Summary

In the first lecture the organizational structure was explained first. A short test was made, to get an overview about the current knowledge state of the course participants. Then some basic terms were explained and defined to have a common basic vocabulary and to prevent misunderstandings during the course. As a first part of “how software breaks” binary exploitation was discussed. Therefore it was explained what happens in the memory if a buffer overflow bug is exploited. Afterwards this was shown as a practical attack and explained in more detail.

1 Organisational Stuff

This course is organized by Sini Ruohomaa and the lectures will mainly be held by Antti Vähä-Sipilä, who also has a industry background and also wants to use this course to evaluate useful methods for teaching software security. These lectures are held without slides. Instead a group of students is responsible to take lecture notes and provide a well readable summary of the lecture afterwards, which can be used by the other course participants to repeat the lecture content or reread it if some points were still unclear. The course can be taken in Bachelor and Master programs.

The course takes place on Wednesdays 16:00 - 18:00 in Room D122 at Kumpulan Kampus. It is compulsory to attend at least 5 of the lectures and there is still a queue of people who want to attend the course. The course does not have an exam in the end. Instead every student has to do homework and hand in a report of lecture notes for one lecture. The homework is organized in weekly projects, which have to be solved and handed in at last 13 days after coming out until 23:55. Homework handed in after the deadline will not be graded. The projects may consist of theoretical exercises, essay writing and drawing diagrams to given exercises. Every student additionally is assigned to a specific lecture. The student is responsible to take notes of this lecture and write a explanatory prose text about the lecture content. Pictures are welcome to be included to make explanations vivid. If you want to use pictures Antti used in the lecture you are free to ask him for these pictures. The first version of the lecture notes has to be returned 1 week after the lecture, the final version is to be handed in 2 weeks after the lecture. All homeworks and the lecture
notes have to be returned via moodle. All homework and lecture notes shall be individual returns and not be copied from other course participants.

All important information will be given via the wiki page\(^1\). Additionally there is an IRC channel, which can be used to ask questions and discuss software security related topics. Instructions are also given on the wiki page. Antti is always accessible there and his phone will ring if you ping him.

If it’s on the wiki, it is official or the wiki is hacked.

For additional reading tree course books are recommended and an additional list of papers related to each lecture is given. This information is also given via the wiki page.

At the end of the organizational explanations the participants were reminded to decide directly after the first lecture whether to take or drop the course, since there is still a queue. A short questionnaire was given to all participants to define their state of knowledge and get information about their background knowledge.

Everyone choose a personal code on their own. The same questionnaire will be filled out after the course is finished to measure how useful the course was.

2 How software breaks (part I)

In the beginning some basic terms were explained and defined to have a common terminology to base on.

| breaking | doing something unintended |
| weakness | potential that something goes wrong |
| vulnerability | weakness which is abusable |
| exploit | leveraging a vulnerability |
| attacker | somebody who tries to misuse the system in any way it was not intended to be used for |
| threat/ threat-actors | possibly evil |
| attack vector | something that carries the attack, if e.g. a house is attacked the attack vector can be an open door, a window or any other weakness |
| attack surfaces | the way the attacker interacts with the system e.g. states, inputs,... |
| reason for the weaknesses | software bug e.g. buffer overflow |
| flaw | logic/ design level issue e.g. wrong technology |
| adversary | something thats against you: attacker, potential attacker, whose intent is against yours |

Weaknesses, which have been found are collected in the Common Weakness Enumeration\(^2\). This is a catalogue of different types of weaknesses. Found vulnerabilities can be registered in the National Vulnerability Database\(^3\). This database can also be searched, every registered vulnerability has its own so called CVE-identifier. An attacker does not have to be a “bad person”, it is generally used in a neutral way, only describing the fact that the system is used in a way is was not made for. Knowing the vulnerabilities of a system is not

\(^1\)https://wiki.helsinki.fi/display/swsec/Software+Security
\(^2\)https://cwe.mitre.org/
\(^3\)nvd.nist.gov
necessarily a bad thing, even if media often considers attackers to be evil people doing bad things. It is moreover helpful to know possible weaknesses in order to design systems is a secure way, reducing the amount of possible vulnerabilities.

2.1 binary exploitation

In case of binary exploitation the attacker somehow gets to execute his code in the native processor command set, which makes it different from crossdescripting, which will be covered in the next lecture. As an example for binary exploitation a buffer overflow was described and an attack was shown as an example afterwards. The explained buffer overflow is a quite old weakness, which is usually tested and prohibited by e.g. modern C compilers. It is a low level class of exploitation but it is very effective.

![Figure 1: Memory containing Stack frames](image)

Figure 1: Memory containing Stack frames

Figure 1 shows the basic structure of memory usage, which needs to be understood for this type of exploit. The text segment contains the program code including all dlls. The Extended Instruction Pointer (EIP) is pointing to the next instruction within this field. The stack is used for subroutine calls and consists of different stack frames. The format of a stack frame is shown on the right side of Figure 1. The frame stores the parameters, which were used while calling the function or subroutine, a return address for jumping back after the subroutine is finished, and a stack for variables used within the subroutine. The attack uses this part of the memory. A big data value is read into a variable e.g. a buffer. If the space needed for storing this big value is bigger than the reserved space and the operating system does not prohibit that (like in Windows XP) other variables can then be overwritten with this specific value. The trick is to find the right position within the read in value to overwrite the right past of the memory, the return address.
3 Demonstration

For demonstrating the exploit the following setup was used: On a Windows XP the server Vulnserver\(^4\) is run. This server is designed to be exploited and therefore has a lot of vulnerabilities. The attack was made from an ubuntu machine. For debugging ollydbg was used. This tutorial\(^5\), which consists of three parts, gives detailed information on how to use the tools. Seeing how easy it is to exploit such a system, is a good way to get a better awareness of system weaknesses and vulnerabilities.

4 Fuzzing

Fuzzing is an easy and useful way to test a system with wrong inputs. It is one of the cheapest, quickest and easiest ways of automated testing. The program Radamsa was shortly introduces here. Radamsa is developed at the University of Oulu. Based on a small set of valid inputs the tool generates a large set of invalid cases. These can automatically fed to the system, which shall be tested. The tool is also used during the first homework and there introduced in a more detailed way.

5 Conclusion

The lecture gave, after an introductory part, a good insight into binary exploitation. Even if the mentioned bug is quite old, it was a good choice because it showed how easy it is to exploit a once detected vulnerability. The attack was shown in a detailed and understandable way.

6 References

\(^4\)http://www.thegreycorner.com/2010/12/introducing-vulnserver.html
\(^5\)http://resources.infosecinstitute.com/stack-based-buffer-overflow-tutorial-part-1-%E2%80%94-introduction/