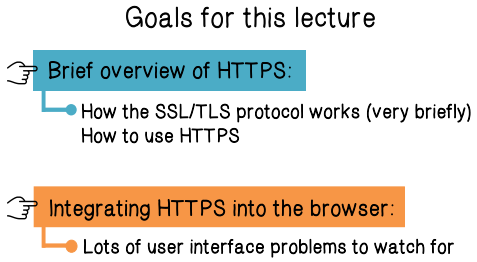
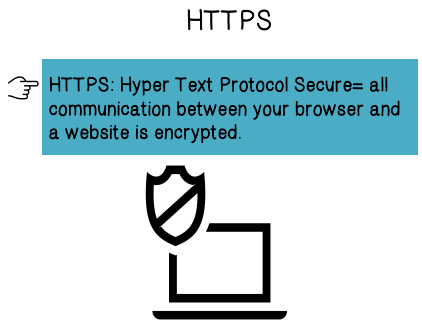
Recommended Readings for this lesson are:

ForceHTTPS: Protecting High-Security Web Sites from Network Attacks

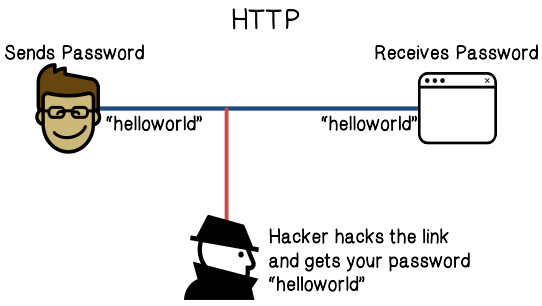
You can find short summary of the paper at the of the document.



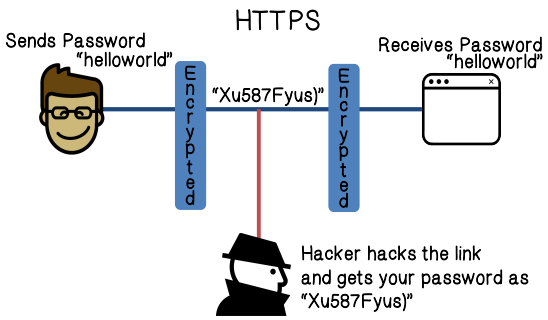
Let us discuss HTTPS and how it is integrated into the web browser. And we are going to discuss a number of security problems with HTTPS.



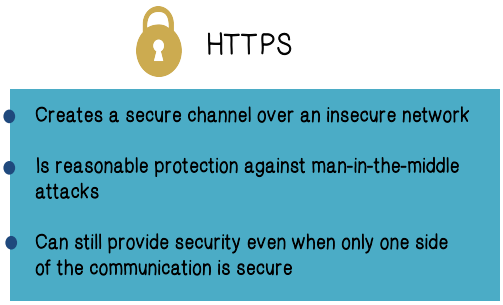
HTTPS is essentially HTTP over SSL, the secure socket layer, which is now called TLS, transport layer security. With HTTPS, all traffic between a web browser and a web site is encrypted, whereas HTTP is a cleartext protocol, meaning that the traffic is not encrypted.



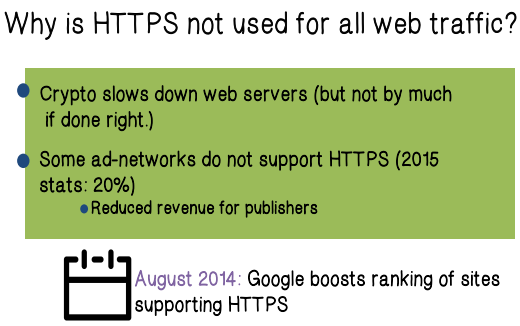
For example, using HTTP, a user sends a password and a web server receives it. Since the traffic data is in cleartext, a network attacker with access to the link can intercept the traffic data and learn the user's password.



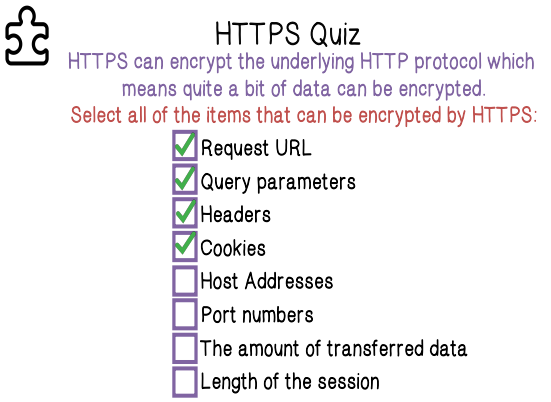
Now with HTTPS, the user still sends the password, but the password is encrypted in transmission. Therefore, even when attacker can access a link, he cannot learn your clear text password.



In summary, HTTPS allows for secure communication over untrusted or public network. It encrypts traffic and uses public key to authenticate the web server and, if possible, even the browser. Even if only the web server's public key is known, many of the man-in-the-middle-attack can still be prevented.

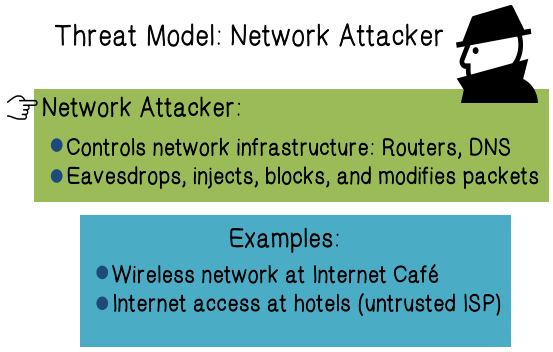


With all these benefits, HTTPS is not yet used for all web traffic. The reason is that crypto operations can slow down the web service, in particular, if it is not implemented right. And some ad networks do not support HTTPS. For example, the ad publishers cannot learn the web contents as being viewed by the users. On the other hand, Google is now trying to encourage websites from adopting HTTPS.

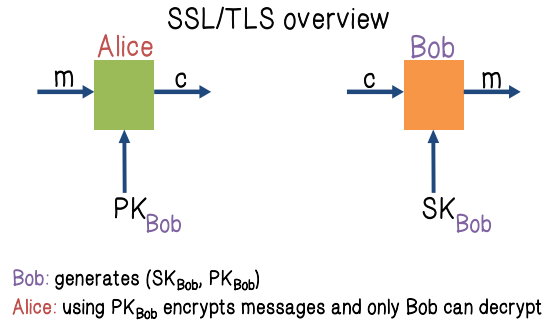


Now let us do a quiz on HTTPS. Select all items that can be encrypted by HTTPS.

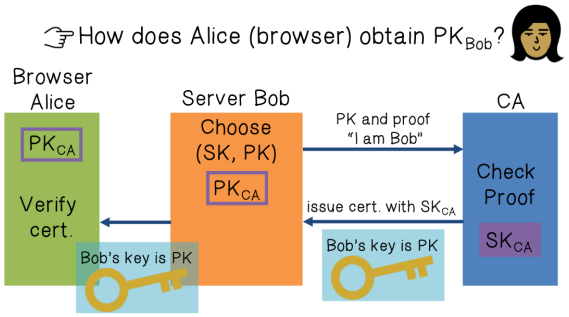
The first four can be encrypted by HTTPS. Host address and port numbers are used to route traffic and so they are not encrypted. The amount of transferred data and length of session can be inferred by observing the traffic. So the attacker can learn this.



Recall that a network attacker can control network infrastructures such as routers and DNS servers. It can eavesdrop to learn traffic contents, inject data, block traffic, or even modify the contents. For example, such a network attacker can sit at an Internet cafe or hotel lobby to compromise the network. HTTPS was designed to thwart such network attackers.



Since HTTPS is HTTP over SSL, let us briefly review SSL/TLS. It uses public key for authentication and key exchange. As a quick review, in public key cryptography, each user, say Bob, has a pair of public key and private key. And Alice, after obtaining Bob's public key, can use Bob's public key to encrypt message into cyphertext, and a cyphertext can only be encrypted by Bob, using the corresponding private key.



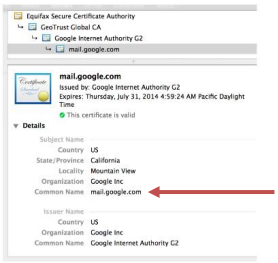
An essential problem in public key cryptography is how Alice can obtain the public key of Bob. The standard is to use certificate issued by a certificate authority, we call it the CA. First, every entity has installed the public key of CA. Then Bob can ask the CA to generate a certificate for his public key. The certificate authority keeps the signing private key to itself. And again, the corresponding public key has been installed in all entities. The CA signs Bob's public key using it's signing private key and the signature is put into the certificate. So, Bob can now present the certificate to Alice. And because Alice has the certificate authority's public key, she can verify that the signature was constructed properly, which means that Bob's public key has been certified by the certificate authority.

Important Fields:

Serial Number	5814744488373890497
Version	3
Signature Algorithm	SHA-1 with RSA Encryption (1.2.840.113569.1.1.5)
Parameters	none
Not Valid Before	Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time
Not Valid After	Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
Public Key Info:	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes: 04 71 6C DD E0 0A C9 76...
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes: 8A 38 FE D6 F5E7 F6 59 ...

Here is an example of public key certificate. Let us go over some important information. First, there is a unique serial number. Second, there is a valid time period. And there is a public key and a signature produced by the CA.

Certificate Examples



Here is an example of certificate information that a user sees on his computer. It identifies that the certificate is for the public key of mail.google.com.

Certificates on the Web

Subject's CommonName can be:

- An explicit name, e.g. cc.gatech.edu
- A wildcard cert, e.g. *.gatech.edu

Matching Rule:

- "*" must occur in leftmost component, does not match "." For example: "*.a.com" matches "x.a.com" but not "y.x.a.com"

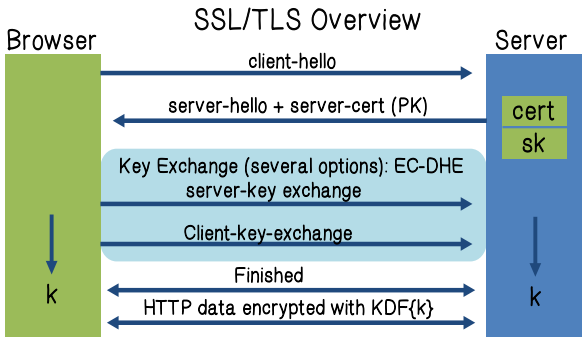
A certificate is for an entity or subject that is identified by the common name. So what is a common name? A common name can be an explicit name, for example cc.gatech.edu. Or it can be a wildcard, for example, *.gatech.edu. If a wildcard is used it can only be the leftmost component, and it does not match a dot. For example: *.a.com matches x.a.com but not y.x.a.com.

Certificates on the Web

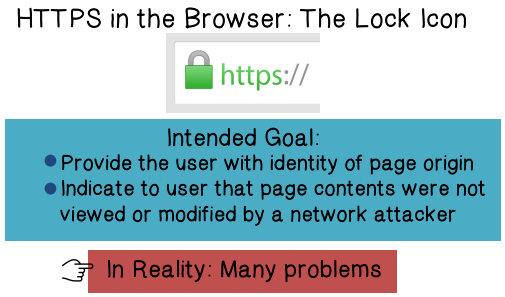
Browsers accept certificates from a large number of CAs:

- Top level CAs ≈ 60
- Intermediate CAs ≈ 1200

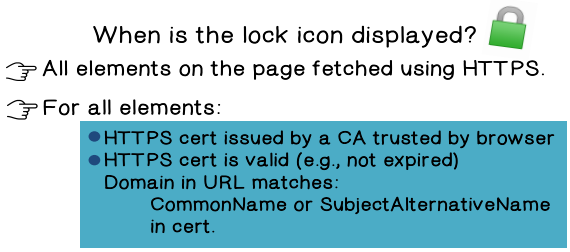
There are large numbers of CAs out there, and a browser typically accepts certificates from 60 top level CAs and 1200 intermediate CAs.



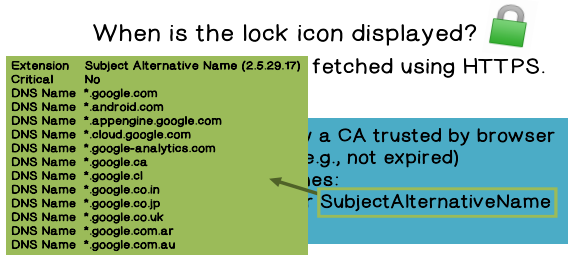
Let us briefly review SSL/TLS. The goal of this handshake is to authenticate the server and optionally the browser and more importantly, at the end, both will have a shared secret key that can be used to encrypt HTTP traffic. The client sends a hello message to a server and the sever's response includes a proper key certificate. The browser verifies the certificate, meaning that now the browser knows the server's valid public key. And with that, the browser can now perform key exchange. For example, it can use Elliptic curve Diffie-Hellman key exchange. With a server's public key, the browser and the server can perform secure key exchange and prevent man-in-the-middle attack. And the result is that they now establish a shared secret key and they can use this shared secret key to encrypt HTTP data.



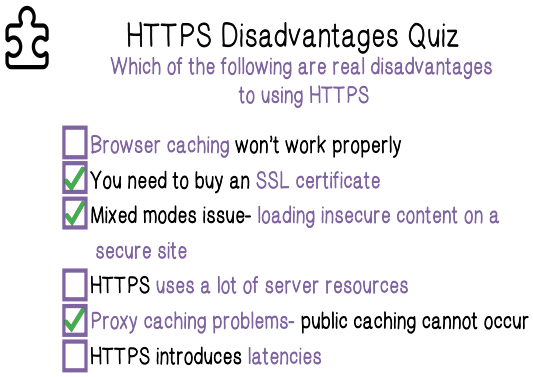
HTTPS is integrated into a browser, or it is indicated in the browser GUI. The goal is to let the user know where a page came from. And it tells the users that the page contents are protected, meaning that they are encrypted so that a network attacker cannot see them or modify them. In reality, there are several security problems.



When the lock icon is displayed on the browser, it means that all the elements on the page are fetched using HTTPS. But for the browser to even accept this HTTPS connection, it means that the browser has trusted the certificate and verified that the certificate is valid. And also, the domain URL matches the CommonName or SubjectAlternativeName in the certificate.

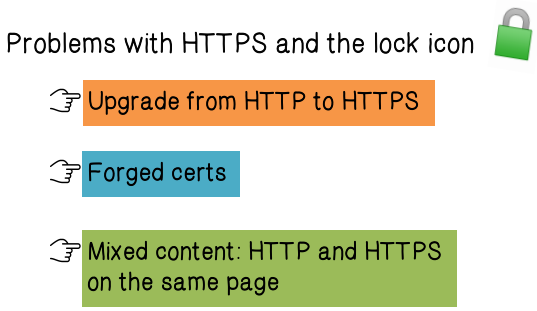


For example, the certificate of google.com can simply supply a list of alternative names.

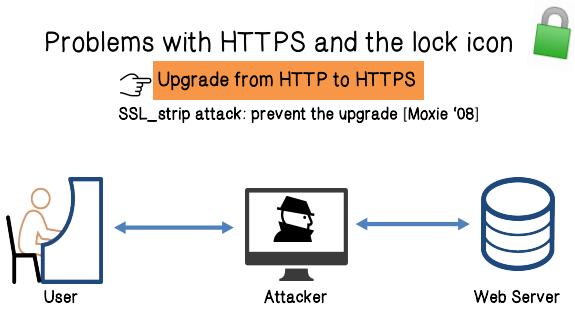


Now, let us take another quiz on HTTPS. Which of the following are real disadvantages of using HTTPS?

You need to buy an SSL certificate. Mixed modes issues, loading insecure content on a secure site. This will continue to be the problem until many, many sites are all using HTTPS. Proxy caching problems, public caching cannot occur. This is because all traffic is encrypted, so there is no public caching possible.

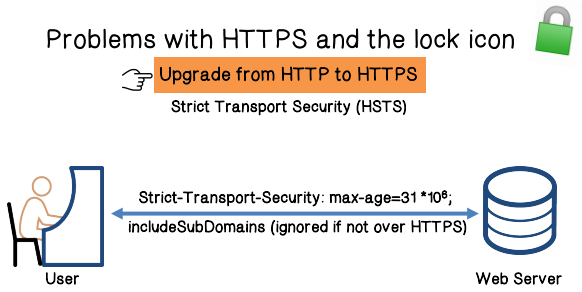


Let us discuss several security problems with HTTPS and the lock icon. These include upgrade from HTTP to HTTPS, and Forged certificates. First, let us discuss upgrade from HTTP to HTTPS. There is an attack method called SSL stripped. It prevents the browser from upgrading.

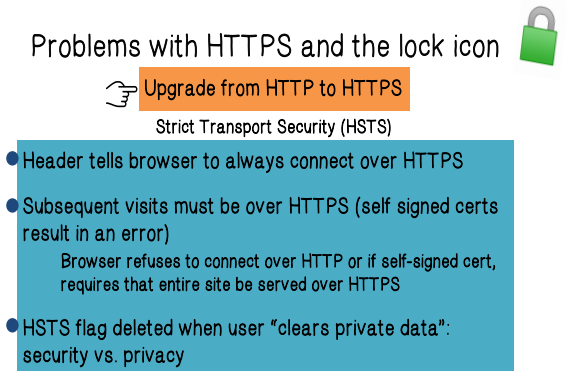


With SSL stripping, the browser will not display any SSL certificate errors, and the user has no clue that such an attack is happening. This attack is also known as HTTP downgrading attack. The connection established between the victim user's browser and the web server is downgraded from HTTPS to HTTP. For example, when a user wants to transfer money to his account using an online banking service, he enters the following URL in the address bar of his browser, www.foobank.com/onlinebanking. Of course, this URL is intended for the web server of the bank. In the background, the user's computer happens to be connected to the attacker's machine. The attacker waits for a response from the bank server. The attacker forwards the request to the bank and waits for the response. The connection between the attacker and the bank is secure. That means the traffic is transferred using an SSL tunnel. Therefore, the login page from the bank's web server will be https://www.foobank.com/onlinebanking. The attacker has access to the login page and can

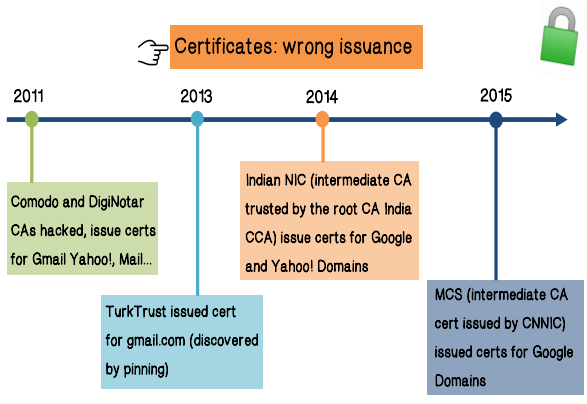
modify the response from the server from HTTPS to HTTP and then forward the login page in HTTP to the user. The user's browser now is connected to http://www.foobank.com/onlinebanking. The user's browser is now connected to the bank's website with an insecure connection. From this point on, all of the user's requests go out in plaintext, and the attacker can access the data and collect the credentials. While the server thinks that it has been using a secure connection, that connection is really just between the web server of the bank and the attacker. On the other hand, the user's browser is using the insecure HTTP connection, thinking that that is what the bank's web server wants it to use.



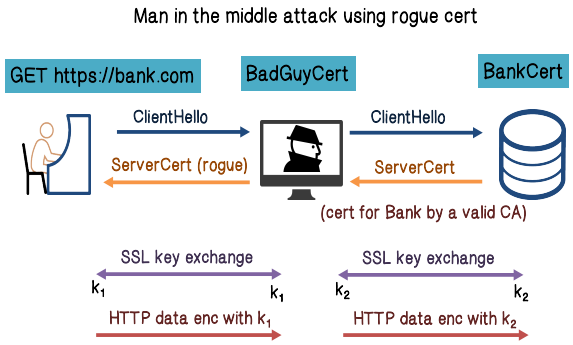
The solution to SSL stripping attack is to use HSTS, which stands for Strict Transport Security. This policy can be set for a maximum of one year. It basically tells the web browser to always use HTTPS, even for its subdomains.



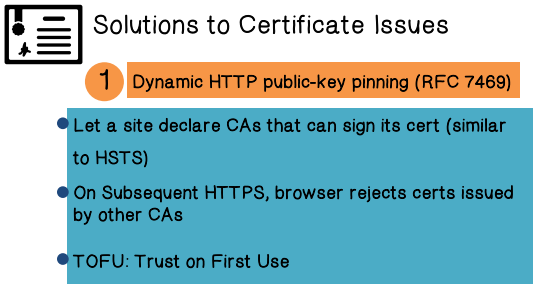
When a web browser visits a website for the first time, the website can tell the browser to always use HTTPS. That is, for any subsequent visit, all connection must be over HTTPS, and HTTP connections will be rejected. A web browser can also have a preloaded list of HSTS websites. Even before web browser visits a site on this list, it knows that it must use HTTPS. The HSTS flag set by a website can be cleared when the user selects clear private data.



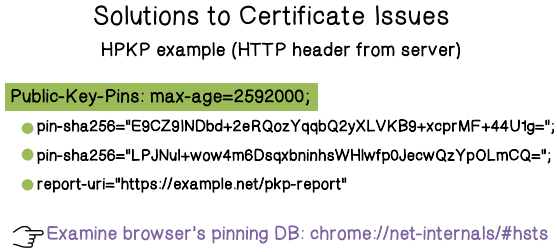
Another serious security problem is forged certificates. For example, if a CA is hacked, the attacker can issue rogue certificates, for example, for Gmail. And once a rogue certificate is issued, now the attacker can set up a fake website and calling itself Gmail. In this website, we have the rogue certificate for Gmail. And several countries have been caught issuing unauthorized certificates, for example, for Google, so the ISPs in these countries can play man-in-the-middle between a user and the real Google server.



This is further illustrated in this example. Suppose a user wants to connect to a bank. There is a bad guy in the middle, and this attacker has a rogue certificate, therefore it can pretend to be the bank. The user may think that he is connected to the bank because the certificate says so, but however, the certificate is rogue, meaning that the user is actually connected to the bad guy. This illustrates that with a rogue certificate, an attacker can play man-in-the-middle, even in HTTPS connection. The attacker plays the bank server to the user and the user to the bank server. And both sides of connections are in HTTPS.



One approach to deal with rogue certificate is to use dynamic public-key pinning. This means that a website will declare the CAs that sign its certificate. When a browser first visits a website, the website tells the browser the list of authorized CAs. Then on subsequent visits, the browser will reject any certificate issued by other CAs.



Very similarly there is a public-key pinning extension for HTTP or HPKP. This feature tells a web browser the list of proper keys to be associated with the website. And it can prevent man-in-the-middle attacks with forged certificates. When the browser visits a website for the first time, the browser sends a list of public-key hashes. And on subsequent visits, the browser expects the server to use one or more of these public keys in its certificates.

Another way to deal with forged certificate is for the CAs to be transparent. That is, the CAs must publish in a public log of all the certificates that they have issued and a browser will only accept a certificate if it is published on a public log. And companies like Google can constantly scan the public logs to look for invalid or forged certificates.

ForceHTTPS: Protecting High-Security Web Sites from Network Attacks

In this paper, the researchers present ForceHTTPS, a browser security mechanism that web sites or users can use to opt in to stricter error processing, improving the security of HTTPS by preventing network attacks that leverage the browser's lax error processing, thus allowing sophisticated users to transparently retrofit security onto some insecure sites that support HTTPS. Another issue browsers have is accepting illegitimate certificates. The authors posit that browsers accept broken certificates and allow embedding of insecure scripts for two main reasons. First, Compatibility: with many web sites having incorrectly configured certificates and embedded insecure scripts, a browser that enforces strict error processing is incompatible with these sites and will lose users to a more permissive browser. Second, Unknown Intent, where some site owners intentionally use self-signed certificates and host portions of their site over HTTP because these mechanisms provide protection from passive attackers and they believe the risk of an active attack is outweighed by the cost of implementing HTTPS in full.

The researchers propose ForceHTTPS, a simple mechanism that security conscious sites can use to opt in to stricter error processing by the browser, essentially giving the browser guidance to be more secure. By setting a ForceHTTPS cookie, a site owner asks the browser to treat HTTPS errors as attacks, not as simple configuration mistakes. Specifically, enabling ForceHTTPS causes the browser to modify its behavior causing Non-HTTPS connections to the site to be redirected to HTTPS, preventing contact to the site without TLS. Also, all TLS errors, including self-signed certificates and common-name mismatches, terminate the TLS session, and attempts to embed insecure (non-HTTPS) content into the site fail with network errors.

The threat models here involve passive network attacks – eavesdropping an insecure HTTP only connection, active network attacks – spoofing DNS server or wireless access points, and inadvertent vulnerabilities introduced by well-meaning but imperfect non-security conscious web developers. The threats of phishing, malware, and browser vulnerabilities remain outside the scope of this paper.

ForceHTTPS can be enabled on the server side, where a security-conscious site can enable ForceHTTPS by setting a cookie with the name ForceHTTPS using a Set-Cookie header in an error-free HTTPS response. The browser will then enable ForceHTTPS for that site as long as the cookie has not expired. Alternatively, it can be enabled on the client side where a security-conscious user can enable ForceHTTPS for a host through the browser user interface. It can only be disabled by an error-free HTTPS response or by the browser's user interface.

ForceHTTPS does come with the caveat that via State Exhaustion, a determined attacker can cause a browser to evict the ForceHTTPS cookie, effectively circumventing the intended

security. In addition, by forcibly enabling ForceHTTPS, an attacker can deny service to a site that only works using broken HTTPS. Furthermore, despite its enhancement to security, ForceHTTPS still has security gaps, more notably: attacks on initialization where a user is unable to establish a secure connection to a server, thus leaving that server unable to set a ForceHTTPS cookie; privacy, where like any cookie, ForceHTTPS leaves a trace on the user's system for each ForceHTTPS site visited; and cross-site scripting (XSS), cross-site request forgery (CSRF), HTTP response splitting, phishing, amongst others.