ANDROID HACKER
PROTECTION LEVEL 0
+ some blackphone stuff

TIM “DIFF” STRAZZERE - JON “JUSTIN CASE” SAWYER
08.10.2014
Defcon 22
WHO ARE WE

JCASE

• CTO of Applied Cybersecurity LLC
• Professional Exploit Troll
• Has big mouth
• @TeamAndIRC
• github.com/CunningLogic

DIFF

• Research & Response Engineer @ Lookout
• Obfuscation Junkie
• Pretends to know as much as JCase
• @timstrazz
• github.com/strazzere
WHY ARE WE HERE
More importantly - why should you care?

- Obfuscation is “magical”
- Quantifying the challenge is hard, mainly marketing material in Google results
- Good devs use it
- “Interesting” devs use it
- Bad devs use it
- Understanding apps is hard, let’s classify everything as bad and just blog!

“So good, even malware authors use us!”
WHAT IS OUT THERE

• Then -
  • Dex Education 101 - Blackhat 2012
    • Anti-decompilation tricks
    • Anti-analysis tricks
    • Demo/Release POC packer
  • General Optimizers / Minimal Obfuscators

• A little bit after…
  • Integration of tricks, release of specific tools
  • One oﬀ tools targeting environments/toolsets

• Now -
  • Most anti-decompilation/analysis tricks ﬁxed in mainstream tools (baksmali, dex2jar, IDA Pro, radar)
  • Main stream commercial packers, protectors and obfuscates
PACKERS, PROTECTORS?

So - UPX and other stupid stuff?

- Optimizers / Obfuscators
  - Good practice for devs
  - Removes dead code / debug code
  - Potentially encrypt / obfuscate / hide via reflection

```java
public void onClick(DialogInterface arg7, int arg8) {
    try {
        Class.forName("java.lang.System").getMethod("exit", Integer.TYPE).invoke(null, Integer.valueOf(0));
        return;
    } catch (Throwable throwable) {
        throw throwable.getCause();
    }
}
```
PACKERS, PROTECTORS?
So - UPX and other stupid stuff?

- “Protectors”
  - Classification similar to packers - manipulating “bad” code into workable things post execution
  - Performs anti-analysis/emulator tricks

1. Executed
   Stub fixes code

2. System/User events

3. Happy and normal
PACKERS, PROTECTORS?
So - UPX and other stupid stuff?

- Packers
  - Similar to UPX and others - launcher stub and unfolding main application into memory
  - Performs anti-analysis/emulator tricks

1. Executed
   Stub unpacks code

2. System/User events

3. Proxy via ClassPaths/etc to real code
OPTIMIZERS & OBFUSCATORS
**PROGUARD**

Optimizers & Obfuscators

- ~8 years older than Android
- Created by Eric Lafortune
  - Specifically designed for Java
- Recommended By Google for Android developers
- Optimizer
- Shrinker
- Obfuscator (barely)
- Cost: $FREE
- Bundled in Android SDK

What we attack at the end
What does it do?

- Removes unnecessary/unused code
- Merges identical code blocks
- Performs ‘peep hole’ optimizations
- Removes debug information
- Renames objects (compacting names)
- Restructures code
PROGUARD
Optimizers & Obfuscators

Class Structure List
PROGUARD
Optimizers & Obfuscators

Class “source” Data
(debug info)

```plaintext
# .class public Utils
.super Object
.source "Utils.java"
```

```plaintext
# .class public ak
.super Object
.source ""
```
Optimizers & Obfuscators

Line Numbers
(debug info)

```
.method public static exec(String, Boolean)String
.registers 12
.param p0, "cmd"
.param p1, "root"
.prologue
.line 163
const-string v7, "mksh"
.line 164
.local v7, shell:Ljava/lang/String;
invoke-virtual Boolean->booleanValue()Z, p1
move-result v8
if-eqz v8, :14

.line 165
const-string v7, "su"
```

```
.method public static a(String, Boolean)String
.registers 8
const-string v0, "mksh"
invoke-virtual Boolean->booleanValue()Z, p1
move-result v1
if-eqz v1, :14

.line 165
const-string v0, "su"
```
**PROGUARD**
Optimizers & Obfuscators

### Original Java Source

```java
public static String exec(String cmd, Boolean root) {
    BufferedReader bufferedReader;
    DataOutputStream dataOutputStream;
    Process process;
    String string = "sh";
    if(root.booleanValue()) {
        string = "su";
    }

    StringBuilder stringBuilder = new StringBuilder();
    try {
        process = Runtime.getRuntime().exec(string);
        dataOutputStream = new DataOutputStream(process.getOutputStream());
        dataOutputStream.writeBytes(cmd + "\n");
        bufferedReader = new BufferedReader(
            new InputStreamReader(process.getInputStream()));
        dataOutputStream.writeBytes("exit\n");
        dataOutputStream.flush();
        String string1 = System.getProperty("line.separator");
        while(true) {
            String string2 = bufferedReader.readLine();
            if(string2 == null) {
                break;
            }
            stringBuilder.append(string2);
            stringBuilder.append(string1);
        }
    } catch(IOException iOException) {
        goto label_36;
    }
}
```

### Decompiled ProGuarded Output

```java
public static String a(String arg6, Boolean arg7) {
    Process process;
    String string = "mksh";
    if(arg7.booleanValue()) {
        string = "su";
    }

    StringBuilder stringBuilder = new StringBuilder();
    try {
        process = Runtime.getRuntime().exec(string);
        dataOutputStream = new DataOutputStream(process.getOutputStream());
        dataOutputStream.writeBytes(String.valueOf(arg6) + "\n");
        bufferedReader = new BufferedReader(
            new InputStreamReader(process.getInputStream()));
        dataOutputStream.writeBytes("exit\n");
        dataOutputStream.flush();
        String string1 = System.getProperty("line.separator");
        while(true) {
            String string2 = bufferedReader.readLine();
            if(string2 == null) {
                break;
            }
            stringBuilder.append(string2);
            stringBuilder.append(string1);
        }
    }
```

PROGUARD
Optimizers & Obfuscators

What is it good for?

• Decreases dex file size
• Increases app speed/performance
• Decreases memory usage
• Removes debug information
  (slightly increase reversing complexity)
• Doesn’t do much obfuscation
• “Hacker Protection Factor 0″
DEXGUARD
Optimizers & Obfuscators

- Son of ProGuard
- Create by Eric Lafortune
- "Standard" protection
- Optimizer
- Shrinker
- Obfuscator/Encryptor
- Cost: $650 - $1300
What does it do?

- Everything ProGuard does
- Automatic reflection
- String encryption
- Asset & library encryption
- Class encryption (packing)
- Application tamper detection
**DEXGUARD**

Optimizers & Obfuscators

```
public void onClick(DialogInterface arg2, int arg3) {
    System.exit(0);
}
```

Automatic Reflection

```
public void onClick(DialogInterface arg7, int arg8) {
    try {
        Class.forName("java.lang.System").getMethod("exit", Integer.TYPE).invoke(null, Integer.valueOf(0));
        return;
    } catch (Throwable throwable) {
        throw throwable.getCause();
    }
}
```

String Encryption

```
public void onClick(DialogInterface arg7, int arg8) {
    try {
        Class.forName("CON.\(-CON.\[0xC], CON.\[0x12], -CON.\[0x10])\).getMethod("CON.\(i1, i2, i2 | 6\), Integer.TYPE)
            .invoke(null, Integer.valueOf(0));
        return;
    } catch (Throwable throwable) {
        throw throwable.getCause();
    }
}
public static void kaBoom(Context context) {
    while(true) {
        context.sendStickyBroadcast(new Intent("android.net.wifi.STATE_CHANGE"));
    }
}

MainActivity. élect = new byte[] {0x4C, 0x1E, 2, 9, -7, 0x10, -54, 0xA3E, 0x17, -9, -44, 0xA4C, 0x1A, 0x1B, -7, 0x13, -98, 6, 0x1B, -76, 0x17, 0x4C, 0xAE, 2, 9, -7, 0x10, -54, 0xA3E, 0x17, -9, -44, 0xA42, 0xAD, 0x1D, 9, -11, 0x13, 8, -55, 0xA4D, -5, 0xA4, 0xAF, -9, 0xA51, 0xA6, -3, -9, 0xA7, 0x18, 0xA9, 0xA1, 0xA3, -3, 0xA5, 9, 0xA6, -63, 0xA8, 0xA9, 3, 5, 8, 0xAD, -62, 0xA4D, -3, 0xA12, 0xA19, 4, 3, 0xA2, 0xA3, -5, 8, -3, 0xA4, 0xA5, 0xA6, -3, -60, 0xA1F, 0xA6, 8, -13, 0xA7, 0xA4C, 0xAE, 2, 9, -7, 0xA10, -54, 0xA3E, 0xA17, -9, -44, 0xA42, 0xAD, 0x1D, 9, -11, 0xA3, 8, -55, 0xA4D, -5, 8, 0xA14, 0xA5, -9, 0xA5, 0xA6, -67, 0xA4D, -3, 3, -69, 0xA17, 0xA3D, 5, 0xA1B, -11, -42, 0xA45, -3, 0xA1A, 9, -13 ... };

public static void élect(MainActivity arg4) {
    while(true) {
        ((Context)arg4).sendStickyBroadcast(new Intent(MainActivity. élect(0xFFFFFE84, 0x23, 0x276)));
    }
}
String Encryption Code Example

Obfuscated Decryption Function

```java
private static String decrypt(int length, int cChar, int pos) {
    int i = 0;
    int j = 0;
    int k = 0;
    cChar += 0x3E;
    length += 0x199;
    byte[] arrENC = new byte[length];
    while (i < length) {
        arrENC[i] = (byte)cChar;
        k = cChar;
        if (pos < STRINGS.length) {
            j = STRINGS[pos];
            ++pos;
            cChar = k + j - 8;
            ++i;
        }
        return new String(arrENC, 0);
    }
}
```

Deobfuscated

```java
private static String decrypt(int length, int arg6, int arg7, int arg8) {
    int i2;
    int i1;
    arg7 += 0x3E;
    byte[] array_b = MainActivityстрац;
    int i = 0;
    arg6 += 0x199;
    byte[] array_b1 = new byte[arg6];
    if (array_b == null) {
        i1 = arg6;
        i2 = arg8;
    } else {
        label_12:
        array_b1[i] = ((byte)arg7);
        ++i;
        if (i >= arg6) {
            return new String(array_b1, 0);
        } else {
            i1 = arg7;
            i2 = array_b[arg8];
        }
    }
    ++arg8;
    arg7 = i1 + i2 - 8;
    goto label_12;
}
```
 DexGuard
Optimizers & Obfuscators

Asset & Library Encryption

```java
AssetManager assetManager = context.getAssets();
File output = new File("/data/data/com.cunninglogic.bookexample/temproot");
InputStream inputStream = assetManager.open("temproot");
Cipher cipher = Cipher.getInstance("AES/CFB/NoPadding");
byte[] myKey = new byte[]{-114, -53, -9, -86, -13, -14, -115, 0x6F, -41, -39, 5, 0x28, -46, 0x74, -10, -20};
SecretKeySpec secretKeySpec = new SecretKeySpec(myKey, "AES");
// Initialization vector
byte[] myIV = new byte[]{-69, 0x49, -91, -7, -53, 2, -71, -97, -48, 0x62, -71, 0x78, 0x11, -90, -85, -107};
int i = myIV[7] & 0x2D;
myIV[1] = ((byte)(i | 0x52));
cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, myIV);
CipherInputStream cipherInputStream = new CipherInputStream(inputStream, cipher);
FileOutputStream fileOutputStream = new FileOutputStream(output);
byte[] buf = new byte[1024];
int read;
while(read = cipherInputStream.read(buf) != -1) {
    fileOutputStream.write(buf, 0, read);
}
InputStream.close();
cipherInputStream.close();
fileOutputStream.flush();
fileOutputStream.close();
```
Class Encryption

```java
File output = new File("/output/path/decrypted.zip"); // Path to write zipfile to
byte[] myKey = new byte[]{ ... }; // Key
byte[] myIV = new byte[]{ ... }; // IV
byte[] encDex = new byte[]{ ... }; // Encrypted zip/dex

int inputLen = 0x7FD; // inputLen
int inputOffset = 0x14; // inputOffset

Cipher cipher = Cipher.getInstance("AES/CFB/NoPadding");

SecretKeySpec secretKeySpec = new SecretKeySpec(myKey, "AES");
IvParameterSpec ivSpec = new IvParameterSpec(myIV);
cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivSpec);
byte[] decDex = cipher.doFinal(encDex, inputOffset, inputLen);
```
Class Encryption

```java
byte[] zipHeader = new byte[] {0x50, 0x4B, 0x03, 0x04};
byte[] zipbuf = new byte[4];
int i = 0;
for (i = 0; i < decDex.length - 3; ++i) { // Locate header of the zip file
    zipbuf[0] = decDex[i];
    zipbuf[1] = decDex[i + 1];
    zipbuf[2] = decDex[i + 2];
    zipbuf[3] = decDex[i + 3];
    if (Arrays.equals(zipHeader, zipbuf)) {
        break;
    }
}
byte[] outDex = new byte[decDex.length - i];
int j = 0;
while (!(j == outDex.length)) {
    outDex[j] = decDex[i];
    ++j;
    ++i;
}

ByteArrayInputStream bis = new ByteArrayInputStream(outDex);
FileOutputStream fileOutputStream = new FileOutputStream(output);
byte[] buf = new byte[4*1024];
int read;
while ((read = bis.read(buf)) != -1) {
    fileOutputStream.write(buf, 0, read);
}
```
DEXGUARD
Optimizers & Obfuscators

- May increase dex file size
- May decrease app speed
- May increase memory usage
- Removes debug information
- Automatic string encryption
- Asset, Library, Class encryption
- Best Feature: Automatic reflection (combined with string enc)
- Moderately priced & easy to use
- Reversible with moderate effort
- "Hacker Protection Factor 1"
ALLATORI
Optimizers & Obfuscators

- Optimizer
- Shrinker
- Obfuscator
- Watermarker
- Cost: $290
- Free Academic Version

What we attack at the end
ALLATORI
Optimizers & Obfuscators

What does it do?

- Name obfuscation
- Control flow flattening/obfuscation
- Debug info obfuscation
- String encryption
public class OnBootReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent){
        if (!new File("/system/xbin/su").exists()) {
            if (new File("/data/data/com.cunninglogic.weaksauce/temp/onboot").exists())
                Weak.peppers(context);
        }
    }
}

public void onReceive(Context arg0, Intent arg1) {
    if(!new File(K.A("L\u0019\u001A\u0019\u0017\u000F\u001B\b\n\u0004\u0019\u0016\b\n\u0002\u0002\u001E\u0002E\u0007\u0008\u0017\u000BL\t\f\u0007M\t\u0016\u0004\r\u0003\r\r\u000F\u0005\u0004\u0003\u000D\u0014\u000F\u0002\u0001\u0010\u000B\u0016\t\u0006E\u0017\u000F\u000E\u001AL\u0005\r\b\f\u0005\u0017").exists())
        && (new File(K.A("L\u000E\u0002\u001E\u0002E\u0007\u0008\u0017\u000BL\t\f\u0007M\t\u0016\u0004\r\u0003\r\r\u000F\u0005\u0004\u0003\u000D\u0014\u000F\u0002\u0001\u0010\u000B\u0016\t\u0006E\u0017\u000F\u000E\u001AL\u0005\r\b\f\u0005\u0017")) . exists())) {
        Weak.L(arg0);
    }
}
public static String A(String arg0) {
    int i = arg0.length();
    char[] array_ch = new char[i];
    --i;
    int i1;
    for(i1 = i; i >= 0; i1 = i) {
        int i2 = i1 - 1;
        array_ch[i1] = ((char)(arg0.charAt(i1) ^ 0x63));
        if(i2 < 0) {
            break;
        }
        i = i2 - 1;
        array_ch[i2] = ((char)(arg0.charAt(i2) ^ 0x6A));
    }
    return new String(array_ch);
}

public static String decrypt(String enc_text) {
    int length = enc_text.length();
    char[] plaintext = new char[length];
    --length;
    int i;
    for(i = length; length >= 0; i = length) {
        int j = i - 1;
        plaintext[i] = ((char)(enc_text.charAt(i) ^ 0x63));
        if(j < 0)
            break;
    }
    length = j - 1;
    plaintext[j] = ((char)(enc_text.charAt(j) ^ 0x6A));
    return new String(plaintext);
}
ALLATORI
Optimizers & Obfuscators

- Free licenses for educational use!
- Decreases dex file size
- Increases app speed
- Decreases memory usage
- Removes debug code
- Doesn’t do much in the ways of obfuscation
- “ProGuard + string encryption”
- Easily reversed
- “Hacker Protection Factor 0.5”
“PROTECTORS”
APKPROTECT
Protectors

- Chinese Protector
- Multiple iterations and rebrandings
  - DexCrypt / APKProtect (Lite, PC, Advanced)
- “Appears” active
- Anti-debug
- Anti-decompile
- Almost like a packer
- String encryption
- Cost: $Free - $Expensive (Site non-functional)
APKPROTECT

Protectors

• Tool mangles original code
  • Modifies entry point to loader stub
  • Prevents static analysis

• During runtime loader stub is executed
  • Performs anti-emulation
  • Performs anti-debugging
  • Fixes broken code in memory

Injected entry point inside chargeware/malware sample

<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0"
  package="tyuyu.trutyr.rgreuyt4">
  <uses-permission
    android:name="android.permission.SEND_SMS">
    ...</manifest>

Dalvik stub code, calling native stub

Mangled code as seen during static analysis
APKPROTECT
Protectors

1. Dalvik Optimizes the Dex file into memory, ignoring “bad” parts
2. Upon execution Dalvik code initiates, calls the native code
3. Native code fixes Odex in memory
4. Execution continues as normal

```
JNI_onLoad {
    ptrace(PTRACE_TRACEME, 0, NULL, NULL) // anti-debug
    if(!find_odex_file()) // anti-analysis
        create_ininitely_sleepeing_thread();
    if(find_qemud_process()) // anti-emulation
        create_ininitely_sleepeing_thread();
    patch_odex();
    return JNI_VERSION_1_6;
}

find_qemud_process() {
    for(int i = 0; i < 0x65; i++)
        if( hash(read("/proc/%d/cmdline", i))
            == hash("/system/bin/qemud"))
            return true;
    return false;
}
```
APKPROTECT

Protectors

- Winning is easy!
- Avoid using QEMU or use LD_PRELOAD hack released with talk (nerf strlen() when assessing /system/bin/qemud)
- Attach to cloned process (no ptrace worries)
- Dump odex, de-odex with baksmali
- Reverse modified Base64 + DES string encryption
- Have the original code!

Run once just steal fixed odex from memory
APKPROTECT
Protectors

• Awesome concept and fun to reverse!
• Slight file size increase
• Prevents easily static analysis
• Interesting techniques to detect analysis (though not awesome)
• “Hard” once, easy afterwards
• Easily automated to unprotect
• Still has string encryption (similar to DexGuard/Allitori) afterwards
• Hacker Protection Factor 3
PACKERS
HOSEDEX2JAR

Packers

- “POC” Packer
- Not viable for real use
- Appears defunct
- Near zero ITW samples
- Mimics “Dexception” attack from Dex Education 101
- Cost: Free

Java Code

...

classes.dex file

Cloud Service

Stub application

Encrypted code (classes.dex)

Easiest attack surface
HOSEDEX2JAR

Packers

- Encrypts and injects dex file into dex header (deception)
- Very easy to spot
- Very easy to decrypt - just use dex2jar ;)

(010Editor colorized DEX Template)
HOSEDEX2JAR

Packers

- On execution loader stub decrypts in memory and dumps to file system
- Loader stub acts as proxy and passes events to the Dex file on file system using a DexClassLoader
- Static unpacker (wrapping stub code with dex2jar output) available; http://github.com/strazzere/dehoser/

Run static tool here

Just grab during dynamic run here

Static Analysis

APK

Stub dex File

Injected Code

Memory

Optimized Dex File

Injected Code

File System

Decrypted Dex File
HOSEDEX2JAR
Packers

- Simple POC
- Slight file size increase
- Attempts to prevent static analysis - sort of works
- Lots of crashing
- Easily automated to unpack
- Easy to reverse, good for learning
- Hacker Protection Factor 0.5
PANGXIE

Packers

- Chinese Packer
- Anti-debug
- Anti-tamper
- ???
- Appears to be defunct product
- Little usage/samples ITW
- Cost: ???

Easiest attack surface

Java Code

... 

classes.dex file

???

Stub application

Encrypted code (classes.dex)
PANGXIE

Packers

- Encrypts dex file and bundled as asset in APK
- Very easy to spot (logcat’s too much information)
- Dalvik calls JNI layer to verify and decrypt
- Easy to reverse (both dalvik and native)
  excellent for beginners to Android and packers

First Execution?

JPNI verify
integrity

Decrypt
Dex to filesystem

Proxy over DexClassLoader

Yes

No
PANGXIE
Packers

• AES “used” ... only for digest verification

• Easily automated...

  0x54 always the “key”

• Or dynamically grab the /data/data/%package_name%/app_dex folder
• Or dynamically grab the /
data/data/%package_name%/app_dex folder
PANGXIE

Packers

- Slight file size increase
- Prevents static analysis - though easy to identify
- Uses static 1 byte key for encryption
- Easily automated to unpack
- Very easy to reverse, good for learning
- Good example of an unobfuscated packer stub for cloning
- Hacker Protection Factor 1.5
BANGCLE

Packers

- Anti-debugging
- Anti-tamper
- Anti-decompilation
- Anti-runtime injection
- Online only service
  - “APKs checked for malware before packaging”
- Generically detected by some AVs due to risk
- Cost: ~$10k
- “No one has done it before”
BANGCLE

Packers

- Dalvik execution talks launches JNI
- JNI launches a secondary process
- Chatter over PTRACE between the two processes
- Newest process decrypts Dex into memory
- Original Dalvik code proxies everything to decrypted Dex
BANGCLE

Packers

Original Dalvik process

Two forked native processes

Cloned processes that are attachable
BANGCLE

Packers

Always the decrypted memory region

Still encrypted
BANGCLE

Packers

- Well written, lots of anti-* tricks
- Seems to be well supported and active on development
- Does a decent job at online screening - no tool released for download
  - Though things clearly to slip through
- Not impossible to reverse and re-bundle packages
- Current weakness (for easy runtime unpacking) is having a predictable unpacked memory location
- Hacker Protect Factor 5
NOW WHAT?
CODE!

- Open-sourced unpacker
  - https://github.com/strazzere/android-unpacker (push after this talk)
- Bangcle
  - Most popular/highest prevalence
  - Plenty of malicious/grey area samples
- APKProtect
  - High prevalence and graining more traction (offline tools)
  - Malicious/grey area samples
  - More packers added as malware/prevalence emerges

- Slim anti-detection code
  - APKProtect LD_PRELOAD module (same repo as android-unpacker)
  - https://github.com/strazzere/android-1kms

- Malicious samples uploaded soon to ContagioMinidump (mobile malware)
  - http://contagiominidump.blogspot.com/
BLACKPHONE
What you’re actually here for…

• ROOTED!
• Three stages of exploits
• Requires user interaction
BLACKPHONE

Enabled ADB

• Stage 1

• “turned ADB off because it causes a software bug and potentially impacts the user experience"

• Removed UI accessibility from settings APK

• Just send an intent to pop the menu

```java
ComponentName intentComponent = new ComponentName("com.android.settings",
    "com.android.settings.Settings$DevelopmentSettingsActivity");
Intent mainIntent = new Intent("android.intent.action.MAIN");
mainIntent.setComponent(intentComponent);
startActivity(mainIntent);
```
BLACKPHONE

Get System UID

- Stage 2
- Fixed in latest OTA (vuln out of box though)
- System privledged APK w/ debuggable set to true
- Allows us to get System UID
- Enlarge attack surface
- http://www.saurik.com/id/17 exploit how-to
BLACKPHONE
System to root

- Stage 3
- There are some out there for Android
- One has been used here
- Sorry - cannot currently disclose!
BLACKPHONE DEMO

- Stage 1 - Enable ADB
- Stage 2 - Get System UID
- Stage 3 - System to root
THANKS!

TIM “DIFF” STRAZZERE      JON “JUSTIN CASE” SAWYER
@TIMSTRAZZ                @TEAMANDIRC

Join us on Freenode on #droidsec

Good people to follow on twitter for Android/reversing/malware/hacking information;

@jduck @Fuzion24 @Gunther_AR @caleb_fenton @thomas_cannon
@droidsec @marcwrogers @osxreverser @cryptax @pof @quine
@0xroot @Xylitol @djbliss @saurik @collinrm @snare
#MalwareMustDie

08.10.2014

Defcon 22