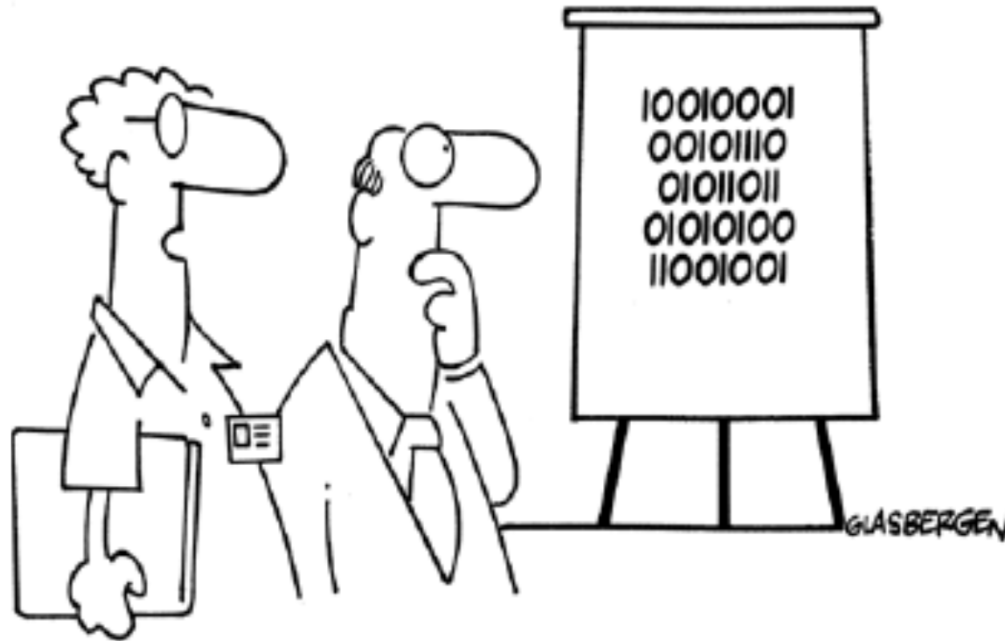


# When Crypto Goes Wrong



**"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](mailto: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..

Key size in bits <sup>[2]</sup>	Permutations	Brute-force time for a device checking $2^{56}$ permutations per second
8	$2^8$	0 milliseconds
40	$2^{40}$	0.015 milliseconds
56	$2^{56}$	1 second
64	$2^{64}$	4 minutes 16 seconds
128	$2^{128}$	149,745,258,842,898 years
256	$2^{256}$	50,955,671,114,250,100,000,000,000,000,000,000,000,000,000,000,000,000,000 years

# 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

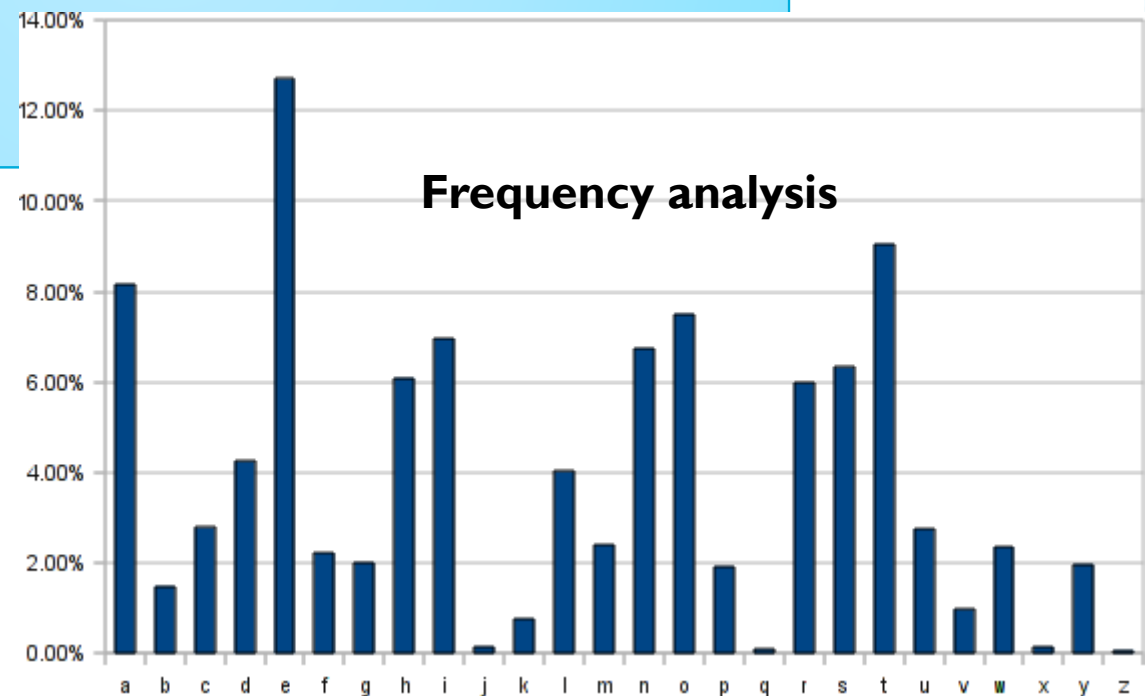
- Bad crypto awareness
- Unauthenticated encryption
- Direct access to cryptographic services
- Exposed hashes
- Insecure keys & wrong crypto schema
- Reply attacks
- Crypto-DOS



# Bad Crypto Awareness

# Home grown algorithms – seen too many of these

```
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();  
}
```





# Outdated crypto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/>



# 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

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

.NET

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

Objective-C  
(Iphone)

# 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.aspx?messageId=3>
  - <http://app/decryptData.jsp?block=51937456432651843>
  - [http://app/getSignature.php?data=some\\_text\\_to\\_sign](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 {
```

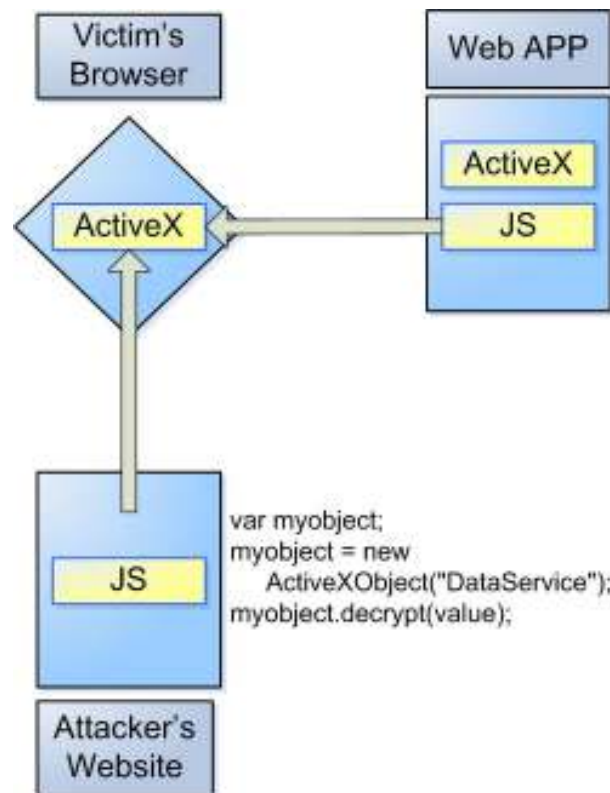
```
...
```

```
virtual HRESULT Encrypt(BSTR* dataToEncrypt, BSTR* output) = 0;
```

```
virtual HRESULT Decrypt(BSTR* dataToDecrypt, BSTR* output) = 0;
```

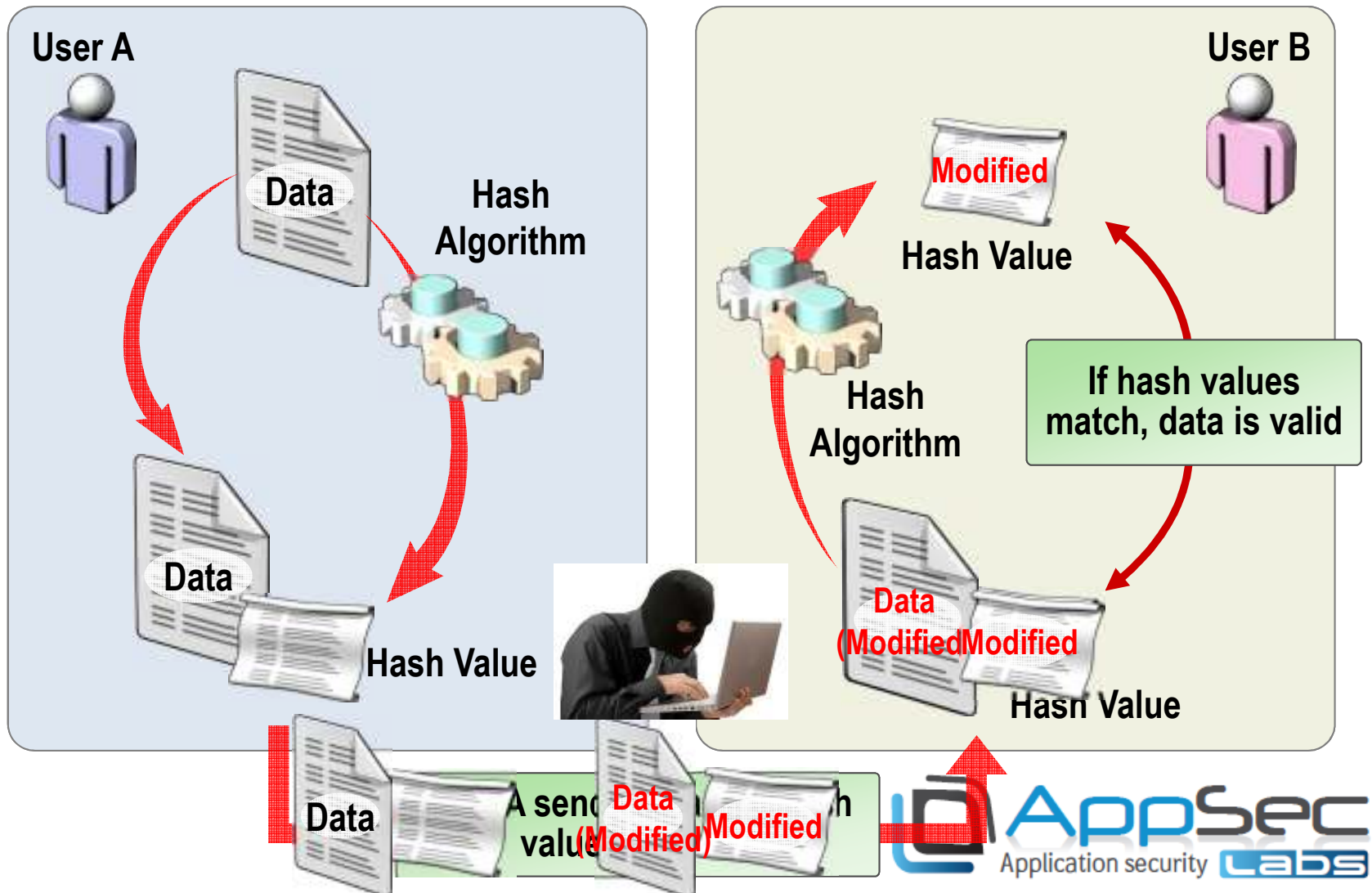
```
...
```

```
};
```



# Exposed hashes

# Sending hash values over an insecure transport



# 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>





Insecure keys & Bad selection of crypto schema

# Leaving the key near the cipher data

Users

username	password	country
david	... ZmRz7jM2NDU2...	USA ...
john	... NDMINDG2dHdl...	Israel ...
michael	... ODK4OTdkc2E= ...	UK

Balance

account	balance
1	... MTK3NjQ= ...
2	... OTA2ODc= ...
3	... MzI2NDU= ...

Encryption Keys

dataType	encryptionKey
password	... )@wmefkj35834...
account	... #\$(sdsjmhfs...
logdata	... :)1592Q\$fgdfss ...

Audit

id	data
1	... dHJodGhqZmdoZGdmaHdydDR5c4UldDQzNTQ1NnIzMzQ1MzVy ...
2	... Njc1ODg3OWhnZnF3ZTZQzMjU2NDVqaGdmcmV3cmVyZXRYZXRIcmU= ...

# Unprotected encryption keys

- Stored in config files
- Can be exposed by remote file include attacks

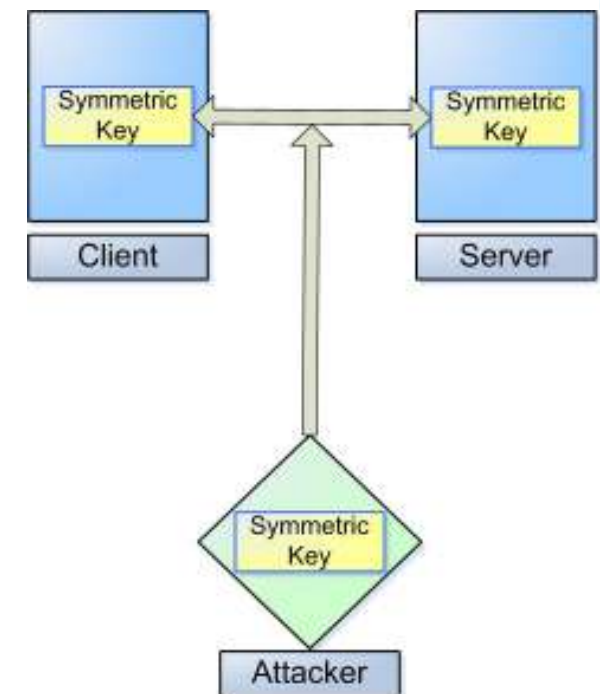
<http://www.victim.com/SendPdf/WebForm1.aspx?file=somefile.pdf>

- ..or simply just stored in code

```
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);
```

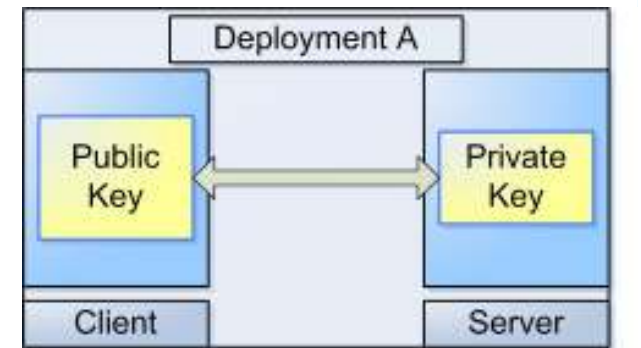
# Same symmetric key for all clients

- 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

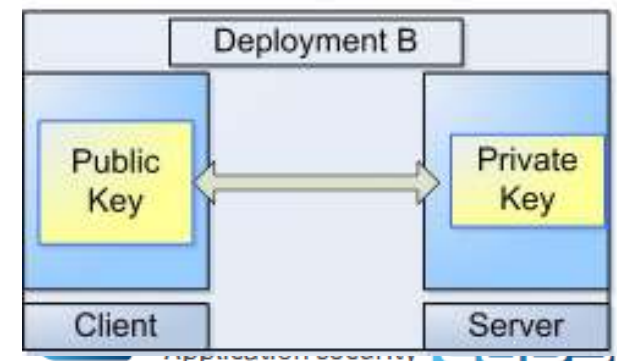


# Same Asymmetric keys, different deployments

- 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!\***



# 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**
- <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 😊

# Replying important encrypted blocks

```
<Transaction>  
  <Source>28747442</Source>  
  <Destination>9837611</Destination>  
  <Amount>10000$</Amount>  
</Transaction>
```

encrypt

```
vd3dKy03D7NiGqouVLYHHqygrukLcEW/HC  
6HRV74pazOScXsDulsLWPQyDWshMdvbln  
jY8mHEsXox3a6SC2EEcWQeCvj+oJbJKyTC  
6B4xMU1uTBMnJalHHzO/JDbZl2uzJmrqdGI  
ptVtKtrAOHEOqMCFXISQVsV+Moby1CyKop.  
ST9GWPXzkvu697ttfmnzrerFzqtHXezM9ZUC  
VdzUUoagzDwRdGI4rTEz0acApuVcjQy9MS.  
EsroGITdJZxmwUS9PtsDI6aLyHboddAJeSL  
m8eq04HuioYkLx22+QqBMcoN++d7BePV03f  
KNcSxsP0+7bArg4jUL0r8YQJ5u4Lf3NcbB12if  
ueYbHA4gNaDhFnsZ1paoCl8f304qBNMGs7  
CwEylBE0qsOBkxPJJGkomouA+olKe+Pclci
```

- 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](mailto:ErezMetula@AppSec-Labs.com)



# Thank You !

**Erez Metula**

[ErezMetula@AppSec-Labs.com](mailto:ErezMetula@AppSec-Labs.com)