

DELIVERY OF THE MALWARE
Developing the Virus Scanner for Images

Dmitrii Maltsev

Bachelor's Thesis
School of Business and Culture
Degree Programme in Business Information Technology

2015

School of Business and
Administration
Bachelor of Business
Administration, Information
Technology

| | | | |
|----------------------------|---|-------------|------|
| Author | Dmitrii Maltsev | Year | 2015 |
| Supervisor | Yrjö Koskenniemi | | |
| Title of Thesis | Delivery of the Malware - Developing the Virus Scanner for images | | |
| No. of pages + app. | 42 + 5 | | |

The appearing of the vulnerabilities for private data has been always there since the internet was born. The measures of protection started to grow because of this fact. The new and creative ways of malicious software delivery have been intensified, as well. Today, no one is surprised by standard methods of delivering the virus - banners, spam, suspicious links, downloading files from untrusted resources. Some people even know about images, which contains executable scripts. But it is almost impossible to detect such viruses, even if antivirus is installed. Due to these facts this thesis has the objective to make a research about modern Information Security, especially about methods of virus delivery. However, main objective is to develop the application which scans the images revealing hidden malicious software.

Action research and case study methodologies were used for this thesis. They were chosen because these methods allow reaching the objectives of the thesis.

As a result, this thesis contains the information concerning the malware and methods of its delivery. The methods of protection are also added to the thesis. Scanning applications for images is implemented.

Most applicable result is an application, which could be useful for users, who need to make sure, that their images do not contain malware. In addition, companies could use this application for selling, and therefore earning money.

Key words Information Systems Security, malware, images

CONTENTS

ABSTRACT

FIGURES AND TABLES

| | | |
|-------|---|----|
| 1 | INTRODUCTION..... | 5 |
| 1.1 | Background and motivation..... | 5 |
| 1.2 | Scope, objectives and research questions..... | 5 |
| 1.3 | Research methodology and limitations | 6 |
| 1.4 | Structure | 7 |
| 2 | DELIVERY OF VIRUSES AND PROTECTION | 8 |
| 2.1 | Malware overview | 8 |
| 2.1.1 | Classification and acting of malware..... | 8 |
| 2.1.2 | Methods of delivery..... | 9 |
| 2.2 | Protection Methods Overview | 11 |
| 2.3 | Malware in different type of files..... | 13 |
| 2.4 | Malware within images..... | 14 |
| 3 | PROSPECTIVES USERS OF THE APPLICATION..... | 18 |
| 3.1 | Users interest..... | 18 |
| 3.2 | Companies interest | 19 |
| 3.3 | Summary..... | 19 |
| 4 | TOOLS AND IMPLEMENTING | 21 |
| 4.1 | Tools | 21 |
| 4.2 | Functions of the application | 22 |
| 4.3 | Graphical User Interface (GUI)..... | 22 |
| 4.3.1 | Scheme of GUI | 22 |
| 4.3.2 | Implementation of the GUI..... | 23 |
| 4.4 | Scripts..... | 28 |
| 4.4.1 | “Choose image” | 29 |
| 4.4.2 | “Check it” | 29 |
| 4.4.3 | “Exit” | 30 |
| 5 | TESTING..... | 31 |
| 6 | CONCLUSION | 38 |
| | REFERENCES | 40 |
| | APPENDICES..... | 42 |

FIGURES AND TABLES

| | |
|--|----|
| Figure 1. Application Delivering Malware..... | 10 |
| Figure 2. Average time to Coverage | 11 |
| Figure 3. Typical EXIF metadata..... | 16 |
| Figure 4. Hidden Malware (Barnett 2013) | 16 |
| Figure 5. Scheme of the GUI | 23 |
| Figure 6. NetBeans GUI editor (NetBeans Tutorial 2014) | 24 |
| Figure 7. Final GUI with scripts | 25 |
| Figure 8. Dialog box “Choose the image” | 25 |
| Figure 9. Application did not find malware | 26 |
| Figure 10. Application has found the virus | 27 |
| Figure 11. Application asks about exit | 27 |
| Figure 12. Libraries | 28 |
| Figure 13. Testing plan | 31 |
| Table 1. Notification “Choose the image” | 32 |
| Table 2. Choosing the image and its printing | 33 |
| Table 3. Checking the image (Malware was not found) | 34 |
| Table 4. Checking the image (Malware was found) | 35 |
| Table 5. Exit (“No”)..... | 36 |
| Table 6. Exit (“Yes”) | 36 |

1 INTRODUCTION

The background and motivation, scope and objectives, and limitations are methodologies to be reviewed first. Sources are also to be reviewed, together with the structure of the thesis.

1.1 Background and motivation

As an active user of the Internet, social media, and in particular, gaming communities often raise the question of the protection of personal data. In gaming communities, it is not just a question of personal data, but also inside-game items, purchased for actual money sometimes. Thus it is very important not to lose an access to personal account. Some of my friends had to experience a stealing of their accounts in gaming communities. In one of such cases of theft, it has been revealed, that malefactors had used malicious code inside the image, which was send via Facebook. Apparently, that was a script, written in JavaScript.

Because of the last incident described above, I decided to explore the phenomenon of malware inside images more closely, along with the study of other delivery methods of such software. This study seeks to find out how to protect information systems using default tools, antiviruses and even developing a program.

My personal motivation comes from interest to protect my data and help others to do the same. Not only by providing of the information about malware, but also developing an application to check the images for viruses.

1.2 Scope, objectives and research questions

This thesis focuses on delivering of the malware and protection against it. Especially it reveals the topic of malware inside different types of files. This particular topic leads to the developing of the scanning application.

The main objective of this study is to develop the application scanning images for malware. In addition, the objective is to make a research about modern Information Security, especially about methods of virus delivery and protection against them.

This thesis study addresses three research questions, which are displayed below. The questions are also reviewed from the point of view of how they contribute to achieving the objectives of this research.

1. How does malware get to a user's computer and what kind of countermeasures could the user perform against malware?

This question refers to the research of ways of the malware delivery and methods of protection. Answer to this question gives important information for this thesis, because it discloses the vulnerabilities of the information system.

2. How does malware get into different type of files?

The answering to this question is necessary for this thesis, because it does justify the development of the application. The answer shows, that malware in images deserves more attention than malware in other types of file.

3. How do images get infected with malicious software and how it is possible to protect them?

Information that has been revealed while answering this question, gives theoretical base for the technical side of scanning application. Technique of getting the concealing malware into the images is needed to know how to create the application. It would be impossible to do, without this knowledge.

1.3 Research methodology and limitations

The main method of this thesis is a case study, because this method allows studying any topic mostly in details, concentrating on the parts of the big topic.

The method allows researcher to use various types of sources, which is important studying such a large and complex phenomenon as information security in different types of files and especially in images. (Denscombe 2010, 30 – 31.) Since the malicious software and methods of protection from it is constantly evolving environment, the annual reports, the data from the network of libraries and some articles are mainly the resources that were used for writing of this thesis. It is possible to use all of these different types of sources, because of case study using.

As technique to find the ways to solve the problems, it is necessary to use action research, because its strategy purpose is to solve a particular problem and to produce guidelines for best practices (Denscombe 2010, 6). The reason to use action research is a particular problem with malware within images. Basically, this method allowing studying infected images and creating the scripts based on received information.

The main limitation, however, is the fact that the researcher is not sophisticated enough in technical side of technologies that modern security and viruses based on. In addition programming part requires some knowledge, which has to be studied before starting of developing.

1.4 Structure

Chapter 2 contains information concerning viruses, ways of malware delivery, and modern methods of protection. Chapter 3 discusses methods of protection and presents arguments in favor of an image scanning application. Chapter 4 explains the tools used for creating the application, and describes the implementation process, with screenshots and tables. Chapter 5 presents the part of testing. Chapter 6 provides the conclusion.

2 DELIVERY OF VIRUSES AND PROTECTION

Common virus delivery methods, information concerning mechanism of malware infection within images, as well as ways to protect systems are reviewed in this chapter.

2.1 Malware overview

2.1.1 Classification and acting of malware

In order to talk about the methods of delivery of the viruses, it is necessary to determine the classification of the malware. The difference between the various types of malicious software takes place not only in mode of activity, but in the delivery method.

The first type is a virus. Despite the fact that this word refers to any malware, virus is a part of the classification. In fact it is a program that replicates itself, once launched. (Support.com, Inc. 2013.) Viruses can use a variety of weaknesses in network configuration, or holes in the security of the operating system to spread themselves (Kaspersky Lab 2015).

The second type is a worm. This type has only one significant difference with the virus. Difference is, that worm program is more covert, does not aggressively interfere with the system and runs on the background of the main processes. (Support.com, Inc. 2013.)

The third type is a trojan. This is the name of programs, which are related to Trojan horse and they act accordingly. These are programs which hide their appointment until the last moment. (Support.com, Inc. 2013.) Trojans are able to delete, block, modify or copy information, but unable to replicate themselves (Kaspersky Lab 2015).

The fourth type is rootkit. This is a special program, purpose of which is to hide the presence of malware in the system. Typically, processes of malware rootkits disguise as conventional processes.

There is also a classification by action. To counteract the malicious software, sometimes it is enough to know the principle of operation of virus.

Adware is advertising software, aggressively distributed by email. Often, they may not be actually malicious. (Support.com, Inc. 2013.) Similar types of malicious software can be found on the websites, in the form of pop-up windows, links to advertising content. To fight adware it is possible to use Adblock or similar types of programs.

Spyware is software that scans computer for information and sends it to its creator. This kind of malware can be considered as one of the most dangerous types, because of jeopardizing the personal information of the user. (Support.com, Inc. 2013.)

Ransomware does blocking the normal operation of the computer, and extorts users money in exchange for the normal operation of the system (Support.com, Inc. 2013). Ransomware is an electronic analog of the usual blackmail and fraud. Usually, this type of malware does not cause any critical damage of the system, in essence being a bluff.

Scareware is a malware that scares the user with non-existent viruses and with general infection of the system (Support.com, Inc. 2013). This software offers to install fake antivirus software for a small amount of money, claims that the system will be protected, or spyware offers to destroy the identified "virus".

2.1.2 Methods of delivery

There are various methods of delivery of viruses. Any resource of information delivery can become supplier of malware. In other words, methods of delivery of viruses are e-mails and suspicious sites. The main objective of methods is to deliver malicious software on the user's computer. In general, it means

downloading a variety of files. It can be e.g. a setup file with malicious software, images, text files, and archive data.

There are a few methods of attracting users to download virus, such as links to some web resource, links or files that are sent via the social networks. Additionally, sometimes malicious software is distributed with some software packages, and in this case a virus penetrates directly into the computer's memory.

Figure 1 displays applications delivering malware. The figure is made in the form of a radial diagram.

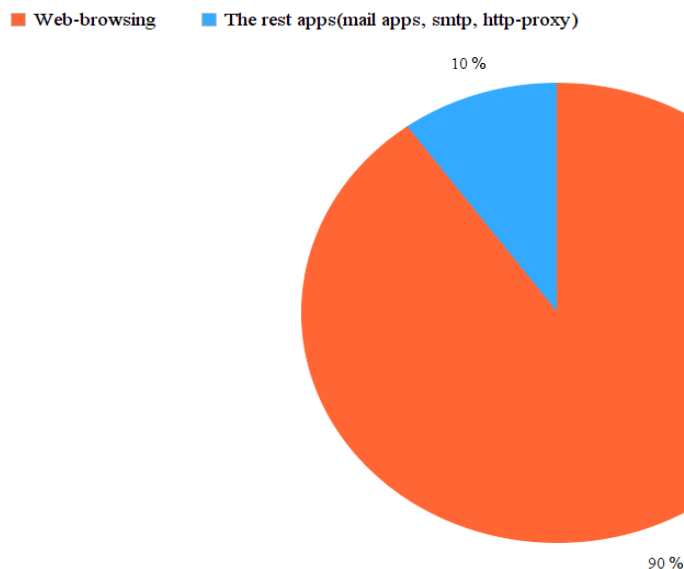


Figure 1. Application Delivering Malware

According to research by PaloAlto Networks (2013, 6), Security Company, in current realities about 90% of undetected malicious software is downloaded in web-browsers. The traffic of e-mail services contains only 10% of new and unknown malware (PaloAlto Networks 2013, 6).

The data of average time to coverage is presented in Figure 2. It shows the bar graph of web transmissions and possibility of antiviruses to cover them.

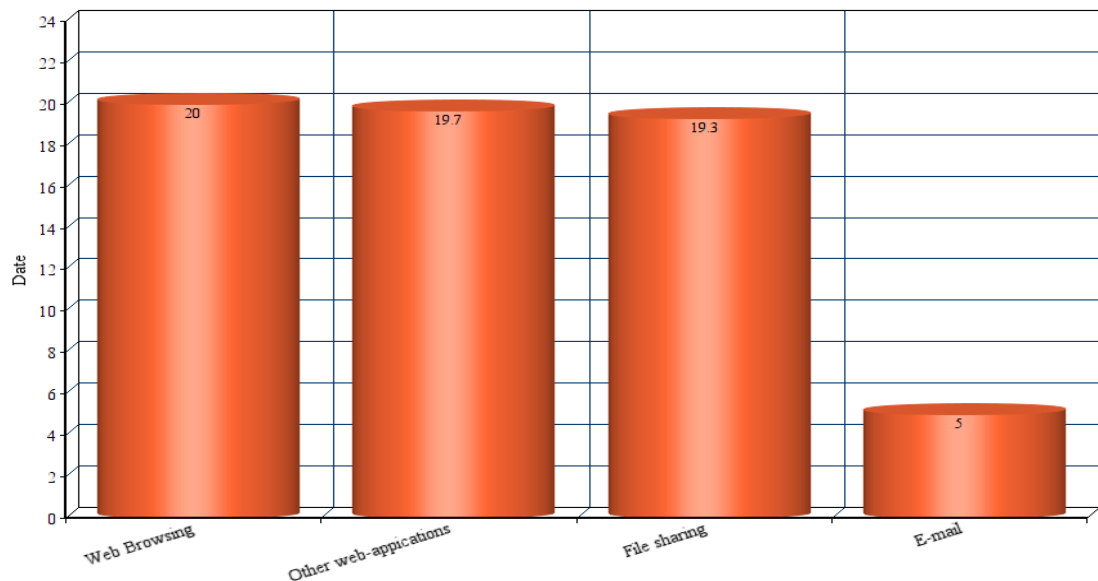


Figure 2. Average time to Coverage

According to the data from the same report, the latest antiviruses are able to make coverage in about 5 days in the case of transmission by electronic mail services. With the transmission via web browsing it is possible to do in 20 days. Almost 20 days needed for file sharing coverage and in a variety of other web applications. (PaloAlto Networks 2013, 7.)

Such statistics explain that the malware via web browsing is typically loaded in real time. However, viruses can be detected by the servers themselves when they are loaded through the mail systems (PaloAlto Networks 2013, 7).

2.2 Protection Methods Overview

There are many ways to protect the information systems from malicious software. The main methods are presented below.

- Using the antivirus software. This is the basic program that scans computer and various files for malicious software. Regularly updated antivirus reduces the risk of infecting the computer significantly. (Microsoft 2015.) Constant

updating of antivirus forces to leave in the past the old ways of spreading viruses and malicious software and makes developers of malware to stand in front of new problems and to extend the time of producing the viruses.

- Using the firewall allows preventing the suspicious activity, if the virus tries to download some other malware via internet (Microsoft 2015).
- Adjusting personal settings in browser. Often, it makes possible to prevent the web-sites from use of private information. (Microsoft 2015.)
- Not opening the suspicious e-mail attachments. Such messages are easy enough to identify. The one who receives e-mails does not know from whom it came; subject and text are in a sense attractive and enticing.
- Installing an extra blocker for advertising and pop-up windows.
- Using complex combinations for passwords. Even the most advanced software needs time to guess passwords; at least, this measure could help delay the hack. (Microsoft 2015.)
- In case of online distribution applications such as Steam or Battle.net client for additional protection it is possible to use special applications for additional authorization - Steam Guard and Battle.net Mobile Authenticator. Using such applications, the primary authentication is not possible without specially sent by the application password. (Steam 2015.)
- Due to the danger of malicious macros within text files it is necessary to disable the possibility of activating the macros in Microsoft Office and similar type of applications. More details concerning macros are in chapter 2.3.

All of the above methods being executed in full measure provide sufficient protection of the systems. However, a better understanding of how viruses enter into various types of files is necessary if user wants to be absolutely protected.

Moreover, the malicious software could be in the most unexpected places, such as video and audio files.

2.3 Malware in different type of files

This chapter contains an overview of methods of infection of files with viruses. All considered information is about video, audio and text type of files.

- In media video files, there are two vulnerabilities, which make appearance of malware possible. The first method of infection the video is fuzzing. Originally this method was designed to test a steadiness of the source code. However, now it has become a tool of infection. In fact, this is a method which allows a programmer to provide invalid or random data to the source code, thereby causing unpredictable behavior of the program. Often it requires considerable knowledge of the file format or the file itself, so this type of viruses is quite rare. (Thiel 2008, 1-3.)
- The second method is embedding the URL inside of media files. The essence of this method is that some video file formats such as Advanced System Format (hereinafter ASF) allows some simple codes to be executed. The scheme usually is simple. This small script allows the user to download the executable file with a virus, which is disguised as required to view the file codec. (OPSWAT, Inc. 2014.)
- In media audio files, there are not many ways to insert malware in. As a matter of fact, viruses cannot be spread with audio files, due to the nature of such formats. For example, there is no way to infect .mp3 files with viruses, because files of this format are not self-executable and even if a hacker sets a code inside the data, the file would still be safe to use. However, already in 2008, Kaspersky Lab found the worm that was able to convert .mp3 files to the Windows Media File (hereinafter WMA) without changing the name of the extension, while adding a link that automatically opens a browser and offers the user to download malicious software (Kaspersky Lab 2008). However, today, this malware is very rare. There is a more applied disguised

.exe files that have names such as test.mp3.exe, the browser and systems are often displayed as test.mp3. This phenomenon is more specifically described in chapter 2.4.

- In text files, the malware could be added/downloaded using macros. Macros are commands, which can perform tasks automatically. Macros are embedded within Microsoft Word or Excel files and written in Visual Basic for Applications (hereinafter VBA). Once a user opens a file with infected macros, it will be launched; therefore such a malware could be quite dangerous. The best way to protect the system is disabling the macros in Microsoft Office applications and open only trusted files. The reason for such actions is that modern macros hide themselves quite well. The hackers usually convert the extension of the text files from .doc or .txt to .mht, but leave the name of extension as .doc or .txt. This extension allows avoiding antivirus checking. However, while the user opens the file, it still runs the malicious macros. (Mosuela 2015.)

In different types of files, there are various types of malicious software. In such types of files as video or audio, it is difficult enough to get the virus. Because of that, such files cause less concern to users. However, by the malicious macros within text files, viruses can cause great harm to the system.

2.4 Malware within images

This chapter is related to the implementation part, defined in the chapter 4. The mechanism of infection image with virus is described in detail.

The first method of delivery of the virus using images is the so-called .jpeg - rar archive. The point this method is to convert the .JPEG file into a special .JPEG archive. In fact, the .JPEG archive also opens and it displays the image, but inside there is also hidden some information that a user can get with the standard program for the opening of the archives, for example WinRar.

It is possible to make sure that this is an archive in two ways. First, ensuring can be done by checking the file size. It is very rare situation, when actual .JPEG file weighs several megabytes. Usually, when .JPEGs weight is more than 1-2 megabytes it could be an archive. The second method is less obvious, because it is in the image metadata. The jpeg-rar archive header is placed on the opposite side of the file with respect to the normal jpeg image.

The second method that uses a double extension is also interesting. This method is simple, but effective. For example, if users want to download or open the file called test.jpeg, the makers of malicious software simply name their programs as.exe programs, which means that they are executable, for example as test.jpeg.exe. Usually, the latest versions of Windows do not show the file extension, if users do not click on it or view the properties. Thus, it is very easy to do such deception, due to carelessness of some users. However, as mentioned above, it is very easy to detect malicious software by simply checking each time the file extension. (Ziff Davis, LLC. PCMag Digital Group 2015.)

The process of virus infection image is simple, because a malicious program is encoded, usually by R57Shell and inserted into the headers of the Exchangeable Image File Format (hereinafter EXIF) metadata of the image (Trustwave Global Security 2013). Antivirus does not identify encoded malware is a malicious injection, identifying the line as usual metadata (Barnett 2013).

Decoding of the software can be conducted by applying simple PHP script using base64_decode function. Execution of this script can be carried out at the opening of the image. (Barnett 2013.)

Figure 3 presents the typical EXIF metadata. Every image contains this kind of metadata.

```

' ",#(7),01444'9=82<.342ЯН С
/bsb-[±X,osIU3fKfKj~OET$gyjKI(енд;црцтк9жлв;3L4"±нцтнвкp'FwEuyufnrvw&XlC.D(K9cFyf±hJ.†V•Y;;$Rk0ny€3'3V~††qYK€JдсK#д фZ~"K,e'ы+INSEh~tosC
XBВ»'NORkцI-εзj6eяIy~n'o-r2-V o~й>R1QqvanstoyE3лH3:oio:"Ret?uJ03лIJ'.sv~bII{A0.[3.ta.ы'»· AIyIZ,mmю6!%~ж8K€]p{be®rdu°Fi>0hI3or4M-6 ж7
лЬW<?€Ye=H<oh/finh~bEdy98:аФ3•kOb04±,аh||qWv3ЙUnJZ%ms{c\qW€Hkiд~o$ZC1b3h'~Kk=гXQ,ssleñIьWYE!пR3»9HЗB)"диОщИЙн+!%r'+)RWñjkцq$S3±'kHw'o~XK
meta/" x:xmp:tk="XMP Core 4.4.0">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <rdf:Description rdf:about="" xmlns:mwg-rs="http://www.metadataworkinggroup.com/schemas/regions/" xmlns:stdim="http://ns.adobe.cc
    <mwg-rs:Regions rdf:parseType="Resource">
      <mwg-rs:AppliedToDimensions rdf:parseType="Resource">
        <stdim:w>640</stdim:w>
        <stdim:h>480</stdim:h>
        <stdim:unit>pixel</stdim:unit>
      </mwg-rs:AppliedToDimensions>
      <mwg-rs:RegionList>
        <rdf:Bag>
          <rdf:li rdf:parseType="Resource">
            <mwg-rs:Extensions rdf:parseType="Resource">
              <apple-fi:Timestamp>229553029</apple-fi:Timestamp>
              <apple-fi:ConfidenceLevel>288</apple-fi:ConfidenceLevel>
              <apple-fi:FaceID>1</apple-fi:FaceID>
              <apple-fi:AngleInfoRoll>270</apple-fi:AngleInfoRoll>
            </mwg-rs:Extensions>
            <mwg-rs:Area rdf:parseType="Resource">
              <stdim:y>0.738</stdim:y>
              <stdim:w>0.344</stdim:w>
              <stdim:unit>normalized</stdim:unit>
              <stdim:x>0.447</stdim:x>
              <stdim:h>0.458</stdim:h>
            </mwg-rs:Area>
            <mwg-rs:Type>Face</mwg-rs:Type>
          </rdf:li>
        </rdf:Bag>
      </mwg-rs:RegionList>
    </rdf:Description>
    <rdf:Description xmlns:xmp="http://ns.adobe.com/xap/1.0/"><xmp:CreatorTool>Microsoft Windows Photo Viewer 6.1.7600.16385</xmp:Cre
xmpmeta>

```

Figure 3. Typical EXIF metadata

The problem is that the sense of this data flow is quite difficult. Metadata of images is a perfect refuge for malicious script.

Figure 4 presents a hidden malware within EXIF metadata. Malware is highlighted in blue.

```

1  ŷ0ŷàJFIF`ŷá!ExifII*эм,/.*/eeval(base64_decode('aWYgKGlzc2V0KCRfUE9T
  VFfienoxIl0pKSB7ZXZhbChzdHJpcHNsYXNoZXMoJF9QT1NUWyJ6eJiXSkpO30='));
  ŷÜC
2
3
4
5  ŷÜCŷÀM, "ŷÄ
6  ŷÄµ}!lAQa"q2'!#B±ARNð$3br,
7  %&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyzf,...†±^%Š'""•—~"šç£¤
  ¥|S""0a23'µ¶•,10ÄÄÄÄÄÇÈÈÈÈÈÖÖÖÖÖ×ØÜÜÜÜÜÄÄÇÈÈÈÈÈÖÖÖÖÖ+øùúŷÄ
8  ŷÄµw!lAQaq"2B'!±Ä #3RðbrÑ
9  $4á%ñ&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz,f,...†±^%Š'""•—~"šç£¤
  ¥¥|S""0a23'µ¶•,10ÄÄÄÄÄÇÈÈÈÈÈÖÖÖÖÖ×ØÜÜÜÜÜÄÄÇÈÈÈÈÈÖÖÖÖÖ+øùúŷÜ?ŷü
  ç1ŷxŷMðš.zİçšivİÜV{È...ŷŽXİÈİÖ5%ŷ-à.âµðp ŷä«Hð0%xA³
10  ŷTŷ$ŷŽİšá?á!~ŷð~à!üÜñtÄL|?ŷ!èÄŷø8ŷŷä" {
11  žÈÈXÈİŽİšá?áŷpððöäŷü[ñÈ?á!>ððöäŷü[ñÈ-Ä"ð0úÄçGwEp,ûL|?
  =|uàñŷç:op9L,ðšððñİfžðskŷç(T*ŷ#*hrós#žçžúÜWÄçÄŷüäðŷÜŷüœŷ'~€È$øÄÄ?
  Ž~n?'B0•fŷ?BfššÜ, 'Ml,W;ú+,ŷgáútnİf?ðomŷç)ÈÜLü?+ŷøN|økBü|?
  asÜY2EB|ÉŷÄÄL|?ŷ!èÄŷø9ŷŷä"ÄL|?ŷ!èÄŷø8ŷŷä"ž..._äçúÄçGwEpŷð0çÜİ?>
  +ŷŷä"žð0üİp+µŷäëü
12  ¶ž+0PbdwTwİñŷÄš...ŌvöB0ðŌfÄi«QèP4Žŷ¶"w'wøŷü6+9$İ+Ä
  'ž;ûNLÜ}hÜ}ilŷÄRÄ Z.Cöü|G6.Yäž~b-ó?µ•û2hzf!«ix+ñjre-ž)>ð~ŷÄSžä...

```

Figure 4. Hidden malware (Barnett 2013)

As it can be noted in Figure 4, this highlighted array is seemingly without any sense, it is quite difficult to notice the malicious software, especially if it is encoded. Even `base4_decode` line is not so noticeable in the middle of this mess.

It can be concluded that problem of searching the viruses inside metadata of images might be solved with the help of relatively simple algorithm. To solve this problem, a script needs to be written that can detect malicious software. Since most viruses are encoded, there is no need to look for the script itself. It is only necessary to find the row of the decoder. As the foolproof, when malicious software is simply inserted into the metadata without encoding, it is also possible to use a mechanism for finding scripts, which are based on PHP or JavaScript.

3 PROSPECTIVES USERS OF THE APPLICATION

Arguments for and against developing of image virus scanner are provided in this chapter. The discussion includes possible benefits for users, organizations, and open source software.

3.1 Users interest

Based on the previous chapter and specifically on the information about the viruses inside the image, it can be concluded that a program for finding malicious software in such files is really needed. Since the viruses are successfully hiding inside the images metadata, there is not always a guarantee that the antivirus will find the malware successfully.

However, another issue arises for users. Users want to know why they need a program, if it is theoretically possible to find the code as that, more independently; simply by opening the image via a notebook and find the PHP-script that encodes the virus.

The benefits are quite simple. Not all users are familiar with the process of virus infection inside the image; many will not be able to determine the location of the PHP-script and the way how it looks. Many of them simply will not have any desire to look for the virus itself, but it is still necessary to check the downloaded image.

Any special software to detect malicious software is not required for other type of files, such as audio, video and text formats. Video and audio files rarely contain viruses. Most of text files, even if they contain malicious software, cannot be activated without macros, which are normally disabled on the computers of cautious user. Therefore, creating an application for images provides more prospects and sense than the creation of something similar to a video / audio or text formats of files.

3.2 Companies interest

As for companies, the interests concerning this application are also obvious. Any application that is potentially useful for many people can be a source of income, especially if the application is multiplatform. It can be placed on platforms such as the App Store or Android Market, earning both from purchases and advertising.

However, the application has a better value as Open Source Software, as such applications already exist. Such applications, including virustotal.com or eset.com, have more features than this malware scanner; for example, they scan not only images, but web-sites and text files as well. However, these are online applications, which user cannot install to the system and use whenever and wherever it is needed. Further, such applications exist in the depths of GitHub, but there is not much information about them. Therefore, it can be assumed that these are similar projects. However, my application can be useful from the point of view of an independent developer, if it will appear on GitHub. Such narrowly focused applications give substantial profits only in conjunction with large anti-virus packages. However, with the competent marketing and a small price/profit policy it is still possible for the companies to use it.

3.3 Summary

This application can be useful for many users. If there is a suspicion that the image may contain a virus, it is better to check and the best way is to use this dedicated application that can scan whether there is a virus in this image.

For development of OSS, software component of this application may also be useful. For example, this application could be useful in the development projects of more advanced applications which capable to detect viruses within all possible files and for each transmission method. Still, there is a need in base for such programs. Thesis application could be this base.

For companies, there is another opportunity to benefit by distributing such applications. For example, scanning application can be ported to mobile platforms, and it is quite possible to do because the application is developed on the Java and this programming language is supported by most types of operating systems. Thus, the mobile application can be sold at a very low cost or free, earning by a built-in advertising. In any case, opportunities for profit clearly exist. Thus, it can be concluded, that creation of such an application is in some ways necessary and its development is justified.

4 TOOLS AND IMPLEMENTING

The platforms and tools, which have been used for developing, are first discussed here. The reasons of choosing the selected technologies are pointed out. This section explains implementation and testing processes.

4.1 Tools

The system of developed application can be implemented in almost any programming language and any development environment. Therefore, I had a full freedom of choice in tool selection.

At the initial stage of development, it was decided to use Python as the programming language, JetBrains PyCharm as integrated development environment (hereafter IDE), and for graphical user interface (hereafter GUI) development use the expanding library for Qt called PyQt. However, the problems of implementation have arisen soon, as it turned out, that Python does not have all the necessary libraries to implement everything as planned and it was way too hard to connect design in PyQt for the graphical user interface(hereinafter GUI) and code itself.

As a result, Java was chosen as the PL for the development. The NetBeans 7.3.1 was used as an IDE. And built-in library Swing was chosen for the development of GUI. During development, it was found that the application cannot be done without use of external libraries, therefore the library of javaxt-core 1.7.4 was used to simplify the work with EXIF metadata.

Basically, this library allows reading and displaying the image metadata easily. The two lines of code only are required in this particular case:

```
Image image = new javaxt.io.Image(path); // getting image from its
location
```

```
HashMap<Integer, Object> exif = image.getExifTags(); // this method  
allows to get metadata, parsed by EXIF tags
```

Therefore, reading of metadata is followed by a list of EXIF tags (JavaXT 2015).

The Application was tested and run on Windows 7. However, since it is written in Java, it can be run with other systems such as Android and MacOS, but the performance with them has not been tested yet.

4.2 Functions of the application

Prior to the development, it was necessary to determine the functions of the application. The main functions, which have been selected are as follows:

- Uploading of image by its selection
- Reading of images metadata
- Searching for encoded malware
- Informing the user about finding/ not finding malware.

4.3 Graphical User Interface (GUI)

This chapter describes the first design and scheme of the Graphical User Interface. Additionally, it shows the implementation of it.

4.3.1 Scheme of GUI

Figure 5 depicts a scheme of the applications GUI. The color scheme was chosen standard, gray and white, text color is black.

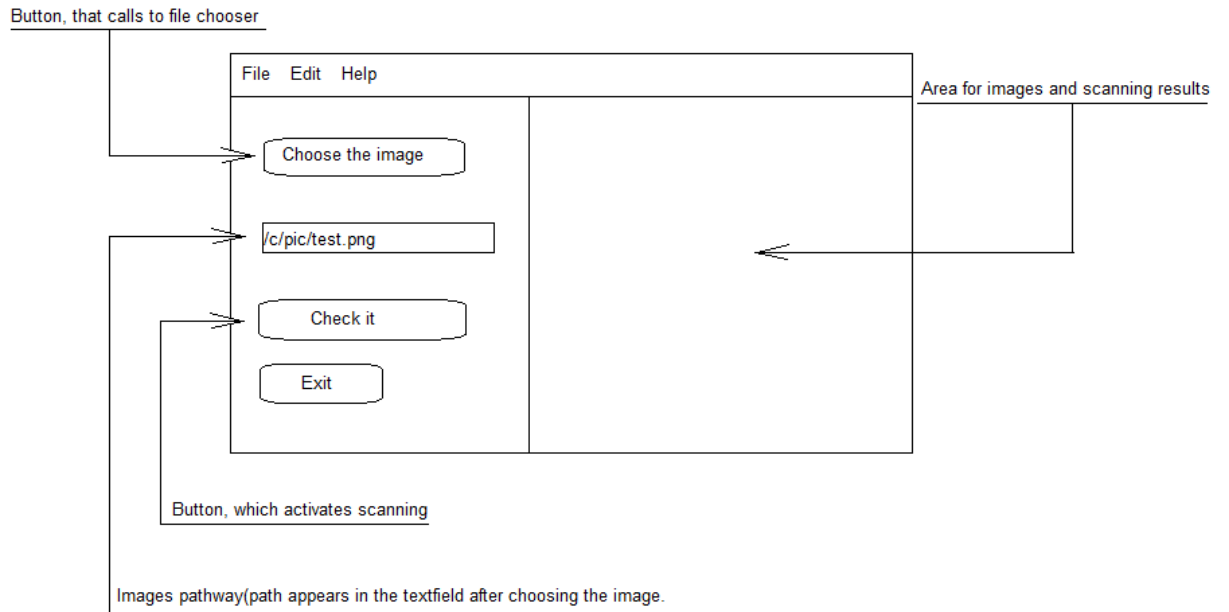


Figure 5. Scheme of the GUI

The structure of graphical interface is quite simple. Initially, there are three buttons and the text field. The first button is called “Choose the image”, the second “Check it”, and the third “Exit”. The process is also simple as the user first downloads the image using the file chooser. This process is activated by the button “Choose the image”. By the click of this button, the image will appear to the right from the buttons, the pathway to the image will appear in the text field, and then the user clicks on the button “Check it”. The image disappears, and the application displays the results of the scan in the released area to the right. Closing of the program takes place by pressing the “Exit” button or by pressing a standard "close box" button.

4.3.2 Implementation of the GUI

Figure 6 presents a NetBeans IDE. Specifically, this screenshot shows the GUI editor and elements of the Swing library.

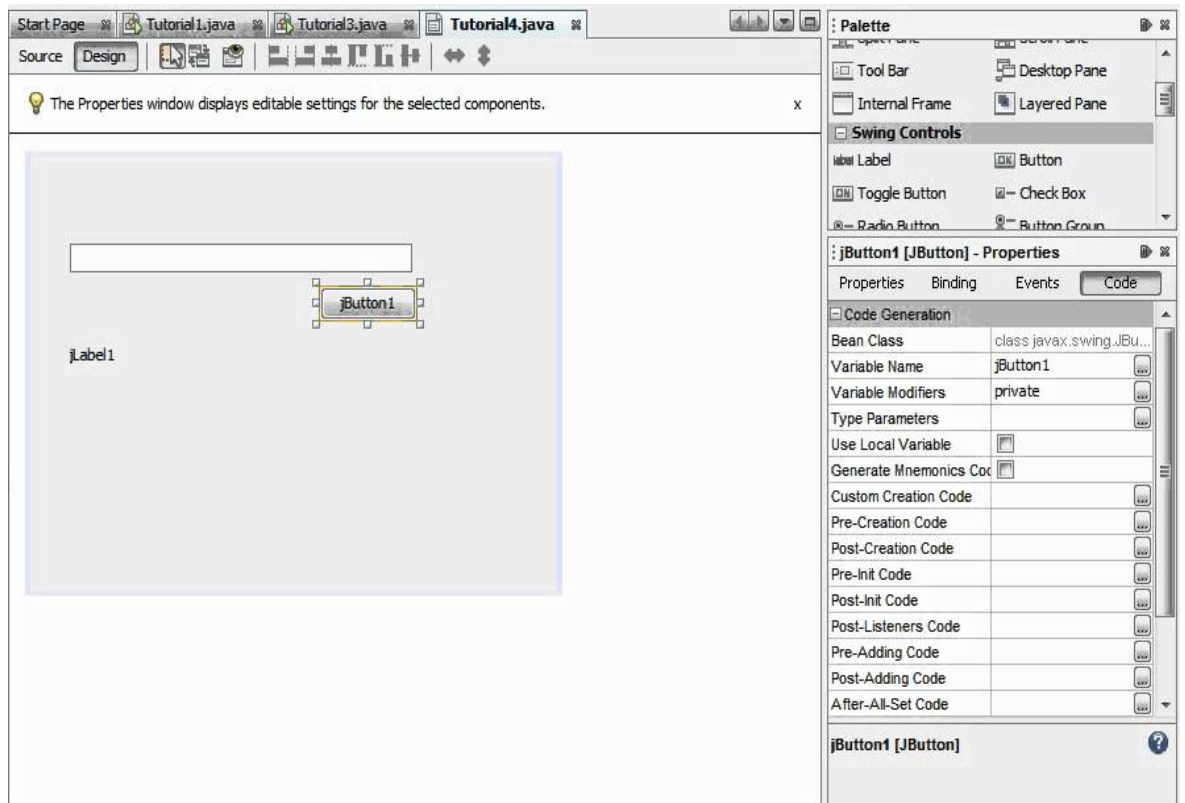


Figure 6. NetBeans GUI editor (NetBeans Tutorial 2014)

As mentioned above in chapter 4.1, the built-in library Swing was used to create a GUI, which allows making elements with modern design. Since the elements can be added using the NetBeans built-in graphics editor, the time that it would take on the intermediate GUI testing is significantly reduced, since it is not necessary to verify where elements are located.

The final version of the graphical interface is the same as that shown in Figure 5. The script-handler is its own for each button, which is discussed in the next chapter in details.

Figure 7 indicates the final version of GUI. All scripts are added and the application is working.

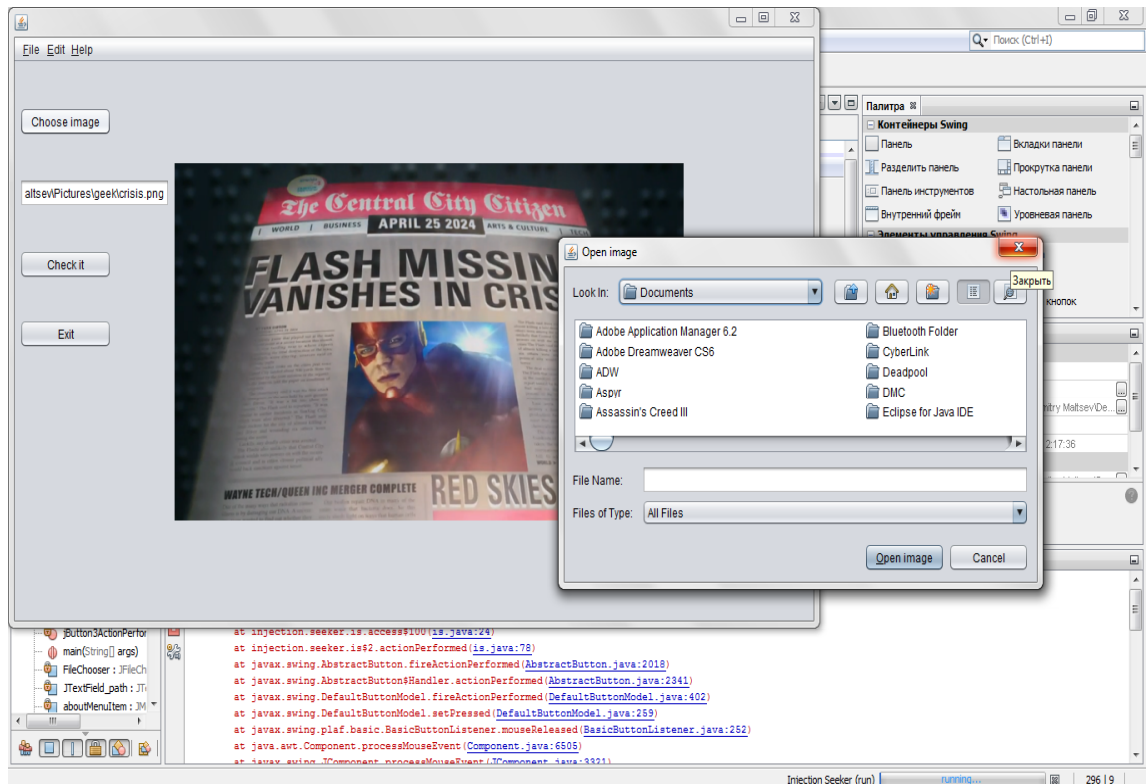


Figure 7. Final GUI with scripts

The algorithm work in the application is as follows: first, the user selects an image by pressing the “Choose image” button and then presses the ‘Check it’. The result appears in the empty space. Figure 8 shows what happens if user did not choose the image first and pressed “Check it”.

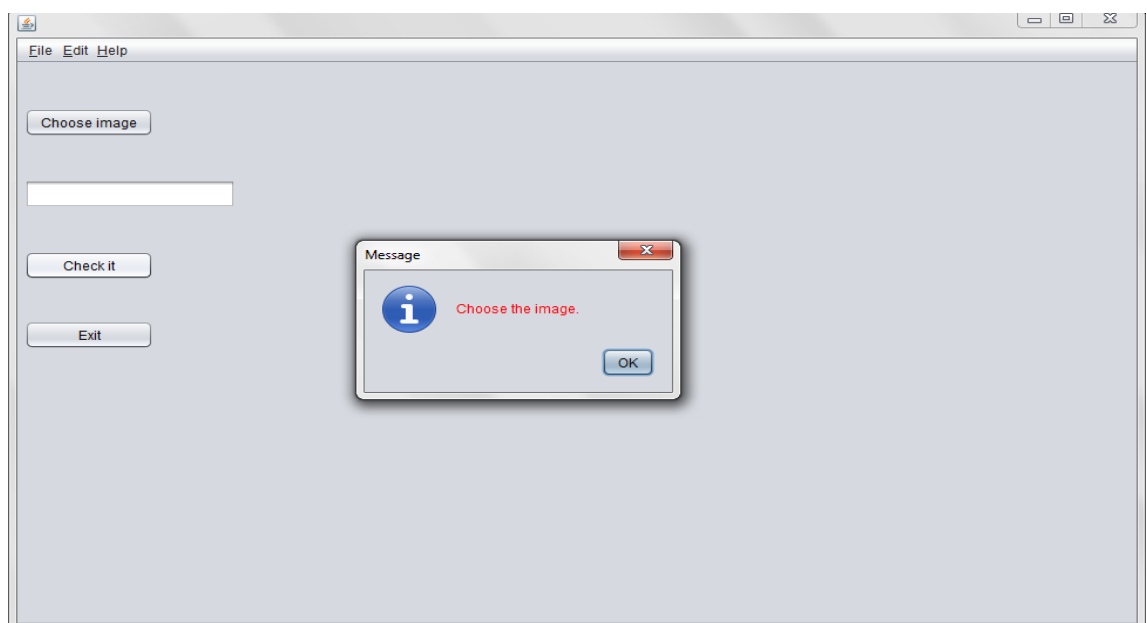


Figure 8. Dialog box “Choose the image”

Application notifies such mistake. It explains that user must first select an image through a dialog box "Choose the image".

In Figure 9 are shown the results after choosing the image and pressing 'Check it' button. In this case application has found that there is no malware inside the image.



Figure 9. Application did not find malware

In case of not detecting the malware, application displays a green message, informing that this image does not contain any malicious software. In addition list of the EXIF tags of the image is printed.

But another appearance of this result exists. Figure 10 reveals what happens, if application has found the virus.

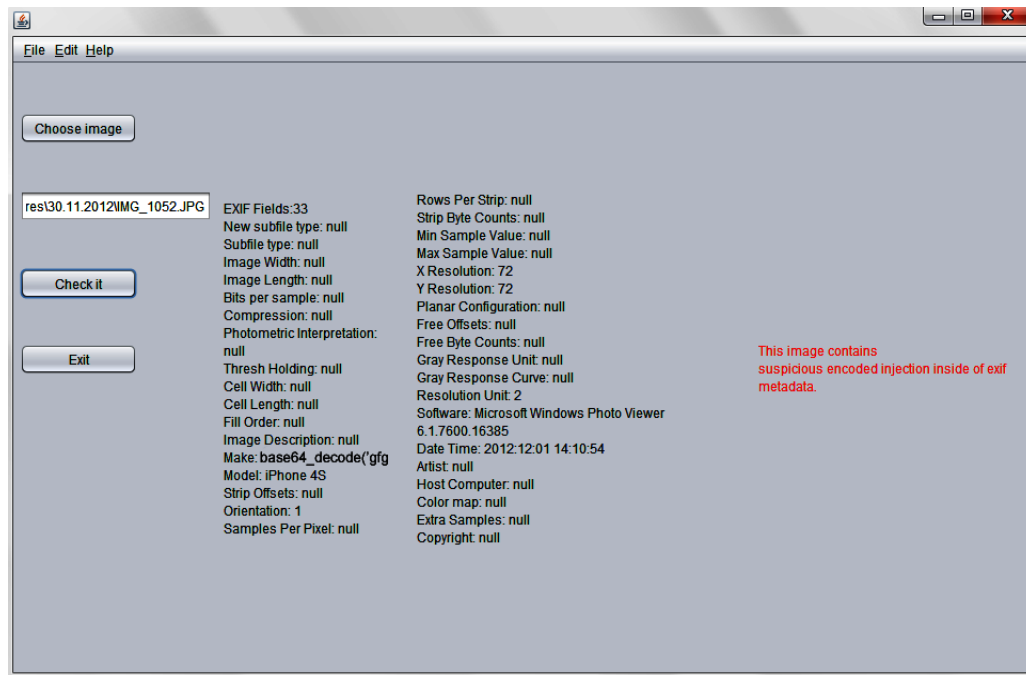


Figure 10. Application has found the virus

The list of EXIF tags appears and php-script will be displayed in one of them which decodes the virus. Application notifies the presence of malicious software, with red inscription. After these steps, the user might want to exit the application. Figure 11 shows dialog box, that asks user about final completion of working.

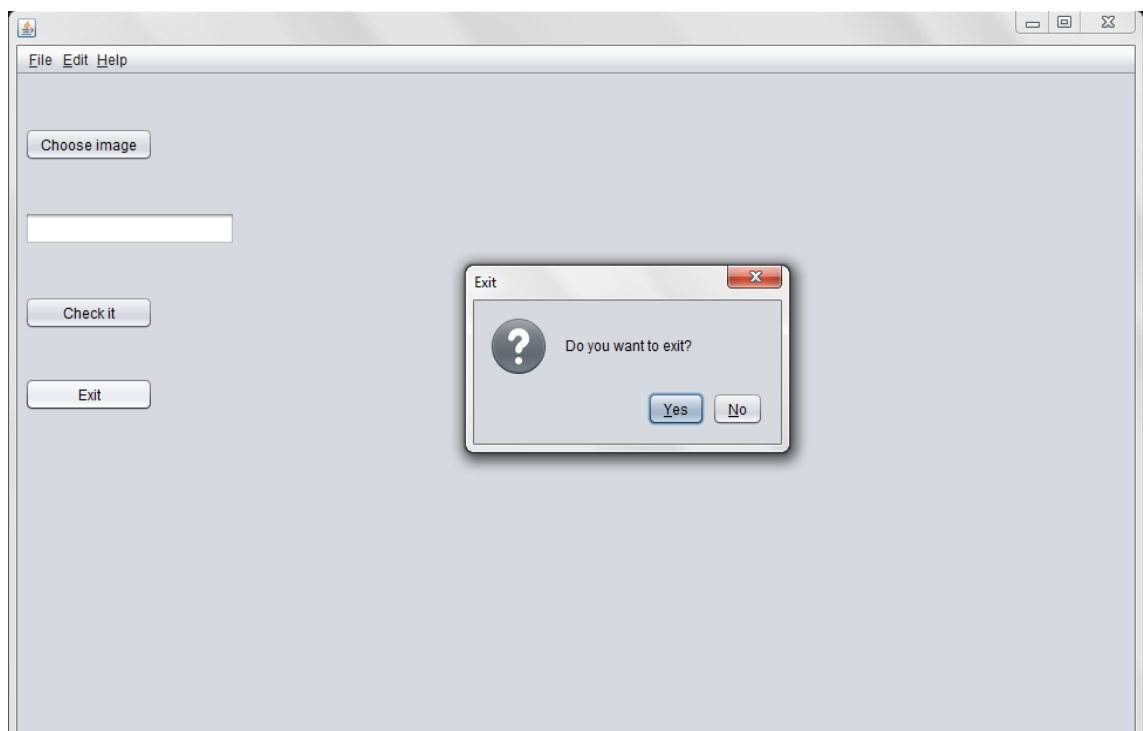


Figure 11. Application asks about exit

It is needed to press the "Exit" button if the user wants to exit the application. However, the exit from the program does not take place immediately, as there is a dialog box that asks the user whether he/she wants to exit or not. If not, dialog box is closing and the user continues to work with the program, and if "yes"-then the program closes.


4.4 Scripts

Scripts are events triggered in application. In general, the code contains three event handlers, i.e. 'Choose image', 'Check it' and 'Exit' for each declared button. Their description discusses in subchapters explaining the scripts for each button.

Each event contains in the Swing element type:

```
private void jButtonActionPerformed(java.awt.event.ActionEvent evt) {
}
```

Figure 12 presents all the libraries, using in the scripts. Screenshot was taken from IDE.



```
package injection.seeker;

import java.awt.image.BufferedImage;
import java.io.File;
import java.util.HashMap;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.io.Image;

/**
 *
 * @author Dmitry Maltsev
```

Figure 12. Libraries

Most of the libraries are default for NetBeans, because of built-in Swing. However, discussed in chapter 4.1 JavaXT library is used for images and imported as `javaxt.io.Image`.

4.4.1 “Choose image”

This event handler contains scripts for choosing the images. More than that the function is featured for displaying the images.

The area for images and results has two `JLabels`. `jLabel_image` displays the images and EXIF tags, and `jLabel_result` displays the message about finding/not finding the malware. The values of both are reset at the beginning of the script to make room for the new images and data. Following the reset, the `JFileChooser` called “chooser” is determined. The next step is writing a filter for chooser, to print only the images, during searching. The method `.setFileFilter` is used. Obtaining of the image is achieved with method `.getSelectedFile`. The method `.getScaledInstance` is used for printing out the image in a certain, specified size. All images are processed as buffered and, consequently, they are given the size of 500 pixels in length and 400 pixels in width. The whole script is shown in Appendix 1.

4.4.2 “Check it”

Scripts in this button execute the most important functions of the applications. They are responsible for extracting the most important metadata and scanning it for viruses.

The script, located at the beginning of handler, blanks `jLabel_image`. Script receives value of `(jTextField)` to the string “path”, where the path to the image is written. The next whole handler is completely encased into if-else structure. The condition implies that if a String “path” is empty, the user did not select an image, or the path to the file would be in `jTextField`. A dialog box "Select an image" appears in this case. Method `.isEmpty` is used.

Handler of “else” represents a script, which reads metadata from the image. This handler prints it out and searches the malware inside of the EXIF tags. It starts with reading the EXIF tags by `javaxt-core` library, using method `.getExifTags`. Then all of the tags are sent to the String “meta” with descriptions of them. In this String, HTML-tags are used, because of design purposes, due to they allow making breaks and tables.

The next stage is the searching mechanism. This script is built with method `indexOf()`. The idea is that script is searching for the index with the lines “base64_decode”, “<script>” and “<?php>”. The `indexOf(“base64_decode”)` is searching for the code that is responsible for decoding the encoded malware. The `indexOf(“<script>”)` and `indexOf(“<?php”)` are the typical “foolproof” if the creator of the virus has not attended to its encoding or decoding, error occurred prior to the time required to open or download the image. In the case of <script>, application is looking for a virus written in JavaScript. <?php > is using for malware written in PHP. If the index with these three lines is not found, the application prints the green text “This image does not contain any malicious codes/viruses”. If a virus is found, the application prints the red text “This image contains suspicious encoded injection inside of EXIF metadata”. The whole handler is shown in Appendix 2.

4.4.3 “Exit”

This handler uses simple `.YES_NO_OPTION` method. It allows to ask, if the user really wants to exit. If the answer is yes, the application closes, using function `System.exit(0)`. If the answer is no, the message will be removed and the user can continue working with the application.

5 TESTING

This chapter describes the process of testing. It contains the alleged work of the application and test process itself. Figure 13 presents a testing plan.

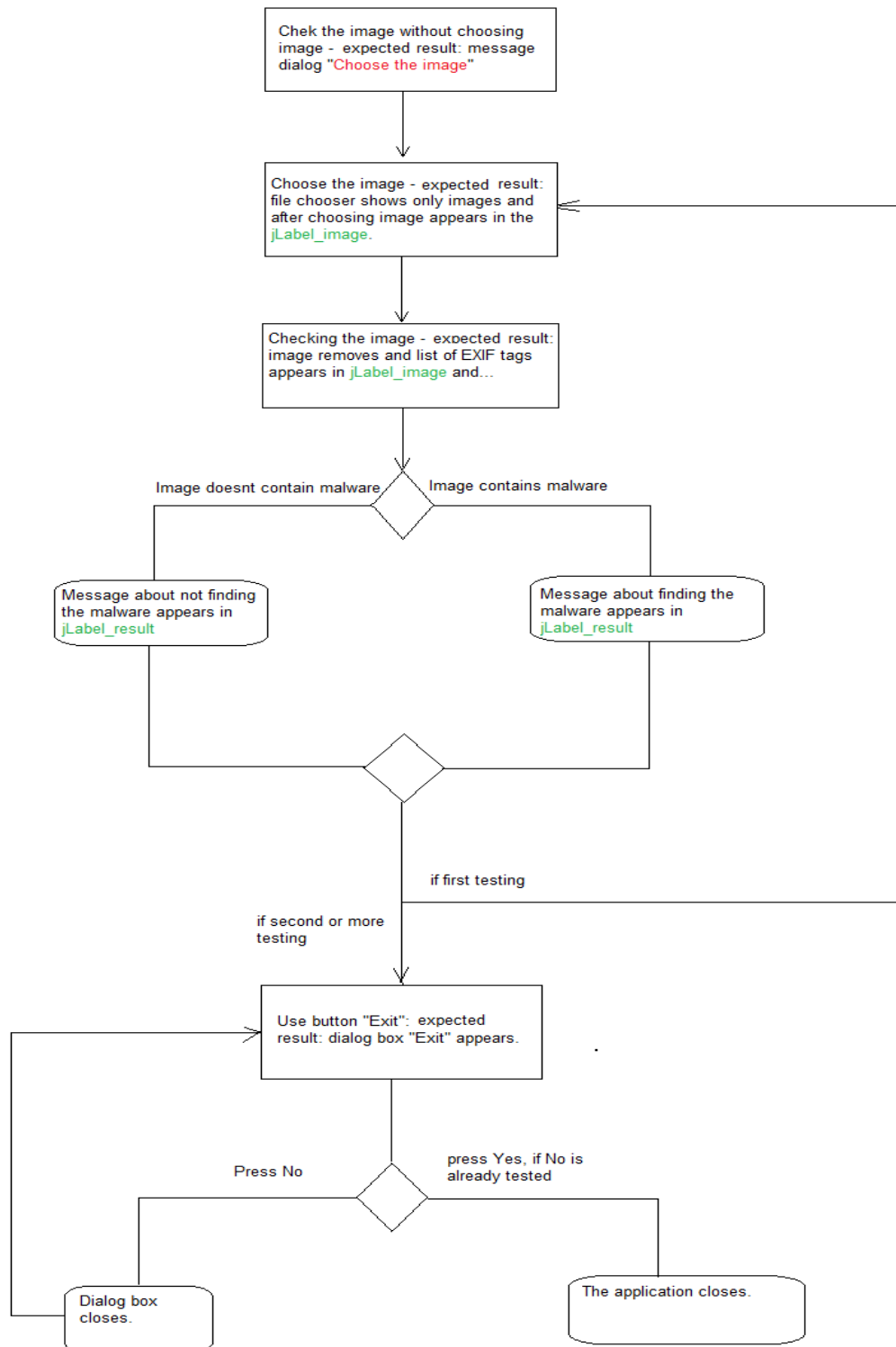


Figure 13. Testing plan

All results of application tests are placed in tables. Testing was performed according to the plan.

The first tested function is a notification that user must first select the image, by clicking on the "Check it" button without selected image or without prescribed path to the file. Test is shown in the Table 1.

Table 1. Notification "Choose the image"

| | |
|-----------------|---|
| Function. | Notification "Choose the image" |
| Test Procedure | <ol style="list-style-type: none"> 1. Run the application 2. Push the button "Check it" |
| Expected result | Appearing of the dialog box with red row "Choose the image" |
| Result | Appearing of the dialog box with red row "Choose the image" |
| Date | 26.10.2015 |

The second function is "Choosing the image". User clicks the button "Choose the image", searching for needed file and image prints out. Test is shown in the Table 2.

Table 2. Choosing the image and its printing

| | |
|-----------------|--|
| Function | Choosing the image and its printing |
| Test Procedure | <ol style="list-style-type: none"> 1. Push the button "Choose the image" 2. Choose some image 3. Press "Okay" |
| Expected result | Chosen image appears in jLabel_image field. File Chooser shows only image formats. |
| Result | File chooser showed only image formats, chosen image has appeared in jLabel_image field. |
| Date | 26.10.2015 |

The third function is "Checking the image". Because it implies to have two different results, such as "Malware was found" and "Malware was not found", it is necessary to complete at least two tests. Results of the tests represented in the Table 3 and Table 4. Table 3 shows testing results for the case, when malware was not found.

Table 3. Checking the image (malware was not found)

| | |
|-----------------|---|
| Function | Checking the image |
| Test procedure | <ol style="list-style-type: none"> 1. Choose the image without any malware 2. Press the button "Check it" |
| Expected result | Disappearing of the image in jLabel_image and printing of the EXIF-tags in the same field. Green message "This image does not contain any malicious codes/viruses." appears in the jLabel_result. |
| Result | The image disappeared, EXIF-tags appeared in the jLabel_image and green message appeared in the jLabel_result. |
| Date | 27.10.2015 |

Table 4 presents testing results for the case, when malware was found. This testing requires same procedures as shown in Table 3 and only result is different.

Table 4. Checking the image (malware was found)

| | |
|-----------------|--|
| Function | Checking the image |
| Test procedure | <ol style="list-style-type: none"> 1. Choose the image with malware 2. Press the button "Check it" |
| Expected result | Disappearing of the image in jLabel_image and printing of the EXIF-tags in the same field. Red message "This image contains suspicious encoded injection inside of EXIF metadata." appears in the jLabel_result. |
| Result | The image disappeared, EXIF-tags appeared in the jLabel_image and red message appeared in the jLabel_result. |
| Date | 27.10.2015 |

Next function is "Exit". This feature also contains two options - a dialog box asks whether user wants to exit or not. Due to this fact, testing is shown in the Table 5 and Table 6. Table 5 presents what happens if user chooses not to exit the application.

Table 5. Exit ("No")

| | |
|-----------------|--|
| Function | Exit (User chooses "No") |
| Test procedure | <ol style="list-style-type: none"> 1. Press the "Exit" button 2. Press "No" in dialog box, which asks "Do you want to exit?" |
| Expected result | The dialog box with message "Do you want to exit?" appears. After clicking to the "No" the dialog box disappears. |
| Result | The dialog box with message "Do you want to exit?" appeared and disappeared after clicking to the "No". |
| Date | 27.10.2015 |

Table 6 indicates the testing of the exit process too. However, in this case user chooses to exit the application.

Table 6. Exit ("Yes")

| | |
|-----------------|---|
| Function | Exit (User chooses "Yes") |
| Test procedure | <ol style="list-style-type: none"> 1. Press the "Exit" button 2. Press "Yes" in dialog box, which asks "Do you want to exit?" |
| Expected result | The dialog box with message "Do you want to exit?" appears. After clicking to the "Yes" the application closes. |
| Result | The dialog box with message "Do you want to exit?" appeared and application closed after clicking to the "Yes". |
| Date | 27.10.2015 |

All tests were completed successfully. Each function is tested during development and shows the results of the final testing in the table.

.

6 CONCLUSION

The objective of this thesis is to create an application that is able to scan the images and find the hidden malware in them. The thesis also provides information on the main methods of delivery of the viruses, as well as methods of protection.

This study has provided three research questions:

1. How does malware get to user's computer and what kind of counter measures could be performed by the user against malware?
2. How does malware get into different type of files?
3. How do images get infected with malicious software and how it is possible to protect them?

These questions were answered in this thesis.

Concerning the first question, it can be stated that malicious software may get into the system very easy. Most of malware is delivered to the information systems through the web browsers. More often malicious software is hidden behind links to the various resources, advertising and files. Smaller part of viruses is spreading using the e-mail systems. Additionally, there are various methods of protection, which, nevertheless, do not give a 100% guarantee, to ensure that systems will be protected. However, the installation of anti-virus, disabling the use of macros and extensions for browsers which removes advertisements could significantly reduce the risk of infecting computer with viruses. Therefore, a significant part of the protection is built on the care of the user.

Answer to the second question reveals that malicious software is adjudged differently at various types of files, and it can penetrate into all possible types of files: video, audio and text files. However, many methods for video and audio files, such as fuzzing, are quite complex for inserting into these types of files and do not cause great concern amongst users. Nonetheless, snag with .exe

extension still can be dangerous. There is still a danger of macros that carry malicious software at the level of a text file.

Answer to the third question states, that malicious software in the images is really a big enough threat for users as it is not always possible to identify whether there is the malicious code somewhere in the depths of metadata. However, there are several ways to detect the virus inside of the picture. One of these methods is developing of the scanning application.

Writing the program took some efforts because I did not have enough knowledge in this area. This fact explains the problems at an early stage of development, when there were troubles with choosing the programming language and IDE. However, later, with the definition of Java as a programming language, everything returned to normal working process, the following technical problems were solved relatively easily.

In the present form of application, which is not very complex and does not contain many functions, it is worth more as an OSS. The application could be a base for potentially stronger anti-virus program or an additional part to already existing projects. However, the application is also suitable for commercial use, since it is possible to easily adapt to other operating systems and mobile applications, and thus possibly to sell / distribute free but with advertising on online distribution platforms.

As for users, this application can also serve as a simple and reliable way to detect malware without the need for knowledge about them. This application is also convenient because even if the user is offline, he / she will be able to check suspicious images.

REFERENCES

NetBeans Tutorial 2014. CIS 2087: Unit 8 NetBeans GUI Tutorial Part 1.

Referenced 28.11.2015

<http://1080.plus/m7kJbyhxy3l.video>

Barnett, R. 2013. Hiding Webshell Backdoor Code in Image Files. Referenced 28.10.2015.

<https://www.trustwave.com/Resources/SpiderLabs-Blog/Hiding-Webshell-Backdoor-Code-in-Image-Files/>.

Denscombe, M. 2010. The good research guide: for small scale research projects. 4th ed. Maidenhead: McGraw-Hill Open University Press. 300.72

DEN & e-book.

JavaXT 2015. Image IO. Referenced 12.11.2015.

<http://www.javaxt.com/javaxt-core/javaxt.io.Image/>.

Kaspersky Lab 2015. What is a Computer Virus or a Computer Worm?

Referenced 29.10.2015.

<http://www.kaspersky.com/internet-security-center/threats/viruses-worms>.

Kaspersky Lab 2008. Kaspersky Lab reports new worm that infects audio files.

Referenced 11.10.2015.

<http://www.kaspersky.com/news?id=207575664>.

Microsoft 2015. Tips for protecting your computer from viruses. Referenced 28.10.2015

<http://windows.microsoft.com/en-us/windows7/tips-for-protecting-your-computer-from-viruses>.

Mosuela L. 2015. New tricks of Macro Malware. Referenced 11.11.2015.

<https://blog.cyren.com/articles/new-tricks-of-macro-malware.html>.

OPSWAT, Inc. 2014. Can A Video File Contain A Virus?

Referenced 12.12.2015.

<https://www.opswat.com/blog/can-video-file-contain-virus#reference-3>.

PaloAlto Networks 2013. The Modern Malware Review; Analysis of New and Evasive Malware in Live Enterprise Networks 1st Edition. 2013.

<http://media.paloaltonetworks.com/documents/The-Modern-Malware-Review-March-2013.pdf>.

Steam 2015, Steam Guard. Referenced 28.10.2015.

https://support.steampowered.com/kb_article.php?ref=4020-ALZM-5519&l=english.

Support.com, Inc.,2013. Malware, Spyware, Virus, Worm, etc... What's the Difference? Referenced 19.11.2015.

<http://www.superantispyware.com/blog/2013/03/malware-spyware-virus-worm-etc-whats-the-difference/>.

Thiel D. 2008. Exposing Vulnerabilities in Media Software. Referenced 10.11.2015.

<http://www.blackhat.com/presentations/bh-europe-08/Thiel/Whitepaper/bh-eu-08-thiel-WP.pdf>.

Ziff Davis, LLC. PCMag Digital Group 2015. Definition of double extension. Referenced 30.10.2015.

<http://www.pcmag.com/encyclopedia/term/41929/double-extension>.

APPENDICES

| | |
|------------|--------------------|
| Appendix 1 | jButton1 (handler) |
| Appendix 2 | jButton2 (handler) |
| Appendix 3 | jButton3 (handler) |

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    jLabel_image.setText(null); //clear the area from previous EXIF-
tags/images

    jLabel_result.setText(null); //clear the area from previous results

    JFileChooser chooser = new JFileChooser();

    FileFilter imageFilter = new FileNameExtensionFilter(

        "Image files", ImageIO.getReaderFileSuffixes());

//Attaching Filter to JFileChooser

chooser.setFileFilter(imageFilter);

//Displaying Filechooser

    chooser.showDialog(null, "Open image");

    File file = chooser.getSelectedFile();

    String filename = file.getAbsolutePath();

    JTextField_path.setText(filename);

//setting the size of images and displaying

    ImageIcon imageIcon = new ImageIcon(new
    ImageIcon(filename).getImage().getScaledInstance(500, 400,
    BufferedImage.SCALE_SMOOTH));

    jLabel_image.setIcon(imageIcon);

}

```

```

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

    jLabel_image.setIcon(null);

    String path = JTextField_path.getText(); //get image path

    if (path.isEmpty()){

        JOptionPane.showMessageDialog(null, "<html><font color =
red>Choose the image.</html>");

    }

    else{

        Image image = new javaxt.io.Image(path);

        HashMap<Integer, Object> exif = image.getExifTags();

        String meta = ("<html><table><tr><td>EXIF Fields:" + exif.size() + "<br>"

            + "New subfile type: " + exif.get(0x00FE) + "<br>"

            + "Subfile type: " + exif.get(0x00FF) + "<br>"

            + "Image Width: " + exif.get(0x0100) + "<br>"

            + "Image Length: " + exif.get(0x0101) + "<br>"

            + "Bits per sample: " + exif.get(0x0102) + "<br>"

            + "Compression: " + exif.get(0x0103) + "<br>"

            + "Photometric Interpretation: " + exif.get(0x0106) + "<br>"

            + "Thresh Holding: " + exif.get(0x0107) + "<br>"

```

+ "Cell Width: " + exif.get(0x0108) + "
"

+ "Cell Length: " + exif.get(0x0109) + "
"

+ "Fill Order: " + exif.get(0x010A) + "
"

+ "Image Description: " + exif.get(0x010E) + "
"

+ "Make: " + exif.get(0x010F) + "
"

+ "Model: " + exif.get(0x0110) + "
"

+ "Strip Offsets: " + exif.get(0x0111) + "
"

+ "Orientation: " + exif.get(0x0112) + "
"

+ "Samples Per Pixel: " + exif.get(0x0115) + "</td>
"

+ "<td>Rows Per Strip: " + exif.get(0x0116) + "
"

+ "Strip Byte Counts: " + exif.get(0x0117) + "
"

+ "Min Sample Value: " + exif.get(0x0118) + "
"

+ "Max Sample Value: " + exif.get(0x0119) + "
"

+ "X Resolution: " + exif.get(0x011A) + "
"

+ "Y Resolution: " + exif.get(0x011B) + "
"

+ "Planar Configuration: " + exif.get(0x011C) + "
"

+ "Free Offsets: " + exif.get(0x0120) + "
"

+ "Free Byte Counts: " + exif.get(0x0121) + "
"

+ "Gray Response Unit: " + exif.get(0x0122) + "
"

+ "Gray Response Curve: " + exif.get(0x0123) + "
"

+ "Resolution Unit: " + exif.get(0x0128) + "
"

```

        + "Software: " + exif.get(0x0131) + "<br>"

        + "Date Time: " + exif.get(0x0132) + "<br>"

        + "Artist: " + exif.get(0x013B) + "<br>"

        + "Host Computer: " + exif.get(0x013C) + "<br>"

        + "Color map: " + exif.get(0x0140) + "<br>"

        + "Extra Samples: " + exif.get(0x0152) + "<br>"

        + "Copyright: " + exif.get(0x8298) + "</td></tr></table></html>");

int decode = meta.indexOf("base64_decode");

int jscript = meta.indexOf("<script>");

int php = meta.indexOf("<?php>");

if((decode == - 1) && (jscript == - 1) && (php == -1)){

    String yeah = ("<html><font color=green>This image does not contain
<br> any malicious codes/viruses.</html>");

    jLabel_result.setText(yeah);

}else{

    String no = ("<html><font color=red>This image contains <br>suspicious
encoded injection inside of EXIF metadata.</html>");

    jLabel_result.setText(no);

}

jLabel_image.setText(meta);

}

}

```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  
    int question = JOptionPane.showConfirmDialog(null, "Do you want to exit?",  
    "Exit", JOptionPane.YES_NO_OPTION);  
    if (question == JOptionPane.YES_OPTION) {  
        System.exit(0);  
    }  
  
}
```