Horror Stories about the Encrypted Web
(and how Let’s Encrypt is helping)

{pde,yan}@eff.org
Horror Story #1

We don’t live in a 100% HTTPS world
Horror Story #2

Setting up TLS is tedious, even in 2016
Purchasing a Signed Certificate from a Certificate Authority (CA)

You can also purchase a certificate directly from a Certificate Authority (CA) and install it in your DreamHost panel. To do this, you'll need a Certificate Signing Request (CSR) which can be found in your panel.

The following steps explain how to obtain this CSR in your panel:

1. Review the Adding Secure Hosting (self-signed certificate) section above to add Secure Hosting and a self-signed certificate to your domain.
2. Go to the (Panel > Domains' > Manage Domains) page.
   The 'Manage Domains' page opens:
3. To the right of your domain, click on the Certificates button.
4. When the 'Secure Hosting' page opens, click the 'Manual configuration' radio button to expose the current certificate information.
   There are several large text fields on this page:

Certificate Settings for dhwiki.dreamhosters.com

- Use a self-signed certificate
- Use a professionally signed certificate

5. COPY (do NOT cut) the text from the Certificate Signing Request field box.
6. Paste the text into the order form from whichever Certificate Authority you'd like to purchase your signed SSL certificate.

When purchasing a signed SSL certificate, you must specify the server type:
- To use the SSL certificate on DreamHost's servers, specify the server type as: Apache 2.X w/ MOD_SSL
- Once you successfully complete your purchase, the CA will send you the signed SSL certificate; you can then replace the self-signed SSL certificate with this signed SSL certificate in your panel.
- For more information, see the instructions in the following Installing a Signed Certificate you've already purchased section.
Horror Story #3

TLS configuration is confusing
It is widely believed that AES-CBC is a secure cipher for the long term, unlike RC4. Choosing AES-CBC provides our customers with long-term forward secrecy, even if it could open them up to a rarely executed noisy active attack if they are using an out of date browser and OS. Choosing RC4 exposes our customers’ data to any government who has advanced enough cryptographic techniques to break it. When faced with this choice, we would rather protect our customers from long term threats by choosing AES-CBC. Experts agree, it’s time to move on from RC4.

Who uses RC4?

Instead of just removing RC4 altogether, we decided first to lower the priority of the RC4 cipher suites on our servers. Typically in HTTPS, the client lets the server know which cipher suites it supports, from which the server picks their favorite. By putting RC4 last in terms of server preference, we will only choose RC4 if it the client does not support anything else. The following chart shows what happened when we changed our cipher preferences.
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;

# Using list of ciphers from "Bulletproof SSL and TLS"
Horror Story #4

Mixed content blocking
1.2. Examples

EXAMPLE 1

Megacorp, Inc. wishes to migrate http://example.com/ to https://example.com. They set up their servers to make their own resources available over HTTPS, and work with partners in order to make third-party widgets available securely as well.

They quickly realize, however, that the majority of their content is locked up in a database tied to an old content management system, and it contains hardcoded links to insecure resources (e.g., http:// URLs to images and other content). Unfortunately, it’s a substantial amount of work to update it.

As a stopgap measure, Megacorp injects the following header field into every HTML response that goes out from their servers:

```
Content-Security-Policy: upgrade-insecure-requests
```

This automatically upgrades all insecure resource requests from their pages to secure variants, allowing a user agent to treat the following HTML code:

```
<img src="http://example.com/image.png">
<img src="http://not-example.com/image.png">
```

as though it had been delivered as:

```
<img src="https://example.com/image.png">
<img src="https://not-example.com/image.png">
```
Horror Story #5

There are too many certificate authorities.
it’s time to fight back
So we started a CA...
(one more CA)
Let’s Encrypt is a new Certificate Authority:

It’s **free, automated, and open**.

In Public Beta
Let’s Encrypt created by

- **Engineering**: EFF, Mozilla, University of Michigan
- **Financial sponsorship**: Cisco, Akamai
- **CA cross-signature**: IdentTrust
- **Housed in a new 501(c)3**, the Internet Security Research Group (ISRG)
Security

How do we decide whether to issue a cert?
Dialog: ACME protocol
Shrubberies: ACME “challenges”
Current status

Private beta through Nov, 2015
Entered public beta on Dec. 3, 2015
Issued 10k certs in <8 hours (1 cert / 3 seconds!)
Almost 400k certs issued so far!
# More statistics

374714 certificates checked (totalling 801637 DNS names)

# adoption statistics

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>names using issued cert</td>
<td>547,200</td>
<td>68.26%</td>
</tr>
<tr>
<td>certs used by all names</td>
<td>162,844</td>
<td>43.46%</td>
</tr>
<tr>
<td>certs used by some names</td>
<td>11,341</td>
<td>3.03%</td>
</tr>
<tr>
<td>certs used by no names</td>
<td>200,529</td>
<td>53.52%</td>
</tr>
</tbody>
</table>
# cipher suite breakdown

<table>
<thead>
<tr>
<th>Cipher Suite</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDHE RSA WITH AES 256 CBC SHA</td>
<td>4,817</td>
<td>(1.33%)</td>
</tr>
<tr>
<td>RSA WITH AES 256 CBC SHA</td>
<td>1,354</td>
<td>(0.37%)</td>
</tr>
<tr>
<td>RSA WITH AES 128 CBC SHA</td>
<td>27,551</td>
<td>(7.63%)</td>
</tr>
<tr>
<td>RSA WITH 3DES EDE CBC SHA</td>
<td>104</td>
<td>(0.03%)</td>
</tr>
<tr>
<td>ECDHE RSA WITH 3DES EDE CBC SHA</td>
<td>30</td>
<td>(0.01%)</td>
</tr>
<tr>
<td>ECDHE RSA WITH AES 128 GCM SHA256</td>
<td>222,517</td>
<td>(61.59%)</td>
</tr>
<tr>
<td>ECDHE RSA WITH AES 256 GCM SHA384</td>
<td>98,427</td>
<td>(27.24%)</td>
</tr>
<tr>
<td>ECDHE RSA WITH AES 128 CBC SHA</td>
<td>6,516</td>
<td>(1.80%)</td>
</tr>
</tbody>
</table>

Lots of forward secrecy!
Alexa top domains using Let’s Encrypt

- archlinux.org
- teamliquid.net (Starcraft news site)
- overclockers.ru (electronics / tech news site)
- gimp.org
- distrowatch.com
- goodlife.tw (shopping promotions site)
- douglas.de (cosmetics site)

More at https://censys.io/domain?q=%28*%29+AND+443.https.tls.certificate.parsed.issuer.common_name%3A+%22Let%27s+Encrypt+Authority+X1%22
Client types and plans...
Multi-server (load balanced)

Bulk hosting (no user shell)

Single server (VPS, self-hosted, managed hosting etc)

Large, custom infrastructures
Bulk hosting (no user shell)

Single server (VPS, self-hosted, managed hosting etc)

Multi-server (load balanced)

Large, custom infrastructures
Bulk hosting (no user shell)

Single server (VPS, self-hosted, managed hosting etc)

Multi-server (load balanced)

Large, custom infrastructures
Bulk hosting (no user shell)

Single server (VPS, self-hosted, managed hosting etc)

Multi-server (load balanced)

Large, custom infrastructures

Bulk hosting (no user shell)
Diverse clients...
Rather old server software!
class IAAuthenticator(IPlugin):
    """Generic Let's Encrypt Authenticator."""

def get_chall_pref(domain):
    """Return list of challenge preferences."""

def perform(achalls):
    """Perform the given challenge."""

def cleanup(achalls):
    """Revert changes and shutdown after challenges complete."""
class IInstaller(IPlugin):
    """Generic Let's Encrypt Installer Interface. """

    def get_all_names():
        """Returns all names that may be authenticated."""

    def deploy_cert(domain, cert_path, key_path, chain_path, fullchain_path):
        """Deploy certificate."""

    def enhance(domain, enhancement, options=None):
        """Perform a configuration enhancement. """

    def supported_enhancements():
        """Returns a list of supported enhancements. """

    def get_all_certs_keys():
        """Retrieve all certs and keys set in configuration. """

    def save(title=None, temporary=False):
        """Saves all changes to the configuration files. """

    def rollback_checkpoints(rollback=1):
        """Revert `rollback` number of configuration checkpoints. """

    def recovery_routine():
        """Revert configuration to most recent finalized checkpoint. """

    def view_config_changes():
        """Display all of the LE config changes."""

    def config_test():
        """Make sure the configuration is valid."""

    def restart():
        """Restart or refresh the server content."""
<table>
<thead>
<tr>
<th>Plugin Type</th>
<th>Installer Type</th>
<th>Use Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authenticator/standalone</td>
<td>Installer/none</td>
<td>6,602</td>
</tr>
<tr>
<td>Authenticator/webroot</td>
<td>Installer/none</td>
<td>5,769</td>
</tr>
<tr>
<td>Authenticator/apache</td>
<td>Installer/apache</td>
<td>4,875</td>
</tr>
<tr>
<td>Authenticator/plesk</td>
<td>Installer/plesk</td>
<td>2,850</td>
</tr>
<tr>
<td>Authenticator/manual</td>
<td>Installer/none</td>
<td>674</td>
</tr>
<tr>
<td>Authenticator/apache</td>
<td>Installer/none</td>
<td>351</td>
</tr>
<tr>
<td>Authenticator/webroot</td>
<td>Installer/apache</td>
<td>92</td>
</tr>
<tr>
<td>Authenticator/standalone</td>
<td>Installer/apache</td>
<td>56</td>
</tr>
<tr>
<td>Authenticator/nginx</td>
<td>Installer/nginx</td>
<td>29</td>
</tr>
<tr>
<td>Authenticator/manual</td>
<td>Installer/apache</td>
<td>28</td>
</tr>
<tr>
<td>Authenticator/s3front</td>
<td>Installer/s3front</td>
<td>23</td>
</tr>
<tr>
<td>Authenticator/nginx</td>
<td>Installer/none</td>
<td>13</td>
</tr>
<tr>
<td>Authenticator/webroot</td>
<td>Installer/nginx</td>
<td>2</td>
</tr>
<tr>
<td>Authenticator/standalone</td>
<td>Installer/null</td>
<td>1</td>
</tr>
<tr>
<td>Authenticator/gandi-shs</td>
<td>Installer/gandi-shs</td>
<td>1</td>
</tr>
<tr>
<td>Authenticator/gandi-shs</td>
<td>Installer/none</td>
<td>1</td>
</tr>
<tr>
<td>Authenticator/webroot</td>
<td>Installer/s3front</td>
<td>1</td>
</tr>
<tr>
<td>Authenticator/manual</td>
<td>Installer/nginx</td>
<td>1</td>
</tr>
</tbody>
</table>
How well configured are we?
# some further work required

# name problems

<table>
<thead>
<tr>
<th>Issue</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid DNS</td>
<td>12106</td>
<td>1.51%</td>
</tr>
<tr>
<td>refused/unavailable</td>
<td>27108</td>
<td>3.38%</td>
</tr>
<tr>
<td>timed out</td>
<td>22628</td>
<td>2.82%</td>
</tr>
<tr>
<td>TLS error</td>
<td>7627</td>
<td>0.95%</td>
</tr>
<tr>
<td>sent incomplete chain</td>
<td>26648</td>
<td>3.32%</td>
</tr>
<tr>
<td>expired cert</td>
<td>5582</td>
<td>0.70%</td>
</tr>
<tr>
<td>self-signed cert</td>
<td>10</td>
<td>0.00%</td>
</tr>
<tr>
<td>cert has wrong names</td>
<td>84172</td>
<td>10.50%</td>
</tr>
<tr>
<td>misc. invalid cert</td>
<td>3</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

# feature usage

<table>
<thead>
<tr>
<th>Feature</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCSP stapled</td>
<td>52797</td>
<td>6.59%</td>
</tr>
<tr>
<td>SCT included</td>
<td>159</td>
<td>0.02%</td>
</tr>
</tbody>
</table>
Vulnerability reporting
Test that our copy of go-jose ignores the JWS "algorithm" field

Remove insecure challenges

Address signature reuse vulnerability

Mitigate signature misuse vulnerability

SimpleHTTP validation accepts "..." components

Do not accept HMAC-based JWS signatures

Verify safety of using "encoding/asn1" parsing (e.g. of CSRs)

WFE should not return arbitrary errors to client

Some endpoints in web-front-end.go need to check for key continuity

Audit Logging

Don't ignore random number generation errors
I recently reviewed draft-barnes-acme-04 and found vulnerabilities in the DNS, DVSNI, and Simple HTTP challenges that would allow an attacker to fraudulently complete these challenges.

I shall describe the DNS challenge vulnerability first, since it is the most serious as it requires no MitM or other network-layer subversion. The assumptions are:
Mallory wants to prove ownership of example.com via DNS challenge

1. Mallory registers RSA key pair [1] for challenge signing
2. letsencrypt issues DNS challenge [2]
6. letsencrypt verifies that [3] is a valid signature from Mallory’s new account key, and issues Mallory cert for example.com.
“The real problem is that ACME makes false assumptions about signatures. It assumes that a signature uniquely identifies a (public key, message) tuple, which RSA does not guarantee.”
This PR addresses the signature reuse vulnerability noted by Andrew Ayer.

https://mailarchive.ietf.org/arch/msg/acme/F71iz6qq1o_QPVhJCV4dqWf-4Yc

It removes the unnecessary signatures on validation objects used in challenges, and replaces it with a simpler structure where the domain holder specifies which account keys are authorized, through which challenges.

Currently WIP, for discussion.
first vuln reported in production

CAA records not verified #1231

AlexanderS opened this issue on Dec 7, 2015 · 4 comments

AlexanderS commented on Dec 7, 2015

I just checked if letsencrypt.org verifies CAA Records and it seems that it does not respect it. I just got a certificate for a domain that has a CAA record, that does not allow letsencrypt.org to issue a certificate.

The requested domain is asulfrian.userpage.fu-berlin.de and this are the CAA records for fu-berlin.de:

```
fu-berlin.de. 86400 IN CAA 0 issue "pki.dfn.de"
fu-berlin.de. 86400 IN CAA 0 iodef "mailto:certificate@fu-berlin.de"
```

There are no CAA records for the sub domains. Reading the relevant section in rfc6844 shows clearly that this records should be the relevant ones. So I should not be able to get this certificate.
- Reported 9:45 PST on 12/7
- Fix deployed 13:11 PST on 12/7
- 6 certs misissued; all revoked
Things we haven’t solved...
Things we haven’t solved...

- Mixed content :(
Mixed content problems

Content Security Policy upgrade-insecure-requests

- Was supposed to help
Except . . .

- Passive mixed content (images) isn’t blocked usually
- Many (most?) HTTP embedded images aren’t available over HTTPS
- Upgrade => more broken than before
https://isnot.org/mixed-uir/
Mixed Content Problems

In theory, report-only CSP is promising

In practice, auto-collecting the reports is tricky
Want to get hacking?

Spec

https://github.com/letsencrypt/acme-spec

Main Client

https://github.com/letsencrypt/letsencrypt

Server

https://github.com/letsencrypt/boulder
(And help us Encrypt the Web, entirely)