



Search


Featured Posts

- 

Project Catalyst - Build for Desktop and Mobile

Post date: July 14, 2019
- 

Android Encryption Updates from I/O 2019

Post date: June 8, 2019
- 

WWDC 2019 Wishlist

Post date: May 31, 2019

Featured Pens

- 


Night and Day

GSAP JavaScript animation that displays either a day or night scene based upon your time of day.
- 


Web Trivia Game

A plain and and simple web trivia game that uses a little GSAP and NO jQuery.

Featured Tutorials

- 

How to Gather URLSession Connection Metrics

If you are like me you may be spending a good deal of development time debugging network...
- 

How to install RVICTL with Xcode 11

How to Diagnose App Transport Security Issues using nscurl and OpenSSL

Submitted by [matt_eaton](#) on Sat, 05/05/2018 - 08:35 AM



Tags: [Networking](#) [Network Security](#)



In the last couple of months I have found myself in situations where I have needed to diagnose transport security issues from from the context of an iOS application. This often can be difficult from the client side perspective as you may not know anything about the minimum TLS version, preferred cipher suite, or the certificate in use on the server. Often, the best move you have, if you do not have a direct line to the server side team, is to test different settings to diagnose what works and what does not. Then confirm your successful tests and transport setting with the server side team when they are available. Well, in debugging these types of issues I discovered the [nscurl](#) tool available in macOS. And that is why I wanted to write this tutorial, to provide a brief explanation on how to use nscurl, and what it can be used for.

NOTE: This tutorial was written and test on macOS version 10.13.5 beta, but should work on any macOS 10.13 version of the operating system as well. I have not tested this on anything below 10.13.

Using the nscurl Utility

If you are on OS X El Capitan or later then you most likely have the nscurl utility at your disposal. To test this out, go to the terminal and type in the following command: `$ /usr/bin/nscurl -h` and you should see a set of options like the one in the screen shot below along with the usage pattern set as `$ nscurl [options...] <URL>`.

```
[Matts-MBP:~ matt_eaton$ /usr/bin/nscurl -h
Usage: nscurl [options...] <URL>
Options:
  -h      --help                Display help message
  -bg     --background          Use the background transfer
                                API to execute the request
                                Marks the transfer as
                                discretionary (only has an
                                effect if --background is
                                passed)
  -D      --dump-header [ARG]  Write the response headers to
                                the specified file (pass '-'
                                for stdout)
  -o      --output [ARG]       Write the response data to
                                the specified file (pass '-'
                                for stdout)
  -i      --include             Include response headers in
                                output
  -L      --location            Instruct nscurl to follow
                                redirects (this is the
                                default behavior)
  --ignore-location            Instruct nscurl to ignore
```

In this micro tutorial I
wanted to cover
installing...

Popular Tags

- BluetoothBluetooth
- Network Security
- Network Security
- Network Security
- Networking
- Networking
- NetworkingMobile
- Mobile
- Network Security
- Networking
- NetworkingAndroid
- Android

So how can this help you diagnose app transport security issues or TLS issues? Well, pointing this tool at a URL and running it with the `--ats-diagnostics` option will provide you the exact information on what version of TLS is configured and whether or not perfect forward secrecy is enable for the secure communication between your application and the server.

So running `nscurl` with the `--ats-diagnostics` option against `www.agnosticdev.com` will produce the results of of what version of TLS the server can communicate with. In the screen shot notice that TLS 1.0, 1.1, and 1.2 all pass, but 1.3 fails with a `-9800 kCFStreamErrorDomainSSL` error. Meaning that TLS 1.3 is not configured yet but are 1.0, 1.1, and 1.2 are all configured and ready to use.

```
$ /usr/bin/nscurl --ats-diagnostics https://www.agnosticdev.com

=====
Configuring TLS exceptions for www.agnosticdev.com
-----
TLSv1.3
2018-05-04 05:42:12.923 nscurl[42605:1674176] NSURLSession/NSURLConnection HTTP load failed (kCFStreamErrorDomainSSL, -9800)
Result : FAIL
-----

TLSv1.2
Result : PASS
-----

TLSv1.1
Result : PASS
-----

TLSv1.0
Result : PASS
-----
```

Next let's take a look at the PFS results, or better known as perfect forward secrecy. Perfect forward secrecy is a security mechanism in secure communication that ensures that if current or future session is compromised that this does not also compromise past sessions, and thus the name "forward secrecy."

The results displayed in the screen shot below are testing with `nscurl` against `agnosticdev.com` where the TLS version is set to a specific version and PFS flag is disabled. Thus testing whether your version of TLS supports PFS running with disabled. The results in the screen shot indicate that all versions of supported TLS can also support PFS disabled. So, the results below would indicate that the following App Transport Security settings would be valid to use in your app:

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSExceptionDomains</key>
  <dict>
    <key>agnosticdev.com</key>
    <dict>
      <key>NSExceptionRequiresForwardSecrecy</key>
      <false/>
      <key>NSExceptionMinimumTLSVersion</key>
      <string>TLSv1.2</string>
    </dict>
  </dict>
</dict>
```

Alternative Approach 👍

One other alternative approach if you are on a unix machine is to use the `openssl s_client` tool suite to validate options about a remote cert or a SSL/TLS setup. This can be very helpful for use cases outside of macOS, on Linux, where you do not have `nscurl` at your disposal. In fact, `openssl` may provide you more granular information about the cipher suites and the certificate authority in use

```
Configuring TLS exceptions with PFS disabled for www.agnosticdev.com
-----
TLSv1.3 with PFS disabled
2018-05-04 05:42:13.387 nsurl[42605:1874176] NSURLSession/NSURLConnection HTTP load failed (kCFStreamErrorDomainSSL, -9800)
Result : FAIL
-----
-----
TLSv1.2 with PFS disabled
Result : PASS
-----
-----
TLSv1.1 with PFS disabled
Result : PASS
-----
-----
TLSv1.0 with PFS disabled
Result : PASS
-----
```

on the remote server than nsurl does. If I had to guess also, nsurl is most likely using openssl under the hood too. (Again, that is just a guess)

```
$ openssl s_client -showcerts -connect www.agnosticdev.com:443
```

In Summary ⌚

In summary I hope this tutorial has been helpful if you are struggling with what the best possible option is to configure for App Transport Security in your application. These settings can be a bit confusing at times, so it is always best to go right to the source if you can and test the remote SSL/TLS communication back to the client. Please leave a comment if you have any questions comments or concerns, and I will try and get back to you as soon as possible. Thank you!

CREDITS: COVER IMAGE DESIGNED BY [FREEPIK](#).



By: Matt Eaton

Long time mobile team lead with a love for network engineering, security, IoT, oss, writing, wireless, and mobile. Avid runner and determined health nut living in the greater Chicagoland area.

[LinkedIn](#) [Twitter](#) [Github](#)

Comments

Venkat

Wed,
10/17/2018
- 09:43 AM

[Permalink](#)

Good post

Nice post. Really this is very useful and informative post. Thanks for sharing this post.

- [Blog Entries](#)
- [Tutorials](#)
- [Sitemap](#)
- [Github](#)
- [Twitter](#)
- [LinkedIn](#)
- [Python](#)
- [Networking](#)
- [Objective-C](#)