

HTTPS FOR LOCAL DOMAINS

Martin Thomson, Sep 2017

Proposal to make origins with non-unique names like *Printer.local* or *192.168.0.1* use an extended origin to allow them to use HTTPS.

INTRODUCTION

As more of the web transitions to using HTTPS, users that access services on their local network are being increasingly marginalized. Servers that run in local networks often cannot easily get a valid server certificate. The majority of devices use unsecured HTTP.

Browsers are progressively reducing the capabilities and features that are available to origins that use unsecured HTTP. In particular, [new features](#) are being developed [exclusively for HTTPS origins](#).

Non-HTTPS origins are vulnerable to a range of attacks, so all of this is easy to understand, however it means that local devices are left with UI that shows [negative security indicators](#), and diminished access to web features.

Servers that operate publicly accessible endpoints have few challenges with getting certificates that allow them to authenticate. The real challenge is in giving servers that are less publicly available access to the same opportunities without compromising the assurances provided to other servers.

We could run HTTP over TLS [without authenticating the server](#), which would provide some of the confidentiality and privacy benefits of HTTPS. However, aside from the exposure to man-in-the-middle attack, a site using RFC 8164 doesn't gain an HTTPS origin.

What is really needed is a way to use HTTPS proper.

USE REAL NAMES

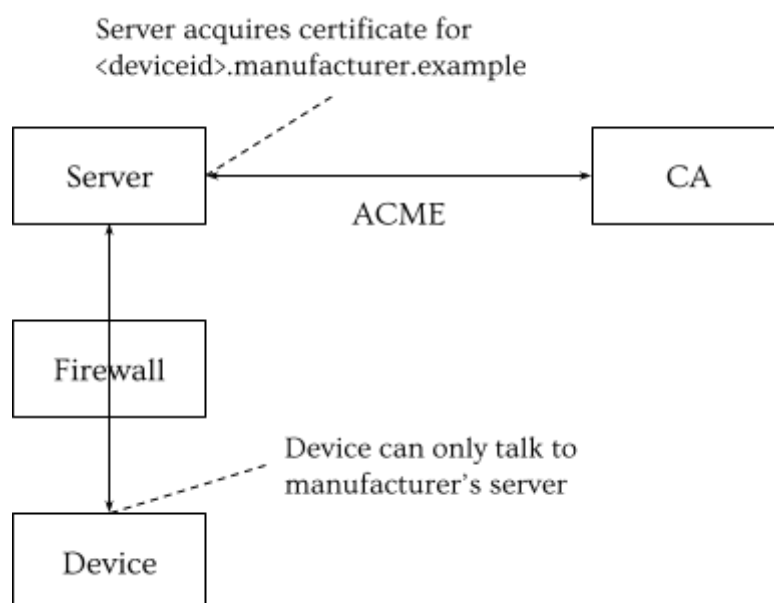
The best option for a device that is installed in a local network might be to use a genuine domain name. The vendor of a device could run a service that assigns a unique name from the global DNS namespace to the device, such as `<deviceID>.devices.example.com`. [Mozilla's IOT Gateway](#) does exactly this, and this is a pattern that is emerging as best practice.

The way that this works is that a service, often operated by the device vendor, provides devices with a unique identity. Typically, this is a subdomain of a domain owned by that vendor. The service might also further enable the acquisition of certificates for devices.

A split horizon DNS ensures that requests for that name produce the correct domain name. The device vendor might also provision the DNS for that name, which could improve reachability in some circumstances, but it could also present a privacy exposure by making network address assignments publicly accessible. This requires that devices have access to a means of controlling DNS. Thus, this is more feasible for a router that also provides DHCP, but less for other devices. Even for a device like a router, a manual configuration of a DNS server makes the name inaccessible.

A redirect from a friendly name (such as gateway.local) to the full device name ensures that the device can be reached by typing a more readily memorable string. However, there is no way to secure this redirect: the name that is entered is not unique and therefore a valid certificate cannot be issued for that name.

Unlike [suggestions to the contrary](#) this does not require that the device itself be exposed to packets from the internet at large, only that it be able to reach one particular server on the internet. Firewall rules could be set to permit only what is necessary to communicate with a certificate provisioning server. That server can proxy requests for certificates; for instance, responding to ACME challenges as needed and relaying the certificate that is received. This only requires that the server authenticate devices. Manufacturers might use keys that are provisioned at the time of manufacture. If the manufacturer is prepared to provide certificates to any device, the device might provide credentials when it first claims an unused name.



For the device to be accessible from the internet, the device manufacturer needs to establish and relay communications. This effectively leads to the device being exposed to the internet, though the manufacturer might be able to provide some measure of protection from attacks like denial of service.

Access to a device using a local name might be used opportunistically. For this, [HTTP Alternative Services](#) can be used to provide an alternative route to the device that does not depend on the external server. The design of alternative services ensures that this is used automatically when available.

This is a good solution. But it isn't without drawbacks. The names that are produced are not generally usable by humans as a consequence of a need to be globally unique. Some deployments address this by providing a second service with a memorable name that manages rendezvous. That in turn has consequences for privacy, because that service needs to mediate interactions between users and their devices.

This approach also imposes an ongoing operational cost on the device vendor. With the plethora of things being released with tiny profit margins, additional operational costs are hard to justify. Support from vendors over the lifetime of the device is not guaranteed. A vendor that goes out of business is likely to cease operation of a certificate enrollment service. This might be managed if the device could be updated with new software that could use a different service. That only reduces the problem to a previously unsolved one: that of ensuring continuity of software updates for devices.

SERVER AUTHENTICATION ON THE WEB

It probably makes sense to take another look at how server authentication is used on the web.

Why does HTTPS need certificates? Unique, memorable names.

The DNS provides centralized management of names and enforces uniqueness of those names. The Web PKI provides security for those names through the issuance of certificates.

A name is important for ensuring that a web browser can faithfully translate user intent into action. When someone types *example.com* into their browser, that name is compared against the name in a certificate to determine if the server is correct.

The [web origin model](#) cares little that the identity of a server is a name. A browser cares more about the uniqueness of identity than its form. The web origin model is built to secure access to the state that a browser accumulates for an origin, it cares more that the identity accessing that state remain constant than how it is identified. A

certificate provides a unique anchor for that state, ensuring that only a single entity controls access to that information.

EXTENDED ORIGINS FOR LOCAL DOMAINS

The key challenge for local domains is that names are not unique. Any printer is entitled to claim the name *printer.local* and use it. Thus, any server has a legitimate claim to that name. If we are to make *https://printer.local* a viable origin, how do we ensure that it is unique?

Extending the origin makes this possible. A web origin is defined as a tuple of scheme, host and port. This is a capability that browsers are building to enable a range of use cases, such as the creation of separate browsing contexts (see for instance [containers](#)). By adding an additional property to that tuple that cannot be used by any other server, we ensure that the tuple is able to uniquely identify that server.

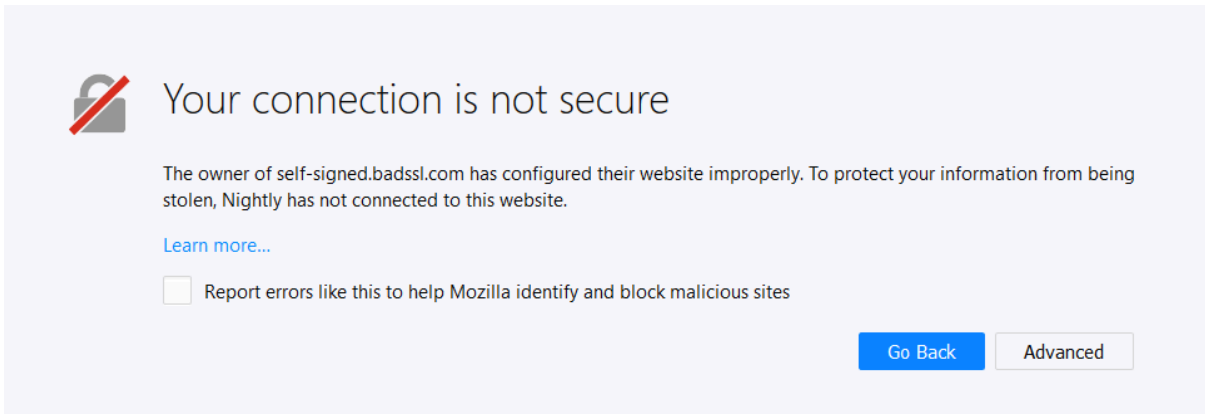
For this, the public key of the server is ideal. No other server can successfully use that identity without undermining the security of the web and any HTTPS server is always able to provide that information.


Adding a public key to the origin tuple creates a separation between origins. Two servers can claim to be *printer.local*, but the browser will ensure that they are distinct entities that are fully isolated from each other. Passwords and form data saved for one printer will not be used by the other printer. The same separation applies to permissions, storage, cookies¹ and other state associated with the origin.

USER INTERFACE

This is what you currently see in Firefox if you happen upon a server that offers a certificate that isn't signed by a certification authority.

¹ This is an important point, and one that adding to the origin tuple does not necessarily guarantee. The rules for isolation for cookies is complicated.



 **Your connection is not secure**

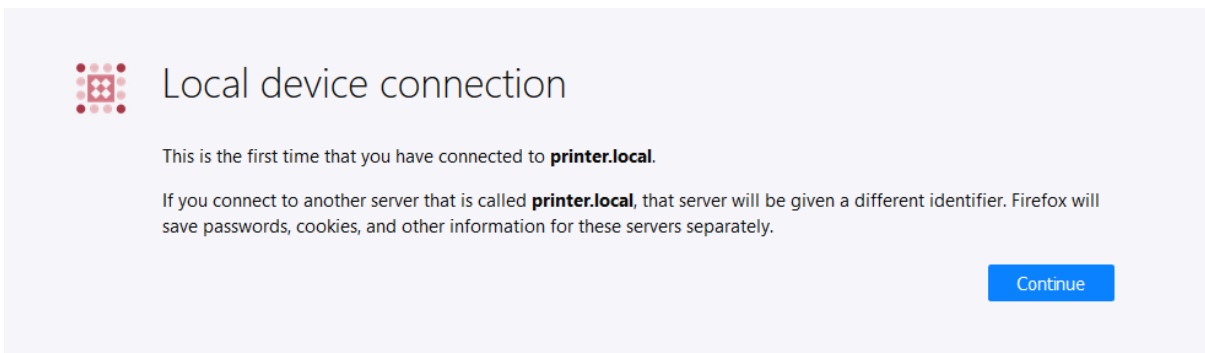
The owner of self-signed.badssl.com has configured their website improperly. To protect your information from being stolen, Nightly has not connected to this website.


[Learn more...](#)

Report errors like this to help Mozilla identify and block malicious sites

[Go Back](#) [Advanced](#)

When a local server is contacted the first time, the following is shown instead²:



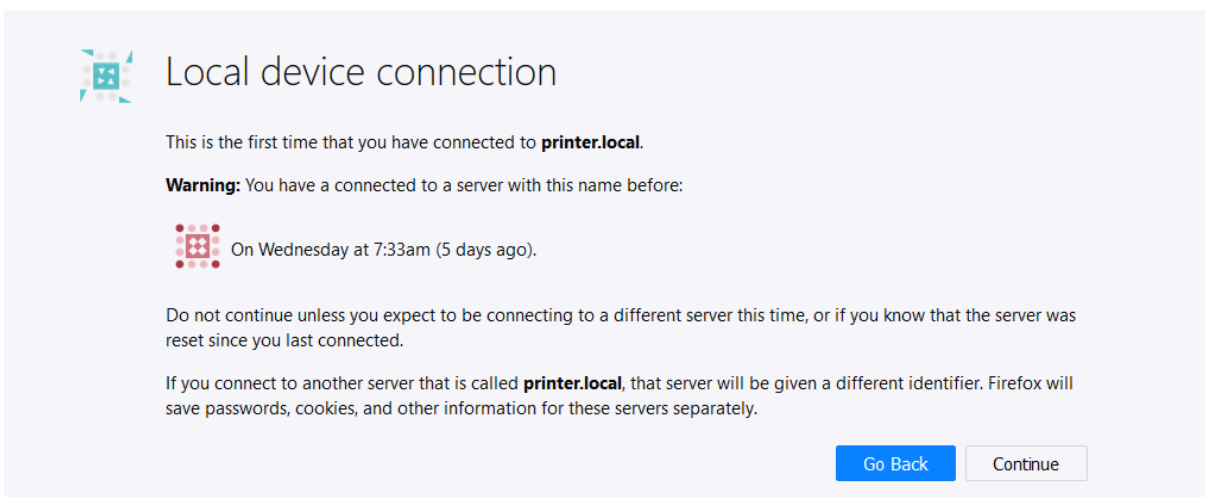
 **Local device connection**


This is the first time that you have connected to **printer.local**.

If you connect to another server that is called **printer.local**, that server will be given a different identifier. Firefox will save passwords, cookies, and other information for these servers separately.

[Continue](#)


To avoid the potential for confusion about identity, a different warning is shown if the same name was previously seen. This shows some information about previous connections to the same name:



 **Local device connection**

This is the first time that you have connected to **printer.local**.

Warning: You have a connected to a server with this name before:

 On Wednesday at 7:33am (5 days ago).

Do not continue unless you expect to be connecting to a different server this time, or if you know that the server was reset since you last connected.

If you connect to another server that is called **printer.local**, that server will be given a different identifier. Firefox will save passwords, cookies, and other information for these servers separately.

[Go Back](#) [Continue](#)

This might also save the title from the page that was last shown, to help with context. When a server is forgotten, this information is also forgotten.

² This is a rough mockup to aid understanding, not a final UX. I am not a UX designer. Also, this interstitial might be replaced with an infobar instead.

IDENTIFYING SERVERS AS LOCAL

The idea here is to do this only for names that are not inherently unique. Domain names like *example.com* are unique and therefore would not get this treatment.

Servers with *.local* ([RFC 6762](#)), *.home.arpa* ([RFC 8375](#)), or *.internal* ([ICANN internal](#)) suffixes will be considered local. These names are specifically designed for local use and are non-unique by design.

Servers with IPv4 literals from the RFC 1918 address spaces (10/8, 172.16/12, and 192.168/16) will be treated as local. Similarly, hosts with link-local literals (169.254/16 or fe80::/64) or Unique Local IPv6 Unicast Addresses (fc00::/7) are considered local.

Servers on loopback interfaces are local. This includes the IPv4 literal (127.0.0.1), the host-scope IPv6 literal (::1), and any origin with the name “localhost” ([draft-ietf-dnsop-let-localhost-be-localhost](#)).

Address literals might reach a server that can also be reached using a domain name. This is not fundamentally different to a server that can be reached by two different names (for example, servers often respond to names both with and without a “www” label: *https://example.com* and *https://www.example.com*). A server that is identified with a URL that includes a domain name has a different identity to the server that is identified with a URL that includes an IP address literal, even if the domain name resolves to that IP address. Servers with multiple identities will be able to use this capability to either provide a secure redirect to a preferred name or to present a different service to clients.

The names that are identified as local are all non-unique and therefore not valid targets for certificates. This means that HTTPS connections to these servers could not otherwise be made using a genuine Web PKI certificate.

Other means of identifying servers as local might be added in future.

ORIGIN SERIALIZATION

The primary drawback of adding more attributes to the origin tuple is the effect it has on applications that use origin in their processing.

For instance, the [postMessage API](#) uses the origin to describe the source or destination of messages. The sender of a message identifies the origin that it expects to receive the message. The recipient of a message is expected to check that the origin matches their expectations.

This proposes a change to the serialization of origins for local servers so that it includes a hash of the server's public key information (SPKI). This is added to the ASCII and Unicode serializations of the origin.

For example, this might use the underscore convention to add the SPKI hash to the domain name, *https://_NPNE4IG2GJ4VAL4DCHL64YSM5BII4A2X.printer.local*, or it could use a separator of some sort to partition off space for a key.

TBD: Does this need a new scheme? Is it a new field, or can it be added to the domain name? What separator would this use? How should the SPKI hash be encoded (base64url, base32, hex)? How many bits are enough? Do we need to signal hash function? Is this a new field at the end, a change to the name, or something else? Should SPKI go in the middle to discourage prefix-matching?

Adding a non-backwards compatible serialization for origins makes these APIs harder to use. For instance, sending a message to a *printer.local* requires learning this value. This is partly intentional. The space of possible names for local servers is limited, and the choice of names for devices like printers even more narrowly limited. Including a server public key in the origin makes it difficult to correctly guess the name that will cause a message to be received by the device. A device can make its name known by sending its own message to other servers (e.g., mDNS).

The other potential problem is that .local names are permitted to use a wider range of characters than domain names. What sort of normalization do we do to avoid confusable characters?

ADVANTAGES AND DRAWBACKS

The key advantage of this approach is that the extension of origin allows local services, including those running on the local machine, to use and benefit from HTTPS.

There are several drawbacks, each of which needs careful consideration.

USER INVOLVEMENT

Part of the security of this system involves user awareness. If *printer.local* at home has a password, then moving to a different network exposes the user to a phishing-type attack where a different *printer.local* attempts to retrieve the password for the home printer.

This is why there is a warning shown on first connection to servers and an enhanced warning is shown after a name-collision. It is possible that other security UX might

be enhanced to better signal the status of local servers. For instance, showing the identicon in the above examples next to the server's chosen favicon.

For password stealing, using a reliable password manager should help. It might be necessary to include notices warning users about the server identity³.

Third-party password managers would need to be enhanced to recognize the additional information in the origin and properly segment the namespace. A password manager that looks at *window.location.host* is likely to broadcast passwords inappropriately.

To mitigate this risk we might consider blocking concurrent use of the same name with different keys if a password manager is installed. However, I don't think that we necessarily know that an extension is a password manager. This probably reduces to some due diligence with the help of the addons team.

KEY CHANGES

Devices that rotate keys will gain a new identity, and lose access to any existing state. This creates an incentive to avoid changing keys, which runs counter to most operational practices. In this case, the advantages of making HTTPS available would seem to far outweigh the risk of using a key over long periods.

It is also possible that a reset of the device might cause keys to be reset, leading to the more alarming user notice.

ADDRESS CHANGES

Devices that are identified by their IP address will receive a new identity when their assigned address changes. This is a consequence of extending the origin tuple to include a public key. This is intentional: a device might intentionally operate multiple identities and wish to preserve separation of origins.

ERGONOMICS OF ORIGINS

As outlined in the construction of [origins for local servers](#), the ergonomics of an origin that includes a SPKI hash is not always ideal.

³ If we had better password interfaces, maybe ones that used a PAKE, this would be unnecessary, but passwords traversing the network is still state of the art for 2018.

ACME GATEWAY FOR DEVELOPMENT

An alternative design that might be better suited to the development and running of servers has the browser generate certificates for local servers that it will accept as valid.

To do that, a browser could offer an ACME server over the loopback interface (127.0.0.1 or ::1).

Accessing the ACME server is only possible from applications on the same host, so the usual process of fulfilling challenges could be circumvented. The ACME server would issue certificates for any name. Certificates would chain to a unique root that would be generated - and trusted - by the browser.

Deploying this capability as an add-on ensures that it is only available to someone who needs this capability.

This creates incentives to implement and use ACME in servers rather than create a reliance on browsers being tolerant of untrustworthy certificates. However, it has some significant downsides.

- Only servers that run on the local machine can access this service.
- It would not be possible to access the servers from other browsers without encountering certificate errors.
- This can interact poorly with servers that choose names that exist on the web. As long as the installed root is treated like a custom root, our code that disables this type of check should ensure that these additional checks don't produce errors.

The primary advantage of this approach is that it is possible to experiment with this approach without relying on significant changes to the browser.