Solid Financial Technologies Inc. Change Notice: AWS Cloudfront Cipher-Suite Upgrade

document version: v002
date: 2023/02/07

Introduction

The DevOps Group at Solid Financial Technologies Inc. ("Solid") sends this Change Notice to partnering Programs to advise them of a soon-coming upgrade to Solid's Amazon Web Services ("AWS") Cloudfront security configuration. This upgrade will improve security for partnering Programs and their end-users, and help Solid stay in line with evolving compliance requirements and standards.

- **What role does AWS Cloudfront play in Solid's infrastructure?**
    - among other things, AWS Cloudfront manages internet connections into Solid's backend infrastructure
    - Solid's backend infrastructure hosts the Solid API, and also web services/sites used by Program personnel and Programs' end-users
- **How is Solid's AWS Cloudfront configuration changing?**
    - the DevOps Group will upgrade their production-level instances ("distributions") of AWS Cloudfront to "*Security Policy TLSv1.2_2021*"
    - the changes involved in the *Security Policy TLSv1.2_2021* are documented at https://aws.amazon.com/about-aws/whats-new/2021/06/amazon-cloudfront-announces-new-tlsv12_2021-security-policy-for-viewer-connections/
        - the upgraded security policy will remove two ciphers from the cipher-suite, while supporting six widely supported ciphers
        - the upgraded security policy also will "*set the minimum negotiated Transport Layer Security (TLS) version to 1.2*"
    - the cipher-suite is used during SSL/TLS/certificate negotiation when establishing secure internet connections between an external client and, in our case here, Solid's backend services, as generally described in: https://en.wikipedia.org/wiki/Cipher_suite
- **When will Solid's DevOps Group apply this change to Solid's production-level AWS Cloudfront?**
    - on, or soon after, **Mar 8, 2023**
- **How might this change affect a Program and its end-users?**

    - the cipher-suite upgrade could possibly break client software that contacts Solid's API or Solid's web services/sites if those clients do not properly support any of the ciphers in the Cloudfront cipher suite
    - **for Programs' end-users:**
        - End-users that use Solid websites via web browsers (Chrome, Microsoft Edge, Safari, Mozilla Firefox, and other desktop/laptop/mobile browser) will usually have up-to-date browsers that support TLS/SSL/certificate negotiation with the updated Cloudfront cipher-suite. If their browsers are very out of date they will have troubles with other consumer-facing websites that are upgrading their security policies as well, and so these end-users will generally need to upgrade their browsers
        - End-users may also use native apps installed on their iOS, Android, or other mobile operating-system devices. These apps typically are written and distributed by Programs themselves, and incorporate third-party or OS-provided libraries for SSL/TLS communication with Solid's API. These SSL/TLS libraries must be compatible with the cipher-suite in the upgraded AWS Cloudfront Security Policy for continued communication with the Solid APIs. *There are more instructions below on how to test compatibility with the upgraded AWS Cloudfront Security Policy*.
    - **for Program-internal-personnel using the Solid Dashboard:**
        - Program personnel use the Solid Dashboard via web browsers in much the same way, as far as the cipher-suite upgrade is concerned, that end-users do for other Solid web sites/services. The conditions for continued successful use by end-users of Solid's websites apply here as well for Program-internal-personnel use of the Solid Dashboard.
    - **for Programs' own backend/server-side infrastructure:**
        - Programs usually have their own server-side/backend infrastructure that will communicate with the Solid API. This software may be written in one, or several, programming languages like Java, Python, go, C#, etc., and will use software libraries or a programming-language runtime to communicate with Solid's API. These libraries usually will handle SSL/TLS/certificate negotiation with Solid's API ... and as mentioned above, on Solid's end, Cloudfront handles this SSL/TLS/certificate negotiation. *A Program's backend software must be able to successfully negotiate SSL/TLS/certificate with Cloudfront's upgraded cipher-suite*. *There are more instructions below on how to test compatibility with the upgraded Cloudfront Security Policy*. A Program's backend software that cannot successfully negotiate SSL/TLS connections through Cloudfront will not reach Solid's API and thus will be broken.
- **How can a Program test compatibility with the upgraded Cloudfront *Security Policy TLSv1.2_2021* and its cipher-suite?**
    - a Program cannot test compatibility with the updated Cloudfront by using the regular production-level Solid web services/sites, or the production Solid API, *as these are not yet upgraded*

- a Program can, however, test connectivity against Solid's QA infrastructure
  - when testing against Solid's QA infrastructure, *a Program does not need to fully authenticate and access the full functionality of the target website or API ... rather, a Program need only ensure that SSL/TLS negotiation happens properly at the browser or API level ... there are clear ways to verify that this SSL/TLS negotiation has happened successfully*

What Access Should Be Tested

Again, the production-level websites and Solid APIs have not yet been upgraded to Cloudfront *Security Policy TLSv1.2_2021* with the newer cipher-suite, whereas the QA websites and QA Solid APIs have been thus upgraded. To test, Programs will will want to point their browsers or built software (native-mobile or backend services) to the QA URLs and ensure they can contact websites / APIs in the Solid's QA and get the *expected outcome ... and that expected outcome does not include full authentication and use of those services and APIs, but rather the cipher-suite successful SSL/TLS negotiation can be verified by showing that software behind Cloudfront is actually reached, though the transaction is incomplete or has an expected error*.

Accessed Solid Resource	**Production URL/Hostname**	**QA URL/Hostname Used for Testing**	**Who/What Is Affected**
Solid "Cards" website	https://solid.cards	https://qa-card.sbx.solidfi.com	Programs end-users
Solid "App" website	https://solid.app	https://qa-superbank.sbx.solidfi.com	Program end-users
Solid Dashboard	https://dashboard.solidfi.com	https://qa-dashboard.sbx.solidfi.com	Program internal personnel
Solid APIs	`api.solidfi.com` `test-api.solidfi.com`	`qa-api.sbx.solidfi.com`	Program-built software: ● native mobile apps (iOS, Android, etc.) ● Program's backend server-side infrastructure

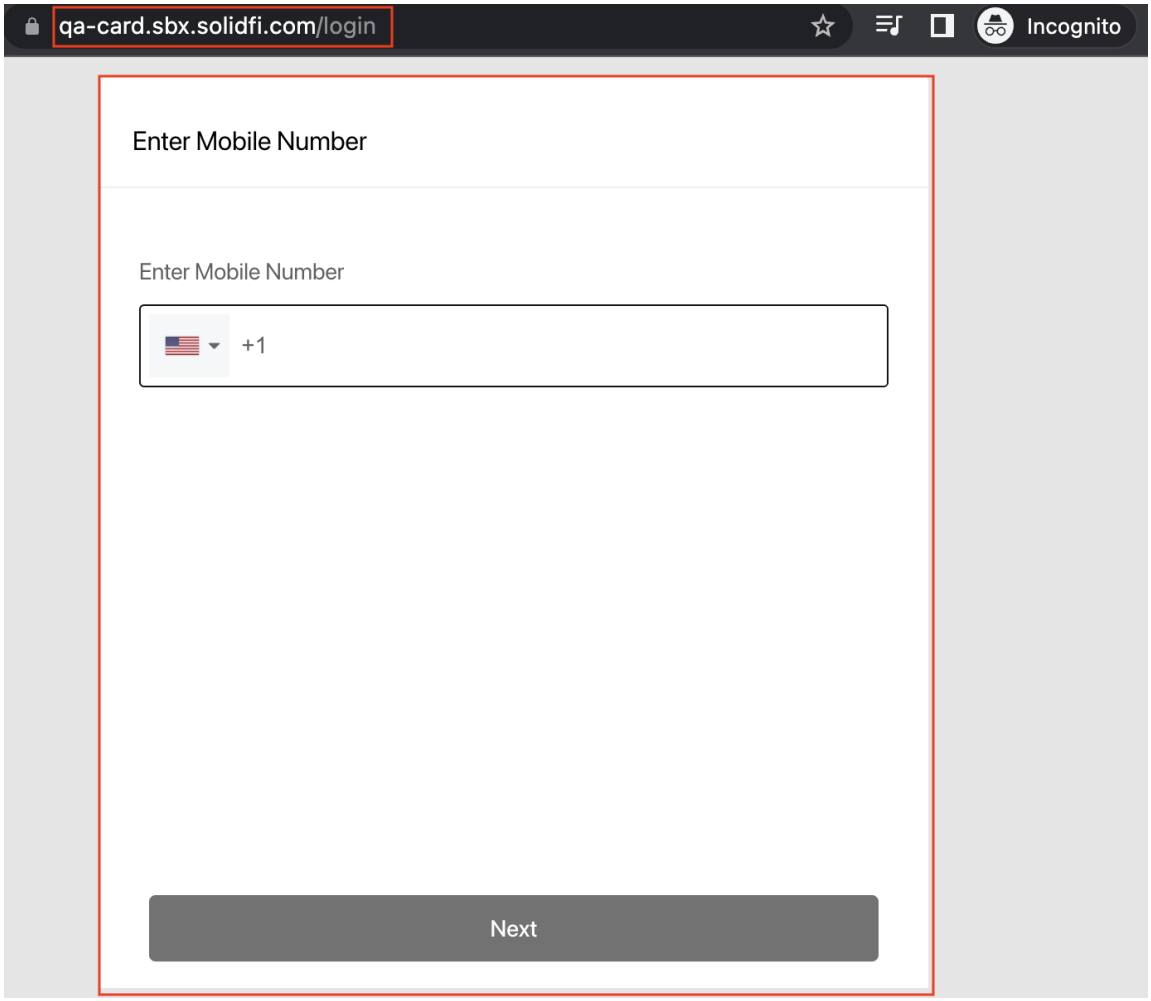
Note that for contexts where web browsers contact a Solid Resource (end-users accessing *Solid "Cards" website* or *Solid "App" website*, Program-internal-personnel accessing *Solid Dashboard*), most users will have up-to-date web browsers that support the upgraded Cloudfront cipher-suite. A user that has an obsolete browser and has trouble accessing the Solid website will likely have trouble accessing many other websites as well. Also, it's likely not feasible for a Program to test every possible operating system / browser combination, but Solid provides test procedures so that Programs can do their own testing and upgrade Program personnel's browsers if required, and also advise end-users about possible issues using the Solid websites.

Note that for contexts where native mobile apps (on iOS, Android, etc.) or Programs' server-side software contact the Solid APIs, it's likely much more feasible for a Program to perform proper testing for all their software stacks that call the Solid APIs. We show below example code that

Programs can *adapt to their own language and native-mobile- or server-side-software contexts* to test API access. In lieu of actual tests, a Program may opt to consult their SSL/TLS software libraries' documentation to ensure they support the Cloudfront upgraded cipher-suite. If those libraries are not compatible, Programs may need to upgrade those library versions or replace them with others that are compatible.

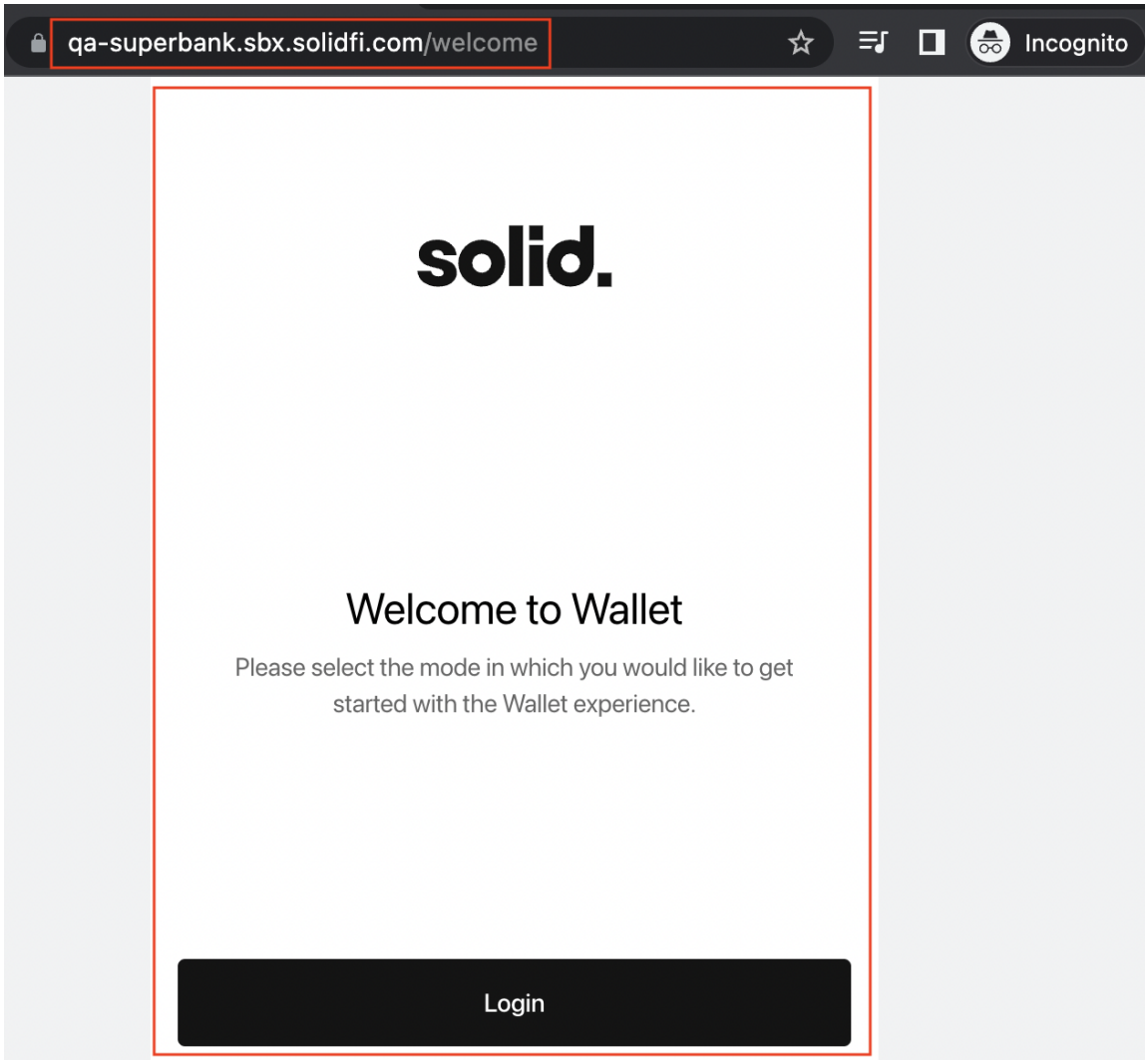
Testing *Solid "Cards" Website*

To test future browser access to https://solid.cards for when its fronting Cloudfront is eventually upgraded, Program personnel can open QA URL https://qa-card.sbx.solidfi.com (already fronted by an upgraded Cloudfront) in the tested browser on the tested device/operating system. If the following content appears in the browser window, the browser is shown compatible with the upgraded Cloudfront.



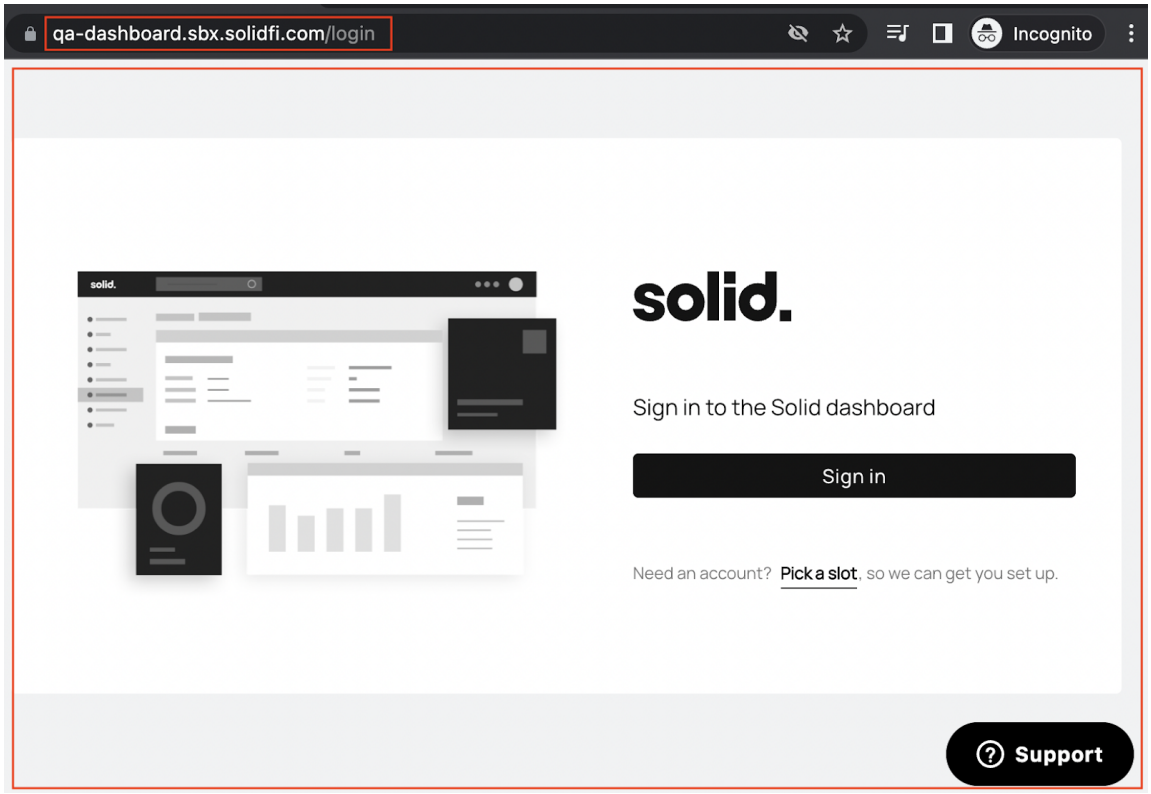
Testing *Solid "App" website*

To test future browser access to https://solid.app for when its fronting Cloudfront is eventually upgraded, Program personnel can open QA URL https://qa-superbank.sbx.solidfi.com (already fronted by an upgraded Cloudfront) in the tested browser on the tested device/operating system. If the following content appears in the browser window, the browser is shown compatible with the upgraded Cloudfront.



Testing *Solid Dashboard*

To test future browser access to https://dashboard.solidfi.com for when its fronting Cloudfront is eventually upgraded, Program personnel can open QA URL https://qa-dashboard.sbx.solidfi.com (already fronted by an upgraded Cloudfront) in the tested browser on the tested device/operating system. If the following content appears in the browser window, the browser is shown compatible with the upgraded Cloudfront.



Testing *Solid APIs*

Each Program will have its own software development stack, or stacks, and surrounding practices, for software that contacts the Solid APIs. These stacks would include:

- operating system
- programming language(s)
- software libraries, including those that manage SSL/TLS negotiation
- build/test procedures
- CI/CD pipelines
- deployment context
    - for server-side software ...
        - in-house/private or rented data-center with managed physical hardware
        - cloud services such as AWS, Azure, Google-Cloud, etc.
        - (perhaps others)
    - for native-mobile software ... apps deployed on ...
        - iOS
        - Android
        - other mobile operating systems

Solid cannot supply tests for all these possible contexts, but can supply examples on which to base tests. *It's important that when testing access to Solid APIs that the same libraries and methods be used to call the QA version of the Solid API (hostname* `qa-api.sbx.solidfi.com`*) as are used to contact the production-level Solid API servers (hostnames* `test-api.solidfi.com` *and* `api.solidfi.com`*).* The examples that Solid provides below should be reproduced in the appropriate "stacks" and contexts a Program wants to test. Also, full authentication and use of the services is not required in order to get a "successful" verification ... it's only important that the built software manage to reach the Solid API servers behind Cloudfront, and get the expected response.

An Example `curl` Snippet

Most Programs will not be using `curl` directly to interact with Solid's API, but it's a useful tool for a quick example. Here's a call that demonstrates a successful call to the QA host `qa-api.sbx.solidfi.com` that manages to get beyond the Cloudfront SSL/TLS negotiation, with the expected outcome:

```
# call QA Solid API

   # some STDERR logging output is excised ... jq tool is used
for pretty-formatting, can omit
$  curl --verbose
https://qa-api.sbx.solidfi.com/v1/webhook/events | jq

...
< HTTP/2 401
< content-type: application/json
...
{
  "requestId":
"webhook-external-deployment-58d775db9-mmlth;6ZmE5QSfSj;qa;req-05
8c824c-cc8c-405e-bb85-b992ddd30b64",
  "method": "GET",
  "status": 401,
  "code": "EC_AUTH_ERROR",
  "message": "unauthorised",
  "sysMessage": "unauthorised",
  "programId": "",
  "env": "qa",
  "createdAt": "2023-02-07T19:18:32Z"
}
```

Even though this `curl` request ends with an error, *the error is generated by the Solid API software itself, after successful SSL/TLS negotiation*. This is an expected outcome and shows that we're able to reach the API.

An Example `python3` Snippet

The `curl` example above does not replicate the software stack or context most Programs will use to contact Solid APIs. Programs will instead want to replicate the functionality in this Python snippet below, especially function `reach_solidfi_qa_backend_api_through_cloudfront_with_upgraded_cipher_suite()`, place it into a deployment context using the software stacks and practices used by their regular software that contacts Solid APIs, and ensure the QA Solid API can be reached. A Program will likely want to change `print(...)` statements to appropriate logging and take other measures to be able to verify that their test properly reaches the QA Solid API.

Here is the Python snippet ...

```
#!/usr/bin/env python3
#
# FILE:  jco5569_reachability_for_upgraded_Cloudfront_cipher_suite.py
#
#   * tested with Python 3.8, should work for Python 3.7+
#   * requires the "requests" library (pip install requests)

import json
import sys
from http import HTTPStatus
from typing import Tuple, Optional

import requests

_SOLIDFI_QA_API_HOSTNAME = 'qa-api.sbx.solidfi.com'


def reach_solidfi_qa_backend_api_through_cloudfront_with_upgraded_cipher_suite() -> Tuple[bool,
Optional[str]]:
    #
    # keeping error-handling simple

    try:
        url = f'https://{_SOLIDFI_QA_API_HOSTNAME}/v1/webhook/events'

        print(f'contacting {url}, expecting {HTTPStatus.UNAUTHORIZED.name} status, code
"EC_AUTH_ERROR"')
        response = requests.get(url)

        try:
            status_code_string = HTTPStatus(response.status_code).name
        except Exception:
            status_code_string = 'UNKNOWN'

        print(f'received response status code {status_code_string}=={response.status_code}')
        if HTTPStatus.UNAUTHORIZED != response.status_code:
            raise ValueError(f'got status code {response.status_code} instead of expected '
                             f'{HTTPStatus.UNAUTHORIZED.name}=={HTTPStatus.UNAUTHORIZED.value}')

        json_content = response.json()
        print(f'received JSON response: {json.dumps(json_content, indent=2, separators=(",", ": "))}')
```

```
        if not isinstance(json_content, dict):
            raise ValueError('response JSON is not a dictionary')
        if 'code' not in json_content:
            raise ValueError('key "code" not in JSON response')
        code = json_content['code']
        if 'EC_AUTH_ERROR' != code:
            raise ValueError(f'got code "{code}" instead of expected "EC_AUTH_ERROR"')

        return True, None

    except Exception as exc:
        return False, f'{exc.__class__.__name__}: {exc}'


if '__main__' == __name__:
    #
    is_success, msg = reach_solidfi_qa_backend_api_through_cloudfront_with_upgraded_cipher_suite()
    if is_success:
        print('successfully reached SolidFi QA "backend" through Cloudfront w/ upgraded cipher-suite')
        sys.exit(0)
    else:
        print(f'failed to reach SolidFi QA "backend" through Cloudfront w/ upgraded cipher-suite,
error: {msg}')
        sys.exit(1)

# END OF FILE.
```

Here is output for a successful test (with exit code 0), as the Python program currently is set up ...

```
contacting https://qa-api.sbx.solidfi.com/v1/webhook/events, expecting
UNAUTHORIZED status, code "EC_AUTH_ERROR"
received response status code UNAUTHORIZED==401
received JSON response: {
  "requestId":
"webhook-external-deployment-58d775db9-pm777;tiVlgJDovA;qa;req-4a36be32-
f5b1-4cf2-807d-61906a45f177",
  "method": "GET",
  "status": 401,
  "code": "EC_AUTH_ERROR",
  "message": "unauthorised",
  "sysMessage": "unauthorised",
  "programId": "",
  "env": "qa",
  "createdAt": "2023-02-07T19:28:42Z"
}
successfully reached SolidFi QA "backend" through Cloudfront w/ upgraded
cipher-suite
```

Here is output from an unsuccessful test (this is just *one example* of a test failure ... in this failure scenario we disconnected from the Internet to set up an inability to reach the target QA Solid API host ... *a failure to communicate with the Solid API due to incompatibility with the upgraded Cloudfront cipher-suite will look different from this, and the failure symptoms would depend on the software stack and content*.)

```
contacting https://qa-api.sbx.solidfi.com/v1/webhook/events, expecting
UNAUTHORIZED status, code "EC_AUTH_ERROR"
failed to reach SolidFi QA "backend" through Cloudfront w/ upgraded
cipher-suite, error: ConnectionError:
HTTPSConnectionPool(host='qa-api.sbx.solidfi.com', port=443): Max retries
exceeded with url: /v1/webhook/events (Caused by
NewConnectionError('<urllib3.connection.HTTPSConnection object at 0x101341bb0>:
Failed to establish a new connection: [Errno 8] nodename nor servname provided,
or not known'))
```

A Note on Using QA URLs / Hostnames

The Solid DevOps Group provides QA website URLs and Solid API hostnames for the limited purpose so far described. Do not use the QA website URLs or hostnames for other purposes, and limit access to just what is required to verify your software stacks (for server-side or native-code Solid API use case access) or browser access (for website access) to Cloudfront with upgrade cipher-suite.

Note that the QA URLs and Solid APIs behind the API hostnames are almost always operational and likely will be available all the time. There's only a small outside chance that the QA website or API URLs / hosts won't be working properly. If the QA Solid API host accessed via the `qa-api.sbx.solidfi.com` hostname doesn't appear to be working when testing the Solid APIs, one can further tell whether the problem is with the tested software stack, or with the QA Solid API endpoint itself, by running the `curl` example above from a different context, e.g., a development laptop. In the unlikely event a QA website or URL is unavailable, it should not be unavailable for long.

Conclusion

This document has advised Programs of upcoming upgrades for Cloudfront "distributions" in Solid's production-level infrastructure to AWS Cloudfront *Security Policy TLSv1.2_2021*, the possible implications for Programs and their end-users, and ways for Programs to do advance testing for interacting software (browsers and Solid API clients) by using Solid's QA infrastructure.