TRASYS
WE GET IT DONE

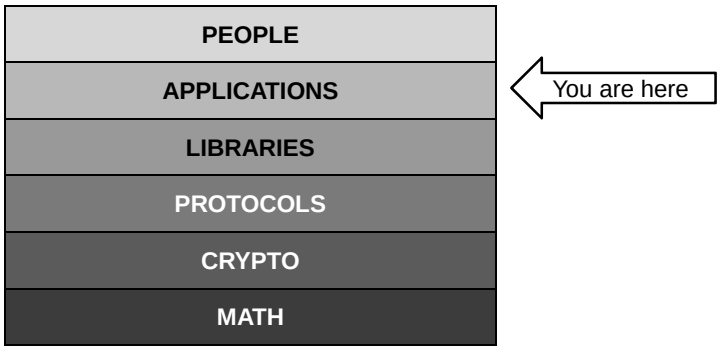
Using SSL/TLS

Thomas Herlea

SecAppDev, 2013-03-07

thomas.herlea@trasysgroup.com

A TLS Stack



Disclaimer

- This lecture is focused on security aspects
- Following the steps from the story does not guarantee
  - Accessibility by all your users
  - Performance
  - Portability to other environments
  - Efficient maintenance
- Not even all known security aspects are covered
  - Please contact me if you find problems before SecAppDev 2014

- The following story is not "The One True Way"
- But you will encounter typical dangers and meet powerful allies
- You will learn what to learn about

ACT I

"Let's get a padlock on our web site!"

Playground

- Linux: Fedora 18 “Spherical Cow”
  - Download: http://fedoraproject.org/en/get-fedora
  - Similar with several major Enterprise Linux distributions
  - Packages rather new versions of software
- Apache 2.4.3, from the newest branch, 2.4

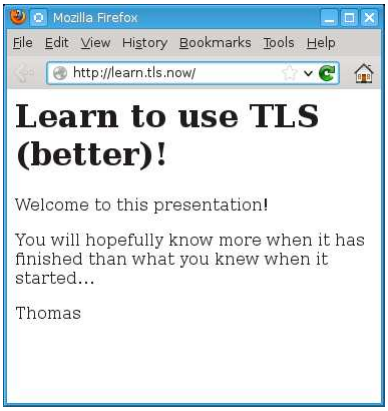
```
$ sudo yum install httpd
$ sudo systemctl start httpd.service
```

  - The document root is in `/var/www/html/`
- Command line interface and editing configuration files, because:
  - You will do it easier remotely
  - It is independent of KDE, Gnome, LXDE, XFCE etc.
- Become root:

```
$ sudo su -
```


Try It!

HTTP woks out of the box

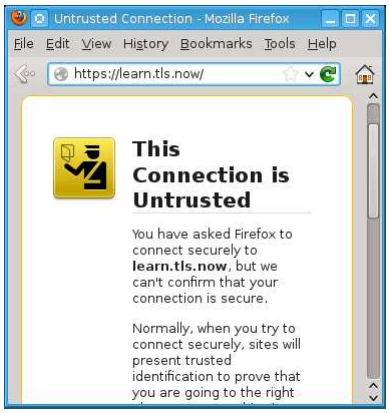


HTTPS doesn’t work yet

Enable “https”

```
# yum install openssl
# yum install mod_ssl
# systemctl restart
  httpd.service
```

- Use **`systemctl restart`**, because we added a module
- Refresh the browser…



- The connection would protect the confidentiality and integrity w.r.t. third parties, but the other endpoint could be anyone

Don’t train your users to continue anyway!

Why Untrusted?

- Firefox reason given for “Untrusted”:
  - “The certificate is not trusted because it is self-signed.”
  - Misnomer: the trusted Root CA certificates are self-signed, too
  - Should be: “… because signer is not trusted.”
- By warning, browser asks user: “Would you trust the signer?”
  - By overruling the warning the user answers “Yes”
  - The “Add exception… permanently” remembers the choice
  - Actually, most users just want to answer “Get out of my way!”
  - Bad idea when it is the certificate of an attacker-in-the-middle
- Lesson: avoid surprising the user with this warning!
  - Prevent it, or
  - Document it

What Certificate?

- You did not do anything about certificates!
- Fedora did:
  - Generated a dummy certificate when we installed **`mod_ssl`**
  - Aim: get you up and running without errors
  - If you're a **novice**, you can learn later to do it properly
  - If you're a **pro**, you anyway have a procedure to do it properly
- Most Linux distros do it this way
- Where is the certificate?

TLS-Enabling in Apache

- With **`mod_ssl`** came **`/etc/httpd/conf.d/ssl.conf`**
- In **`ssl.conf`** you can find the directives:

```
Listen 443 https
# Global TLS Configuration
...
# TLS Configuration per virtual host
<VirtualHost _default_:443>
    SSLEngine on
    SSLCertificateFile
        /etc/pki/tls/private/localhost.key
    SSLCertificateKeyFile
        /etc/pki/tls/certs/localhost.crt
    ...
</VirtualHost>
```

Crash Course OpenSSL CLI

- In Fedora 18 the version of OpenSSL is 1.0.1c
- OpenSSL has a command mode:
  # openssl SUBCOMMAND OPTIONS
- Which subcommands?
  - Run **man openssl**, see the COMMAND SUMMARY section
  - Then run **man SUBCOMMAND**
- Options are in single dash format
  - E.g.: **-noout**, "do not re-display raw input"
- Some options can have parameters
  - E.g.: **-in INPUTFILE**, "read input from INPUTFILE, not stdin"

Inspect with OpenSSL

 # cd /etc/pki/tls/

- What site is the certificate for?
  # openssl x509 -in certs/localhost.crt –subject -noout
  subject= [...]/CN=localhost.localdomain/[...]
  - The site is localhost.localdomain: it's a dummy certificate
- Does it appear to be self-signed?
  # openssl x509 -in certs/localhost.crt –issuer -noout
  issuer= [...]/CN=localhost.localdomain/[...]
  - Yes, the Issuer has the same value as the Subject above
- Does the certificate correspond to the key?
  # openssl x509 -in certs/localhost.crt –pubkey -noout
  # openssl pkey -in private/localhost.key –pubout
  - Yes, the public keys are the same

Generating Certificates

- There are numerous helper tools and wizards
  - Fedora recommends **`genkey`** from package **`crypto-utils`**
  - Fedora provides OpenSSL GUI package **`xca`**
  - Fedora provides heavyweight set of packages **`pki-*`**
  - JDK comes with **`keytool`**
  - OpenSSL wrapper: **`/etc/pki/tls/misc/CA`**
  - OpenSSL subcommand: **`openssl ca`**
  - OpenSSL: **`openssl req –newkey ARGS –x509`**
  - OpenSSL Makefile: **`/etc/pki/tls/certs/Makefile`**
- You take the route that shows each step
  - Inspiration comes from the OpenSSL Makefile above

Generate a Keypair

- You can already generate a key pair as simply as:

```
# openssl genpkey -algorithm rsa -out private/keys.key
```

  - The conventional file extension is **`.key`**
- But is it any good?
  - Algorithm could be DSA, but RSA procedure is simpler
  - Default keyfile format is PEM, accepted by Apache
  - Default key length is 1024 bits – should be 2048 bits
    - NIST Special Publication 800-57 (July 2012)
    - CA/Browser Forum Baseline Requirements (November 2011)
  - The public exponent has by default value “65537” – 17 bits
    - 2006: Belichenbacher attack against exponent “3” PKCS #1 v1.5
    - SecAppDev 2012, Bart Preneel: “use >32 bits”

Generate a Keypair (continued)

- So far, recommended key generation:

```
# openssl genpkey -algorithm rsa -out private/keys.key
  -pkeyopt rsa_keygen_bits:2048
  -pkeyopt rsa_keygen_pubexp:0x100000001
```

- Private key protection?
- Option 1: password protect the keyfile
  - Add cipher option, e.g. **`-aes-128-cbc`**
  - By default, passphrase provided interactively at generation
    - See **`man openssl`**, section PASS PHRASE ARGUMENTS
  - Passphrase must be provided every time Apache starts up
    - See docs for SSLPassPhraseDialog
- Option 2: keyfile born protected at filesystem and SELinux level

```
# umask -S u=rw,go=
# openssl genpkey ...
```


Generate a Self-Signed Certificate

- You can already generate a self-signed certificate as simply as:

```
# openssl req -new -x509 -key private/keys.key
  -out certs/selfsigned.crt
```

  - You will be prompted for Distinguished Name information
  - Only “Common Name”==“Web server FQDN” is needed
    - Let’s say your site will be **`http://learn.tls.now`**
    - Automate with: **`-subj "/CN=learn.tls.now/"`**
  - The conventional file extension is **`.crt`**
- But is it any good?
  - Default hash algorithm is SHA1, good (do not use MD5)
  - Default certificate validity is 30 days, not enough for production
  - It thinks it's a CA cert! (X509v3 Basic Constraints: CA:TRUE)
  - For quick’n’dirty testing on your own this cert is good
- Add it in Firefox at most as a non-persistent exception

A Better Self-Signed Certificate

- Longer validity period with **`-days 365`**
- Not claiming it's a CA certificate: control the x509v3 extensions
- Not done on the command line, but in a configuration file
- Config file can be default or specified with **`-config CONFIGFILE`**
  - On Fedora: **`/etc/pki/tls/openssl.cnf`**
- Sections are introduced by lines like **`[ section_name ]`**
- Section **`[ v3_req ]`** has good values:
  - basicConstraints = CA:FALSE
  - keyUsage = nonRepudiation, digitalSignature, keyEncipherment
- Use that section with **`-extensions v3_req`**
- Pulling things together:

```
# openssl req -new -x509 -subj "/CN=learn.tls.now/"
  -days 365 -extensions v3_req -key private/keys.key
  -out certs/selfsigned.crt
```


Deploy Certificates to Apache

- The files are already in the folders expected by Apache on Fedora
  - Therefore they have inherited the correct SELinux context
- Let **`mod_ssl`** know about them
  - Replace old values in **`/etc/httpd/conf.d/ssl.conf:`**

```
SSLCertificateFile /etc/pki/tls/certs/selfsigned.crt
SSLCertificateKeyFile /etc/pki/tls/private/keys.key
```

- Let Apache know about the change

```
# systemctl reload httpd.service
```

  - Use **`systemctl reload`**, because we just changed configs

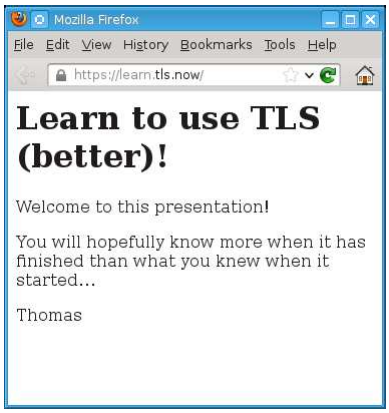
Trust the Certificate – Manually

- This works on a small scale: you and a few friends
- Obtain the "fingerprint" of the installed certificate:

```
# openssl x509 -in certs/selfsigned.crt -fingerprint
SHA1 Fingerprint=18:94:A1:5F:9B:23:3F:B6:5E:71[...]
```

- Users obtain the fingerprint from Firefox:
  - Reload https://learn.tls.now
  - Expand "I Understand The Risks", click on "Add exception…"
  - Click on "View…"
  - Check known value
    - With your friends: over the phone, in a letter, by carrier pidgeon…
  - Click on "Close"
- Only if the fingerprints matched:
  - Click on "Confirm Security Exception"

It Worked!



Beyond Friends, The Enterprise

- You cannot ask all employees to do this… every 365 days
- 365 days is a long time during which to protect the private key file
  1. It has to be present on the web server
  2. The web server is accessible from the intranet
  3. The intranet is teeming with Advanced Persistent Script Kiddies
  4. …
  5. Profit! (not for you)
- You don't want to repeat the exercise every time!

Trust the Certificate – by Delegation

- Web Browser trusts:
  - Long-term Company Keypair, which certifies:
    - Yearly Website A Keypair, Yearly Website B Keypair
    - Six months later, Yearly Website C Keypair
    - The next Long-Term Company Keypair
- Long-term private key:
  - Stored on USB stick in CEO's safe
  - Used only on off-line, freshly installed computer
  - CTO and CIO each know half the passphrase
- Public key, in a 5-year "Certification Authority" certificate:
  - Installed by IT whenever it provisions a corporate computer
  - Also published on the web site
  - Fingerprint available through multiple (authentic?) channels

The Company Keypair

- Generate keypair directly to the USB disk:

```
# cd /media/usbdisk/
# umask -S u=rw,go=
# openssl genpkey -algorithm rsa
  -out company_keys_2013.key
  -aes-128-cbc -pkeyopt rsa_keygen_bits:2048
  -pkeyopt rsa_keygen_pubexp:0x100000001
.....................................................+++
.....................................+++
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

- CIO and CTO discreetly enter their own half of the pass phrase
- The filename distinguishes this keypair from its predecessors

The Company CA Certificate

- Issue certificate for 5*365 days
- In the DN, mention the organisation, not the web site
- Let it be a CA certificate (done by the default settings)

```
# openssl req -new -x509 -days 1825 –subj
  "/C=BE/L=Brussels/O=TLS Now/CN=TLS Now Root CA
  2013/emailAddress=info@tls.now" –key
  company_keys_2013.key -out company_cert_2013.crt
Enter pass phrase for company_keys_2013.key:
```

- Preinstalling the CA certificate depends on browser and OS
- E.g. for Firefox 19 on another computer with Fedora 18

```
# yum install nss-tools
# for DIR in /home/*/.mozilla/firefox/*.default/; do
  certutil -A -n "TLS Now" -t "T,c,c" –I
  /media/otherusb/company_cert_2013.crt -d $DIR; done
```

The Yearly Website Certificate

- Generate a CSR (Certificate Signing Request)

```
# openssl req -new -subj "/CN=learn.tls.now/"
  -days 365 -extensions v3_req
  -key /etc/pki/tls/private/keys.key
  -out /etc/pki/tls/certreq.csr
```

- Sign it with the Company Keypair
  - Prepare a file to keep track of the serial numbers
  - Prepare a file to keep track of the issued certificates
  - Configure various other values in **`openssl.conf`**

```
# openssl ca -config /etc/pki/tls/openssl.cnf
  -in certs/certreq.csr -keyfile company_keys_2013.key
  -cert company_cert_2013.crt
  -out /etc/pki/tls/certs/corpsigned.crt
```

- Update **`ssl.conf`** to point to the new certificate

More Than One Hostname

- Co-hosting of related web sites (one IP address):
  - http://learn.tls.now:443
  - http://www.tls.now:443
  - http://tls.now:443
- Same web server, same configuration, one certificate
- The alternate names are specified at CSR generation time
  - Keep main name in the **`-subj`** argument
  - In the configuration file, in the **`[v3_req]`** section, add:
    `subjectAltName=DNS:www.tls.now,DNS:tls.now`
  - Generate the CSR as before

Trust the Certificate – Between Strangers

- Secure access by friends and employees is not enough
- Need to be able to prove it's you to strangers
- Employ Trusted Third Parties: professional Certification Authorities
- Luckily, browsers come with a bunch of them preloaded
  - Or unfortunately, in case one of them is corrupted
- Replace Step "Sign it with the Company Keypair" with the CA's procedure.
  - Certifying your certificate usually has a price tag

Extended Validation Certificates

- What exactly is certified?
- It should be: "requester's ownership of the FQDN"
- How stringent checks?
  - Verify photo ID of the requester?
  - Verify business ownership of the FQDN?
  - Verify control of the indicated e-mail address?
  - Verify possession of the private key?
  - Don't verify, just issue a limited-time "testing" cert?
- "Extended Validation" certificates
  - a.k.a. "the way it should have been done in the first place"
  - Has minimum validation requirements
  - Certificates contain a reference to the CA's verification policy
- Current browsers display additional positive assurance for EV certs

Certificate Chains

- There are sub-CAs and sub-sub-CAs, …
- Browser only knows root CA certificate in the beginning
- Web site must provide the intermediate links in order:
  - TLS RFC says so: each certificate certifies the one before it

```
-----BEGIN CERTIFICATE-----
MIICzzCCAbegAwIBAgIJAIXRQJSXK2HAYD
Ap  Sub-sub-CA certificate  FLVQQD
U2olbGhvc3QubG9jYWxkb
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDCTCCAnKgAwIBAgICNtQwDQYJKoDBVs
MR    Sub-CA certificate    ETb2Nh
IGS25sSHgrz8OU28eA==
-----END CERTIFICATE-----
```

- The cert chain goes into the file specified in ssl.conf

```
SSLCertificateChainFile
    /etc/pki/tls/certs/server-chain.crt
```


Certificate Expiration and Revocation

- At expiry, assume private key to be compromised
  - Cryptanalysis: deriving it from the public key
  - Breach: breaking into web server and copying the file
  - Brute force 1: trying enough candidates for a match
  - Brute force 2: “convince” a sysadmin to disclose it
- Generate a new keypair for every new certificate, it’s cheap
- Reasons for renewing the keypair even before expiration:
  - You have evidence of key compromise (see above)
  - You have lost control of the private key (no disaster recovery)
- Inform the CA, who updates a CRL or an OCSP server
- Browsers would ignore the old cert if they encountered it

“Padlock Security” Limitations

- Attacker lures victim to https://learn.t1s.now
- Attacker lures victim to https://learn.tls.now@attacker.corp
  - Here, learn.tls.now is a user name, not a domain name
  - Or hide attacker domain name: https://learn.tls.now@4.3.2.1
  - Or hide IP address, too: https://learn.tls.now@67305985
  - But browsers can highlight URL parts: http://**user@***host*/path
- Attacker spoofs your HTTP site and uses a padlock favicon
  - But browsers can be made to have unambiguous UI
  - Put padlock somewhere else than favicon, use colours

A CommonName Attack

- Attacker registers domain containing a ‘\0’ character
  - Special domain: learn.tls.now**\0**attacker.corp
- CA signs certificate on the full string
- Browser is written in C in which strings are terminated by zero:
  - It will display learn.tls.now
- But domain registrars and CAs now check for NULL characters
- And browsers are not fooled so easily any more, either

Another CommonИαмє Attack

- Attacker registers Unicode domain with characters resembling “/”
  - Combining Short Solidus Overlay (\u0337):⟋
  - Combining Long Solidus Overlay (\u0338):/
  - Fraction Slash (\u2014):⁄
  - Division Slash (\u2215):∕
- Special domain: learn.tls.now⁄ignore⁄attacker.com
- A legitimate certificate can be obtained!
- But browsers can display them in “Punycode” (RFC 3492)
  - http://ジェーピーニック.jp becomes
  - http://xn--hckqz9bzb1cyrb.jp, no more confusion

ACT II

“Configure TLS well on the server!”

Protocol Versions

- SSLv1: by Netscape, not public, insecure
- SSLv2: 1995 by Netscape, insecure
- SSLv3: 1997 by Netscape
  - 2002 theoretically insecure, 2011 practically insecure
- TLSv1.0 == SSLv3.1: by IETF
  - 2002 theoretically insecure, 2011 practically insecure
- TLSv1.1: 2006 by IETF, fixes 2002 attack, still not widely supported
- TLSv1.2: 2008 by IETF, still not widely supported

Protocol Versions

- The only option should be TLSv1.2 (OpenSSL is new enough)

```
<VirtualHost _default_:443>
    SSLProtocol TLSv1.2
</VirtualHost>
```

- But support in some browsers stops at SSLv3 and TLSv1.0
  - You may need to support those versions a while longer

```
<VirtualHost _default_:443>
    SSLProtocol +TLSv1.2 +TLSv1.1 +TLSv1 +SSLv3
</VirtualHost>
```

- Don't use **`SSLProtocol all`**
  - Older versions of Apache include insecure SSLv2 in **`all`**
- See later for avoiding SSLv3 and TLSv1 problems

Cipher Suites Specification

- Apache mod_ssl allows restricting OpenSSL crypto suites
  `SSLCipherSuite SPEC1:SPEC2:SPEC3:…`
  - Restrictions are inherited from the surrounding context
- Each SPEC can be a combination:
  - Key Exchange, Authentication, Encryption, Mode, Hashing
  - E.g. **`DHE-RSA-AES128-GCM-SHA256`**
  - Defaults can be omitted: e.g. **`AES128-SHA`** (RSA, RSA, CBC)
- SPEC can be a wildcard
  - **`AES`** == “all suites containing AES 128 or 256 encryption”
  - **`AES+SHA1`** == “all suites containing both AES and SHA-1”

Cipher Suites Specification (continued)

- SPEC can also have modifiers:
  - SPEC == add SPEC to the allowed suites
  - -SPEC == remove SPEC
  - !SPEC == remove SPEC and descendants cannot add it back
  - +SPEC == if SPEC was in the list, move it to the end (see later)
- SPEC can be an alias:
  - ALL == all supported cipher suites
  - HIGH == encryption with keys of at least 128 bits (except some)
  - TLSv1 == cipher suites specified by TLSv1.0
  - EXP == export grade ciphers
- List all supported cipher suites:
  `# openssl ciphers -v`

Exclude Short-Key Cipher Suites

- There are some insecure choices:
  - eNULL == no encryption (you get only other properties of TLS)
  - EXP == 40-bit and 56-bit ciphers, easy to crack on PCs
  - RC2, DES, IDEA – now deprecated
- Reconfigure Apache to allow only ciphers with sufficiently long keys
  `SSLCipherSuite –ALL:HIGH:MEDIUM:!RC2:!IDEA`
- If list is too specific, there may be no match with the browser
- With aliases the risk is that an undesired one will be included
  - Always expand specification list and check for undesired ciphers
  `# openssl ciphers –v "SPEC1:-SPEC2:+SPEC3"`

CBC Mode in Cipher Suites

- The BEAST attack exploits a design flaw in the use of CBC mode
  - BEAST == “Browser Exploit Against SSL/TLS”
  - Allows the theft of session cookies, leading to impersonation
- Design flaw had been removed in TLSv1.1, but the world waited…
- If you must support SSLv3 or TLSv1.0, allow:
  - Non-CBC-mode symmetric ciphers
  - Fallback to RC4 (smaller(?) problems than BEAST)
  - Fallback to CBC-mode ciphers if the priorities are:
    - Availability >> Integrity + Confidentiality
- Force the order of preference of the server:
  `SSLHonorCipherOrder on`
- List the categories in order, for example
  `SSLCipherSuite –ALL:AESGCM:RC4:HIGH:MEDIUM:!RC2:!IDEA`

Hashes in Cipher Suites

- MD5 collision resistance is badly broken
- TLS uses MD5 second preimage resistance, which is not broken
- But cryptologists say it is just a question of time
- We have better hashes:
  - SHA-1 – well supported in TLS implementations
  - SHA-256, SHA-384
  - SHA-3 – too new to be widely supported
- Do you want to wait and have another BEAST situation?
- Let's all kill this beast!

```
SSLCipherSuite ...:!MD5
```

- One day we will have to do it to SHA-1, too…

Avoid Insecure Renegotiation

- TLS supports security parameters renegotiation:
  - Server-initiated: needed when switching between site areas with different SSLCipherSuite configurations
  - Browser-initiated: needed… never?
- No-Auth to With-Auth: the renegotiation request is unauthenticated
- Initial negotiation and renegotiation used to look the same
- Attackers could sneak in SSL blocks before the legitimate ones
- The SSL layer passed to the HTTP layer a tampered request:

```
POST /transfer_money HTTP/1.1
Dummy-Header: GET /original_url HTTP/1.1
```

- Use a TLS library that supports secure renegotiation:
  - OpenSSL >= 0.9.8l
- Do not enable the old behaviour:

```
SSLInsecureRenegotiation off
```

Avoid Renegotiation DoS

- Renegotiation is more CPU-intensive on the server side
- The client side can trigger it sending ‘R’ on one line
- “R\nR\nR\nR\nR\nR\nR\nR\nR\nR\nR\n” == AppDoS
- Net-IDS sees only one TCP connection, it’s not a flood
- Turn off client-initiated renegotiation completely
  - Apache > 2.0.64 turned it off if OpenSSL was < 0.9.8l
  - With OpenSSL >= 0.9.8l, I couldn’t find how

Disable TLS Compression

- The CRIME attack exploits a flaw involving compression
  - CRIME == “Compression Ratio Info-leak Made Easy”
  - Allows the theft of session cookies, leading to impersonation
- For the time being compression should be disabled:
  `SSLCompression off`

Configuration Limitations

- The attacker can intercept packets and refuse to forward them
  - It is Denial of Service
  - But it is fail-securely
- An attacker can still attack through a secure channel
  - Password brute force attacks
  - Exploit vulnerabilities in the web app framework
  - Exploit vulnerabilities in the web app itself

ACT III

"Leverage TLS in the web app!"

Put No Secrets In The URL

- TLS will protect the URL in transit, but not:
  - On-screen, against shoulder surfing
  - In bookmarks
  - In the browser history
  - In the “restore last open tabs”
  - When the users themselves share it
- If you give the user a secret:
  - Prepare it for returning in the body of HTTP POSTs or in cookies
  - Refuse it via GET to educate users
- Configure web frameworks not to fall back on the URL when cookies are not available

Leave No Secrets Behind (On Proxies)

- Proxies that do not open the TLS tunnel don’t understand the traffic
- TLS-terminating proxies should know not to cache the traffic
- Defense in depth: add HTTP response headers for cache control
  - HTTP/1.1: **`Cache-Control: no-cache, no store, must-revalidate`**
  - HTTP/1.0: **`Pragma: no-cache`**
  - For maximum compatibility, add both:

```
void doGet(HttpServletRequest req,
           HttpServletResponse resp) {
    [...]
    resp.addHeader("Cache-Control",
        "no-cache, no store, must-revalidate");
    resp.addHeader("Pragma", "no-cache");
    [...]
}
```

Do Not Mix Secure With Insecure

- Don't Mix in a page HTTP and HTTPS resources
  - Even resources from third-party sites
- An attacker will tamper with the HTTP ones
  - JavaScript can be injected into HTML, JavaScript and CSS
  - Applets, Flash Movies can be hijacked
- Malicious and legitimate content may interact, leading to:
  - Tampering of supposedly integrity-protected content
  - Disclosure of supposedly confidential content
- URLs can be fingerprinted by what insecure resources they load
- By the way, session cookies should not be accessible to JavaScript
  - Set session cookies with the "HttpOnly" attribute

```
Cookie cookie = getMyCookie("myCookieName");
cookie.setHttpOnly(true);
```


Securely Submit Secrets

- Submitting the login form only via TLS goes without saying
  - Otherwise an eavesdropper might capture them
- Retrieving the login form must also be over TLS
  - Otherwise an attacker might tamper with it
  - … and make it send the credentials over HTTP
  - … or to a spoofed site
- But we may have been talking to the attacker from the beginning
  - Attacker does HTTP with us and HTTPS with the web server
  - Attacker never lets us see an HTTPS link!
- Educate users to expect correct HTTPS for retrieving the forms
  - Never make the login form accessible over HTTP yourself

Either HTTP, Or HTTPS

- Extend that idea from login forms to other sensitive content:
  - Serve all sensitive content only over HTTPS
  - Non-sensitive content can also be served over HTTPS
    - common images, style sheets, external JavaScript
- Sensitive content accessible over HTTP can be found through:
  - Guessing
  - Mis-typing of the URL by a legitimate user
- It decreases the care of legitimate users in protecting the sensitive content

TLS-Only Cookies

- Once logged in, the session cookie is the new temporary secret
- It should not be sent over HTTP, only HTTPS
  - But browsers send cookies whenever the domain matches
  - HTTP or HTTPS does not matter
  - A session cookie will not be expired, so it is eligible
- "But I don't provide HTTP links any more!"
  - An attacker's page could contain:

```
<IMG src="http://learn.tls.now/">
```

- "But I don't even have port 80 open!"
  - Attacker: `<IMG src="http://learn.tls.now:443"/>`
- Set any confidential cookies with the "Secure" attribute

```
Cookie cookie = getMyCookie("myCookieName");
cookie.setSecure(true);
```

TLS-Only Sites

- Extend that idea from cookies to the entire site
- There are browser plugins to try every link as HTTPS first
- The web site should be allowed to control this
  - "HTTP Strict Transport Security" to the rescue!
- Have the web application send the HTTP response header
  `Strict-Transport-Security: max-age=duration`
- Browser honors the header if it was retrieved over HTTPS
  - Otherwise it may be an attacker playing games
  - Exception: a preconfigured list of sites
- Browser honors the header for **`duration`** seconds
  - But every new response will push back that deadline

Sync App And TLS Identities Often

- TLS allows the server to authenticate the client, too
- Client certificate can be stored on the computer or on a smartcard
- Good idea: two-factor authentication with Belgian eID + PIN
  - National PKI: instant enrollment!
  - The user identity is in the client certificate: instant login!
- But pulling the card out of the reader is not logout!
  - Attacker inserts own eID and visits the site before timeout
  - TLS tunnel is set up in the attacker's name
  - Old session cookies mean the web app does not do a login
  - Web app sees victim, when it is the attacker
- Check correspondence session cookie vs. client certificate
  - At every request or
  - At every sensitive request

Keep Managing Sessions Carefully

- Just because the attacker can not eavesdrop and tamper on the connection does not mean game over for the attacker.
- The attacker can be at the scene before the victim arrives
- Attacker plants a session cookie and leaves
- The victim's browser sends the stale cookie together with the login credentials
- The web application does not generate a fresh session cookie, just "blesses" the old one
- Now the attacker can start sending requests with that cookie
  - "Session Fixation"!
  - The attacker can impersonate the victim
- Always generate fresh session cookies at successful login

TLS-Protected Web App Limitations

- While the attacker may not be able to take over the victim's session, the attacker's game is not over yet
- Attacker targets the victim's browser as confused deputy and attempts to insert requests into the legitimate session
- CSRF == the browser is tricked into making a request representing a command, while the user did not intend to issue it
- ClickJacking == the user issues commands thinking that he/she is interacting with a different site
- In both these cases TLS and session cookies give a false sense of trust in the authenticity of the requests in question

EPILOGUE

"Build a mobile app for our site!"

You Still Need TLS

- Not protecting the transport layer is #3 in OWASP Top10 Mobile Risks
- Current smartphones possess the battery and the CPU to do it
- The mobile network may have good security, but there are still segments over the public Internet
- Take into account the limited interface
  - Less room for status information
    - The app must decide for the user
  - Limited data input
    - Inconvenient to have to type in long secrets
    - Therefore, remember secrets for a longer time
    - But you may fail if device is stolen
- Non-browser apps are limited-purpose
  - Do certificate pinning

More Applications

- Drones
  - Control channel
  - Telemetry and video channels
- Smart meters
- Virtual Private Networks

Bibliography

- Fedora 16 System Administrator's Guide, Ch. 12 Web Servers: http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/System_Administrators_Guide/ch-Web_Servers.html#s2-apache-mod_ssl
- Apache 2.4 Module mod_ssl: http://httpd.apache.org/docs/2.4/mod/mod_ssl.htm
- OpenSSL man pages and HOWTOs: http://www.openssl.org/docs/
- OWASP Transport Layer Protection Cheat Sheet: https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet