TLS / SSLv3 renegotiation vulnerability explained

Thierry ZOLLER
contact@g-sec.lu
http://www.g-sec.lu
http://blog.zoller.lu

G-SEC™ is a non-commercial and independent group of Information Security Specialists based in Luxembourg.

1. Table of Contents

2. Synopsis

Around the 09/11/2009 Marsh Ray, Steve Dispensa and Martin Rex published details[1] about a vulnerability affecting the TLS and the SSLv3 protocol. The vulnerability is being tracked under CVE-2009-3555[2] | VU#120541[3] and affects a multitude of platforms and protocols, the impact of this vulnerability varies from protocol to protocol and from application to application. There is extensive research required in order to assess

When speaking of a "Man in the Middle" attack, it is often assumed that data can be altered or changed. Indeed an attacker that sits in the middle of a connection (hence it's name) is often able to do so. In this particular case however the attacker piggybacks an existing authenticated and encrypted TLS sessions in order to (prefix) inject arbitrary text of its choice. The attacker may not read/alter the other TLS session between the "client" and the "server".

This paper explains the vulnerability for a broader audience and summarizes the information that is currently available. The document is prone to updates and is believed to be accurate by the time of writing.

Important: This vulnerability **is not limited to HTTPS**, this vulnerability potentially affects every application/protocol that implements TLS or SSLv3.

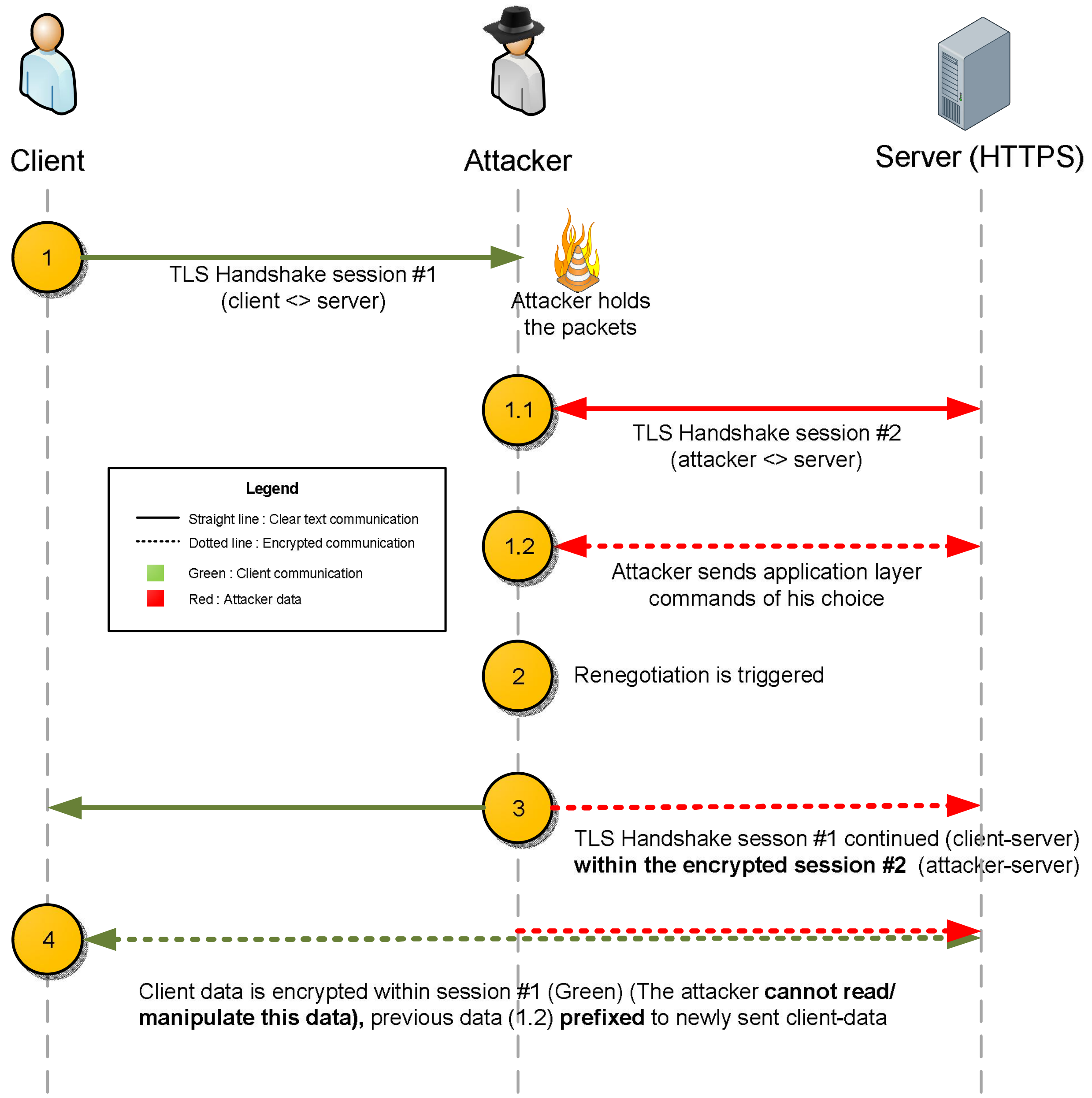
This paper is referenced by the US-CERT, DFN-CERT, BELNET-CERT, SWITCH-cert,Nessus, Qualys, c't Heise, and many more. Furthermore it has served as a internal Training paper for a major OS vendor.

[1] http://www.extendedsubset.com/
[2] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555
[3] http://www.kb.cert.org/vuls/id/120541

3. Revisions

Version	*Date*	*Annotations*
0.8	09.11.2009	Initial draft
0.81	10.11.2009	Adding general and specific example
0.9	12.11.2009	Added vulnerability requirements, protocol overview
0.91	12.11.2009	Initial public draft release at http://www.g-sec.lu/
0.92	13.11.2009	Corrected few errors
0.93	17.11.2009	Added test cases and SMTP over TLS details
0.95	24.11.2009	Added FTPS details, fixed syntax and formatting errors, added IIS7 clarifications
0.96	25.11.2009	New test cases
0.97	27.11.2009	Added HTTPS TRACE and HTTPS to HTTP downgrade attack
0.98	29.11.2009	Added 2 POC files for the TRACE and HTTPS to HTTP downgrade attack
0.99	22.12.2011	Added the correction sent in by Alun Jones
1.0	23.12.2011	Grammar, Better PDF support, release of Final version

4. Generic TLS renegotiation prefix injection vulnerability


Client
Attacker
Server (HTTPS)
1
TLS Handshake session #1
(client <> server)
Attacker holds
the packets
1.1
TLS Handshake session #2
(attacker <> server)
Legend
Straight line : Clear text communication
Dotted line : Encrypted communication
🟩 Green : Client communication
🟥 Red : Attacker data
1.2
Attacker sends application layer
commands of his choice
2
Renegotiation is triggered
3
TLS Handshake sesson #1 continued (client-server)
within the encrypted session #2 (attacker-server)
4
Client data is encrypted within session #1 (Green) (The attacker **cannot read/**
manipulate this data), previous data (1.2) **prefixed** to newly sent client-data

4.1. Details

1 “Client” starts the TLS handshake – Attacker does not forward these immediately

1.1 The attacker negotiates a new session performs a full TLS exchange

1.2 The attacker sends application level commands over the previously established TLS session (#2)

2 Renegotiation is triggered either

1. because of Certificate based auth (server sees get /dir and decides it needs an certificate for „directory“)
2. due to different cipher requriements on different ressources (Server initiated)
3. by the client

3 The TLS handshake started at 1 and hold back by the attacker, is now being let to the server which performs a new TLS Handshake **over the previously established encrypted TLS session #2** (Attacker<>Server)

4 The TLS endpoint, due to the renegotiation has to take into the account the previously sent data (per spec), the endpoint believes the previous data (1.2) to have been send from the same client. As such this request is **prefixed to the one issued by the client in 4** (See HTTPS example for a more explicit example)

5. HTTPS

There are multiple ways to abuse HTTPS in order to inject traffic into an authenticated stream. An additional attack vector was uncovered by Frank Heidt (Leviathan Security) but not published and rediscovered by Thierry Zoller (G-SEC) for this paper, this vector allows downgrading an existing SSL session to plain text.

This paper will present 2 new methods to leverage the TLS renegotiation vulnerability

1. **Injecting plaintext and abusing using X-Ignore:/n** (Marsh Ray) or using unfinished POST to a web application reflecting content (Anil Kurmus)

   Summary: The attacker injects (prepends) GET/POST HTTP commands and does not terminate the last command (i.e no CRLF) that way when both http requests (Attacker, Victim) merge, part of the victim requests are ignored)

2. **Downgrading from HTTPS to HTTP** and performing active Man-in-the-Middle – according to an online article this was discovered by Frank Heidt but choosen not to disclose[4] the details; details have been rediscovered for this paper by Thierry Zoller (G-SEC)

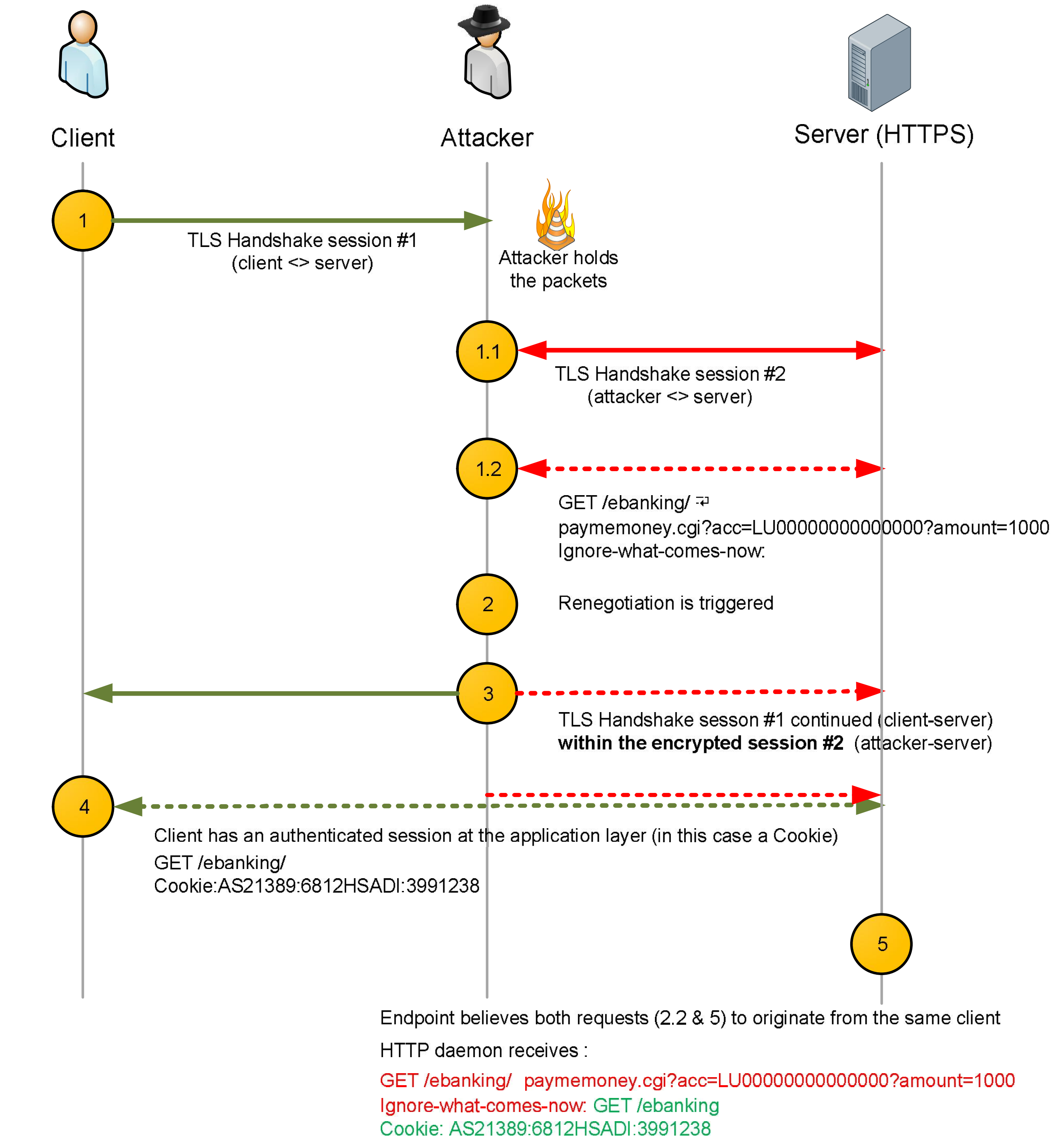
   Summary: The attacker injects (prepend) a HTTP request to a resource reachable over SSL but redirecting the client to HTTP when requested. Such behavior is a common occurrence.

3. **"When TRACE comes back to bite you"** – After ideas to use TRACE method to leverage this flaw appeared in twitter (olle@toolcrypt.org and sirdarckcat) this method was researched and turned into a POC by Thierry Zoller.

   Summary: The attacker injects a TRACE command, by doing so the attacker can control the content that is send from the server to the victim over HTTPS

[4] http://www.pcworld.com/article/182720/security_pro_says_new_ssl_attack_can_hit_many_sites.html

5.1. First method - Injecting commands into an HTTPS session


Client
Attacker
Server (HTTPS)
1
TLS Handshake session #1
(client <> server)
Attacker holds
the packets
1.1
TLS Handshake session #2
(attacker <> server)
1.2
GET /ebanking/
paymemoney.cgi?acc=LU00000000000000?amount=1000
Ignore-what-comes-now:
2
Renegotiation is triggered
3
TLS Handshake sesson #1 continued (client-server)
within the encrypted session #2 (attacker-server)
4
Client has an authenticated session at the application layer (in this case a Cookie)
GET /ebanking/
Cookie:AS21389:6812HSADI:3991238
5
Endpoint believes both requests (2.2 & 5) to originate from the same client
HTTP daemon receives :
GET /ebanking/ paymemoney.cgi?acc=LU00000000000000?amount=1000
Ignore-what-comes-now: GET /ebanking
Cookie: AS21389:6812HSADI:3991238

5.2. Details

This is one example of how this vulnerability might be used to affect HTTPS. We are aware that in this case a simple XSRF[5] attack could have achieved the same effect, however this is a easy to understand example. This attack can be used to abuse specific features of the affected web application, for example a POC has been demonstrated on how to steal Twitter credentials using this flaw[6].

1.1 The attacker negotiates a new session performs a full TLS exchange

1.2 The attacker sends a GET request to a fictional weak e-banking application. Note that the attacker can **add multiple requests** due to HTTP1.1 pipelining but that only the last request usurps the cookie.

2 Renegotiation is triggered

3 The TLS handshake started at 1 and hold back by the attacker, is now being let to the server which performs a new TLS Handshake **over the previously established encrypted TLS session #2** (Attacker<>Server)

4 The TLS endpoint, due to the renegotiation has to take into the account the previously sent data (per spec), the endpoint believes the previous data (1.2) to have been send from the same client

The requests

1.2 : Attacker -> server

```
GET /ebanking/  paymemoney.cgi?acc=LU00000000000000?amount=1000
Ignore-what-comes-now:
```

And

4: Client->server

```
GET /ebanking
Cookie: AS21389:6812HSADI:3991238
```

5 The request is prefixed to the request issued by the client in (4) and is merged into

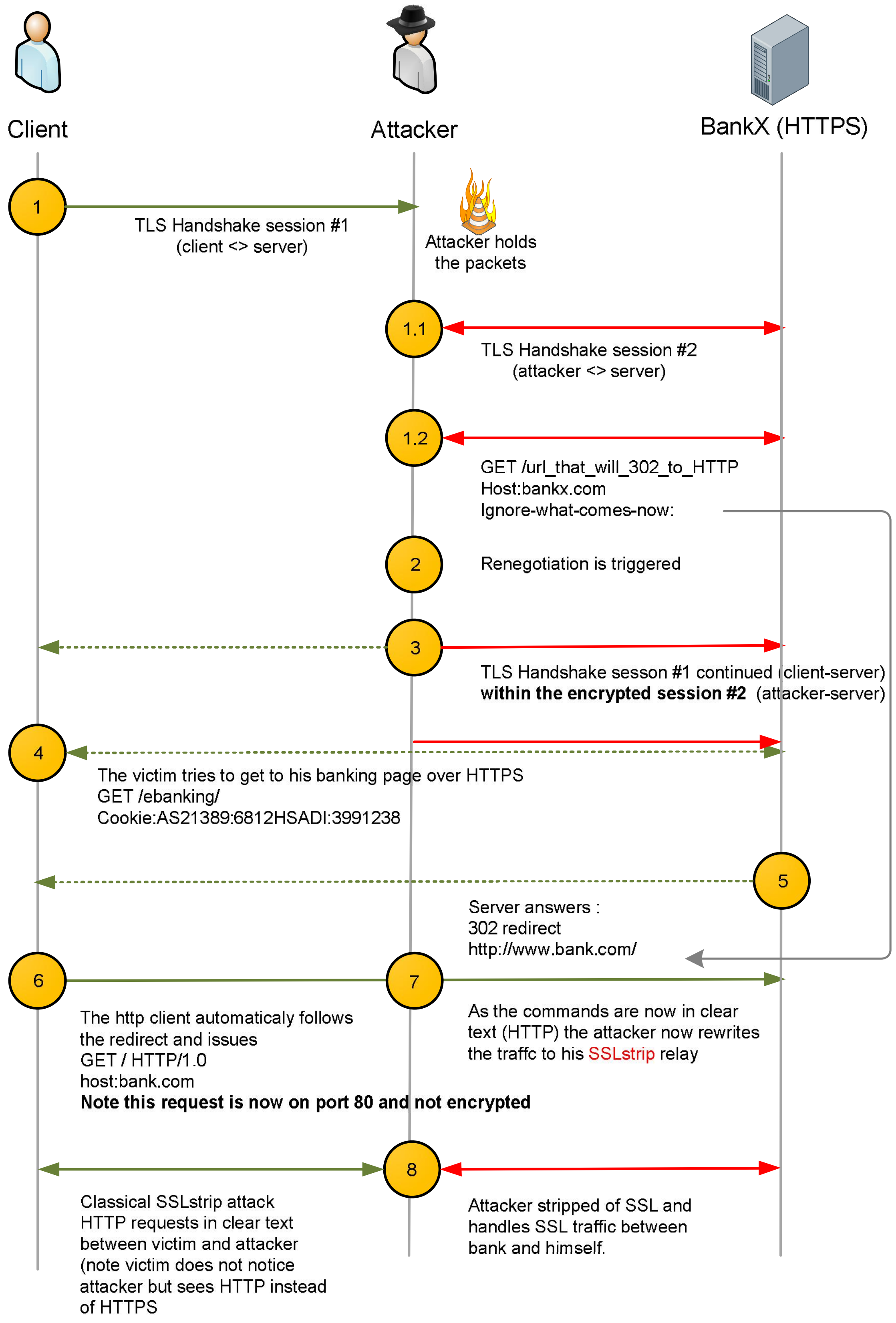
```
GET /ebanking/  paymemoney.cgi?acc=LU00000000000000?amount=1000
Ignore-what-comes-now: GET /ebanking
Cookie: AS21389:6812HSADI:3991238
```

Interpreted by the HTTP daemon as :

```
GET /ebanking/  paymemoney.cgi?acc=LU00000000000000?amount=1000
Cookie: AS21389:6812HSADI:3991238
```

[5] http://en.wikipedia.org/wiki/Cross-Site_Request_Forgery
[6] http://www.securegoose.org/2009/11/tls-renegotiation-vulnerability-cve.html

5.3.Second method - HTTPS to HTTP downgrade attack


Client
Attacker
BankX (HTTPS)
1
TLS Handshake session #1
(client <> server)
Attacker holds
the packets
1.1
TLS Handshake session #2
(attacker <> server)
1.2
GET /url_that_will_302_to_HTTP
Host:bankx.com
Ignore-what-comes-now:
2
Renegotiation is triggered
3
TLS Handshake sesson #1 continued (client-server)
within the encrypted session #2 (attacker-server)
4
The victim tries to get to his banking page over HTTPS
GET /ebanking/
Cookie:AS21389:6812HSADI:3991238
5
Server answers :
302 redirect
http://www.bank.com/
6
The http client automaticaly follows
the redirect and issues
GET / HTTP/1.0
host:bank.com
Note this request is now on port 80 and not encrypted
7
As the commands are now in clear
text (HTTP) the attacker now rewrites
the traffc to his SSLstrip relay
8
Classical SSLstrip attack
HTTP requests in clear text
between victim and attacker
(note victim does not notice
attacker but sees HTTP instead
of HTTPS
Attacker stripped of SSL and
handles SSL traffic between
bank and himself.

5.4. Details

SSLstrip[7] is a tool presented by Marlin Spikes at Blackhat 2009 - it allows to perform an active MITM attack by stripping of SSL from the connection of the victim. The attack had one particular drawback: it was not possible to downgrade an existing SSL session, and only worked if the user accesses his bank over HTTP first then trying to submit his credentials to HTTPS.

Abusing the TLS renegotiation vulnerability however **it is now possible to even apply SSLstrip to established SSL connections.**

The Proof of concept for this attack can be found at : http://www.g-sec.lu/tls-ssl-proof-of-concept.html

1.2 The attacker sends a GET request he knows will redirect the HTTP client to a non HTTPS page on the server.

2 Renegotiation is triggered

3 The TLS handshake started at 1 and hold back by the attacker, is now being let to the server which performs a new TLS Handshake **over the previously established encrypted TLS session #2** (Attacker<>Server)

4 The TLS endpoint, due to the renegotiation has to take into the account the previously sent data (per spec), The request is prefixed to the request issued by the client in (4) and is merged into one request. The attacker effectively replaced the GET request.

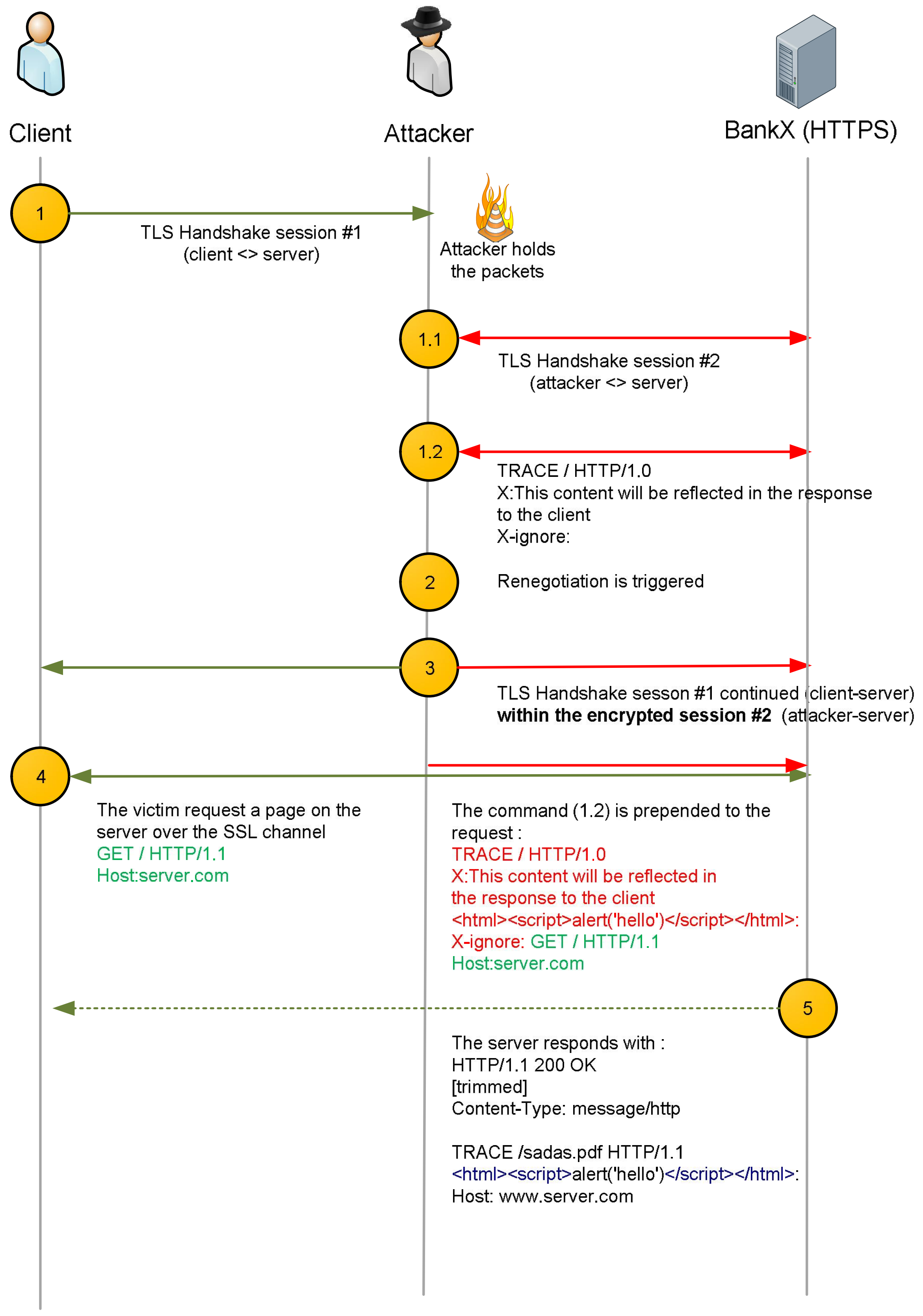
5 The server replies with a 302 redirecting the victim to an HTTP page

6 The HTTP browser of the victim automatically follows the redirect the server has sent and requests the HTTP page.

7 The attacker is now seeing the clear text requests and can rewrite the HTTP request from the victim to his liking – from this point on the attacker continues the attack with SSLtrip

[7] http://www.thoughtcrime.org/software/sslstrip/

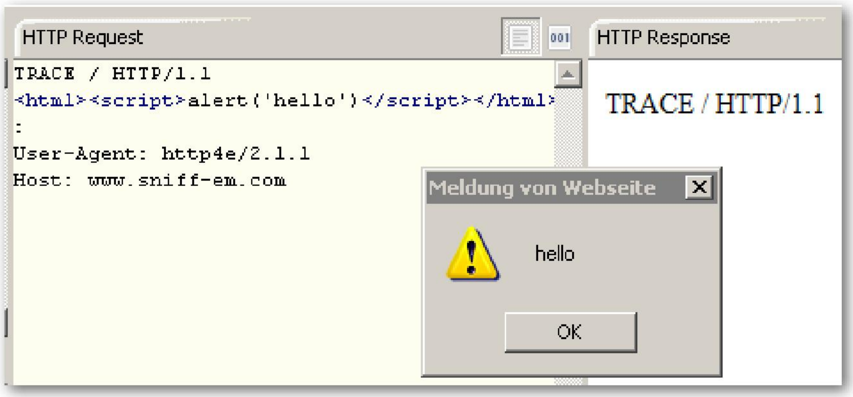
5.5. Third method - Injecting custom responses through TRACE


Client
Attacker
BankX (HTTPS)
1
TLS Handshake session #1
(client <> server)
Attacker holds
the packets
1.1
TLS Handshake session #2
(attacker <> server)
1.2
TRACE / HTTP/1.0
X:This content will be reflected in the response
to the client
X-ignore:
2
Renegotiation is triggered
3
TLS Handshake sesson #1 continued (client-server)
within the encrypted session #2 (attacker-server)
4
The victim request a page on the
server over the SSL channel
GET / HTTP/1.1
Host:server.com
The command (1.2) is prepended to the
request :
TRACE / HTTP/1.0
X:This content will be reflected in
the response to the client
<html><script>alert('hello')</script></html>:
X-ignore: GET / HTTP/1.1
Host:server.com
5
The server responds with :
HTTP/1.1 200 OK
[trimmed]
Content-Type: message/http
TRACE /sadas.pdf HTTP/1.1
<html><script>alert('hello')</script></html>:
Host: www.server.com

5.6. Details

TRACE allows the attacker to control the response from the server to the client, contrary to the original attack that only allowed controlling the **request** to the server, using trace gives control over the response (within certain limits).

At the moment is believed that TRACE is unlikely to be used to execute client-side javascript code, this is due to the "content-type:message/http" header the servers adds to the response and prompts the browser to initiate a download. Binary content injection through TRACE also appears not to be possible as the filename the browser saves the data into is not controlled by the attacker. There are several third-party browsers that use own sockets to send/receive http data and use the TRIDENT engine (mshtml.dll) to render the WebPages. **These implementations are vulnerable to JavaScript injection**. The reason is that the IE component does not see the HTTP header and renders that data as if it would be HTML.



```
TRACE / HTTP/1.1
<html><script>alert('hello')</script></html>
:
User-Agent: http4e/2.1.1
Host: www.sniff-em.com
```

TRACE / HTTP/1.1

hello

The TRACE method can also be abused for example in situations where custom code is used that ignores the content-type and just parses for specific data.

For instance one can imagine that several automatic updates protocols and server to server communications are vulnerable to this attack. As the client expects a response to a GET request it is likely that developers have not spend time to look into whether the response really comes from a GET request.

Summary: The attacker injects a TRACE command, by doing so the attacker can control the content that is send from the server to the victim over HTTPS

The Proof of concept for this attack can be found at : http://www.g-sec.lu/tls-ssl-proof-of-concept.html

6. SMTPS

There are 2 major ways to use TLS with SMTP – STARTTLS and TLS from the beginning. With STARTTLS you connect to the SMTP port using plain text and then request a TLS connection using the command "STARTTLS".

T. ZOLLER (G-SEC) as well as W.VENEMA (Postfix) have researched this protocol independently, the following represents a summary of what is currently known. VENEMA has published a PDF that summarizes his views[8].

Discussions with M. VENEMA resulted in the following information based attacks in-line with protocol requirements. A successful attack requires an SMTP server that uses a TLS engine that reads the data as soon as it arrives, vendors are required to assess the products in order to determine if their product is vulnerable. Currently there are no independant research results for SMTP. As an example of software that uses TLS engine in a way required for these attacks to work, VENEMA cited STUNNEL.

Protocol vulnerability matrix

The following information is believed to be correct by the time of writing

The attacker does NOT have an account on the SMTP server

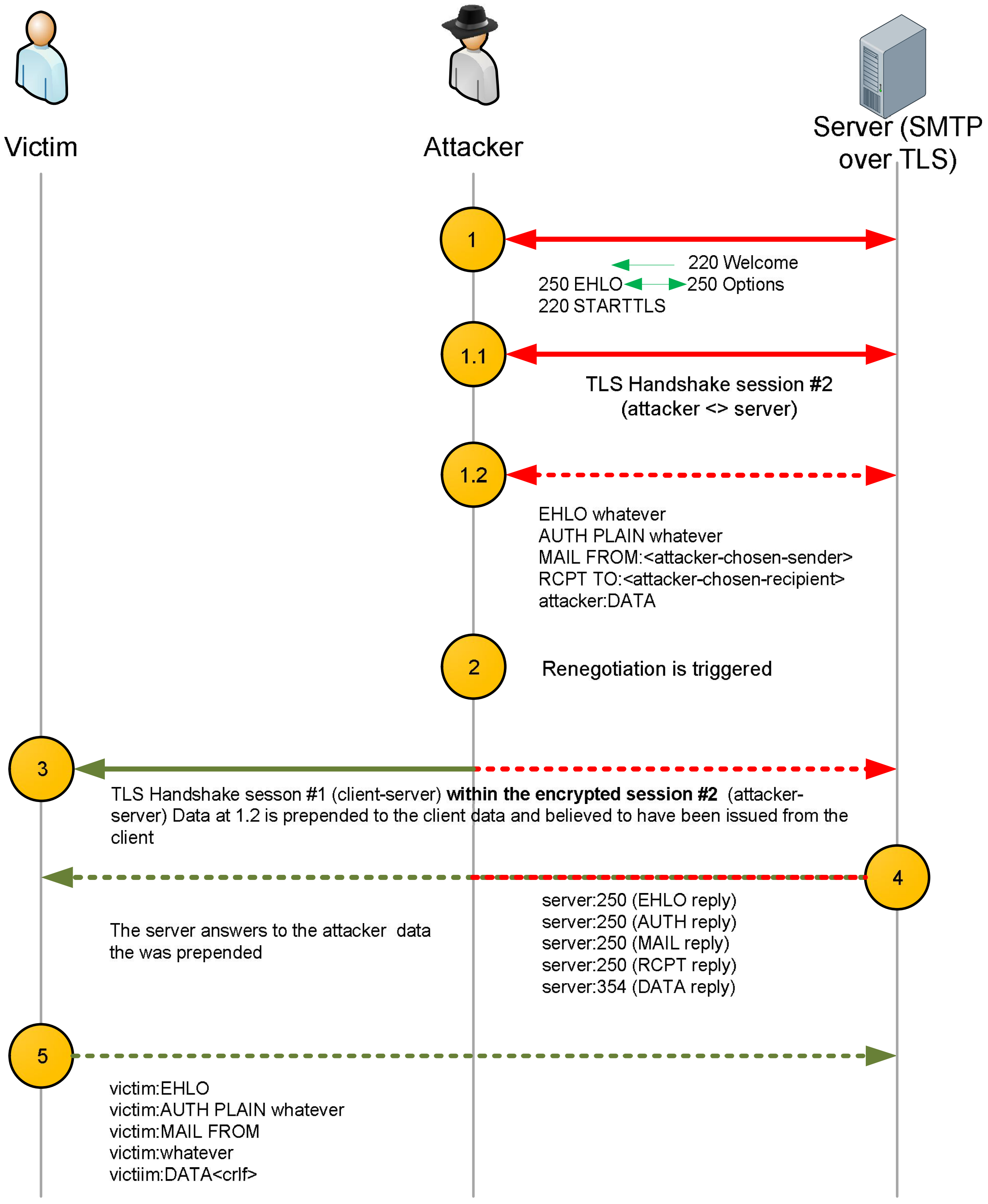
Attack theoretically possible if	TLS private cert authentication without SASL
	SMTP over TLS without SASL

The Attacker has an account on the SMTP server

Attack theoretically possible if	TLS private cert authentication without SASL
	TLS private cert authentication with SASL
	SMTP over TLS with SASL
	SMTP over TLS without SASL

[8] http://www.porcupine.org/postfix-mirror/smtp-renegotiate.pdf

Attack scenario - SMTP STARTTLS (110)


Victim
Attacker
Server (SMTP over TLS)
1
220 Welcome
250 EHLO
250 Options
220 STARTTLS
1.1
TLS Handshake session #2 (attacker <> server)
1.2
EHLO whatever
AUTH PLAIN whatever
MAIL FROM:<attacker-chosen-sender>
RCPT TO:<attacker-chosen-recipient>
attacker:DATA
2
Renegotiation is triggered
3
TLS Handshake sesson #1 (client-server) **within the encrypted session #2** (attacker-server) Data at 1.2 is prepended to the client data and believed to have been issued from the client
4
server:250 (EHLO reply)
server:250 (AUTH reply)
server:250 (MAIL reply)
server:250 (RCPT reply)
server:354 (DATA reply)
The server answers to the attacker data the was prepended
5
victim:EHLO
victim:AUTH PLAIN whatever
victim:MAIL FROM
victim:whatever
victiim:DATA<crlf>

Details

This is a complex example of how this vulnerability could be used to exploit SMTP over TLS (STARTSSL) **if the attacker has an account**

1 Attacker connects to the SMTP server and initiates a TLS session (STARTTLS)

1.1 The attacker negotiates a new session performs a full TLS exchange

1.2 The attacker sends SMTP commands to the server but does not end the SMTP session, in this example the attacker controls the source and destination e-mail addresses.

2 Renegotiation is triggered

3 Attacker initiates a TLS session (TLS HELLO) and the victim performs a new TLS Handshake **over the previously established encrypted TLS session #2** (Attacker<>Server)

4 The TLS endpoint, due to the renegotiation has to take into the account the previously sent data (per spec), the endpoint believes the previous data (1.2) to have been send from the same client

As such the client now receives the answers from the attacker injected commands (note this is a way to detect this attack on the client-side).

5 The victim SMTP client now issues his commands to send mail – Those commands end up in the BODY of the mail previously started by the attacker.

The SMTP server receives:

```
EHLO whatever
AUTH PLAIN whatever
MAIL FROM:<attacker-chosen-sender>
RCPT TO:<attacker-chosen-recipient>
attacker:DATA
victim:EHLO
victim:AUTH PLAIN whatever
victim:MAIL FROM
victim:whatever
victiim:DATA<crlf>
```

As such the :<attacker-chosen-recipient> receives a mail containing the authentication data aswell as the other data.

Client side attack detection

Contrary to HTTPS protocol the client has a way to detect that he was attacked at the application layer as the server replies arrive before the victim even sent the commands.

Important Note

To our knowledge POSTFIX is not affected by this vulnerability.

7. FTPS

FTPS is a SSL/TLS based implementation of FTP, it is **not to be confused with SFTP** (Alun Jones: "SFTP is a completely different protocol – a sub-protocol of SSH. For example, where FTP uses commands of three or four letters, SFTP uses binary single byte instructions.").

Alun Jones[9] author of WFTPD has done an impact analysis on FTPS implementations and the potential vulnerabilities that might be present, the analysis contains an interesting note about degrading encryption for sake of NAT compatibility which has impact beyond the TLS/SSL renegotiation vulnerability.

FTPS is particularly interesting because it has two channels, the CONTROL channel and the DATA channel which can be requested to be encrypted separately. Please refer to the description at http://www.allaboutjake.com/network/linksys/ftp/ for more insight into FTP connections.

The following information is believed to be correct by the time of writing; the bespoken possible vulnerabilities are depended on specific FTP client and FTP server implementations. Vendors need to look into these particular implementations and fix them accordingly.

[9] http://msmvps.com/blogs/alunj/default.aspx

Client certificate based authentication (Control Channel)

Contrary to HTTPS, FTPS Client-certificate based auth is unlikely to be vulnerable. Indeed HTTPS client certificate based authentication it is vulnerable due to a particular circumstance being the necessity to receive a particular request to a directory prior to choosing whether a certificate is required or not, the HTTP **server then needs to renegotiate**.

This is not the case with FTPS, the connection is encrypted from the very start or it isn't, **it is unlikely the server supports renegotiation at that point**. "FTPS resets the command state and commands are executed in the context that existed at that time, rather than the newly-authenticated context."

Renegotiations due to NAT support (Data Channel)

NAT devices need to keep track of connections and have support to rewrite FTP connections on the fly in order to allow FTP to work over NAT. The problem that appeared with FTPS is that the NAT devices could not peek into the PASV or PORT commands any longer and as such would not be able to NAT FTP.

Because of this and to be able to offer FTPS over NAT, several vendors added support for the CCC [10]command (Clear Command Channel). **The FTP server will drop the secure connection** in order for the NAT device to be able to rewrite PASV and PORT commands. This allows the attacker to see the control channel in clear text. This allows the attacker to know WHEN and WHAT files are currently being transferred, if the server accepts TLS renegotiations the attacker might then, timed to the clear text control channel, inject data into files being uploaded, by renegotiating at the beginning of a new file transfer.

Abuse case : Client uploads a binary file, attacker injects binary code of his choice.

Resetting the TCP connection and injecting in mid transfer

According to Alun Jones[11], a lot of FTP clients do not properly terminate the TLS Session. Instead of sending a TLSshutdown messsage, the clients terminate the TCP session beneath (RST,FIN), for this reason a lot of FTP servers have support for these border cases and do not report these conneciton terminations as errors.

This however allows for a clever attack to be done. The Attacker can close the TCP connection between the victim and the server by sending the specific TCP packet. The FTP client will the n try to resume the upload using REST[12] as the attacker (due to CCC) has access to this data he exactly knows what part of the file the victim will resume and by renegotiation TLS he can

[10] http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=/rzaiq/rzaiqserversubcommandccc.htm
[11] http://msmvps.com/blogs/alunj/default.aspx
[12] http://www.math.ntnu.no/mirror/www.qmail.org/koobera/www/ftp/retr.html

prepend parts of the transfer. Additional the attacker may modify the REST command so that the server resumes at the byte location he wants.

8. The Impact on other protocols using TLS

The impact of this vulnerability is different from one protocol to another. Several stateless protocols like HTTP for instance, merge both sessions into one, making it possible for the attacker to inject arbitrary plain text into the stream that is processed by the end stream as coming from the same destination

This breaks a principal assumption made by application developers and has impacts on innumerable number of custom implementations.

9. Summary

Protocol	Impact analysis available	Current status
HTTPS	Yes	1. Vulnerable to a certain degree, impact depends on application level logic and structure of the HTTP requests. 2. If server supports TRACE command, attacker can control the response 3. Attacker can downgrade to HTTP (sslstrip)
EAP-TLS	Online discussions[13]	Believed to not be vulnerable
IMAPS	No	Unknown
POP3S	No	Unknown
LDAPS	No	Unknown
SMTPS	Yes	Vulnerable only if certain requirements are met
FTPS	Yes	Vulnerable - Further research required

Application	Impact analysis available	Current status
OpenVPN	Partially (vendor)	Not vulnerable, does not rely on openssl session capabilities – session handling was hardened after disclosure reports[14]
Tomcat	Partially (vendor)	Vulnerable[15] - mitigations exist
Apache	Available	Vulnerable – short term patch available[16]
IIS 7 <=7.5	Available	Vulnerable – not vulnerable to client initiated renegotiation requests.
GNUtls	Available	Vulnerable – patch status unknown, IETF proposal currently being implemented
OpenSSL	Available	Vulnerable – short term patches available

[13] http://www.ietf.org/mail-archive/web/tls/current/msg04109.html
[14] http://www.pubbs.net/openvpn/200911/19535/
[15] http://www.mail-archive.com/users@tomcat.apache.org/msg69335.html
[16] http://marc.info/?l=apache-httpd-announce&m=125755783724966&w=2

Please refer to VU#120541 and BID36935 for an updated list of applications

EAP-TLS

EAP-TLS is not believed to be vulnerable if implemented as per specification[17].

- There is no application layer protocol involved when EAP-TLS is executed
- Only the TLS key material is used, the tunnel is not used.
- EAP re-authentication not the same as TLS renegotiation which is executed in the previous TLS tunnel

[17] http://www.ietf.org/mail-archive/web/tls/current/msg04109.html

10. Solutions

1. Proposed IETF solution

The IETF draft proposed by E. Rescorla, M. Ray, S. Dispensa, N. Oskov offers an elegant way to solve the problem.

The Draft proposes a new TLS extension that cryptographically binds TLS sessions to clients and further allows informing clients about renegotiations. Furthermore the proposed solution allows working with a defined rule set that allows either - Never to renegotiate - Only renegotiate if TLS negotiation extension is being used or Renegotiate anyways

As to our information all major vendors are currently implementing above proposed solution.

11. Patching TLS

From the conditions that emerged in "Vulnerability conditions" the patching requirements might be:

Client

- Mid-term : Implement the IETF proposal for a TLS extension tracking and handling renegotiation requests[18] (draft-rescorla-tls-renegotiation-00.txt)

Server

- Short-term : Remove renegotiation capabilities altogether
- Mid-term : Implement the IETF proposal for a TLS extension tracking and handling renegotiation requests[19] (draft-rescorla-tls-renegotiation-00.txt)

12. Patching SSLv3

The only way to fix the renegotiation vulnerability for SSLv3 is to disable renegotiation on the server side completely. SSLv3 **does not support extensions** and as such cannot use the proposed extension mentioned above.

[18] https://svn.resiprocate.org/rep/ietf-drafts/ekr/draft-rescorla-tls-renegotiate.txt
[19] https://svn.resiprocate.org/rep/ietf-drafts/ekr/draft-rescorla-tls-renegotiate.txt

13. Testing for a renegotiation vulnerability

The toolset provided by Openssl[20] offers the simplest way to test whether a server allows for client-side renegotiation in the established tunnel. Note: This doesn't necessarily mean that the application beneath is vulnerable to attacks over this channel, but indicates the server allows attacks to happen.

Vulnerability requirements

The preconditions for a TLS or SSLv3 connection to be vulnerable are

1. The server acknowledges and accepts full TLS renegotiations in the middle of a connection and after the initial handshake
   and
2. The server assumes that both TLS sessions were negotiated with the same client
   and
3. The server treats both sessions as one and merges them at the application layer

As such this vulnerability might not been seen as a vulnerability in TLS but the as the bad choice to merge two different requests together by the endpoint.

Generic Example

```
Openssl s_client -connect yourserver.com:443
GET / HTTP/1.0
Host:yourserver.com
R (Triggers renegotiation – if this works, the server accepts renegotiations
within an existing TLS session Req. #1)
CRLF
<server responds with content> (server merged both sessions Req. #2)
```

Patched server with disabled renegotiation

```
Openssl s_client -connect yourserver.com:443
R (Triggers renegotiation)
2860:error:1409444C:SSL routines:SSL3_READ_BYTES:tlsv1 alert no
renegotiation:./ ssl/s3_pkt.c:1053:SSL alert number 100
```

[20] http://www.openssl.org/

14. Conclusions

The vulnerability lies within the core of TLS and SSLv3, and will rear its ugly head for years to come; the custom applications that are potentially vulnerable are innumerable.

Servers

- Servers that do allow mid-connection renegotiations are vulnerable
- Applications that handle 2 TLS sessions as coming from the same client are vulnerable

Clients

- Clients have no means (pre TLS extension) to check if a renegotiation is happening and are vulnerable

Sources

1. http://www.securityfocus.com/bid/36935
2. https://svn.resiprocate.org/rep/ietf-drafts/ekr/draft-rescorla-tls-renegotiate.txt
3. https://bugzilla.mozilla.org/show_bug.cgi?id=526689
4. http://blog.ivanristic.com/2009/11/ssl-and-tls-authentication-gap-vulnerability-discovered.html
5. http://www.leviathansecurity.com/pdf/ssltlstest.zip
6. http://extendedsubset.com/renegotiating_tls_20091104_pub.zip
7. https://bugzilla.redhat.com/show_bug.cgi?id=533125
8. http://www.mail-archive.com/users@tomcat.apache.org/msg69335.html
9. http://www.apache.org/dist/httpd/patches/apply_to_2.2.14/CVE-2009-3555-2.2.patch
10. http://sid.rstack.org/blog/index.php/373-tls-tout-le-monde-en-parle-pourquoi-pas-moi
11. https://www.mikestoolbox.net/
12. http://extendedsubset.com/
13. http://extendedsubset.com/Renegotiating_TLS.pdf
14. http://extendedsubset.com/Renegotiating_TLS_pd.pdf
15. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555
16. http://www.chappellseminars.com/files/nutter-stevedispensa-sslgap.mp3
17. http://www.phonefactor.com/sslgap/ssl-tls-authentication-patches

Thanks

We would like to thank Marsh Ray, Alun Jones, Wietse Venema, Alexandre Dulaunoy, Noam Rathaus, j.clausing and Simon Zuckerbraun.

15. Disclaimer

Information is believed to be accurate by the time of writing. As this vulnerability is complex this document may be prone to revisions in the future.