Cryptography for mobile malware obfuscation

Axelle Apvrille

RSA Conference Europe, October 2011
Session ID: NMS-305
Summary

Introduction
Session objectives
Mobile malware, what are they and how advanced?
Why are malware authors using cryptography?

Simple obfuscation
Crypto apprentices
XOR encryption
DES and AES
Conclusion
## Session objectives

<table>
<thead>
<tr>
<th>Get familiar with real life mobile malware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discuss (very) recent malware</td>
</tr>
<tr>
<td>Wrong ideas:</td>
</tr>
<tr>
<td>▶ &quot;This never happens, I need not be concerned&quot;</td>
</tr>
<tr>
<td>▶ &quot;They do not use exploits, no interesting stuff in there&quot;</td>
</tr>
</tbody>
</table>

## How To See What’s Hidden!

- Spot encryption routines in assembly listings
- Spot the key
- Decrypt!

Step by step examples with real malicious samples!
Mobile malware

> 200,000 downloads!!! - single sample of Android/Plankton
Mobile malware

> 200,000 downloads!!! - single sample of Android/Plankton

FUN FAME ??? $$$MONEY MONEY MONEY$$$
Mobile malware

> 200,000 downloads!!! - single sample of Android/Plankton

FUN FAME ??? $$$MONEY MONEY MONEY $$$

Basic malware: very successful
Advanced: exploits, polymorphic code (=code mutates), botnets, crypto…
Crypto in mobile malware

Motivations

▶ Obfuscation
▶ Hide maliciousness
▶ Harden reverse engineering
▶ Harden detection
▶ Keep control of their own malicious network
### Motivations
- Obfuscation
- Hide maliciousness
- Harden reverse engineering
- Harden detection
- Keep control of their own malicious network

### Algorithms
Encryption algorithms only: Base64 → **AES**
Hash functions, signatures: not used (yet)
### What’s encrypted?

<table>
<thead>
<tr>
<th>Encrypted payload</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS short codes and bodies</td>
<td>Analysis difficulty?</td>
</tr>
</tbody>
</table>
### What’s encrypted?

<table>
<thead>
<tr>
<th>Encrypted payload</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS short codes and bodies</td>
<td>Analysis difficulty?</td>
</tr>
<tr>
<td>URLs of remote servers or C&amp;C to contact</td>
<td>Hide maliciousness</td>
</tr>
</tbody>
</table>
## What’s encrypted?

<table>
<thead>
<tr>
<th>Encrypted payload</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS short codes and bodies</td>
<td>Analysis difficulty?</td>
</tr>
<tr>
<td>URLs of remote servers or C&amp;C to contact</td>
<td>Hide maliciousness</td>
</tr>
<tr>
<td>Communication with remote servers</td>
<td>Keep control + Harden reversing</td>
</tr>
</tbody>
</table>
### What’s encrypted?

<table>
<thead>
<tr>
<th>Encrypted payload</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS short codes and bodies</td>
<td>Analysis difficulty?</td>
</tr>
<tr>
<td>URLs of remote servers or C&amp;C to contact</td>
<td>Hide maliciousness</td>
</tr>
<tr>
<td>Communication with remote servers</td>
<td>Keep control + Harden reversing</td>
</tr>
<tr>
<td>Variable names, keywords, filename</td>
<td>Harden reversing</td>
</tr>
</tbody>
</table>
### What’s encrypted?

<table>
<thead>
<tr>
<th>Encrypted payload</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS short codes and bodies</td>
<td>Analysis difficulty?</td>
</tr>
<tr>
<td>URLs of remote servers or C&amp;C to contact</td>
<td>Hide maliciousness</td>
</tr>
<tr>
<td>Communication with remote servers</td>
<td>Keep control + Harden reversing</td>
</tr>
<tr>
<td>Variable names, keywords, filename</td>
<td>Harden reversing</td>
</tr>
<tr>
<td>Exploits or nested executables</td>
<td>Hide maliciousness + Harden reversing &amp; detection</td>
</tr>
</tbody>
</table>
## What’s encrypted?

<table>
<thead>
<tr>
<th>Encrypted payload</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS short codes and bodies</td>
<td>Analysis difficulty?</td>
</tr>
<tr>
<td>URLs of remote servers or C&amp;C to contact</td>
<td>Hide maliciousness</td>
</tr>
<tr>
<td>Communication with remote servers</td>
<td>Keep control + Harden reversing</td>
</tr>
<tr>
<td>Variable names, keywords, filename</td>
<td>Harden reversing</td>
</tr>
<tr>
<td>Exploits or nested executables</td>
<td>Hide maliciousness + Harden reversing &amp; detection</td>
</tr>
<tr>
<td>Binaries: polymorphic exec.</td>
<td>Harden reversing &amp; detection</td>
</tr>
</tbody>
</table>
Simple obfuscation

Crypto apprentices

XOR encryption

DES and AES

Conclusion
Decompiled source code

```java
this.jdField_b_String = a(b("L1RodW1icy5kYg=="));
```

b() does Base64 decoding, and a() reads a resource.
Decompiled source code

```java
this.jdField_b_String = a(b("L1RodW1icy5kYg=="));
```

b() does Base64 decoding, and a() reads a resource.

Decode the string

```
$ echo "L1RodW1icy5kYg==" | openssl base64 -d
/Thumbs.db
```
Decompiled source code

```java
this.jdField_b_String = a(b("L1RodW1icy5kYg=="));
```

b() does Base64 decoding, and a() reads a resource.

Decode the string

```
$ echo "L1RodW1icy5kYg==" | openssl base64 -d
/Thumbs.db
```

Read Thumbs.db

The file contains base64-encoded data:

```
U01TTn ... HJ1ZmYuLi4NCg==
```
Decode Thumbs.db

SMSNum-1: 3353
SMSText-1: xesss 3689
SMSNum-2: 3353
SMSText-2: xesss 3689
SMSNum-3: 7132
SMSText-3: xesss 3689
end of preff...
Simple obfuscation in mobile malware 2/2

Decode Thumbs.db

SMSNum-1: 3353
SMSText-1: xesss 3689
SMSNum-2: 3353
SMSText-2: xesss 3689
SMSNum-3: 7132
SMSText-3: xesss 3689
end of preff...

Base64 obfuscates:

1. the filename
2. the payload (SMS numbers and text)
Summary

Introduction

Simple obfuscation

Crypto apprentices
Android/PJapps, Java/Konov.S, WinCE/Sejweek, SymbOS/ShadowSrv

XOR encryption

DES and AES

Conclusion
Crypto apprentice no. 1: Android/PJapps

▶ Discovered in 2011, affects Android phones
▶ Remotely controls the phone: send SMS, add bookmark, visit URL, install app

Code builds this URL:

```java
StringBuilder localStringBuilder1 =
    new StringBuilder("http://");
String str1 = Base64.encode(
    "alfo3gsa3nfdsrfo3isd21d8a8fccosm", 1);
...

This is not base64 + it’s decoding:

alfo3gsa3nfdsrfo3isd21d8a8fccosm
log.android188.com
http://log.android188.com
Using a hand-made and obscure algorithm

```java
public String encryptSFrom = "R$...THE KEY";
String str = "";
char [] paramString = param.toCharArray();
int i = (encryptSFrom.toCharArray()).length - 1;
char [] enc = encryptSFrom.toCharArray();
int j = paramString.length - 1;
for (int l = 0; l <= j; ++l) {
    int k = -1;
    for (int i1 = 0; i1 <= i; ++i1)
        if (enc[i1] == paramString[l]) {
            k = i1; break;
        }
    if (k != -1) {
        if (k == 0) k = i; else k -= 1;
        paramString[l] = enc[k];
    }
    str = str + paramString[l];
}
return str;
```
Java/Konov.S!tr: Decrypting the ciphertext

- Encrypts a file named /numbers.cfg (=ciphertext)
- No need to understand the algorithm, just to decrypt the ciphertext!
- Write a basic Java class, copy / paste the algorithm, call it on the ciphertext:

```java
String str = decodeCes(getText("/numbers.cfg"));
System.out.println("Decoding string: "+ str);
```

Result: SMS numbers, body and corresponding price

7122::suksa1837::241.55py6.
7132::suksa1837::141.66py6.
8355::suksa1837::86.00py6.
Simple cryptographic substitution

```cpp
Parameters::codeTable->AddShifrRow(S"YGL", S"1");
Parameters::codeTable->AddShifrRow(S"HKR", S"2");
Parameters::codeTable->AddShifrRow(S"DPO", S"3");
Parameters::codeTable->AddShifrRow(S"WHR", S"4");
Parameters::codeTable->AddShifrRow(S"MKT", S"5");
...
```
Looks like a video downloading application, but sends SMS messages...

```
LDR R1, Y0Rloij[cR?dijWbbRH[]_ijhoR?dij...
```

Simple cryptographic translation (0x0A)
```
c:\System\Install\Registry\Install.reg
```

Install.reg is encrypted too (XOR).
## Summary

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple obfuscation</td>
</tr>
<tr>
<td><strong>Crypto apprentices</strong></td>
</tr>
<tr>
<td>Android/PJapps, Java/Konov.S, WinCE/Sejweek, SymbOS/ShadowSrv</td>
</tr>
<tr>
<td><strong>XOR encryption</strong></td>
</tr>
<tr>
<td>General techniques</td>
</tr>
<tr>
<td>Hands On: SymbOS/Yxes</td>
</tr>
<tr>
<td>XOR encryption in Java and Android malware</td>
</tr>
<tr>
<td>DES and AES</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
XOR encryption is basic, is it really used?

Yes.

- Java/Espaw.D!tr, Java/Swapi.AF, Java/Konov.K...
- WinCE/Pmcryptic (2008)
- SymbOS/Yxes (2009)
- SymbOS/Shurufa (2009)
- SymbOS/ShadowSrv (2010)
- SymbOS/Zhaomiao (2010)
- Android/DrdDream (2011)
- ...

Also found on Pushdo (PC malware)

*Historical*: red phone USA / Russia.
Why XOR ?!

Reasons

1. Perfect algo in theory if key truly random + as long as cipher
2. Efficient: 1 instruction
3. Easy to code

Malicious implementations

- Often 1-byte key
- Simple to break: frequency analysis
Spotting an XOR-encryption routine in Symbian

- a function with a buffer and a key as parameter

```assembly
LDR R1, R4 ; R4 is a counter
LDR R0, R6 ; R6 is the buffer
```

- load one byte of the buffer: LDRB

```assembly
BL _ZNK6TDesC83AtCEi ; TDesC8::AtC(int)
```

- apply the XOR key: EOR

```assembly
EOR R3, R8, R3 ; R3 = R3 XOR R8 (key)
```

- increment a counter ADD

```assembly
ADD R4, R4, #1 ; increment counter
```

- loop until all buffer has been processed: CMP, BL

```assembly
CMP R4, R7 ; R7 is the maximum value
BLT loop ; loop if not finished
```
Finding the value of the XOR key

Method 1. Close to the calling function

Method 2. Break it!

Use **XORSearch** tool (Didier Stevens)

```bash
$ wine XORSearch.exe -s Srv.cfg http
Found XOR 57 position 0000: http://[CENSORED]banw.com/api/
```

Taken from SymbOS/Shurufa.A!tr
Hands On Symbian Yxes Worm
What is it?

SymbOS/Yxes!worm is a *worm* for mobile phones. It sends SMS and connects to Internet. Discovered in 2009.

Why is it important?

1. High bills for victims
2. Said to have affected "*hundreds of thousands*" devices in China [source: Daniel Hoffman, CTO of Smobile]
3. First malware for Symbian OS 9
4. Advanced: hidden connections to Internet and SMS sending...
Where is the config file SisInfo.cfg?

Contents of Symbian package:
- Resource file [20026CA5].rsc: No
- Main malicious executable: AcsServer.exe No
- Installer: 0x20026CA6.exe Try here

Downloaded from a URL? which URL?
Hands On Yxes: Identify the encryption key

Reversing of the installer

Read section in Symbian package:

huh? why?

SUB R0, R11, #0xBC
MOV R1, #0xBF
BL Yxes_func

Key = 0xBF
Decryption routine = Yxes_func
Hands On Yxes: decryption routine

```
SUB    R11, R12, #4
SUB    SP, SP, #0x1C
STR    R0, [R11,#buffer]
MOU    R3, R1
STRB   R3, [R11,#key]
LDR    R0, [R11,#buffer]
BL     Axelle_getLength
STR    R0, [R11,#var_20]
MOU    R3, R0
STR    R3, [R11,#position]
LDR    R2, [R11,#position]
LDR    R3, [R11,#var_20]
ADD    R3, R2, R3
STR    R0, [R11,#max]

Axelle_moreToDecode
LDR    R0, [R11,#buffer]
LDR    R1, [R11,#position]
BL     Axelle_atC  ; returns a pointer on the data buffer at the ith position
MOU    R4, R0
LDR    R0, [R11,#buffer]
LDR    R1, [R11,#position]
BL     Axelle_atC  ; returns a pointer on the data buffer at the ith position
LDRB   R2, [R0]   ; load encoded character
LDRB   R3, [R11,#key]  ; load offset (key)
EOR    R3, R2, R3  ; exclusive OR
STRB   R3, [R4]   ; write decoded character
LDR    R2, [R11,#position]
ADD    R3, R3, R1  ; increment position
STR    R3, [R11,#position]
B      Axelle_haveWeFinished
```

```
Axelle_haveWeFinished
LDR    R2, [R11,#position]
LDR    R3, [R11,#max]
CMP    R2, R3
BCE    Axelle_DecodingDone
```
Hands On Yxes: decryption routine

Initial Test XOR Loop End

Axelle_moreToDecode
LDR R0, [R11, #buffer]
LDR R1, [R11, #position]
BL Axelle_att; returns a pointer on the data buffer at the ith position
LDN R4, R0
LDR R0, [R11, #position]
LB Axelle_att; returns a pointer on the data buffer at the ith position
LDR R2, [R0]
LDRB R3, [R11, #key] ; imul dstset (key)
EOR R3, R2, R3 ; exclusive OR
STRB R3, [R4]; write decoded character
LDR R2, [R11, #position]
ADD R3, R3, #1 ; increment position
STR R2, [R11, #position]
B Axelle_haveWeFinished
Decrypt it! Method 1

Read the configuration file on the phone

**Easy but beware:** ensure you do not propagate the virus
Decrypt URLs from a debug session - Method 2
Decrypt URLs from a debug session - Method 2
Decrypt URLs from a debug session - Method 2

MEMORY: 006001BC  DCB  0xD7
MEMORY: 006001BD  DCB  0xCB
MEMORY: 006001BE  DCB  0xCB
MEMORY: 006001BF  DCB  0xCF
MEMORY: 006001C0  DCB  0x85
MEMORY: 006001C1  DCB  0x90
MEMORY: 006001C2  DCB  0x90
MEMORY: 006001C3  DCB  0xC8
MEMORY: 006001C4  DCB  0xC8
Decrypt URLs from a debug session - Method 2

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>DCB</th>
<th>0x68</th>
<th>; h</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x74</td>
<td>; t</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x74</td>
<td>; t</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x70</td>
<td>; p</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x3A</td>
<td>; :</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x2F</td>
<td>; /</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x2F</td>
<td>; /</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x77</td>
<td>; w</td>
</tr>
<tr>
<td>MEMORY</td>
<td>DCB</td>
<td>0x77</td>
<td>; w</td>
</tr>
</tbody>
</table>
Apply the XOR key to the ciphertext - Method 3
Apply the XOR key to the ciphertext - Method 3
Java/SmsBoxer.F!tr

- the package contains an encrypted file a.zip and Java classes
- we decompile the classes
- one of the classes shows it loads the a.zip resource and then:

```java
new a(...).field ^ 0x78 ^ 0x78;
```

where the constructor of `a` does:

```java
this.a = (paramInt ^ 0x78);
```

XOR key

value xor 0x78 xor 0x78 xor 0x78 = value xor 0x78
Quick Perl script to decrypt `a.zip`

```perl
$ cat a.zip |
perl -ne 'print pack "C*", map {$_^0x78} unpack "C*", $_'
06159395 smswap 473151350 vsxwap 473159395
smswap 473147122 gywap 473159395 smswap
473159395 smswap 473159395 smswap 473159395 smswap
473159395 smswap 473
```
In the Android Market (removed)

Root the phone (Rage against the cage)

Leak private info, install without consent

Ciphertext and key are hard-coded

```java
arrayOfByte[0] = 94;
arrayOfByte[1] = 42;
arrayOfByte[2] = 93;
...
KEYVALUE = "6^)(9-p35...";
```

XOR encryption

```java
int l = arrayOfByte[j];
int i1 = KEYVALUE[i];
int i2 = (byte)(l ^ i1);
```

Decrypted value

```
http://[CENSORED]45.17:8080/GMServer/GMServlet
```
Introduction

Simple obfuscation

Crypto apprentices

XOR encryption

**DES and AES**
Locating the encryption code
Identifying the key
Decrypting

Conclusion
Modern algorithms in malware

- Android/Geinimi (Jan 2011): DES
- Android/Hongtoutou (Feb 2011): DES
- Android/DrdLight (June 2011): DES
- Android/DroidKungFu (June 2011): AES
- ...
Locating the encryption code: search for KeySpec

In Android/DroidKungFu.A!tr

```java
SecretKeySpec localSecretKeySpec = new SecretKeySpec(arrayOfByte, "AES");
```

In Android/DrdLight.A!tr (smali code)

```java
new-instance v1, Ljavax/crypto/spec/DESKeySpec;
const-string v2, "DDH#X%LT"
invoke-virtual {v2}, Ljava/lang/String;->getBytes()[B
move-result-object v2
invoke-direct {v1, v2}, Ljavax/crypto/spec/DESKeySpec;-><init>([B)V
```

Beware

Several advertisement kits use encryption!
Identify the key

- Search for hard-coded constants

In Android/Geinimi.A!tr

```java
b = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
```

In Android/Hongtoutou.A!tr

```java
const-string v0, "48734154"
```

- Search for the code that creates the KeySpec

In Android/DroidKungFu.A!tr

```java
private static byte[] defPassword = { 70, 117, 99, 107, 95, 115, 69, 120, 121, 45, 97, 76, 108, 33, 80, 119 };
byte[] arrayOfByte = defPassword;
SecretKeySpec localSecretKeySpec = new SecretKeySpec(arrayOfByte, "AES");
```

- Search in assets or resources for unusual data
Decrypt it!

It’s not difficult!

Write a standalone program, and copy/paste the malware’s encryption code!

MyDecrypt.java for Android/Hongtoutou.A!tr

```java
public class MyDecrypt {
    private byte[] desKey;
    public String kk;
    public static String CIPHER = "39...";
    public MyDecrypt() { this.kk = "48734154"; }
    public static String decrypt(String paramString1, String paramString2) throws Exception {
        byte[] arrayOfByte1 = convertHexString(paramString1);
        Cipher localCipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
        byte[] arrayOfByte2 = paramString2.getBytes("UTF-8");
        ... 
    }
}
```
The result

Result=B#1#963a_w1|http://[CENSORED]2.105/g/g.ashx?w=963a_w1|1|http://[CENSORED]2.105/add/pk.aspx$B#1#961a_w1|
http://[CENSORED]2.105/g/g.ashx?w=961a_w1|1|
http://[CENSORED]2.105/add/pk.aspx$B#1#964a_w1|
http://[CENSORED]2.105/g/g.ashx?w=964a_w1|1|
http://[CENSORED]2.105/add/pk.aspx$B#1#978a_w1|
http://[CENSORED]2.105/g/g.ashx?w=978a_w1|1

What is this?

Result=Parameters | URL | Params ...
<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple obfuscation</td>
</tr>
<tr>
<td>Crypto apprentices</td>
</tr>
<tr>
<td>XOR encryption</td>
</tr>
<tr>
<td>DES and AES</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
Noticeable Trends in Crypto

2009
- Handmade
- XOR
- Base64
- Translation
- Java/Swapi
- Java/Esplaw
- WinCE/PmCryptic

2010
- XOR Substitution
- 3DES
- XOR
- SymbOS/Zhaomiao
- WinCE/PhoneCreeper
- Java/SmsBoxer
- Dial/SmsBox
- SymbOS/Yxes
- SymbOS/Shurufa
- WinCE/SejWeek

2011
- Handmade
- DES
- XOR
- AES
- Android/Geinimi - Jan
- Android/Hongtoutuou - Feb
- Android/PjJApps
- Android/DrdDream
- March
- Android/DroidKungFu
- Android/DrdLight
- Android/BaseBridge
- Android/JSmsHider
- June
Noticeable Trends in Crypto

- **2009**: Handmade
  - Java/Swapi
  - Java/Konov
  - Java/Espaw
  - WinCE/PmCryptic
  - Java/SmsBoxer
  - Dial/SmsBox
  - SymbOS/Yxes
  - SymbOS/Shurufa
  - WinCE/SeiWeek
  - SymbOS/Zhaomiao
  - WinCE/PhoneCreek

- **2010**: XOR
  - Base64
  - XOR
  - Substitution
  - 3DES
  - XOR
  - DES
  - XOR
  - AES

- **2011**: DES, AES
  - Handmade
  - Android/Geinimi - Jan
  - Android/Hongtoutu - Feb
  - Android/PJApps
  - Android/DrdDream
  - March
  - Android/DroidKungFu
  - Android/DrdLight
  - Android/BaseBridge
  - Android/JSMSHider
  - June
Noticeable Trends in Crypto

Explanations

- DES, AES in Android API
- Botnets
How efficient is it?

**Efficiency against detection**

**Poor**: AV signatures usually not based on encrypted data. Only efficient against *basic* hash/checksum-based signatures.

**Efficiency against analysis**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time to reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64</td>
<td>😊</td>
</tr>
<tr>
<td>XOR</td>
<td>😞</td>
</tr>
<tr>
<td>AES, DES</td>
<td>😞</td>
</tr>
<tr>
<td>Custom</td>
<td>😞</td>
</tr>
</tbody>
</table>

- Hard-coded keys and ciphertext quite easy to spot
- Assembly more difficult to reverse
You are a ... mobile phone user

*Do not trust your mobile phone (yet)*

You are a Security Researcher, Architect, Cryptographer

- Mobile malware **IS** an issue.
- Mobile malware use crypto, exploits etc. Spread the word.

You are a (nice) developer

- Secret? Don’t put it in the code ;)
- Have a look at Key Agreement schemes

You study malware

- Get your hands on a few mobile malware samples
- Spot the algo, the key, write your own decrypt code
References

- Description of Android/PJApps.A!tr
- Description of Java/Konov.S!tr
- Description of Java/SmsBoxer.F!tr
- Description of Java/SmsBoxer.N!tr
- Description of SymbOS/ShadowSrv.A!tr
- Description of SymbOS/Shurufa.A!tr.dldr
- Description of Android/DrdDream.A!tr

- Description of Android/Hongtoutou.A!tr
- A. Apvrille, Android/DroidKungFu uses AES encryption, June 9, 2011
- A. Lelli, A Smart Worm for a Smartphone – WinCE.PmCryptic.A, June 29, 2009
- T. Strazzere, T. Wyatt, Geinimi Trojan Teardown, January 6, 2011
- D. Maslennikov, Trojan-SMS.WinCE.Sejweek, December 17, 2009
Follow us on http://blog.fortinet.com
or twitter: @FortiGuardLabs

Axelle Apvrille
aka Crypto Girl
/mobile malware reverse engineering/
aapvrille@fortinet.com

Slides edited with LOBSTER