Abstract

Basically, reverse engineering is group of techniques that are used for extracting the design and architecture of a finished product. It has not officially become the part of software engineering until about 10 years ago. Still, the most of programmers are not familiar with reversing, and the benefits they can get by using it in real world scenarios. Reverse engineering is a primary tool in some security-related fields, and can surprisingly be used in software development to save a lot of time and resources. The demand for more quality and secure software is always present, so it is expected of programmers to resort to reversing much more in the future.

1. Introduction

Reverse Engineering has been used long before the computers were invented. It was used heavily in various industries. Manufacturers used to buy a successful products from different companies and then tried to copy it, or make it even better, and then sell it as their own product. This is one example of reverse engineering. Other, more primitive example could be found in everyday life, especially before when the old radios were used. Enthusiasts used to break pieces of electronics into pieces so they could understand how it works, and how it was manufactured. So, in a way, reverse engineering is similar to science, but instead of trying to figure out how some natural phenomena works, reverse engineers try to figure out design and architecture of something that is man-made, but they don't have a “blueprint” for it.

In this paper the main interest is survey of the Reverse Engineering of software, or Reverse Code Engineering. Throughout the text abbreviation RCE, which signifies Reverse Code Engineering, will be used for Reverse Engineering of software. Common name for the action performed by the reverse engineer is Reversing, and this term is used heavily in this text. Since this is closely related to software
development and computer science in general, standard related terms are used such as: OS-operating system or open source software, DBMS-database management system, IS-information system etc.

When people hear the term Reverse Engineering, usually the first thing that comes to their mind is the word illegal. That is because most people connect it only to stealing products, cracking and removing copyright protection. There are much more in Reverse Engineering than that. Actually, reversing is heavily used in preventing piracy and other security-related fields. In the last decade Reversing became a sub-discipline of Software Engineering and can be used in software development to insure interface interoperability or compatibility with third-party software. Also, reversing can save a lot of time in maintenance of existing software, especially when there is no detailed documentation. Not all reversing is illegal, and in this paper we will try to present important applications of reversing with emphasis on legal rights related to them and we will try to show the benefits of reversing over other fields in certain scenarios. So, the main criteria according to which reversing is classified in this paper is its application in a real world scenarios.

2. Definition of Reverse Engineering

“Reverse engineering is the process of discovering the technological principles of a device, object or system through analysis of its structure, function and operation. It often involves taking something (e.g., a mechanical device, electronic component, or software program) apart and analyzing its workings in detail to be used in maintenance, or to try to make a new device or program that does the same thing without using or simply duplicating (without understanding) any part of the original” [1].

“Reverse Engineering of software, or Reverse Code Engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction.” [2] (wiki). It can also be seen as “going backwards through the development cycle “ [3]. In real world it usually means looking at the binary file and trying to extract meaning from it, decompile if it is a program, decipher it if it is a file format etc.

![Figure 1](image.png)

Figure 1 - The figure illustrates the process of reversing the product and extracting its design
Reversing is a base for some very important computer fields. For example it is used extensively in security-related software, on both sides of the fence. It is used by malware developers and contrary by those developing antidotes. Also, it is used for testing the security of software, for example researcher reverses encryption product and evaluates the level of security it provides. Beside security, reversing can be a big factor in software maintenance. Maintenance of existing systems is estimated to consume between 50% and 80% of the resources in the total software budget [4]. Within the maintenance function, reverse engineering activities (“comprehension”) require 47% and 62% of the total time for enhancement and correction tasks, respectively [5]. Also, reversing can reduce huge amount of time in software development while maintaining interoperability with third-party software, because in reality a lot of times the documentation is not detailed enough and it takes a lot of time for waiting for the improved version of documentation.

If we would live in a perfect world, where all software would be perfectly designed, with fully detailed documentation, and where there is no piracy or malicious software, there would be not much need for reverse engineering. But, history has learned us differently, and we can only assume that there will always be these things. In the future, software is only going to be more and more complicated and it will be even harder to write fully detailed documentation of layers upon layers of program abstractions. Beside that, people are more and more connected with computer networks. When internet became popular, it made fertile ground for all malicious software to emerge. Before that it was hard to spread the virus between users, but now it is very easy because in every moment there are million of users connected via internet. Now we have even more connection with things like cloud computing becoming popular, so it is even easier to spread the malicious software to the millions of users in one second, so there is a huge need for great security products and software protection in general. So when looking at these facts we can only assume that, when people accept the good sides of reversing it will become more and more popular in the days to come.

3. Existing solutions of the problem and their criticism

Here, we will try to classify reverse engineering, based on its applications and elaborate the benefits of using reversing. Of course, as in every classification, there are certain things that are hard to classify in one class only, but we will try to do it as good as possible. Based on this criteria, there are two major applications of reversing: security-related reverse engineering and reverse engineering in software development.

In security-related reverse engineering, reversing is used on both end of the fence, both by people who are trying to do harm, and by those who are trying to prevent them. Since this is a science paper both sides will be elaborated, but we strongly encourage readers to apply reversing only in accordance with law. There are few disciplines here, such as: Reversing malicious software, reversing cryptographic algorithms, Digital Rights Management and Auditing program Libraries.
Reverse Engineering in software development is often very underestimated application of reversing. Reversing can be used to maintain interoperability of different software, protocols of data formats. Also, it can be a big part of maintaining the old software, especially if it is poorly designed or not-well documented. Here we can extract these different disciplines: Achieving interoperability with proprietary software, developing competing software, evaluating software quality and robustness.

3.1. Classification criteria and the classification tree

The classification criteria of interest for this research, as well as the justification there-off, are given in Table 1. All selected classification criteria are explained in the caption of Table 1, and elaborated in the paragraph to follow.

<table>
<thead>
<tr>
<th></th>
<th>Security-related RCE</th>
<th>RCE in software development</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>C2</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>C3</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>C4</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C5</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C6</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C7</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1 - Classification criteria. Legend: Security-related RCE, RCE in software development. Explanation: The table shows the grouping of class in relation to the applications of reverse engineering.

The idea for these classification criteria came from the excellent book about reversing [4]. The classification tree, derived from the above introduced classification criteria, is presented in Figure 3. Each leaf of the classification trees given a name, as described in the caption of Figure 3. For each leaf, the list of existing solutions is given.
3.2.1. C1: Reversing malicious software

Malicious software (or malware) is any program that works against interests of system user or owner. Since internet became so popular, malware became very easy to distribute. Before internet, malicious software was very hard to spread, having to be copied by diskettes, CD's or some other medium. Now, with internet, it is spread instantly to a millions of users. So, obviously, today malware represents a big threat to system owners or users. There are few types of malicious software that are encountered today such as: viruses, worms, trojan horses, backdoors, mobile code, spyware/adware etc.

Reversing is the strongest weapon security programmers have against creators of malware. Anti-virus researchers use reversing heavily while attempting to analyze latest malicious programs, determine just how dangerous they are, and learn there weaknesses so they can produce effective anti-virus programs. The most popular and effective way for anti-virus program to identify and remove malicious software is by using unique signatures. In order to develop signatures for latest malicious software, programmers spend many hard hours reversing malicious software looking for the sequence they could use to identify it, and looking for a weakness they could use to successfully remove it from the system.

This is a best example of how reversing could be used in security for improving protection. Of course, it might, and it is, also used by those developing malicious software. Security is a field that is going to be popular as long as there is crime, and judging by modern history, crime is going to be present for a long time, and the best tool we have against it is Reverse Engineering.

Figure 2 - Legend: SRR – Security-related reversing; RSD – Reversing in software development. Explanation: The figure shows the classes derived using classification criteria presented in Table 1
3.2.2. C2: Reversing cryptographic algorithms
Cryptography has always been based on secrecy. Cryptographic algorithms can be roughly divided into two groups: restricted algorithms and key-based algorithms. The most popular and effective are the latter, key-based algorithms. Those algorithms are usually not secret. Only secret parts are the non-public keys that are used for deciphering a massage. This makes reverse engineering kind of pointless because the algorithm that is used is made public with all it's details. Still there are cases where it makes sense to reverse private implementations of those algorithms. Even though, algorithm is public, it's concrete implementation can have some unexpected weaknesses that can be discovered during reversing. So, in order to test the quality of algorithm implementation, programmer could reverse the code and try to find weaknesses and repair it later if found. This can make an attackers attempt of reversing algorithm implementation pointless. While this is not a really popular application of reversing it certainly is not the one to look by, because errors in implementations are very common in real world scenarios and not spending enough effort on testing can make a very complex and good algorithm be vulnerable to simple attacks.

3.2.3. C3: Digital rights management
Modern computers are not used just for programming or business purpose anymore. They became a multimedia devices and turned most types of copyrighted materials into digital information. Music, movies and even books, which were once available on analog mediums primarily are now distributed digitally. This is a mixed blessing. While it is great for the consumers it provides great complications for copyright owners and content providers. For users this means easy access to high quality information at lower cost. While, this also means a bigger sale numbers for provider it also makes and impossible mission of controlling the flow of such content.

The real problem is that digital information is very easy duplicated and spread illegally. Because of this, providers made technologies to control the flow of these contents. These technologies are called DRM (Digital rights management) technologies.

This topic is highly related to Reverse Engineering because crackers routinely use reversing to defeat the protection of DRM technologies. They reverse the DRM software finding weaknesses they can use, of reverse the media file with protection while trying to remove protection so they can copy and distribute it freely. As always, only defense of reversing is reversing itself. Developers of DRM technologies use reversing for testing there's product, so they ensure that a products potential weaknesses are harder or impossible to find.

3.2.4. C4: Auditing program libraries
One of the downside of non-open source software is that this it has been tested only by the developers. Open source software is viewed by many impartial software engineers, so the weaknesses are easier discovered during design and use. For proprietary software, for which source code is unavailable, reversing becomes a viable, yet admittedly limited, alternative for searching for security vulnerabilities. Of course, reversing cannot make proprietary software nearly as accessible and readable as open source
software, but strong reversing skills can enable a programmer to view code and assess various security risks it poses.

3.2.5. C5: Achieving interoperability with proprietary software
This is closely related with the class “Auditing program libraries”. The techniques used are similar in both classes, only it’s application differs. Here, the emphasis is on application in software development. Interoperability is where most software developers can benefit from reversing almost daily. In a real world scenarios there are lots of times where library documentation is not sufficient for solving the concrete problem. Regardless of the amount of effort the library vendor puts into making a fully detailed documentation, programmers usually find some unanswered questions. In these scenarios, most developers would be persistent in trying to somehow get things work, or they would contact vendor and wait for documentation improvement. Problem is that this usually takes a lot of times, and still when they receive the answer they cannot be sure they have the right information they desired. This is a main problem of proprietary software, because the users have to put complete trust in the software vendor, in this case library vendor. Developers with strong reversing skills will often find dealing with those situations remarkably easy. Using reversing, it is possible to resolve many of these problems with very little time and with relatively small effort. Besides that, looking at the certain parts of library code gives a developer more trust about the library quality. We would like to warn the readers that it is not always legal to reverse proprietary software, and programmers should always consult the legal adviser before involving themselves into risky reversing sessions.

3.2.6. C6: Developing competing software
In most industries this is, by far, the most popular application of reverse engineering. In software industry however this is a least popular application of reversing. Reason is that software tends to be much more complex than products from other industries and reversing an entire software product in order to make a competing product just doesn’t make sense. Is is usually faster and easier to develop product from scratch or license the complex parts from third-party then develop them “in-house”. Exception could be some unknown, unique, highly complex design or algorithms that are very difficult or costly to develop. Beside this kind of situations, reversing is not used for developing competing software. Also, it is not ethic to steal other people ideas and solutions and used them as your own.

3.2.7. C7: Evaluating software quality and robustness
Similar to auditing program binaries for evaluating level of security it provides, it is possible to reverse a program binaries for purpose of evaluating a quality of it’s design and coding practices applied in program. Open source software is an open book that allows it's users to evaluate the quality of it before using it. Contrary to that, vendors of proprietary software don't publish there software code, and by that they expect from their customers to completely trust them.

The need of having access to source code of key software such as operating systems has been made clear by large corporations. For example, Microsoft has special deals for “big” customers, allowing them to see source code of OS for evaluation purposes. Those who lack the purchasing power to convince the big companies to grant them access need to take the company's word that the product is well built or
resort to reversing. Again, reversing cannot produce nearly as much information as the source code, but can be very helpful. We must, again, point out that reversing is not legal in a lot of situations, so programmers who like to use reversing must be very careful when it is allowed to use it and when it is not.

4. Trends and optimal solutions for future
As we mentioned before, some applications of reversing are going to be even more popular in future. Today, reversing is still looked upon with doubt, especially because of the lack of the proper legal system. Lots of reversing scenarios that should be legal are still in shade area. In the future it is expected that programmers and software developers will become more accustomed of using reversing more often. Also regarding computer networks and malicious software, the rise of cloud computing and similar services which require from users to put all their trust into the service vendor must be considered. Besides that, vendors of those services must have unbreakable security, which will again raise a demand for techniques of reverse engineering.

5. Conclusion
In this paper, we tried to classify the reverse engineering by it's application groups. The reason for this is to try to make the readers more familiar with the practices they could benefit from. Most of programmers are still skeptical about using reversing in software development, mostly because they usually think it is illegal and they don't clearly see the benefit of it. The truth is, however, that a reversing is the main technique used in some fields such as malicious software detection and removal, while it can be used daily in other fields such as software engineering. Because reversing is just starting to be used more and more, in the future we can only expect even greater benefits. There are more than few small companies that started making frameworks, for which research based on the reversing is required, that help anti-virus companies do they work better and faster. Also, even though open source software is more and more popular, there will always be proprietary software that could be put to test by reversers. Having said all that, who can put a software package to a stronger security or quality test, than a programmer with strong reversing skill, who can track the interaction of particular software with OS, third-party libraries and other resources from the both low-level and high level perspective.