When Crypto Goes Wrong

“We’ve devised a new security encryption code. Each digit is printed upside down.”

Erez Metula (CISSP), Founder
Application Security Expert
ErezMetula@AppSec-Labs.com
Breaking modern crypto is impractical...

- Suppose a device existed that could brute-force a 56-bit key in 1 second
- It would take it 149.7 trillion years to brute-force a 128-bit encryption key.

<table>
<thead>
<tr>
<th>Key size in bits</th>
<th>Permutations</th>
<th>Brute-force time for a device checking $2^{56}$ permutations per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2^8$</td>
<td>0 milliseconds</td>
</tr>
<tr>
<td>40</td>
<td>$2^{40}$</td>
<td>0.015 milliseconds</td>
</tr>
<tr>
<td>56</td>
<td>$2^{56}$</td>
<td>1 second</td>
</tr>
<tr>
<td>64</td>
<td>$2^{64}$</td>
<td>4 minutes 16 seconds</td>
</tr>
<tr>
<td>128</td>
<td>$2^{128}$</td>
<td>149,745,258,842,898 years</td>
</tr>
<tr>
<td>256</td>
<td>$2^{256}$</td>
<td>50,955,671,114,250,100,000,000,000,000,000,000,000,000,000,000,000,000,000 years</td>
</tr>
</tbody>
</table>
To motto of this presentation

- Most ciphers cannot be just cracked in a seasonable time - but must we break it?
All our slides are made from real ingredients.

Real world examples collected from day-do-day application penetration testing.

Based on the true story.
Agenda

- Bad crypto awareness
- Unauthenticated encryption
- Direct access to cryptographic services
- Exposed hashes
- Insecure keys & wrong crypto schema
- Reply attacks
- Crypto-DOS
Bad Crypto Awareness
Home grown algorithms – seen too many of these

```csharp
public static string Encrypt(string textToEncrypt) {
    StringBuilder inSb = new StringBuilder(textToEncrypt);
    StringBuilder outSb = new StringBuilder(textToEncrypt.Length);
    for (int i = 0; i < textToEncrypt.Length; i++) {
        char c = inSb[i];
        c = (char)(((c ^ 153)*2-3)^123);  //data is XORed with some value
        outSb.Append(c);
    }
    return outSb.ToString();
}
```

Frequency analysis
Outdated crypto

- Crypto, like food, can be expired
  - Expired food can make you feel ill
  - Expired crypto can make your data to be exposed
- Examples: MD5, DES
- **DEMO** (md5 collision)
  
  [http://www.mscs.dal.ca/~selinger/md5collision/](http://www.mscs.dal.ca/~selinger/md5collision/)
Bad crypto modes

- Bad crypto is sometimes worse than not doing crypto at all. It gives a false sense of security
  - Bad crypto algorithms & modes
  - Example: good encryption (AES), bad mode (ECB)

Before (cleartext)

After (AES encryption with ECB)
Unauthenticated Encryption – trusting the other side
Forgetting to verify certificates

Often caused by ignorance or by the usage of self signed certs

```
TrustManager[] trustAllCerts = new TrustManager[]{
    new X509TrustManager()
    {
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }
        public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
            String authType) {
        }
        public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
            String authType) {
        }
    };
```

JAVA
Forgetting to verify certificates

```csharp
public static bool ValidateRemoteCertificate(object sender,
X509Certificate certificate, X509Chain chain, SslPolicyErrors policyErrors) {
    return true; //force any the certificate to be accepted
}
```

```objective-c
- (void)connection:(NSURLConnection *)connection
didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
    [challenge.sender useCredential:[NSURLCredential credentialForTrust:challenge.protectionSpace.serverTrust]
    forAuthenticationChallenge:challenge];
    [challenge.sender
        continueWithoutCredentialForAuthenticationChallenge:challenge];
}
```
Forgetting to require https

- HTTPS provides the client with:
  - Transport level encryption
  - Server authentication (based on its cert)
- Breaking the encryption is hard, and replacing the cert will probably fail

- But what happens if we fool it to accept HTTP in the first place?
- **DEMO (if time permits..)**
  - SSLstrip
Direct access to cryptographic services
Direct access to server side crypto functions

- Many times the crypto business logic is exposed at the server side
  - “Please encrypt/decrypt” my data!
- Some examples:
  - http://app/GetEncryptionKey.asmx?messageId=3
  - http://app/decryptData.jsp?block=51937456432651843
  - http://app/getSignature.php?data=some_text_to_sign
Direct access to client side crypto functions

- Often some kind of phishing is involved
  - Client has some kind of client-side component (example: activex) responsible for crypto
  - Client is tricked into visiting the attacker’s site
  - The attacker executes client’s crypto logic
Example – Exposed ActiveX crypto

interface IDataService : IUnknown {

    ...
    virtual HRESULT Encrypt(BSTR* dataToEncrypt, BSTR* output) = 0;
    virtual HRESULT Decrypt(BSTR* dataToDecrypt, BSTR* output) = 0;
    ...
};

```
var myobject;
myobject = new ActiveXObject("DataService");
myobject.decrypt(value);
```
Exposed hashes
Sending hash values over an insecure transport

If hash values match, data is valid
Not using salts (and/or pepper!)

- Having sensitive values (such as passwords) stored as hash is not enough
- Suppose the hashes are somehow stolen
  - Network sniffing
  - SQL Injection
  - Insiders such as admin, DBA’s, etc.
- Hashes without any protection such as salt and/or shared secret MAC (a.k.a “pepper”) are exposed to various attacks

- **DEMO (sha-1 dictionary attack)**
  - [http://www.victim.com/sqlinjectweb](http://www.victim.com/sqlinjectweb)
Insecure keys & Bad selection of crypto schema
Leaving the key near the cipher data

### Users

<table>
<thead>
<tr>
<th>username</th>
<th>password</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>david</td>
<td>ZmRz7JM2NDu12...</td>
<td>USA</td>
</tr>
<tr>
<td>john</td>
<td>NDM1NDc2Hd...</td>
<td>Israel</td>
</tr>
<tr>
<td>michael</td>
<td>Odk4OTdkc2E=...</td>
<td>UK</td>
</tr>
</tbody>
</table>

### Balance

<table>
<thead>
<tr>
<th>account</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MTK3NjQ=...</td>
</tr>
<tr>
<td>2</td>
<td>OTA2Odc=...</td>
</tr>
<tr>
<td>3</td>
<td>MzI2NDu=...</td>
</tr>
</tbody>
</table>

### Encryption Keys

<table>
<thead>
<tr>
<th>dataType</th>
<th>encryptionKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>@wmefkj35834...</td>
</tr>
<tr>
<td>account</td>
<td>$%$(sdsjmhfss...</td>
</tr>
<tr>
<td>logdata</td>
<td>1582Q$fdfsf...</td>
</tr>
</tbody>
</table>

### Audit

<table>
<thead>
<tr>
<th>id</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>dhJodGh2mZdoZGdmaHedydDR50-UldDQzNTQ1NhIzMzQ1MzVy...</td>
</tr>
<tr>
<td></td>
<td>Njc1ODg3QWhnZvF32TQzMjU2NcVqaGdmemV3cmWy2XRyZXRlniu=...</td>
</tr>
</tbody>
</table>
Unprotected encryption keys

- Stored in config files
- Can be exposed by remote file include attacks
  
  http://www.victim.com/SendPdf/WebForm1.asp
  x?file=somefile.pdf

- ..or simply just stored in code

```java
String secret = "lkre943yu943ujf";
byte[] key = key.getBytes();
Cipher c = Cipher.getInstance("AES");
SecretKeySpec k = new SecretKeySpec(key, "AES");
c.init(Cipher.ENCRYPT_MODE, k);
byte[] encryptedData = c.doFinal(dataToSend);
```
Same symmetric key for all clients

- **Scenario:**
  - Legitimate client and server
  - Messages are encrypted using symmetric encryption
  - Encryption key is the same for all users
  - Attacker who puts his hands on the client side app can intercept the communication
Same Asymmetric keys, different deployments

- **Scenario:**
  - Legitimate client and server
  - Messages are encrypted using an Asymmetric encryption
  - Public-Private keypair is the same for all deployments
    - Think of 2 different organizations
  - Attacker who puts his hands on the server side app can expose the private key
  - He can now intercept everything, for each deployment out there.
Same keys, different encryption needs

- Same encryption keys are used for different encryption needs
  - “one key does it all !!”
  - Put all the data at risk, in case compromised

- Scenario:
  - App can be tricked to encrypt/decrypt data of type X where type Y is expected
  - Often combined with chosen plaintext attacks

- DEMO
  - [Link](http://owasp.victimsite.com/getboo/books.php?folderid=CwsL%2BWGKzrc%3D)
Reply attacks
Replying password hashes

- **Scenario**
  - Login page displayed at some client side application
  - Passwords are saved as hash (example: in DB)
  - Since passwords can be sniffed, the developer “protects” the password by calculating a hash at the client side before sending it to the server
  - Login succeeds by comparing the received hash to the stored hash

- But sniffed hash values are as good as the password 😊
Replies important encrypted blocks

- Data is encrypted...
- But what happens in case the attacker reply the same encrypted message again and again?
- Well the message is legitimate 😊
Combining unrelated encrypted blocks

- The application encrypts different values, each pretty much protected by itself
- No correlation between the encrypted blocks
- The attacker combines unrelated legitimate encrypted blocks and sends them to the application!
Crypto-DOS
Crypto-DOS

- Crypto often requires high computational processing power
- We can abuse services making use of crypto behind the scenes to DOS the application

- DEMO – RSA DOS the application by signing large amounts of data

http://www.victim.com/SignatureRSA/RSADoS.aspx
Summary

- In the real world, breaking the crypto function itself is unlikely
- Crypto is often bypassed by exploiting a flaw in the crypto mechanism
- Flaws are caused from various reasons – from lack of awareness related to crypto to logical flaws in the application design, unrelated to crypto at all.
Questions?

Erez Metula

ErezMetula@AppSec-Labs.com
Thank You!

Erez Metula

ErezMetula@AppSec-Labs.com