Criminals, ATMs and a cup of coffee

SL securelist.com/criminals-atms-and-a-cup-of-coffee/91406/

Authors

Expert Konstantin Zykov

In spring 2019, we discovered a new ATM malware sample written in Java that was uploaded to a multiscanner service from Mexico and later from Colombia. After a brief analysis, it became clear that the malware, which we call ATMJaDi, can cash out ATMs. However, it doesn't use the standard XFS, JXFS or CSC libraries. Instead, it uses the victim bank's ATM software Java proprietary classes: meaning the malware will only work on a small subset of ATMs. It makes this malware very targeted.

Kaspersky products detect the sample as Trojan.Java.Agent.rs

Technical Details

First, as with most other ATM malware, the attackers must find a way to install the malware on the target ATMs. The malware can't be controlled via the ATM keyboard or touchscreen, because it runs a self-crafted HTTP server web interface for its purpose. So the criminals

must have network access to the target ATM. This makes us believe that the criminals have compromised the bank's infrastructure to gain access to the network that the ATMs are connected to.

Once installed and executed, the malware, in the form of a Java archive file called "INJX_PURE.jar", looks for the process that controls the ATM and injects itself into it, giving it control of the legitimate ATM process. After injection, the malware prints a message on the terminal simultaneously in several languages: Russian, Portuguese, Spanish and Chinese. However, all the other messages or strings used by the malware are in English. The different language phrases shown in the output can be translated into English as "Freedom and glory". This is followed by the additional Russian message "отдельный", which means "separate". We believe this might be a false flag, because native Russian speakers would never use this word in this context.

- Свобода и слава
- Liberdade e glória
- Libertad y gloria
- 自由与荣耀
- отдельный

Next, an HTTP server is started that accepts commands using predefined URL paths. They are:

- **/d** to dispense or to get the ATM cassette to carry out actions (the proper action is determined by the passed parameters);
- **/eva** to evaluate (run) user-supplied code on the victim ATM;
- **/mgr** for the manager, which gives criminals access to a list of all running classes for the attached Java virtual machine, so that they can call any function they desire, supplying the arguments if needed;
- **/core** allows the criminals to load a specified *.jar* file from the victim file system;
- **/root** path accepts a POST request body and passes it as a shell command to cmd.exe, returning the resulting output.

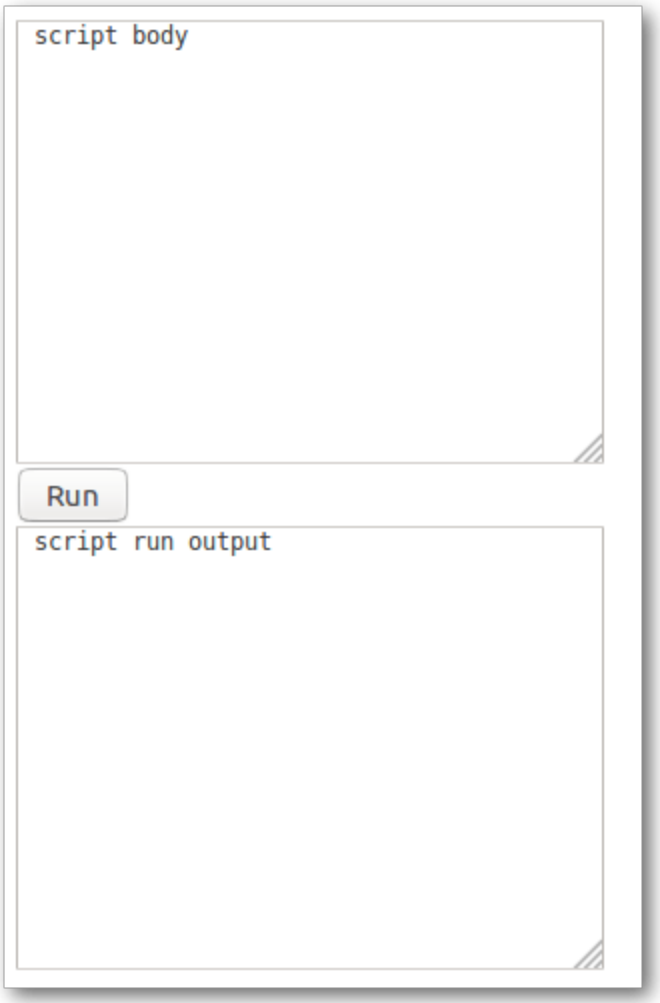
The dispensing and "run a shell" path do not have an interface page with forms and buttons, but instead only accept pre-prepared HTTP POST requests and print the raw text results to a page, omitting HTML tags. So, in the case of the dispensing request, the malware response will be the 'ok' string. The "get cash units information" request will be followed by a string describing the ATMs cash units status (see the example below).

1:1000;5:700;10:100;20:30;

This string consists of four groups, each group separated with a semicolon. It is a list that corresponds to the ATM cash cassette and consists of two values, separated by a colon: the denomination and the actual number of bills in the cassette. In the example above, the first

cassette has 1000 banknotes of denomination 1, 700 banknotes of denomination 5, etc.

Other than the “run a shell”, “dispense” and “get cash unit”, the “eva”, “mgr” and “core” paths have interface pages. Below is a screenshot of the evaluation page:



/eva path interface example screenshot

It allows the criminals to paste and run any JavaScript code on a victim ATM and see what it returns. Why JavaScript? Because Java allows the use of external engines, and the criminals used a JavaScript one. Below is the function that the malware uses to run the passed JavaScript code.

```
public static String runjs(String script)
{
    ScriptEngineManager manager = new ScriptEngineManager(HTTPServ.class.getClassLoader());
    ScriptEngine engine = manager.getEngineByExtension("js");
    StringWriter sw = new StringWriter();
    PrintWriter pw = new PrintWriter(sw);
    engine.getContext().setWriter(pw);
    engine.getContext().setErrorWriter(pw);
    try
    {
        engine.eval(script);
    }
    catch (ScriptException e)
    {
        e.printStackTrace(pw);
    }
    String outx = sw.getBuffer().toString();

    return outx;
}
```

Malware sample code that can evaluate JavaScript

Conclusions

The targeted nature of ATMJaDi shows that the criminals studied the victim very well before writing their malware. It's clear that they must have had access to an ATM where the custom Java classes were running and most likely to the Java program source code as well.

Also, the way the malware can be controlled shows that the criminals planned to gain network access to the infected ATM, most likely through the bank's internal network.

What banks can do to prevent these types of attacks:

- Set up special anti-targeted attack solutions such as KATA (Kaspersky Anti Targeted Attack Platform) to protect the bank's network and other solutions to protect ATMs from malware;
- ATM file allowlisting;
- The bank ATM network has to be isolated and access to it must be highly restricted;

This is not a complete list of actions: the issue of information security requires constant attention and action.

Criminals, ATMs and a cup of coffee
