to have more perspective on the importance of securing code through Security By Design. I have audited more than 4000 vulnerabilities in C++/JAVA programs. And I must say that depending on the company where I audit, I can face developers who are much more aware of the stakes involved in securing code and therefore trust and listen to us auditors. Most of the vulnerabilities I've seen are related to the trust we place too easily in the inputs/variables we later include in our code. They can lead to numerous attack techniques that can be exploited to break into the server and even lead to an intrusion into the SI. I thought it would be much easier to understand what is at stake. To prove my point, I decided to create a small program in C++ to exploit a BufferOverflow vulnerability.

In the code below, I write two functions called “Exploit” that I never use and “Vulnerable\_func” where I use strcpy reputable to be vulnerable. In the code opposite, I don't perform any check on the argv[1] argument. This leaves me the possibility to inject any command. I then use this argument as a variable in the vulnerable\_func function that I take care to copy into a buffer of size 64 using the strcpy function. Voluntarily, I use this function because it doesn't evaluate the size of the variable that is copied into the buffer. You will have understood that it is thanks to this vulnerability that I will be able to generate a buffer overflow.

A buffer overflow is caused by certain conditions where a running program is writing data outside the memory buffer. By injecting (shell)code and redirecting the execution flow of a running program to that code, an attacker is able to execute that code. This is called arbitrary code execution. With arbitrary code execution an attacker is able to gain (remote) control of a specific target, elevate privileges or cause a denial of service on the target. Buffer overflows can be proactively prevented and mitigated with several techniques. Programmers should write secure code and test it for buffer overflows. When a buffer overflow is not prevented from happening it can still be mitigated with reactive methods like protecting memory from being written to. To exploit this vulnerability, the idea is to exceed the size allocated for "char buffer [64]". So I repeat the character A 72 times and concatenate to this pattern the hexadecimal address of the Exploit() function in order to force the stack to execute it. As can be seen below the program called the Exploit function and displayed "You did the Exploit with SUCCESS".



It doesn't matter if you don't necessarily understand why I managed to perform a function that was not initially supposed to be performed. The main thing is that you understand that as an attacker, if you let me have this vulnerability, I will be able to exploit it and execute whatever I want! I will even go further in my demonstration, I will run a shell in order to take control of the server. Of course, this kind of case is light compared to what an attacker can itself generate from these vulnerabilities. I pushed my scenario a bit further to demonstrate it to you. Exploiting this bufferoverflow allows me to inject a certain number of patterns to fill the stack and force the stack to execute what I want. Here, my shellcode which is nothing more than the translation of a simple command "execve('/bin/sh')" allowing me to get a bash through an input.



The consequences are such that the attacker has free access to the server and can continue the development of his intrusion scenario. For more explication about all technique which he can use, I invit you to have a look on MITRE ATT’CK website.

As an auditor, the first thing I look at is input validation because it allowed me during several intrusion tests to obtain information on the system I was targeting. For this type of case, it is mainly a matter of performing the checks below :

* Respect Best Practices about the language you use to program
* Check the len of the source buffer; by copy it to a transient buffer for example
* Allocate the right number of bytes for the destination buffer (strlen +1)
* Copy all bytes of the transient buffer to the destination buffer (via a memcpy for example). If this a string, don't forget to add a NULL terminated character if you use a string manipulation function.
* Idealy, don’t use arg[], check if the source buffer contain special characters and reject them.

Securing the code at the earliest stage of a project's lifecycle saves time and money, just like writing an important document that has a significant impact on your business. Studies have shown the difference in cost between a vulnerability that is detected upstream and a vulnerability that is detected in production. The cost is 30 times higher, simply because the developer must go through several test phases before production. Whereas if he integrates this security at the beginning of his cycle, he saves time for correction throughout the cycle and he also gains more security. In this context, Checkmarx have some pluggins which you can add to your IDE and it allow you to discover what you can improve during the development of your codes.

If I can give you a suggestion, I will say that the best solution you can do in order to reduce the number of vulnerabilities. It is to introduce Security By Design in amont of your project.

Notes:

Protection binaires au niveau des constructeurs

Graph du débordement de tampon