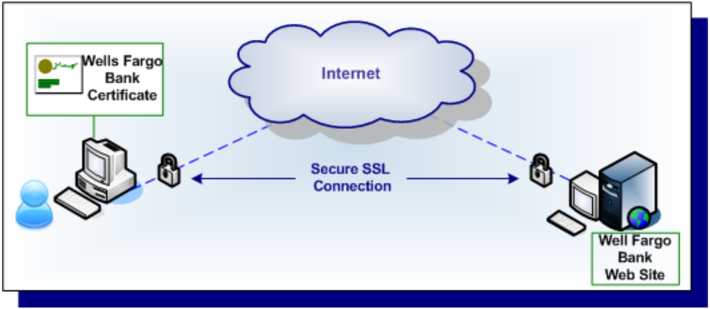
The SSL/TLS protocol

Myrto Arapinis
School of Informatics
University of Edinburgh

October 30, 2017

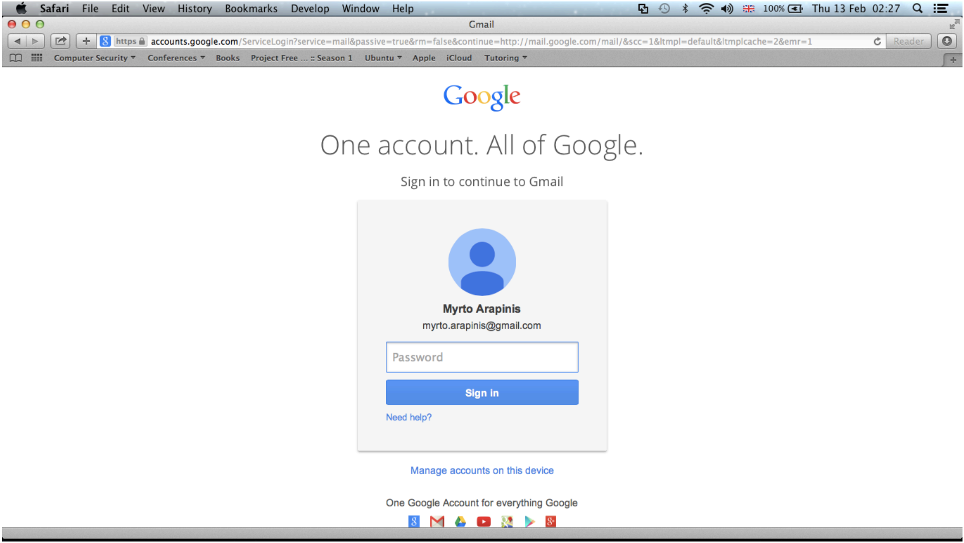
SSL/TLS protocol

Goals: Confidentiality, Integrity, Non repudiation

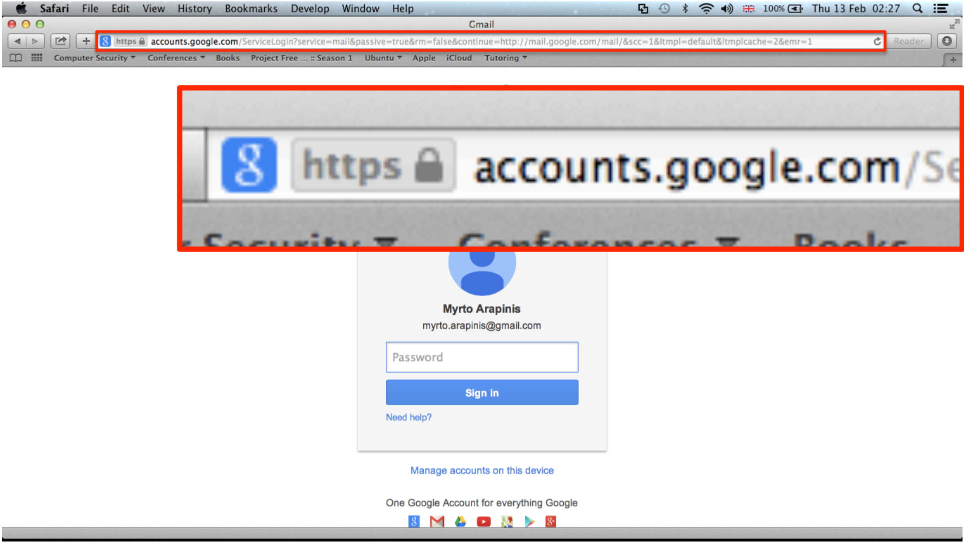


SSL/TLS use X.509 certificates and hence asymmetric cryptography to exchange a symmetric key. This session key is then used to encrypt subsequent communication. This allows for **data/message confidentiality**, and message authentication codes for **message integrity** and thus, **message authentication**.

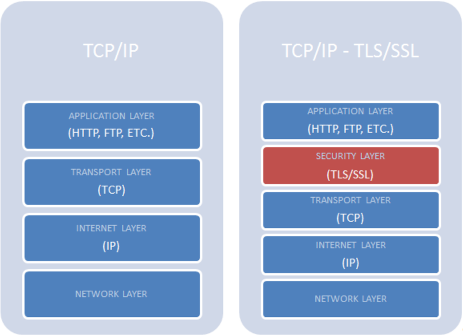
SSL/TLS protocol



SSL/TLS protocol

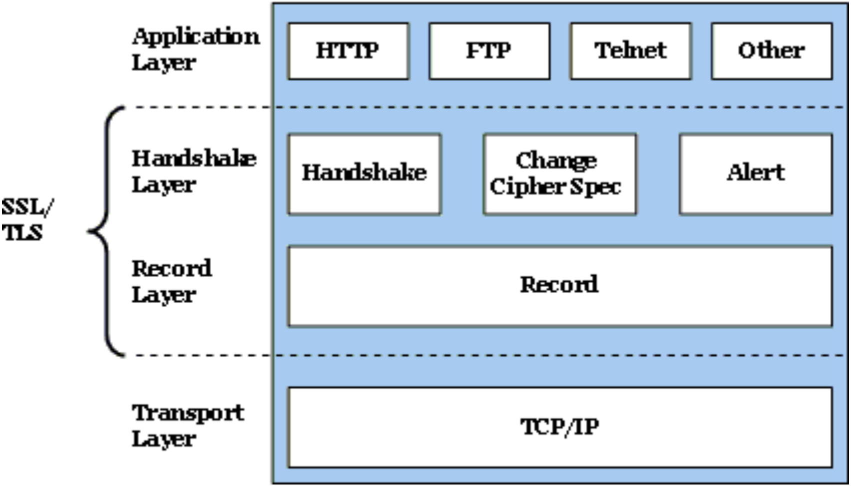


TCP/IP protocol stack

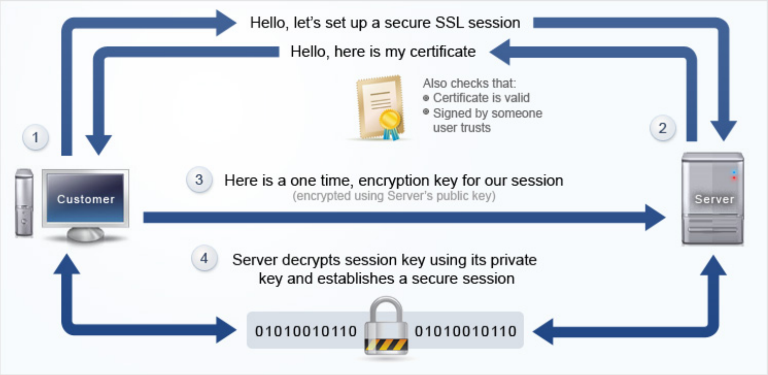


- TCP/IP provides end-to-end connectivity and is organized into four abstraction layers which are used to sort all related protocols according to the scope of networking involved
- The SSL/TLS library operates above the transport layer (uses TCP) but below application protocols

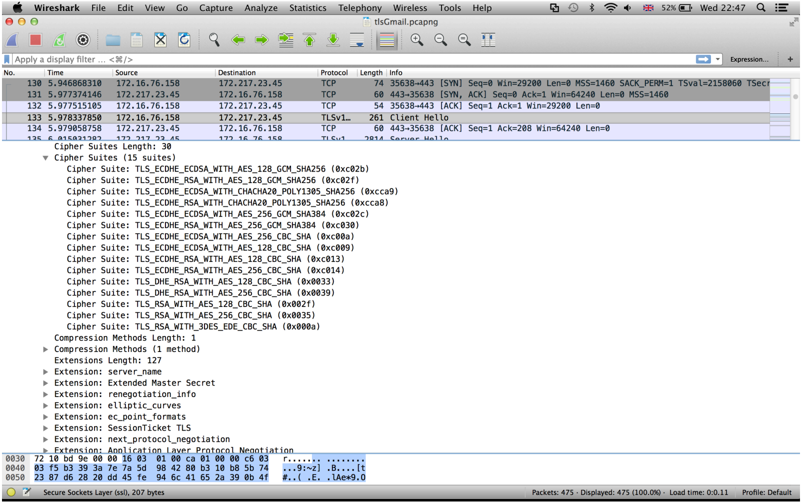
SSL/TLS protocol layers



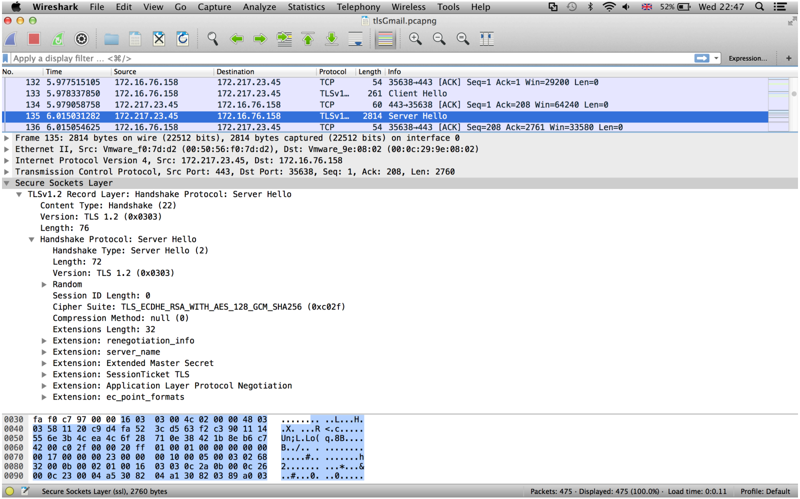
SSL/TLS handshake protocol



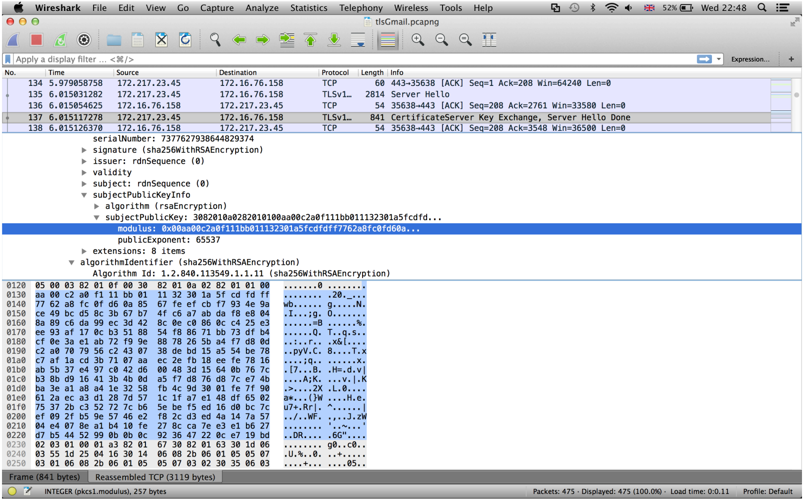
Client Hello



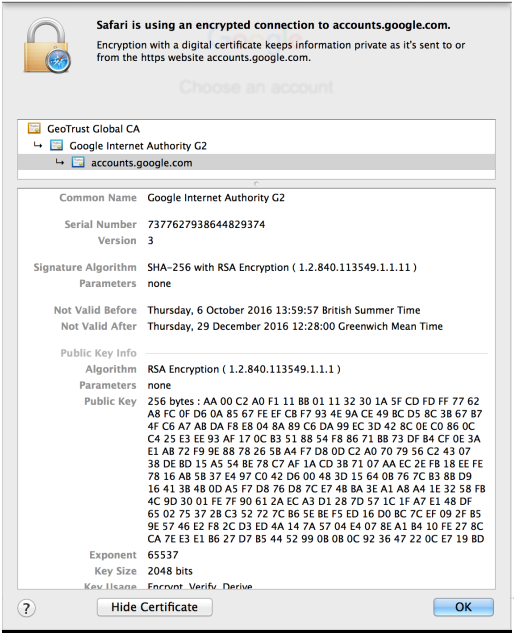
Server Hello



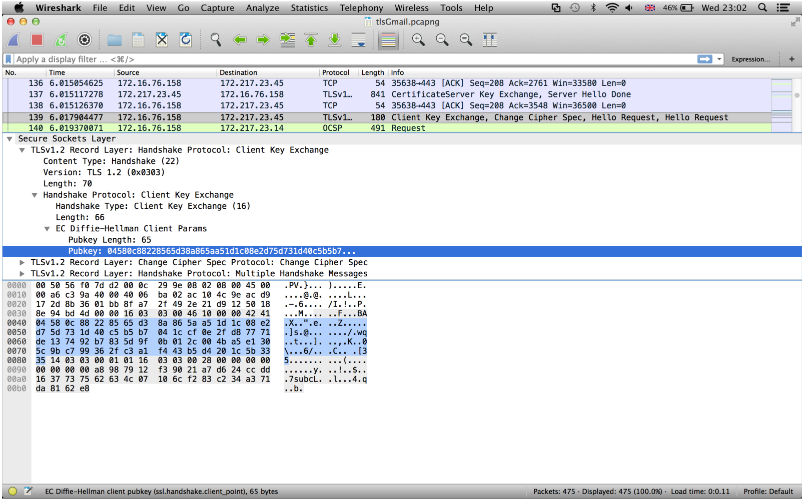
Certificate



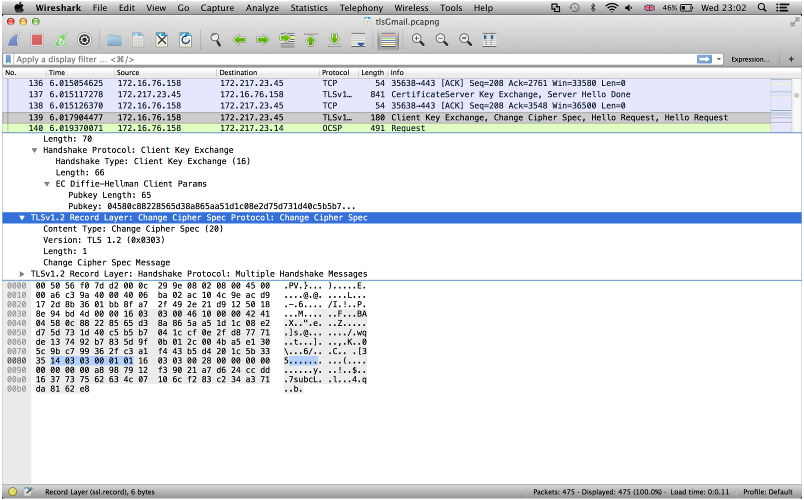
www.gmail.com's certificate



Key exchange



Change cipher spec



www.mitls.org

miTLS, Triple Handshake, SMACK, FREAK, Logjam, and SLOTH

mitls.org/pages/attacks

SimSec ▾ CSexam ▾ La cryptogra...ts dévoilés Conferences ▾ ResearchProfiles ▾ Security-Club ▾ Teaching ▾ Tutoring

miTLS Publications Attacks Code FlexTLS People

Alert 3SHAKE VHC SMACK Logjam SLOTH

Protocol Cryptographic Implementation Deployment

A Zoo of TLS attacks

Attacks on TLS that break the intuitive security property of a virtual recreation of a physically secure channel can be categorized along three dimensions.

1. Protocol logic vs. cryptographic design flaw
2. Specification/Standard vs. Implementation errors
3. TLS vs. Context

Flaws in the protocol logic

Attacks targeting the protocol logic may for instance cause the client and server to negotiate the use of weak algorithms even though they both support strong cryptography.

If the faulty negotiation logic conforms to the specification, then the attack is on the specification itself (as, e.g., partially enabled by the **False Start** modification), if an implementation deviates from the specification to implement a faulty negotiation logic [**Dimcev**, **Langley**] it is an attack on the implementation. As many aspects of the standard can be underspecified or ambiguous, it is not always possible to distinguish between these two cases.

Another class of protocol logic flaws are state-machine bugs [**Early CCS Attack**, **SMACK Attack**].

The attack can also be either an attack on TLS proper, or on its context, e.g. if the attacker can just change the configuration files to deactivate strong cryptography. As the TLS standard does not describe APIs or configuration file formats, context specific attacks are always implementation specific.

The renegotiation attack [**TLS_Reneg_Attack**] is a logical attack on the TLS standard, where one peer believes it is running the first handshake on a connection, while the other peer is running a re-handshake. miTLS prevents the renegotiation attack by **implementing the renegotiation extension**.

More generally, the TLS specification is vague about how applications should handle data coming from consecutive sessions, e.g. whether it is safe to join them and consider them as a single stream, or if the user should be notified of the change of context. The renegotiation extension partially fixes the problem, but it still leaves room for our **alert attack**, where the attacker can turn any authentic fatal alert into a warning alert, which gets ignored by default.

Much more seriously, resuming the attacker controlled session on a different connection re-enables the renegotiation attack. This attack is known as the **triple handshake attack** and is an instance of a larger class of attacks resulting from inadequate channel binding in compound authentication protocols. The miTLS security theorem does not promise channel binding across different connections and is thus not violated by the attack. To be secure, applications making use of miTLS have, however, to be carefully designed to make use of the provided security cues. We give a basic HTTPS client, miHTTPS, as an example for such an application. Subsequently the flaw is also being patched at the TLS level using a new **extension**.

Cryptographic design flaws

Attacks exploiting cryptographic design flaws may simply result from cryptanalytic progress against the cryptographic building blocks of TLS. They can, however, also result from improper non-blackbox use of otherwise secure cryptographic constructions. An example for this is chosen ciphertext chaining (CBC) mode encryption. Early versions of TLS allow using knowledge of the next initialization vector (IV) to set up adaptive plaintext attacks, see, e.g. **OpenSSL archive** for a first mention of the '**BEAST** attack'.

SSL/TLS renegotiation

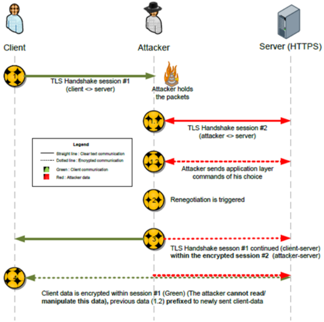
Client and server are allowed to initiate renegotiation of the session encryption in order to:

- Refresh keys
- Increase authentication
- Increase cipher strength
- ...

Client or server can trigger renegotiation by sending a hello meesage

SSL/TLS renegotiation weaknesses

- Renegotiation has priority over application data!
- Renegotiation can take place in the middle of an application layer transaction!



(Detailed on the board)

Incorrect implicit assumtion: the client doesn't change through renegotiation

Marsh Ray's plaintext injection attack on HTTPS

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Result:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
```

Cookie:victim_cookie

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Result:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
```
Cookie:victim_cookie

⇒ **Server uses victim's account to send a pizza to attacker!**

Anil Kurmus' plaintext injection attack on HTTPS

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password
User-Agent:  curl/7.19.5
Host:  twitter.com
Accept:*/*
Content-Length:  140
Content-Type:  application/x-www-form-urlencoded
status=
```

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password
User-Agent:  curl/7.19.5
Host:  twitter.com
Accept:*/*
Content-Length:  140
Content-Type:  application/x-www-form-urlencoded
status=
```

Victim:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password...
```

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization: Basic username:password
User-Agent: curl/7.19.5
Host: twitter.com
Accept:*/*
Content-Length: 140
Content-Type: application/x-www-form-urlencoded
status=
```

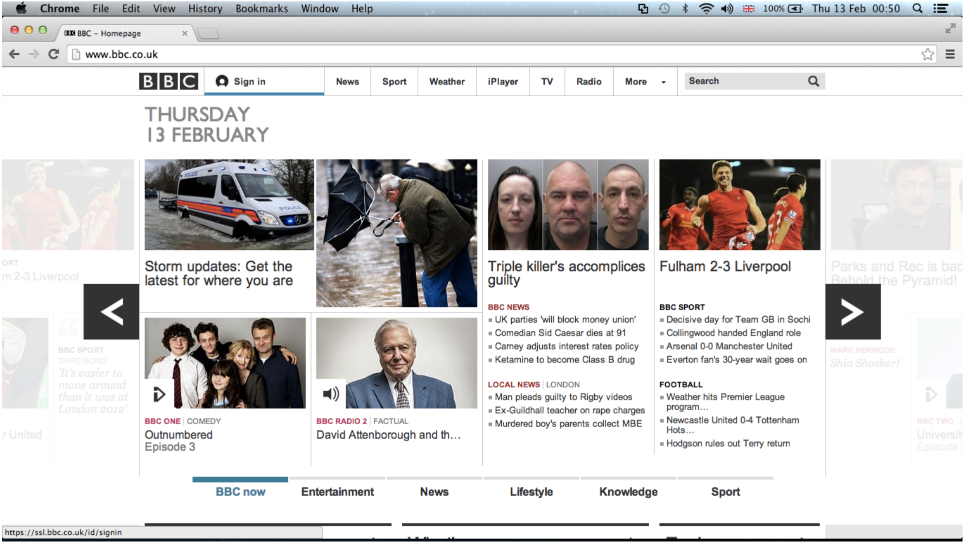
Victim:

```
POST /statuses/update.xml HTTP/1.1
Authorization: Basic username:password...
```

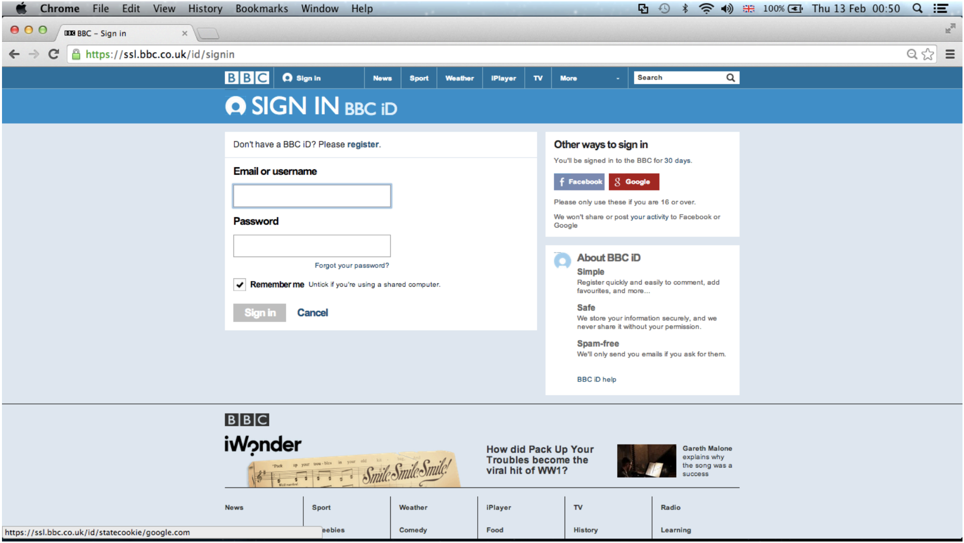
$\Rightarrow$ **the attacker gets the user name and password of the victim!**

The SAML Signle Sign On (SSO) protocol

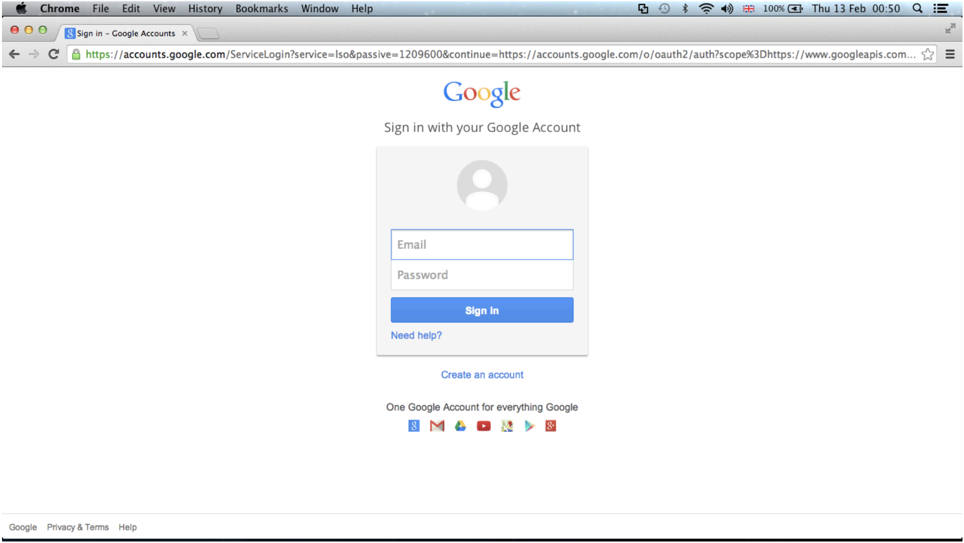
SAML SSO protocol



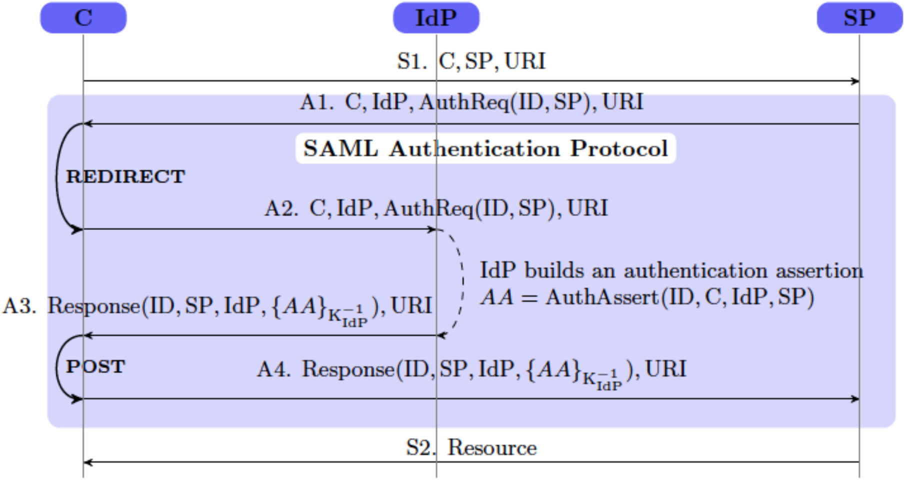
SAML SSO protocol



SAML SSO protocol



SAML SSO protocol (OASIS 2005)



Google's implementation of SSO

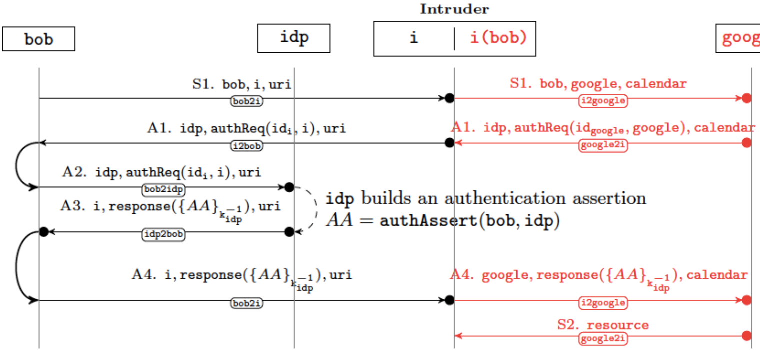
Google's SAML-based Single Sign-On for Google Applications deviates from the above protocol for a few, seemingly minor simplifications in the messages exchanged:

G1. *ID* and *SP* are not included in the authentication assertion, *i.e.* $AA = \mathrm{AuthAssert}(C; IdP)$ instead of $\mathrm{AuthAssert}(ID; C; IdP; SP)$;

G2. *ID*, *SP* and *IdP* are not included in the response, *i.e.* $Resp = \mathrm{Response}(\{AA\}_{K_{IdP}^{-1}})$ instead of $\mathrm{Response}(ID; SP; IdP; \{AA\}_{K_{IdP}^{-1}})$.

Attack Google's SSO implementation

[A. Armando, R. Carbone, L. Compagna, J. Cullar, L. Tobarra, "Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps", (FMSE'08)]



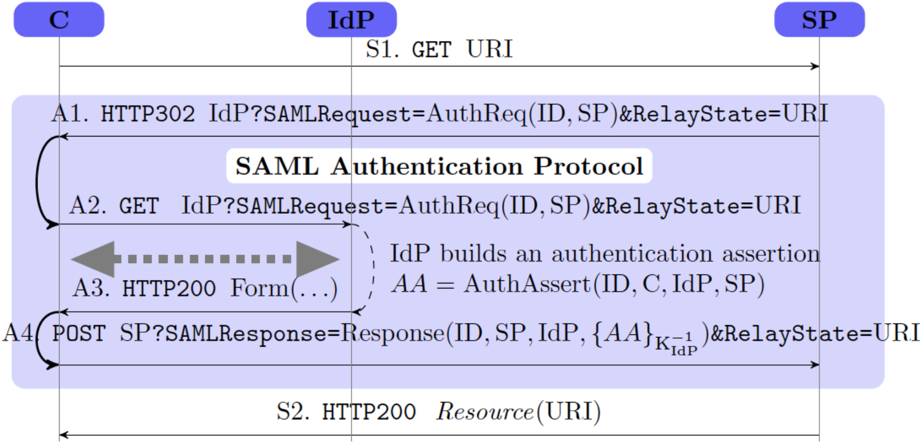
Legend:

$A \xrightarrow[ch]{M}\bullet B$: A sends M on ch confidential to B

$A \bullet\!\xrightarrow[ch]{M} B$: A sends M on ch authentic for A

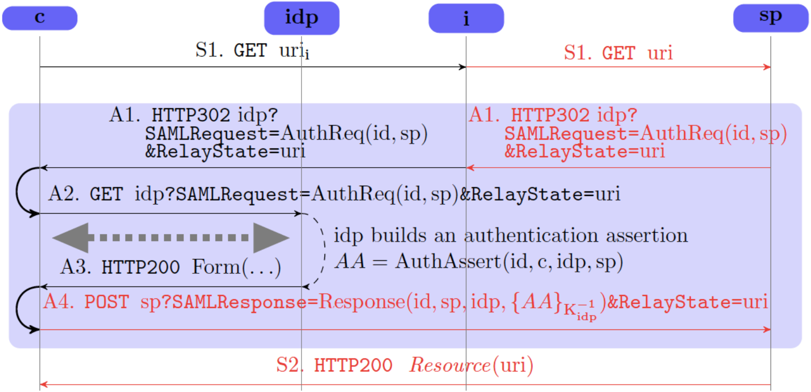
$A \bullet\!\xrightarrow[ch]{M}\bullet B$: M is sent on ch authentic for A and confidential to B

SAML SSO protocol (OASIS 2012)



Attack SAML SSO protocol (OASIS 2012)

[A. Armando, R. Carbone, L. Compagna, J. Cullar, G. Pellegrino, A. Sorniotti, "From Multiple Credentials to Browser-Based Single Sign-On: Are We More Secure?", Chapter in Future Challenges in Security and Privacy for Academia and Industry]



⇒ XSS attack on SAML-base SSO for Google Apps