Risk and vulnerability analysis of antivirus software in C++

Blekinge Institute of Technology
DV1620
05/03/2022
by Oliver Bölin
Oliverbolin97@gmail.com

Summary

The purpose of the analysis was to make a risk assessment for an antivirus software and present these risks and consequences for comparison. How critical do we judge them to be, and what risk should be prioritized? In the risk analysis there were 10 different risks found where 3 were high priority, 5 were medium priority and 2 were low priority. In the discussion it is written how these risks are a vulnerability to the software, and in the conclusion it is written how they should be handled and also how expensive it would be.

1. Introduction

1.1 Background

Malicious code is constantly circulating on the internet, and therefore it is important to have antivirus software that works accordingly. The risk assessment is for the antivirus software created in the course DV1620. Risk assessments are a basis in information security and since people have a complete trust in antivirus software, it is critical for it to be as secure as possible. The antivirus is written in C++ with the latest version as ISO/IEC 14882:2020. The antivirus works as a scanner with a database. If the user wants to scan a directory, it then tells the user if any of the files in the directory has hashes found in the database.

1.2 Risks

- Rogue copies
- Vulnerabilities in the software
    - Buffer overflow
    - Traversal directory attack
    - Arbitrary code execution
    - System crashes
    - Permission and privilege problems
- Human vulnerabilities
- User Experience (UX)
- Manipulation of data
- Ransomware

1.3 Question statements

- What risks are critical?
- What consequences are critical?
- What vulnerabilities should be handled first?
- How can the risks be minimized?
- What would the approximated cost be for solving the security issues?

2. Method

2.1 Explanation of risks

A1: Rogue copies
If a software became populated with users, rogue copies are not unusual. A rogue copy could be a version of the software that doesn't work as intended to make malware pass through it, it could be a trojan itself or just a type of bloatware. [1]

A2: Vulnerabilities in the software
A vulnerability in the software can mean that it's not up-to-date and different methods can be used to exploit the software or the system that uses the software. Some of these methods could be as following:

- **A2.1:** Buffer overflow
  - Buffer overflow is an attempt to gain unauthorized access to the antivirus systems by filling the host buffer memory that is allocated to the antivirus. This is usually done by injecting memory such as strings into the buffer to overflow it. After the buffer is breached, the software could either crash or send over the data to the next block. This could cause a manipulation or an execution of code that the user is not privileged to do. [2]
  -
- **A2.2:** Traversal directory attack
  - While the user could input whatever directory they want, it could be a problem if a user inserts a top directory such as */etc/shadow* or simply */etc*. This could cause sensitive information to be stored in the antivirus and a user could get this information using this method. [3]
- **A2.3:** Arbitrary code execution
  - If a major security bug exists, an attacker could use an arbitrary execution on the software, meaning an attacker could take over the running process of the antivirus and run whatever code they want. The attacker would typically not need privileged access to run an arbitrary code execution on the antivirus. [4]
- **A2.4:** System crashes
  - Would the antivirus have severe bugs making it unusable, the system would be an easy target for exploits. It could also have major faults making it crash, which would render it useless.
- **A2.5:** Permission and privilege problems
  - The software could have some bugs making a user not being able to scan directories that it does not have the privilege to

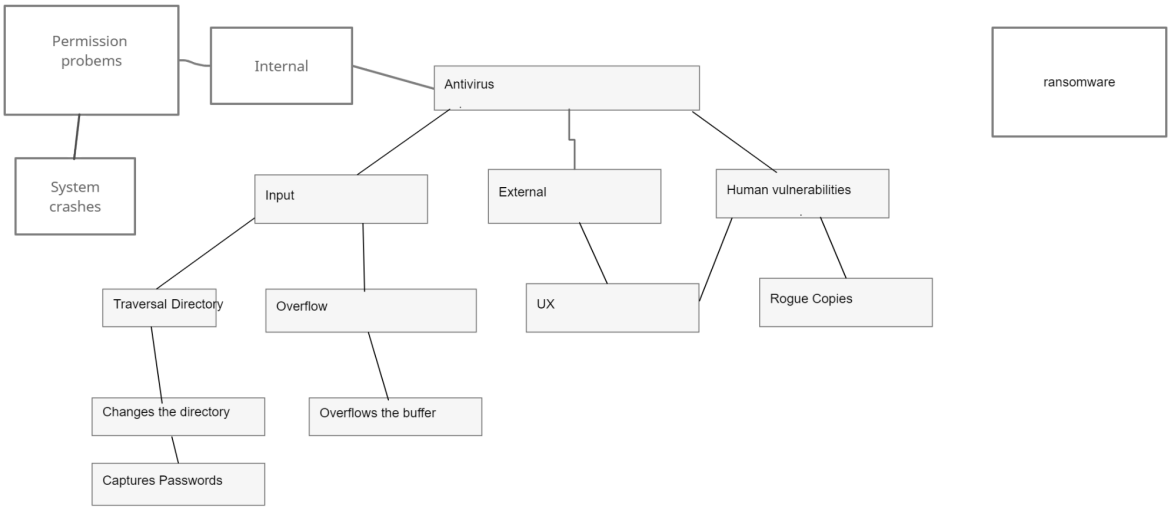
A3: Human vulnerabilities
A human vulnerability could be an effect of a bad UX, or just mistrust in the system. It could also be a social engineered attack for the user to change the signature database. There's not much to gain in reconnaissance and information gathering from the user, except the information where the antivirus is located. The user could use the software poorly even if the UX is in a good state and create mishaps. Perhaps the database could be accidentally deleted or not even used, which creates no protection against malware.

A4: User Experience (UX)
The UX is what the usual user sees. If there is a problem with the UX, a human vulnerability can happen. For example, the UX makes the user think an input should be a certain way, and then the antivirus handles the input differently than the UX tells the user. This creates a UX vulnerability and the antivirus would not work as intended.

A5: Manipulation of data
If an attacker could get into the system, it is entirely possible that they could manipulate the signature database. If the signature database is just stored on the host computer, it could be manipulated with ease. But first an attacker has to gain control over the host. If it were stored externally, it could be vulnerable to data manipulation without attacking the host first.

A7: Ransomware
Ransomware is a type of malware that locks the entire computer with encryption, and stays locked until a type of ransom is paid. It usually encrypts files or entire hard drives. Once deployed, it's increasingly difficult to break free from ransomware

Diagram 1: A diagram showing the paths an attacker could take.



3. Results

3.1 Data

Figure 1: Scale for consequences and risks.

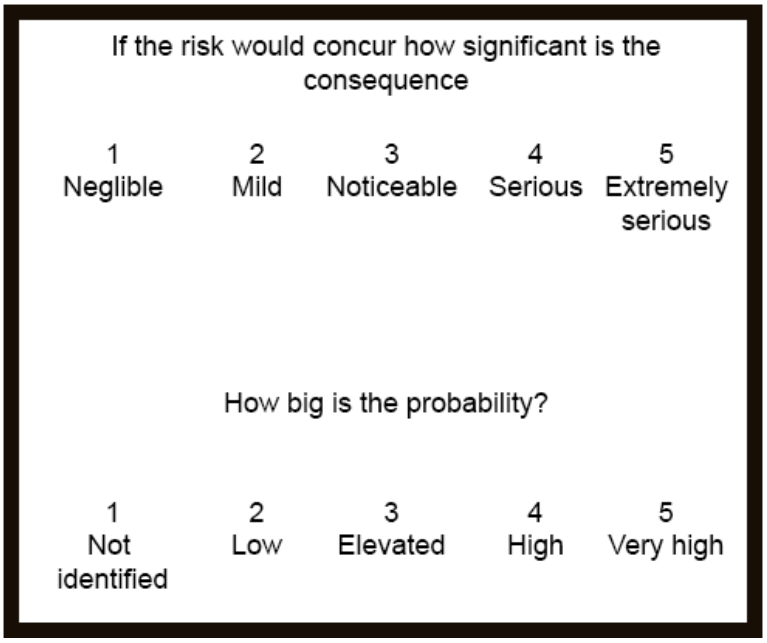


Table 1: A table for the risks and equivalent consequences.

	Risk	Consequences	Probability
A1	Rogue copies	3	1
A2.1	Buffer overflow	4	4
A2.2	Traversal directory attack	5	4
A2.3	Arbitrary code execution	5	3
A2.4	System crashes	2	4
A2.5	Permission/privilege problems	3	5
A3	Human vulnerabilities	4	1
A4	User experience (UX)	3	3
A5	Manipulation of data	4	2
A6	Ransomware	5	5

Table 2: A table for the probability x consequence.

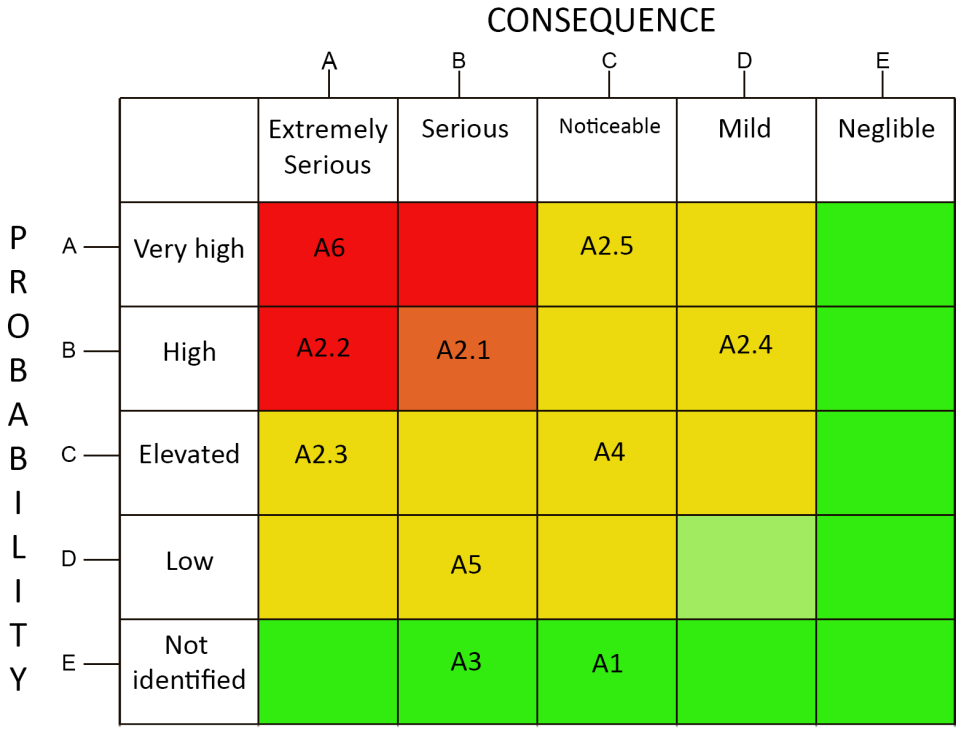


Table 3: A table for the probability x consequence that shows the equivalent value.

	Risk	Consequence x Risk
A1	Rogue copies	3
A2.1	Buffer overflow	16
A2.2	Traversal directory attack	12
A2.3	Arbitrary code execution	15
A2.4	System crashes	8
A2.5	Permission/privilege problems	15
A3	Human vulnerabilities	4
A4	User experience (UX)	9
A5	Manipulation of data	8
A6	Ransomware	25

4. Discussion

4.1 High priority.

4.1.1 Buffer overflow

Since the software is written with static memory allocation, a buffer overflow attack is not too difficult to achieve. If an attacker got into the system and would try to escalate their privilege, this would be a common technique. Perhaps the attacker would like to bypass the antivirus and by using buffer overflow it could bypass it by injecting and executing external code.

4.1.2 Ransomware

The antivirus has no method against a ransomware attack, it would be devastating for the user since it's such a common attack. The ransomware completely bypasses the antivirus before it even has time to scan the files, and could effectively activate the malicious code.

4.1.3 Traversal directory attack

When writing this software, there was no knowledge about traversal directory attacks, and thus it does not have any means of security against it.

4.2 Medium Priority.

4.2.1 User Experience (UX)

User experience, as in the name, depends on the user. If the user has knowledge in terminal-based software, it could be an easy task to use it. But most users are not knowledgeable in terminal-based software such as the antivirus in question. This could lead to a user that fails the scans without knowing it, thinking the files are safe when it could be a trojan hiding.

4.2.2 Manipulation of data

Manipulation could be used within the means of other exploits, but being used alone the risk is limited. Before a manipulation could happen there has to be reconnaissance, gaining access to the host, a privileged escalation, just to change the data on the antivirus. Therefore, the probability is low, but the consequence is high.

4.2.3 Arbitrary code execution

Arbitrary code execution is almost thought of as remote code execution, but not done from a remote computer. The attacker would need to gain some sort of access and from there continue the code execution. By manipulating some pointers and combining it with privilege escalation, the attacker could turn the entire system into a zombie computer via the antivirus. [5]

4.2.4 Permission and privilege problems

The software is built during a short time and therefore a bunch of problems could come into existence that were not known before. Permission and privilege concerns could be one of those, and it could also lead into easier ways to use privilege escalation methods later. A user not being able to use the software as intended because of privilege difficulties could be detrimental. But because the software has not been tested for a long time and on different users, this risk is hard to evaluate. It could be a huge issue, or it could be no problem at all. Therefore, it's in the middle.

4.2.5 System crashes

Just as in permission and privilege concerns, the software needs more testing to evaluate the risk of the program. But since it’s not a continuation of scan but more of a scan once, a system crash once in a while would not do anything, since the user would see this they would just reopen the software and *rescan* the files. A issue would arise if the software would be crashing so much it would render it useless.

4.3 Low priority.

4.3.1 Rogue copies

Rogue copies could be a concern for major antivirus software distributed online, the antivirus in question is a private antivirus not meant for distribution. This would effectively eliminate any rogue copies. If the antivirus would be distributed to more users, the best way to combat this would be having an official website where it’s distributed.

4.3.2 Human vulnerabilities

While UX could be more of a difficulty, human vulnerabilities are not as common here. Since the antivirus is not a complex software, there would be less risk of a user doing something that would be detrimental to the user's security. While an inexperienced user still could perhaps delete some important files, it would not be common.

5. Conclusion

The best way to handle the high priority risks, *buffer overflow* and *traversal directory attack* is to patch the software. For buffer overflow, there is a method called *input validation.* The method contains exceptions which are thrown whenever an overflow happens.

Since some files could be read with a traversal directory attack, it has to be handled. It could be handled in various ways, but making the software from exceptions when there's a directory that shouldn't be touched is probably the easiest way. Hard-coding it this way and possibly making it so root-users could add some directories that should not be touched. This would make it harder to succeed in this attack. [7][8]

Implementing this into the software would possibly take 40-80 hours of work, depending on how fail-proof it should be.

A *ransomware* attack is on a different level than the before said risks. *Ransomware* attacks are so common now, both in huge enterprises such as *WannaCry* and even sometimes in private computers. To counter ransomware, you would need to add these to the database, and always run the antivirus on the downloaded content before you do anything else. If the ransomware is not sophisticated enough, the antivirus should be able to detect it. This would not take a lot of time to implement, but it is very important to do so. [9]

Implementing something like this would not take too much time. The most time would be searching for the signatures that would suffice for security against ransomware attacks. Approximately 8 to 24 hours of work. There is a possibility to have the antivirus on a continuous scan, and when it detects a sort of malicious software it pauses it from being executed, and alarming the user that it found something. Implementing something like this would take around 24 more hours of work.

These risks mentioned should definitely be handled first, and after them the medium priority risks could be thought of. User Experience is something that can be improved with time, seeing and watching what users like, and don't like. Both system crash and permission & privilege problems need more testing to fix.

Arbitrary code execution gets its probability minimized when the other risks get fixed. A more stable software does not have as many ways to get fiddled with. A serious reverse engineer could possibly still find a way to use the software in a malicious way, and more testing would be needed. This would be handled along the way while the software is being used, but around 4 hours of work per week, while it's being used.

The approximate price for solving these security issues is 11000 Swedish crowns. There are always more security issues that could come up in the future and therefore the price could be much more.

6. References

1. Bloatware, Kim Fernandez, 2022
2. How Cybersecurity Really Works, Sam Grubb, 2021
3. PortSwigger, 2021
4. CompTIA Security+ SY0-401: 3.5, Mark Ciampa, 2014
5. KernelCare, KernelCare team, 2021
6. Vlad Rișcuția, 2021
7. The Web Application Security Consortium, Robert Auger, 2010
8. Common Weakness Enumeration, PLOVER, 2006