HTTPWG
Internet-Draft
Intended status: Best Current Practice
Expires: September 12, 2019

B. Sniffen
M. Bishop
E. Nygren
R. Salz
Akamai Technologies

March 11, 2019

# Best practices for TLS Downgrade
## draft-richsalz-httpbis-https-downgrade-00

Abstract

Content providers delivering content via CDNs will sometimes deliver content over HTTPS (or both HTTPS and HTTP) but configure the CDN to pull from the origin over cleartext and unauthenticated HTTP. From the perspective of a client, it appears that their requests and associated responses are delivered over HTTPS, while in reality their requests are being sent across the network in-the-clear and responses are delivered unauthenticated. This exposes user request data to pervasive monitoring [RFC7258]; it also means response data may be tampered with by active adversaries. Terminating TLS connections on a load balancer and contacting a backend over cleartext has long been common within data centers, but doing this TLS termination and downgrade to HTTP at a CDN introduces additional risk when the unprotected traffic is sent over the general Internet, sometimes across national boundaries.

While it would be nice to say "never do this," customer demand, content provider use-cases, and market forces today make it impossible for CDNs to not support downgrade. However, following a set of best practices can provide visibility into when this is happening and can reduce some of the risks.

This Internet-Draft will expire on September 12, 2019.



Table of Contents



## 1. Background and Motivation

Browsers are helping drive a push to universal HTTPS through a variety of mechanisms, including:

- Show HTTP as "not secure"
- Showing mixed-content warnings when images or advertisements are HTTP on an HTTPS base page
- Making "powerful" new web features available only for HTTPS

On mobile, app stores sometimes require HTTPS for acceptance.

These factors have pushed many content providers to quickly enable HTTPS, even when their origin infrastructure is not ready or not

perceived as being ready. Being able to use a CDN to convert HTTPS to HTTP has been looked at as a fast path for getting onto HTTPS quickly. Doing this has value in protecting requests and responses over the last mile, but admittedly does not address or worsens some other types of attacks (such as pervasive monitoring, or corruption and manipulation of content crossing national boundaries).

Delivering content over HTTPS but fetching it insecurely over HTTP is done for a variety of reasons, some of which have historic motivations with better alternatives today, but where content providers are resistant to change. This includes:

- Lack of HTTPS support in origin infrastructure, such as due to using load balancing hardware that does not support HTTPS, has bad performance characteristics with HTTPS, or which only supports SSLv3.

- A perception that HTTPS is more expensive to deliver. In some cases content providers may have origin infrastructure using old hardware where this is a real challenge and they lack the budget to upgrade to servers or load balancers that can handle HTTPS well.

- A perception that using HTTPS introduces performance issues, such as due to the additional round trips required to establish connections. This can be a real issue for origins that lack persistent connection or session resumption support.

- Challenges in managing origin certificates, or a perception that it is hard to get TLS certificates. Automation with providers such as LetsEncrypt help here, but some content provider origins may be using software or hardware elements that don't yet integrate well with Auto-DV or may have organizational policies against using DV certificates.

- Delivering the same library of content to end-users over both HTTP and HTTPS, but wanting the CDN to cache any given object only once. This can be better addressed by always fetching content via HTTPS and storing in a cache accessible for both HTTP and HTTPS requests, but this faces challenges for transitioning from an entirely HTTP-fetched-and-served content library to one that is served over a mixture of HTTP and HTTPS.

- A perception that there is no risk to their users or brand reputation, sometimes due to thinking of pervasive monitoring and content manipulation as esoteric threats that don't apply in their case. For example, content providers delivering on-demand

streaming movies may not see a threat from using HTTP and may view DRM as addressing most of their immediate concerns.

There is also a closely-related issue where content delivered over HTTPS has been pushed to origin infrastructure over an insecure protocol. For example, content uploaded to a storage service over an insecure protocol such as FTP, or live streams pushed from encoders to ingest entry points over an insecure protocol. This has the added risk that authenticators may be unprotected on-the-wire.

## 2. Recommended alternatives

The "right thing" to do is to use modern secure protocols and cryptography for secrecy and authentication for the request and the response when interacting with content origin sources: HTTPS for pull-through caches, and protocols such as SCP or SFTP or FTP-over-TLS or HTTPS POSTs for pushed data.

Origin sites that avoided TLS for fear of a performance hit should collect data on the actual costs with modern implementations and modern crypto-support hardware. These are expected to be under 2% CPU overhead, especially when persistent connections are enabled. Auto-DV certificate management can make origin certificate management straight-forward and automateable.

## 3. Potential risk mitigations

An intermediate cache can take several actions to reduce the risk of unpleasant consequences from using TLS downgrade - though these practices do not eliminate that risk. They take two general strategies:

1. Informing the endpoints that this downgrade is in place. End points have more information about the details of the connection, and can expose details to human controllers. For example, returning a response header such as "Protocol-To-Origin: cleartext" and preventing customers from removing it. Clients may then choose some manner in which to expose this to end-users. (Some other proprietary implementations of this response header have included "X-Forward-Proto: http" and "CDN-Origin-Protocol: http".)

2. Restricting the sort of data in transit when downgrading from HTTPS to cleartext HTTP. Examples of this include:

   * Limiting to GET methods. This prevents unauthenticated writes to the origin.

* Refusing to downgrade requests for "/" , "/index/", or "/index.html". This prevents accidental delivery of the entire site. The goal is to rapidly detect a misconfiguration with too much downgrading by breaking the site.

* Limiting the content types or file extensions (e.g., to streaming media or other static media assets).

* Stripping outgoing request headers containing potential identifiers (Cookie, etc)

* Stripping query strings

In practice, stripping query strings breaks an enormous amount of Web traffic: searches, beacons, and the selection apparatus of streaming media clients. Mechanisms that rely on lists of what is allowed (file extensions) or what is banned (such as "Cookie" headers) rely on an implausibly detailed and up-to-date models of Web use.

Other headers that may wish to be stripped from outgoing requests include "X-Forwarded-For", "Origin", "Referer", "Cookie", "Cookie2", and those starting with "Sec-" or "Proxy-".

## 4. Recommendations

It is recommended that CDNs do at least the following as default behaviors as part of TLS downgrade:

o Providing and encouraging better alternatives (such as always fetching securely over HTTPS but making static objects available in a shared cache that can also be accessed via HTTP requests).

o Returning a "Protocol-To-Origin: cleartext" response header (which may be a comma-separated list of protocols when multiple hops are involved).

o Limiting downgrade requests to GET.

o Refusing requests for "/" , "/index/",or "/index.html".

o Strip at least some headers that may include personal identifiers or sensitive information.

## 5. Alternative approaches

Some other approaches may also help address the risks:

- Use a VPN or IPSEC or other secure channel between the CDN and the origin.

- Validate asymmetric signatures of content at the CDN before serving, such as for software downloads. This helps with integrity, but still exposes confidentiality risks.

## 6. Security Considerations

### 6.1. Risks of doing downgrade

Downgrades allow protection of last-mile connections to end-users, but they make it easier for adversaries who control the network between CDN caches and origin (such as at national boundaries) to poison caches or perform surveillance (as correlation attacks are possible, even if ostensible PII information is stripped at the CDN.)

> TODO: It is easy to say that everyone should use TLS everywhere (cite). Let's support that position by explaining which adversaries can win which games here - including some where the adversary has an advantage in an HTTP-to-HTTPS connection _over a connection with no TLS at all_.

### 6.2. Control of the network between the cache and the origin

ISPs on the HTTP path, including nation states at their borders, can surveil traffic. They can expect to get end-user IP information from "X-Forwarded-For" or similar. In some circumstances, they can learn more from correlated timing and sizes. This is principally a risk to _secrecy_.

Active adversaries can also corrupt or modify content.

For executable content (such as software downloads or javascript) this can be used to compromise clients or web pages, especially if no end-to-end secure integrity validation is performed. Even when software downloads have signature validation performed, this can provide a potential exposure for downgrade attacks, depending on client-side implementations.

For site and media content, modification can be used to make content appear as authoritative to a user (delivered via HTTPS from a "trusted site") while actually containing selective modifications of the attackers choice, such as for the financial or political benefit of the attacker.

### 6.3. Confused-deputy issues at the browser or origin

HTTP clients make different decisions based on whether they are using HTTPS or HTTP - for example, they send Secure cookies (cite), they enable certain Web features (high-resolution location, Service Workers). This is principally a risk to _authentication_.

This attack is only available with downgrade. A related attack is available in the case of HTTP _upgrade_, in which a server makes a similar decision based on seeing HTTPS on its end of the connection. In cases where HTTP requests are upgraded to HTTPS, CDN or proxy operators need to work with origin operators to control this complexity and prevent the complementary attack, such as by only performing upgrades for cache-able, static, and idempotent content. (TODO: Future versions of this draft should expand on this topic in more detail.)

## 7. Normative References

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

## Appendix A. Acknowledgements

Thank you to Suneeth Jayan and others at Akamai who have helped develop best practices. Future versions of this draft hope to also incorporate best practices developed elsewhere.

## Authors' Addresses

Brian Sniffen
Akamai Technologies
145 Broadway
Cambridge 02139
US

Email: bsniffen@akamai.com

Mike Bishop
Akamai Technologies
145 Broadway
Cambridge 02139
US

Email: mbishop@akamai.com

Erik Nygren
Akamai Technologies
145 Broadway
Cambridge 02139
US

Email: nygren@akamai.com

Rich Salz
Akamai Technologies
145 Broadway
Cambridge 02139
US

Email: rsalz@akamai.com