

2016



MICT1 – Exercise Week 5

Group1

Zuyd University

19-Mar-16

MICT1 – Exercise Week 5

Reverse Engineering Data – Exercise week 5

Group	Group 1	
Students	Delano Cörvers	(1306669)
	Davy Heutmekers	(1309730)
	Rik Kierkels	(1354442)
Module	MICT1 – Reverse Engineering Data	
Assignment	Exercise – Week 5	
School year	2015 – 2016	

Table of content

1	Our understanding and ideas	4
2	Our attempt at reverse engineering the Merlok 2.0 application	6
2.1	Decompiling the APK	6
2.1.1	APK & Cache retrieval	6
2.1.2	APK & Cache structure	6
2.1.3	Dynamic Link Libraries (DDL)	10
2.1.4	Assets	16
2.1.5	Classes.dex	16
2.1.6	Datafiles	17
2.1.7	QR Code scanner	19
3	Discussion	20
4	Conclusion	21
5	Tools	21

1 Our understanding and ideas

In order to understand how the android application scans the shields and other QR codes, we have to take a look at what scanning methods are used. The app uses traditional QR codes in order to share powers with friends and their own custom codes to scan the shields. It is also possible to use QR codes to instantly level up a shield / power to the maximum level, which is level 5.

We started our research with the process of scanning normal shields. To figure out how this process actually works, we downloaded a few samples of existing shields from ShieldPowerList.com. We were able to scan each sample multiple times as we had three Android phones to test with. First, we scanned the original image. This worked as expected. The next step was to clear out the background image of the shield, so only the border and 'bits' remained. This was correctly recognized and processed by the app. Next we changed the color of the border and bits to see if the color affected the scanning process. We believed this was a possible scenario as different characters in the game (Clay, Lance, Axl) had different color on their shields. This, however, made no difference to the app.

We also tried to change the shape of the shield, by deleting the bottom portion of the shield. The app could not recognize the modified shape. We also tried to modify the bits on the shield, to try to create our own combinations, or to verify if certain bits affected the scanning process. After adding and removing multiple bits, some shields were recognized and some weren't. It also important to note that the modified shield was recognized as the same shield as the original shield.

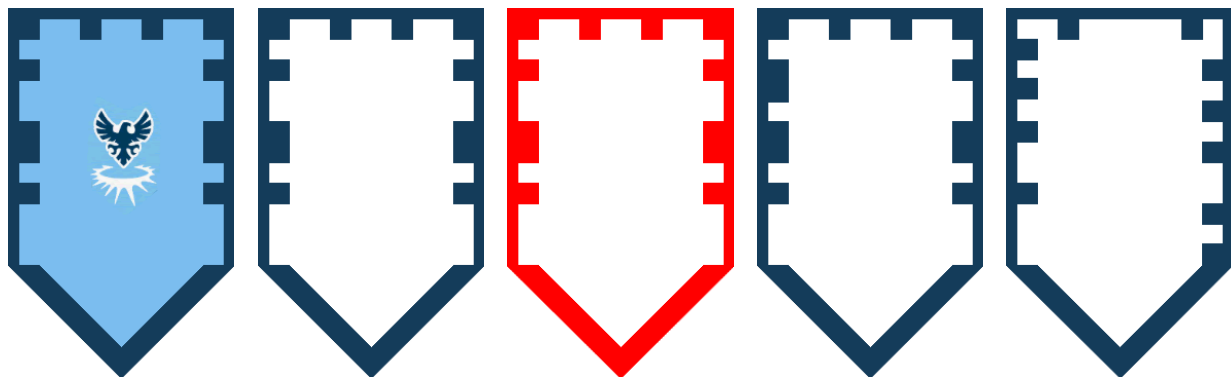


Figure 1: Original shield.

Figure 2: Removed background.

Figure 3: Changed border color.

Figure 4: Added bits.

Figure 5: Added and removed bits

Now that we know what affects and what doesn't affect the scanning process, in terms of how the shields are displayed or constructed, we can start to figure out how these shields are actually being scanned. Since this seems to be a custom scanning mechanism from this specific app, we believe that each extended block of the border resembles a bit. A block that is the same color as the border is considered a '1' bit. The other one is a '0' bit. However, we were not able to find any data that supports our theory. We started to count the total amount of places where a bit could be placed, which is 32. The average shield has '1' bits ranging from 11 to 18 positions out of the 32. There are never more than 3 bits of the same kind in a row (for example: 0001 is possible; 00001 is not possible).

We thought the bits in the shield are counted from left to right, starting in the lower left corner and ending in the lower right corner.



Figure 6: Sample shield with arrows.

Following this method the shield above would result in the following binary sequence: 00010010001001001101001010100101. This is a 32 bit sequence.

We were searching on the internet for more information about these shields and ended up on a Reddit page where someone was talking about his own shield generator which was web based. We checked out his tool and used it to confirm our method using the provided sequence and other sequences we typed out based on the ShieldPowerList.

Our idea was to try to reverse engineer the APK file through the (de)compiled Unity libraries and the Game Cache using various different tools.

2 Our attempt at reverse engineering the Merlok 2.0 application

In this chapter we start our reverse engineering process to try to find the missing shields.

2.1 Decompiling the APK

2.1.1 APK & Cache retrieval

The Lego Nexa Knights Merlok 2.0 APK and OBB (Game cache) was downloadable from a few links, we specifically looked for version 1.1.1 which is the latest version available in the Google Play Store. The APK is roughly 24 MB in size and the OBB game cache is roughly 350 MB. The APK is usually downloaded from the Google Play Store and the OBB game cache is downloaded the first time the application is booted.

2.1.2 APK & Cache structure

APK files and OBB caches can be unzipped using WinRAR or a similar program. The retrieved APK file had the following file structure.








 assets	15-3-2016 17:27	Bestandsmap	
 lib	15-3-2016 17:27	Bestandsmap	
 META-INF	15-3-2016 17:27	Bestandsmap	
 res	15-3-2016 17:27	Bestandsmap	
 AndroidManifest.xml	16-12-2015 22:34	XML-document	8 kB
 classes.dex	16-12-2015 22:32	DEX-bestand	3.895 kB
 resources.arsc	16-12-2015 22:32	ARSC-bestand	1.945 kB

Figure 7: Merlok 2.0 APK structure

The assets folder contains the shared assets used by the application, it only contains the fundamental assets, almost all of the textures and sound files are stored in the Cache. The Assets folder also contains all managed assets, these are .dll files that contain C# code.

















 Managed	15-3-2016 17:27	Bestandsmap	
 Resources	15-3-2016 17:27	Bestandsmap	
 mainData	16-12-2015 22:34	Bestand	3.937 kB
 settings.xml	16-12-2015 22:34	XML-document	1 kB
 sharedassets0.assets.split0	16-12-2015 22:34	SPLIT0-bestand	1.024 kB
 sharedassets0.assets.split1	16-12-2015 22:34	SPLIT1-bestand	1.024 kB
 sharedassets0.assets.split2	16-12-2015 22:34	SPLIT2-bestand	1.024 kB
 sharedassets0.assets.split3	16-12-2015 22:34	SPLIT3-bestand	1.024 kB
 sharedassets0.assets.split4	16-12-2015 22:34	SPLIT4-bestand	1.024 kB
 sharedassets0.assets.split5	16-12-2015 22:34	SPLIT5-bestand	1.024 kB
 sharedassets0.assets.split6	16-12-2015 22:34	SPLIT6-bestand	1.024 kB
 sharedassets0.assets.split7	16-12-2015 22:34	SPLIT7-bestand	1.024 kB
 sharedassets0.assets.split8	16-12-2015 22:34	SPLIT8-bestand	1.024 kB
 sharedassets0.assets.split9	16-12-2015 22:34	SPLIT9-bestand	202 kB
 splash.png	16-12-2015 22:29	PNG-bestand	766 kB
 unity default resources	16-12-2015 22:34	Bestand	1.616 kB

Figure 8: Merlok 2.0 APK Assets






























 ArabicSupport.dll	16-12-2015 22:34	Toepassingsuitbre...	13 kB
 Assembly-CSharp.dll	16-12-2015 22:34	Toepassingsuitbre...	2.253 kB
 Assembly-CSharp-firstpass.dll	16-12-2015 22:34	Toepassingsuitbre...	39 kB
 BestHTTP.dll	16-12-2015 22:34	Toepassingsuitbre...	942 kB
 FrimaEngine.AtlasMaker.dll	16-12-2015 22:34	Toepassingsuitbre...	36 kB
 FrimaEngine.AudioManager.dll	16-12-2015 22:34	Toepassingsuitbre...	21 kB
 FrimaEngine.CheatManager.dll	16-12-2015 22:34	Toepassingsuitbre...	21 kB
 FrimaEngine.CommandQueue.dll	16-12-2015 22:34	Toepassingsuitbre...	8 kB
 FrimaEngine.CommonGloo.dll	16-12-2015 22:34	Toepassingsuitbre...	7 kB
 FrimaEngine.CommonInterface.dll	16-12-2015 22:34	Toepassingsuitbre...	6 kB
 FrimaEngine.Console.dll	16-12-2015 22:34	Toepassingsuitbre...	31 kB
 FrimaEngine.Core.dll	16-12-2015 22:34	Toepassingsuitbre...	47 kB
 FrimaEngine.Debug.dll	16-12-2015 22:34	Toepassingsuitbre...	13 kB
 FrimaEngine.EnumMapper.dll	16-12-2015 22:34	Toepassingsuitbre...	7 kB
 FrimaEngine.Log4Net.dll	16-12-2015 22:34	Toepassingsuitbre...	51 kB
 FrimaEngine.ManagerLauncher.dll	16-12-2015 22:34	Toepassingsuitbre...	9 kB
 FrimaEngine.MultiSprite.dll	16-12-2015 22:34	Toepassingsuitbre...	52 kB
 FrimaEngine.PoolManager.dll	16-12-2015 22:34	Toepassingsuitbre...	27 kB
 FrimaEngine.ProfileManager.dll	16-12-2015 22:34	Toepassingsuitbre...	27 kB
 FrimaEngine.Profiling.dll	16-12-2015 22:34	Toepassingsuitbre...	20 kB
 FrimaEngine.QualityManager.dll	16-12-2015 22:34	Toepassingsuitbre...	18 kB
 FrimaEngine.RData.dll	16-12-2015 22:34	Toepassingsuitbre...	20 kB
 FrimaEngine.ResourcesLoader.dll	16-12-2015 22:34	Toepassingsuitbre...	42 kB
 FrimaEngine.ScreenManager.dll	16-12-2015 22:34	Toepassingsuitbre...	47 kB
 FrimaEngine.ScreenResolution.dll	16-12-2015 22:34	Toepassingsuitbre...	10 kB
 FrimaEngine.UIExtensions.dll	16-12-2015 22:34	Toepassingsuitbre...	8 kB
 FrimaEngine.VirtualScrollList.dll	16-12-2015 22:34	Toepassingsuitbre...	80 kB
 FrimaTools.dll	16-12-2015 22:34	Toepassingsuitbre...	37 kB
 log4net.dll	16-12-2015 22:34	Toepassingsuitbre...	157 kB

Figure 9: Merlok 2.0 APK DLL's

The lib folder contains the libraries for plugins used by the application. The application uses a number of plugins namely;

Vuforia

An augmented reality plugin that takes care of the scanning process and is vital to this report.

QRcode

A QR code scanner used to scan shared powers.

Crittercism

A logging tool that tracks users interaction and behavior.

The META-INF folder contains certificates that are created when the application is signed and build by the developers. The res folder contains icons for different DPI's. The classes.dex file contains JavaScript classes and is discusses in a later chapter.

The OBB game cache contains 2 folders. Bin and QCAR, the bin folder contains roughly 8000 assets files that contains textures and sound files used by the application.


 bin	14-3-2016 23:53	Bestandsmap
 QCAR	15-3-2016 17:24	Bestandsmap

Figure 10: Merlok 2.0 Cache

The QCAR folder contains 2 files: shield.xml and Shields.dat and Shields.xml. The shields.xml file is a reference to the shields.dat file. When opened, the shields.dat file contains code similar to a PKZIP file. We changed the file extension of Shields.dat to Shields.zip and used PKZIP to open the zip file. It contained a file named; config.info. Config.info contains information that is directly linked to scanning a shield.

It contains the coordinates that the scanner uses to identify the bit blocks of a shield and it contains the coordinates of the shield border. It also contains a large list of 8 byte blocks that we suspect to be hashed or encrypted shield border ID's.


```
<?xml version="1.0" encoding="UTF-8"?>
<QCARInfo version="1.5" xsi:noNamespaceSchemaLocation="Vuforia_Dataset_info_4.3.xsd" xmlns:xsi="http
  <TargetSet version="4.3">
    <ContourMarker targetId="9dae5ac0df7311e488300800200c9a66" name="shields" toolVersion="4.3.2
      <Shape contour-corners="M -15.591,15.272 -15.589,-19.574 15.589,-19.574 15.591,15.272 0,
        bit-locations="M -12.373,13.748 -12.373,10.998 -12.373,8.249 -12.373,5.499 -12.373,2.75
        -12.373,-5.499 -12.373,-8.249 -12.373,-10.998 -12.373,-13.748 -12.373,-16.498 -9.624,-16
        1.375,-16.498 4.124,-16.498 6.874,-16.498 9.624,-16.498 12.373,-16.498 12.373,-13.748 12
        12.373,2.75 12.373,5.499 12.373,8.249 12.373,10.998 12.373,13.748" bit-radius="0.97" />
      <Code format="eh" blockLength="32" messageLength="26" minDistance="4" alphabetSize="2">
        <Restrictions>
          <![CDATA[0x0167e6b2 0x02881172 0x02381c72 0x0203c072 0x038bd1f2 0x033bdcf2 0x03c
        </Restrictions>
        <Exceptions>
          <![CDATA[0x06018060 0xa8118815 0x80100801]]>
        </Exceptions>
      </Code>
    </ContourMarker>
  </TargetSet>
</QCARInfo>
```

Figure 11: Config.io file with possible shieldborderID's

Based on the XML file above, we can reverse engineer the shape based on the contour-corners and bit-location arrays. Using this we were able to create the following sketch, which could be used in the future to create our own generator for all power shields. This XML file also confirms our theory of the bit locations (left to right) and the total amount of bits per shield.

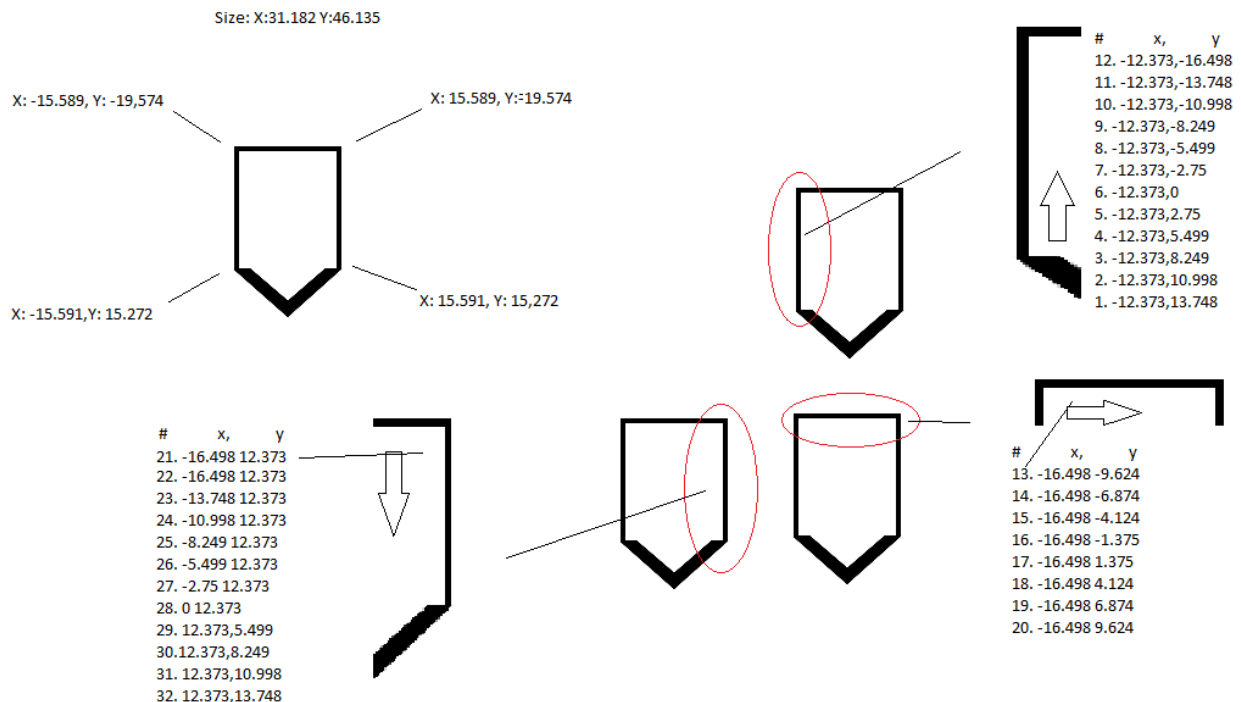


Figure 12: Sketch with all coordinates.

2.1.3 Dynamic Link Libraries (DDL)

The dll files contain all of the source code of the application, FrimaEngine dll's are code used by a part of the game engine. Most dll's are not very interesting, Assembly-CSharp.dll however contains all of the code written by the Lego developers. This includes the code that fires when a user scans a shield.

We used a .NET decompiler named JetBrains DotPeek to decompile the .dll files, inside the Assembly-CSharp.dll we found a list of all the powers and we got an understanding of how the scanning process works.

Analyzing the C# Assembly DLL using dotPeek.

As said before the C# Assembly DLL contains a lot of useful classes, like each shield and the data that pairs with the shield. This information, however, is only valuable for in-game situations such as the power, damage, range, etc. These shield classes did not contain any links to the actual data that was used to validate the shields.

We also found a file called ShieldScanner.cs and ShieldScannerUI.cs. The ShieldScanner C# Class contains most of the functionality regarding scanning shields and QR codes. There are two different scanning processes. The first one is for actual QR codes that friends can share with each other. To do this, the app uses a QR extension (xzing.dll). The class responsible for this process, generates a new QR code and embeds a decimal value in the code. The app can decode the QR and read out the decimal value and give the receiving user the corresponding shield and power. The decimal value converted to binary uses the same 32 bit sequence like the shields use. This process is not useful for our case, but it is worth mentioning regardless.

In the ShieldScanner.cs has a function called 'UpdateReadyToScanState()'. This function is called when a shield has been detected and is recognized as a valid shield (blue crosshair). When a shield is actually detected it will be checked to see if it is a 'PlainShield' or not. A 'PlainShield' only contains the border and four '1' bits in each corner (just like the shield samples on the lego boxes). To refer back to the recovered XML file, one exception in that XML file is a 'PlainShield'. If a PlainShield is detected it will cause an error regarding an unknown border. If it is not a PlainShield the app proceeds to the 'ValidateIfPowerAvailable' method.

```
private void UpdateReadyToScanState()
{
    if (ProgressionManager.PowerProgressionManager.IsPopupPoppedUp || MonoSingleton<VideoPlaybackManager>.Instance.IsPlaying || this.m_ErrorPopupActive)
        return;
    this.UpdateTrackingNearestShields();
    for (int index = 0; index < this.mDetectedShields.Count && index < 3; ++index)
    {
        ShieldTrackerFeedback trackedShieldData = this.mScannerUI.GetShieldTrackerFromTrackedShieldData(this.mDetectedShields[index].m_TrackedShield);
        Vector2 targetFrameSize = this.mScannerUI.GetTargetFrameSize();
        bool flag = false;
        if ((UnityEngine.Object) trackedShieldData != (UnityEngine.Object) null)
        {
            float aPixelDistance;
            if (this.IsDetectedShieldInTargetPosition(trackedShieldData, out aPixelDistance) && !flag)
            {
                MonoSingleton<CrittercismManager>.Instance.BeginTransaction(MonoSingleton<CrittercismManager>.Instance.m_ScanPowerTransaction);
                if (this.IsPlainShield(trackedShieldData))
                {
                    MonoSingleton<CrittercismManager>.Instance.LoggingBreadcrumbs(MonoSingleton<CrittercismManager>.Instance.m_ScanPowerTransaction + " Scan Plain Border Shield");
                    trackedShieldData.ClearTrackedShield();
                    this.EnterPlainShieldState();
                    MonoSingleton<CrittercismManager>.Instance.FailTransaction(MonoSingleton<CrittercismManager>.Instance.m_ScanPowerTransaction);
                }
                else
                {
                    this.ValidateIfPowerIsAvailable(trackedShieldData);
                }
            }
            else if ((double) aPixelDistance < (double) Mathf.Min(targetFrameSize.x / 2f, targetFrameSize.y / 2f))
                trackedShieldData.SetColor(this.mScannerUI.m_NearScanningShieldColor);
            else
                trackedShieldData.SetColor(this.mScannerUI.m_TrackingShieldColor);
        }
    }
}
```

Figure 13: Method 'UpdateReadyToScanState'.

```
private bool ValidateIfPowerIsAvailable(ShieldTrackerFeedback aVisualTracker)
{
    uint markerId = aVisualTracker.GetTrackedShieldData().ContourMarker.MarkerID;
    this.m_CurrentScannedPower = PowerUtil.GetByShieldBorderID(markerId.ToString());
    this.m_CurrentScannedPowerLevel = 0;
    bool flag = (UnityEngine.Object) this.m_CurrentScannedPower != (UnityEngine.Object) null;
    if (flag)
    {
        MonoSingleton<CrittercismManager>.Instance.LoggingBreadcrumbs(MonoSingleton<CrittercismManager>.Instance.m_ScanPowerTransaction + " Shield border id found");
        this.EnterWaitingForScanResultState();
    }
    else
    {
        MonoSingleton<CrittercismManager>.Instance.LoggingBreadcrumbs(MonoSingleton<CrittercismManager>.Instance.m_ScanPowerTransaction + " Shield border id not found: " + markerId.ToString());
        this.EnterShieldNotAvailableState(aVisualTracker, markerId);
        MonoSingleton<CrittercismManager>.Instance.FailTransaction(MonoSingleton<CrittercismManager>.Instance.m_ScanPowerTransaction);
    }
    aVisualTracker.ClearTrackedShield();
    return flag;
}
```

Figure 14: Method 'ValidateIfPowerAvailable'.

The method 'ValidateIfPowerAvailable' checks the MarkerID of the shield that is currently being tracked (green crosshair). The MarkerID is related to the ContourMarker which is scanned by the Vuforia extension. Using the MarkerID the application will check to see if the BorderID corresponds to the shield. Next the BorderID will be checked to see if it corresponds to a UnityEngine.Object, If that is the case the scan state will be changed to 'EnterWaitingForScanResultState' and a callback will be executed to the PowerProgressionManager.

```

PowerMap.cs  EPower.cs  RData.cs  PowerData.cs  PowerUtil.cs X  ShieldTrackedData.cs  ShieldTrackerFeedback.cs  ShieldScanner.cs  ShieldScannerUI.cs  PowerSharer.cs  ContourMarker.cs

public static PowerData GetPowerData(EPower aPower)
{
    PowerData powerData = EPowerMap.GetPowerData(aPower);
    if ((UnityEngine.Object) powerData == (UnityEngine.Object) null)
        Debug.LogError((object) (aPower.ToString() + " Not in EPowerMap"));
    return powerData;
}

public static Texture2D GetPowerShieldTexture(EPower aPower)
{
    return PowerUtil.GetPowerData(aPower).PowerShieldTexture.target as Texture2D;
}

public static PowerData GetByShieldBorderID(string aID)
{
    if (PowerUtil.mPowerDataByShieldBorderID.ContainsKey(aID))
        return PowerUtil.mPowerDataByShieldBorderID[aID];
    return (PowerData) null;
}

public static List<PowerData> GetBasicPower()
{
    return PowerUtil.mBasicPower;
}

public static List<PowerData> GetAllUnlockedReleasedPower()
{
    List<PowerData> list = new List<PowerData>();
    PowerProgressionManager progressionManager = ProgressionManager.PowerProgressionManager;
    for (int index = 0; index < PowerUtil.mAllPower.Count; ++index)
    {
        EPower aPower = PowerUtil.mAllPower[index].PowerID;
        if (progressionManager.IsPowerUnlocked(aPower) && PowerUtil.IsPowerReleased(aPower) && !(PowerUtil.mAllPower[index] is BonusPowerData))
            list.Add(PowerUtil.mAllPower[index]);
    }
    return list;
}

public static List<PowerData> GetAllPower()
{
    return PowerUtil.mAllPower;
}

public static List<PowerData> GetAllReleasedPower()
{
    List<PowerData> list = new List<PowerData>();
    for (int index = 0; index < PowerUtil.mAllPower.Count; ++index)
    {
        if (PowerUtil.IsPowerReleased(PowerUtil.mAllPower[index].PowerID) && !(PowerUtil.mAllPower[index] is BonusPowerData))
            list.Add(PowerUtil.mAllPower[index]);
    }
    PowerData[] array = list.ToArray();
}

```

Figure 15: Method 'GetByshieldBorderID'.

```

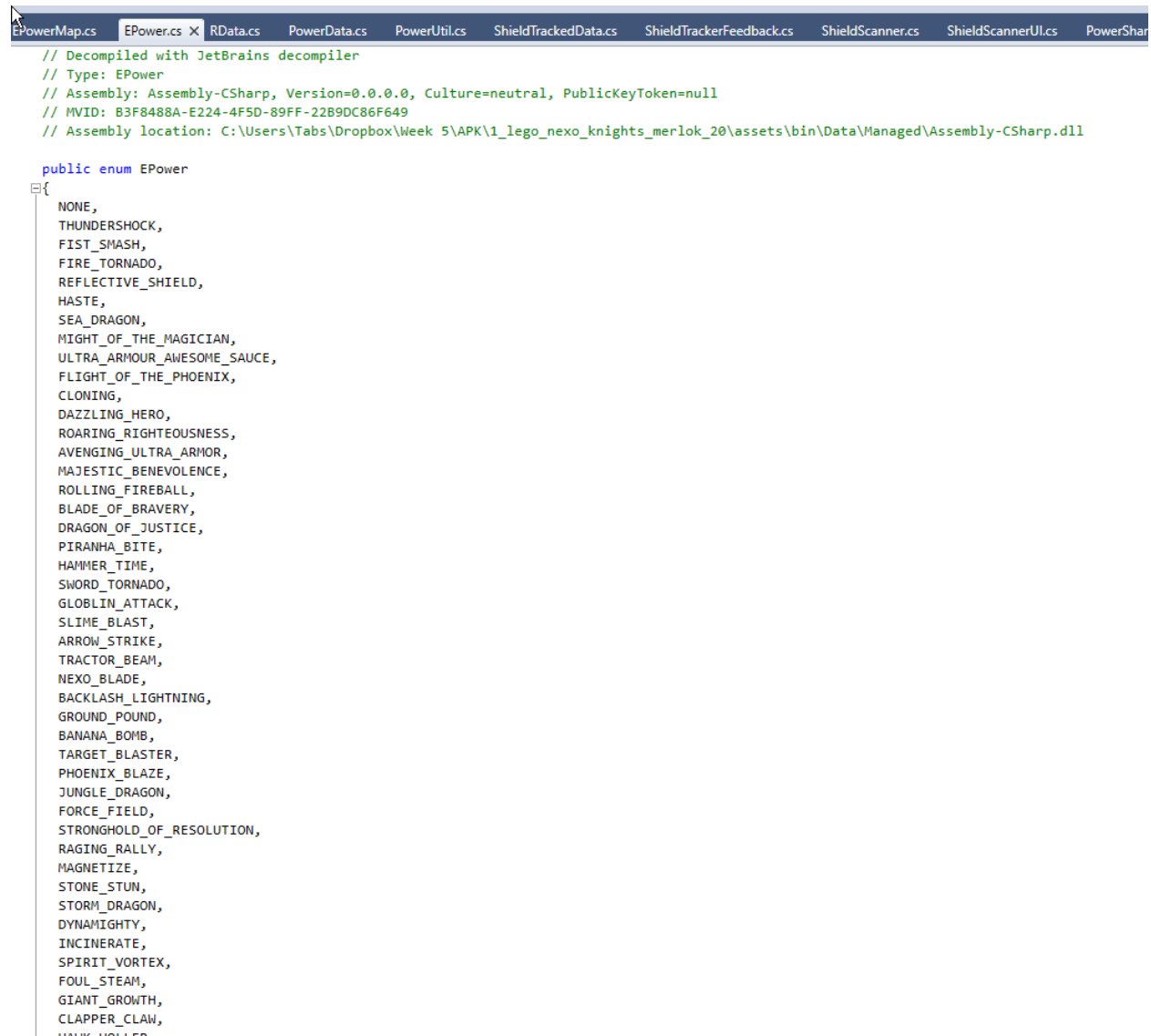
public void ScanPower(EPower aPowerID, int aPowerLevel = 0, Action<bool> aCallback = null, bool aFreeScan = false)
{
    // ISSUE: object of a compiler-generated type is created
    // ISSUE: variable of a compiler-generated type
    PowerProgressionManager.<ScanPower>c__AnonStorey2Bf powerCAnonStorey2Bf = new PowerProgressionManager.<ScanPower>c__AnonStorey2Bf();
    // ISSUE: reference to a compiler-generated field
    powerCAnonStorey2Bf.aCallback = aCallback;
    // ISSUE: reference to a compiler-generated field
    powerCAnonStorey2Bf.scanSuccessful = false;
    string aDescription = string.Empty;
    float aPowerUpgradeCost = (float) this.m_PowerUpgradeCosts[0];
    bool flag = this.IsPowerUnlocked(aPowerID);
    PowerData powerData = PowerUtil.GetPowerData(aPowerID);
    PowerProgression progression = this.GetProgression(aPowerID);
    if (PowerUtil.IsPowerReleased(aPowerID))
    {
        MonoSingleton<CritterismManager>.Instance.LoggingBreadcrumbs(MonoSingleton<CritterismManager>.Instance.m_ScanPowerTransaction + " power released: " + aPowerID.ToString());
        if (powerData.PreScanCheck())
        {
            if (!flag || progression != null && progression.StarCount < this.m_MaxPowerUpgrade)
            {
                if (!((powerData is BonusPowerData) && progression != null))
                {
                    aPowerUpgradeCost = (float) this.m_PowerUpgradeCosts[Math.Min(progression.StarCount, this.m_PowerUpgradeCosts.Length)];
                }
                if ((double) this.ScanEnergy >= (double) aPowerUpgradeCost)
                {
                    MonoSingleton<CritterismManager>.Instance.LoggingBreadcrumbs(MonoSingleton<CritterismManager>.Instance.m_ScanPowerTransaction + " Scan energy available");
                    if (!aFreeScan)
                    {
                        this.ScanEnergy -= aPowerUpgradeCost;
                        MonoSingleton<ProgressionManager>.Instance.ResetDailyBonusTimer();
                        if (!(powerData is BonusPowerData))
                        {
                            if (flag)
                            {
                                this.RescanPower(aPowerID, Math.Max(progression.StarCount, aPowerLevel));
                            }
                            else
                            {
                                this.UnlockPower(aPowerID, aPowerLevel);
                            }
                        }
                    }
                    // ISSUE: reference to a compiler-generated field
                    powerCAnonStorey2Bf.scanSuccessful = true;
                }
                else
                {
                    MonoSingleton<CritterismManager>.Instance.LoggingBreadcrumbs(MonoSingleton<CritterismManager>.Instance.m_ScanPowerTransaction + " Scan energy missing " + (aPowerUpgradeCost - this.ScanEnergy));
                    if ((double) aPowerUpgradeCost > (double) this.ScanEnergyMax)
                    {
                        if (!(UnityEngine.Object) ScreenManager.Instance.GetScreen(EScreenMap.Value[EScreen.NexoChargeLevel].m_Screen) == (UnityEngine.Object) null) || this.m_IsLevelPopUpLoading)
                        {
                            return;
                        }
                        // ISSUE: reference to a compiler-generated method
                        MonoSingleton<CoroutineManager>.Instance.StopCoroutine(this.ShowNexoChargeLevelPopUp(new Action(powerCAnonStorey2Bf.<>m__15)));
                        // ISSUE: reference to a compiler-generated method
                        MonoSingleton<CoroutineManager>.Instance.StartCoroutine(this.ShowNexoChargeLevelPopUp(new Action(powerCAnonStorey2Bf.<>m__16)));
                    }
                }
            }
        }
    }
}

```

Figure 16: Scan state 'EnterWaitingForScanResultState'.

The PowerProgressionManager uses a list of Powers and a list of data (enum) about these powers called EPower. The enum APowerID is used to determine if the shield or power has been released and what its level is. After this a few more callbacks are used to finish of the scanning process.

The enum EPower returns multiple times in the C# Assembly DLL. As said before, this is the list of all the powers in the game. Besides this there is a so called 'EPowerMap' which is also referenced multiples times in the code. It seems like EPowerMap retrieves information about the power / shields by calling 'Rdata'. This is where we lose track of what is going on, because it seems like it uses some sort of serialization. Besides we have no clue what 'Rdata' actually does.



The screenshot shows a code editor with several tabs at the top: EPowerMap.cs, EPower.cs, RData.cs, PowerData.cs, PowerUtil.cs, ShieldTrackedData.cs, ShieldTrackerFeedback.cs, ShieldScanner.cs, ShieldScannerUI.cs, and PowerShar. The active tab is EPower.cs, which contains the following code:

```
// Decompiled with JetBrains decompiler
// Type: EPower
// Assembly: Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
// MVID: B3F8488A-E224-4F5D-89FF-22B9DC86F649
// Assembly location: C:\Users\Tabs\Dropbox\Week 5\APK\1_lego_nexo_knights_merlok_20\assets\bin\Data\Managed\Assembly-CSharp.dll

public enum EPower
{
    NONE,
    THUNDERSHOCK,
    FIST_SMASH,
    FIRE_TORNADO,
    REFLECTIVE_SHIELD,
    HASTE,
    SEA_DRAGON,
    MIGHT_OF_THE_MAGICIAN,
    ULTRA_ARMOUR_AWESOME_SAUCE,
    FLIGHT_OF_THE_PHOENIX,
    CLONING,
    DAZZLING_HERO,
    ROARING_RIGHTEOUSNESS,
    AVENGING_ULTRA_ARMOR,
    MAJESTIC_BENEVOLENCE,
    ROLLING_FIREBALL,
    BLADE_OF_BRAVERY,
    DRAGON_OF_JUSTICE,
    PIRANHA_BITE,
    HAMMER_TIME,
    SWORD_TORNADO,
    GOBLIN_ATTACK,
    SLIME_BLAST,
    ARROW_STRIKE,
    TRACTOR_BEAM,
    NEXO_BLADE,
    BACKLASH_LIGHTNING,
    GROUND_POUND,
    BANANA_BOMB,
    TARGET_BLASTER,
    PHOENIX_BLAZE,
    JUNGLE_DRAGON,
    FORCE_FIELD,
    STRONGHOLD_OF_RESOLUTION,
    RAGING_RALLY,
    MAGNETIZE,
    STONE_STUN,
    STORM_DRAGON,
    DYNAMIGHTY,
    INCINERATE,
    SPIRIT_VORTEX,
    FOUL_STEAM,
    GIANT_GROWTH,
    CLAPPER_CLAW,
    ...
}
```

Figure 17: Enum 'EPower'.

```
// Decompiled with JetBrains decompiler
// Type: EPowerMap
// Assembly: Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
// MVID: 83F8488A-E224-4F5D-89FF-22B9DC86F649
// Assembly location: C:\Users\Tabs\Dropbox\Week 5\APK\1_lego_nexo_knights_merlok_20\assets\bin\Data\Managed\Assembly-CSharp.dll

using FrimaEngine;
using System;
using System.Collections.Generic;

public class EPowerMap : EnumMapper<EPower, RData>
{
    private static EPowerMap m_Mapper;

    public static EPowerMap Value
    {
        get
        {
            if ((UnityEngine.Object) EPowerMap.m_Mapper == (UnityEngine.Object) null)
            {
                EPowerMap.m_Mapper = new RData("02b6f5a6ea7e0d4a903c7ff6b4c1165").target as EPowerMap;
            }
            return EPowerMap.m_Mapper;
        }
    }

    public static RData GetValue(EPower aKey)
    {
        if (((List<EPower>) EPowerMap.Value.m_SerializedT).Contains(aKey))
        {
            return EPowerMap.Value[aKey];
        }
        return (RData) null;
    }

    public static PowerData GetPowerData(EPower aKey)
    {
        if (((List<EPower>) EPowerMap.Value.m_SerializedT).Contains(aKey))
        {
            return EPowerMap.Value[aKey].target as PowerData;
        }
        return (PowerData) null;
    }

    public static List<RData> GetPowerRDataList()
    {
        List<RData> list = new List<RData>();
        foreach (int num in Enum.GetValues(typeof(EPower)))
        {
            EPower index = (EPower) num;
            if (index != EPower.NONE)
            {
                list.Add(EPowerMap.Value[index]);
            }
        }
        return list;
    }
}
```

Figure 18: Class 'EPowerMap'.

If and when someone scans a QR code of a shield, which upgrades the shields power to level 5), a string of data is embedded in the QR. This string contains multiple elements:

```
string aTextToEncode = "SHDPWR/" + (object) powerData.ShieldBorderID + "/" + (string)
ProgressionManager.PowerProgressionManager.GetProgression(aPower).StarCount +
"/" + (string) (object) dateTime2.Ticks + "/" + (string) (object) dateTime3.Ticks + "/" + (string)
(object) utcOffset.Ticks;
```

The string from the QR code contains the ShieldBorderID, hidden as the second element. This is the decimal number we talked about earlier on. When this decimal number is converted to a binary number, we get the same 32bit sequences like the ones we found early on (the way we presumed they were displayed). When a QR code is scanned, the ShieldBorderID is retrieved and used as a decimal number that forms the BorderID. Through this we can conclude that the MarkerID is a decimal number as well, because this is used in the same way as the BorderID is used.

2.1.4 Assets

There are roughly 70 assets files, most of them are split into different files. We used the Hex Neo Editor to re-assemble all of the assets files and then used UnityAssetsExplorer to view the contents of the assets files. This however yielded no results, most of the files stored in the assets are texture and sound files.

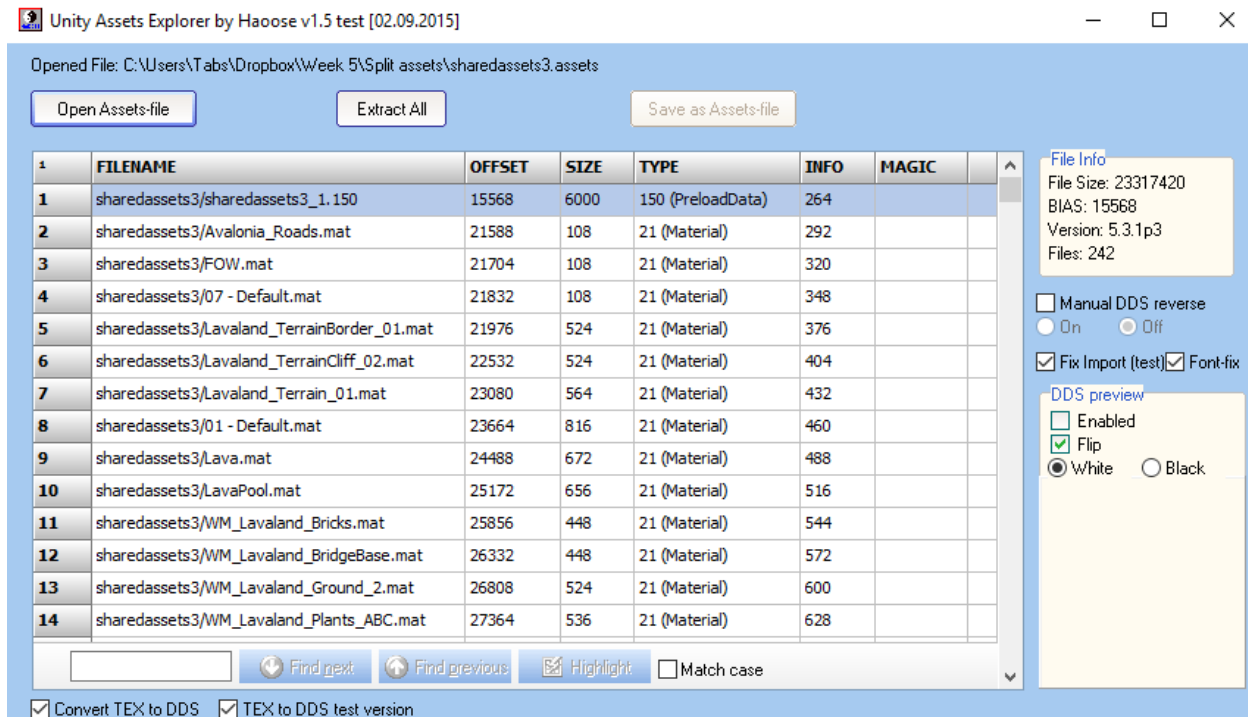


Figure 19: Opened assets file

2.1.5 Classes.dex

The classes.dex file contains all of the JavaScript classes used by the application. We used dex2jar to convert the dex file to a Jar file. Then we used JD Java Decompiler to open the Jar file and get a full overview of all of the used JavaClasses. Most of the JavaClasses seem to contain code specific to certain plugins, but no code vital to the scanning process. However a lot of the code contained in the JavaClasses is obfuscated, which makes it hard to tell exactly what a function does and what variables it effects.

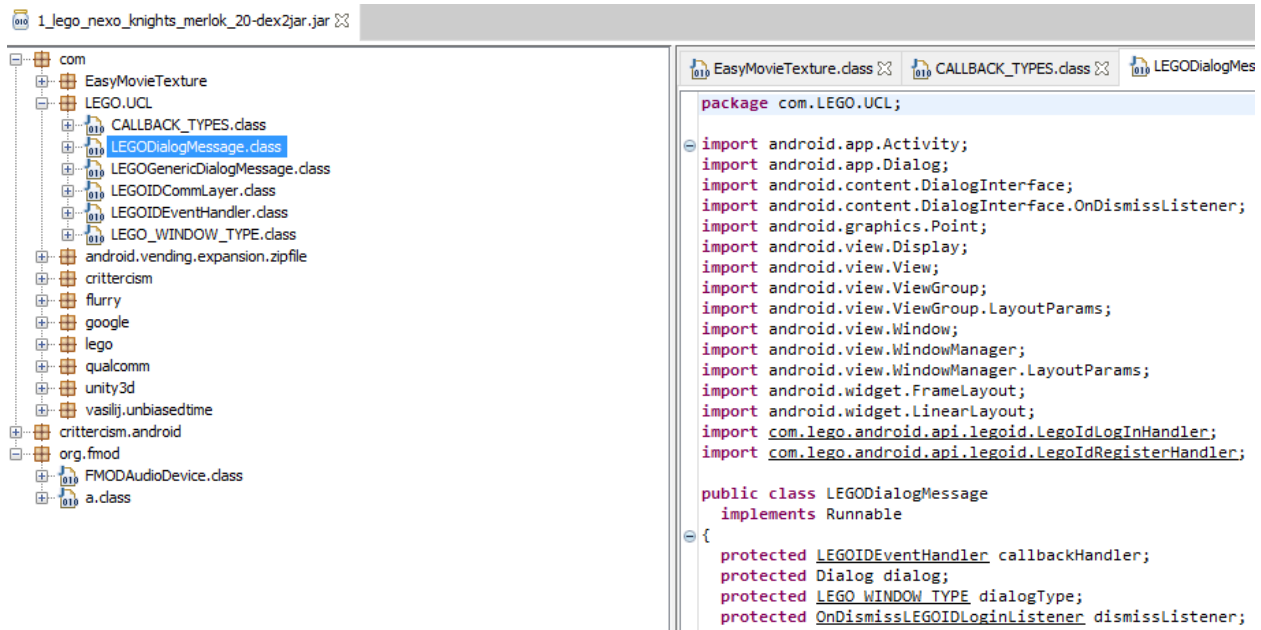


Figure 20: JavaClasses

2.1.6 Datafiles

Inside the Game Cache there are roughly 8000 data files with what seems hashed names, they are most likely serialized and contain data related to game objects, textures and sound files. We used Bulk Extraction Viewer to load the 8000 files and used a wordlist to look through the files. This makes searching for specific information in a large amount of files easier.

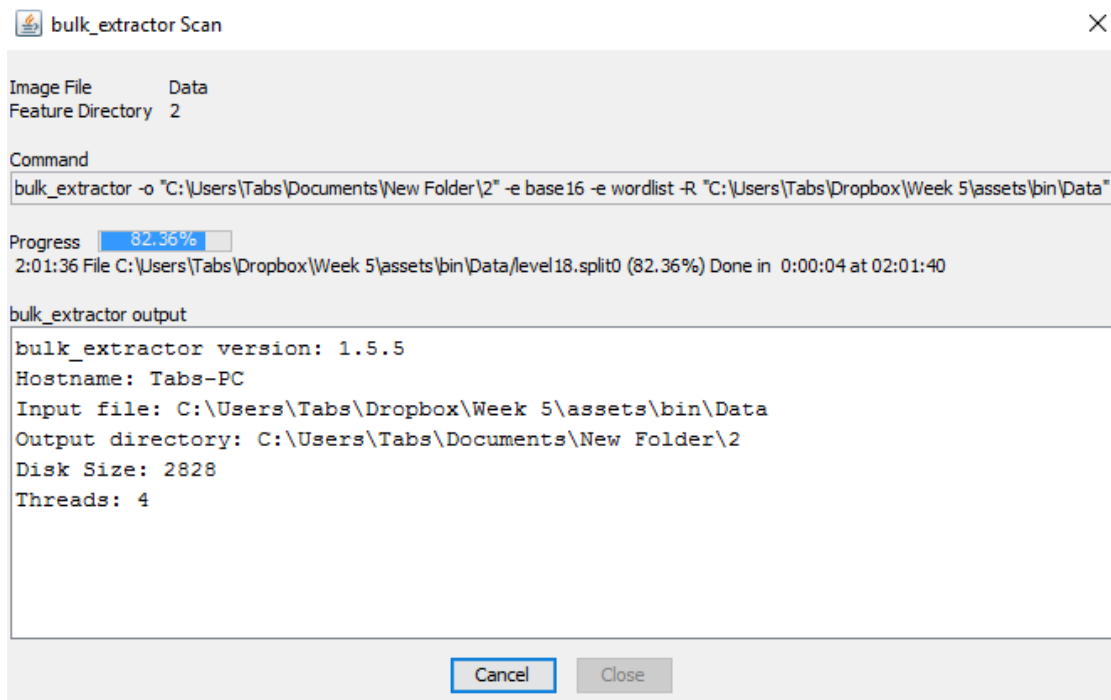


Figure 21: Analysing all hashed files with BE viewer

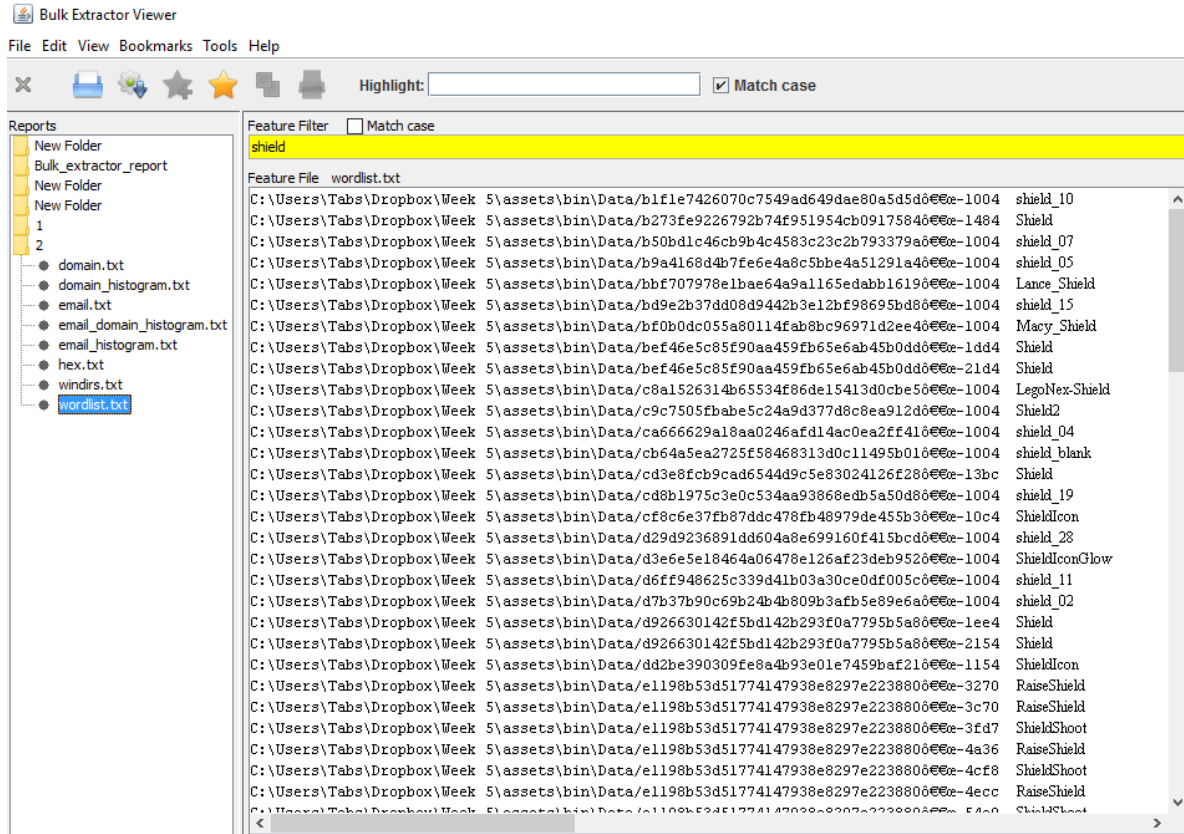


Figure 22: BE Wordlist

Our search criteria were (match case turned off):

- Shield
- BorderID
- Power
- EPower
- APower
- PowerMap
- QR
- MarkerID
- RaptorBite
- Vector2D
- Vector3D
- ContourMarker

Almost all of the information inside the files is binary or hashed/encrypted, therefore we did not retrieve valuable information from all of the data files.

2.1.7 QR Code scanner

The application can also scan regular QR codes, they are used to share powers with other people. The QR code contains the shieldborderID of a shield and is a decimal value.

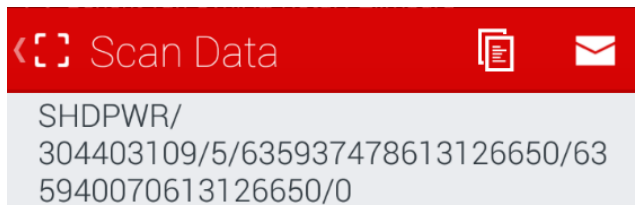


Figure 24: QRCode Rolling fire shield



Figure 23: Rolling Fire shield

When we take a look at the border of the Rolling fire Shield we can count its Border and get a binary number of: 00010010001001001101001010100101. When we scan the QR code of this shield we get the following string:



SHDPWR stands for Sharedpower. The number 304403109 is the shieldBorderID, when we turn 304403109 into a binary number we find: 00010010001001001101001010100101. This is exactly the same number as the borderID we calculated earlier.

3 Discussion

During the process of writing the documentation, we were wondering if the abovementioned “Vuforia” was a Virtual Reality or an Augmented Reality plugin. When we were browsing for more information about Vuforia we stumbled on the Vuforia homepage, which showed us the following picture.

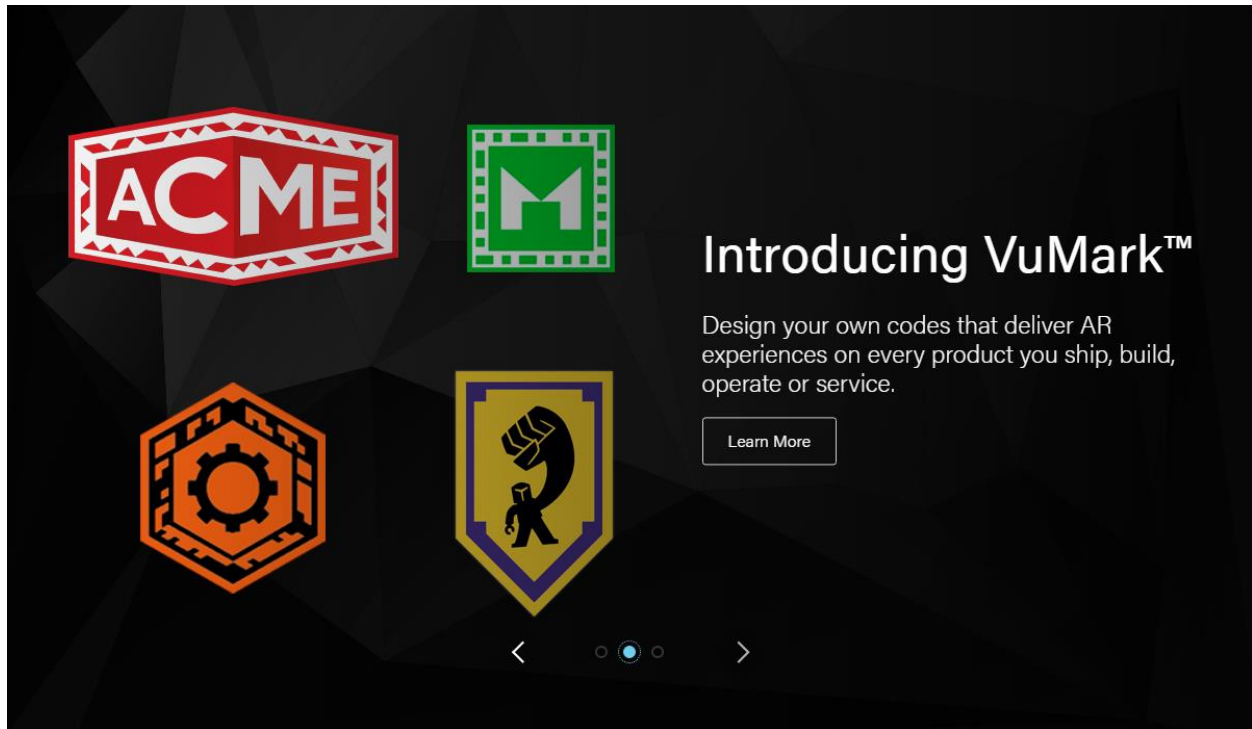


Figure 25: VuMark SDK.

According to this, VuMark can be used to make custom barcodes by using your own design. The picture also shows a few examples, including one of the Shields from Nexu Knights.

4 Conclusion

It's possible to find the undiscovered shields by brute forcing with the QR code scanner, but it would require a specific setup where all of the current shields and their border ID ranges are excluded. It would also take a lot of time to scan millions of different shield combinations. The best chance at recovering the missing shields is by looking at VuMarker and how it interprets scanned shields, then reverse engineering the APK to find out if there is a way to retrieve the missing shields.

5 Tools

Hex Editor Neo

Hex Editor Neo is a free hex editor tool which is optimized to work with large files.

<http://www.hhdsoftware.com/free-hex-editor>

Notepad++

Notepad++ is a free text editor which supports several plugins, which can be downloaded in the application.

<https://notepad-plus-plus.org/>

HEX-Editor

The HEX-Editor plugin converts the selected text into a HEX view.

PKZIP

PKZIP compresses and decompresses files.

<https://www.pkware.com/software/pkzip/>

JetBrains DotPeek 10.0.2.

Used to decompile .NET files and view the assembly.

<https://www.jetbrains.com/decompiler/>

UnityAssetsExplorer 1.5

Used to view and extract individual files within a compiled unity assets file.

<http://zenhax.com/viewtopic.php?f=9&t=36>

Dex2Jar-2.0

Tools to work with android .dex and java .class files

<https://github.com/pxb1988/dex2jar>

JD Java decompiler GUI 1.4.0

JD-GUI is a standalone graphical utility that displays Java source codes of “.class” files.

<http://jd.benow.ca/>

Bulk Extractor Viewer

Bulk Extractor Viewer (BEViewer) is a User Interface for browsing features that have been extracted via the bulk_extractor feature extraction tool.

https://github.com/simsong/bulk_extractor/wiki/BEViewer

ZombieVision Nexo Knights Generator

Nexo Knights Generator is a web-based tool that generates all possible shield combination without any restrictions. It start at one '1' bit block and gradually goes up to 32 '1' bit blocks.

<http://zombievision.net/nexoknights/>