# Web Sites Should Not Need to Rely On Users to Secure Communications*†‡

Andy Ozment
MIT Lincoln Laboratory

Stuart E. Schechter
MIT Lincoln Laboratory

Rachna Dhamija
Harvard University

March 8, 2006

*This work is sponsored by the I3P under Air Force Contract FA8721-05-0002. Opinions, interpretations, conclusions and recommendations are those of the author(s) and are not necessarily endorsed by the United States Government.

†This work was produced under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College, and supported under Award number 2003-TK-TX-0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

‡This paper was presented at the: *W3C Workshop on Transparency and Usability of Web Authentication*. New York, NY, USA: March 15-16, 2006.

# 1 Position Overview

The `https` scheme has been the switch used to turn on secure channels since support for HTTPS was first added to web browsers. The insecure HTTP protocol remains the default, and it is applied when users enter domain names into their browser's address bar. Thus the burden of specifying the level of security for a web site has fallen on the user of the browser, rather than the operator of the site that the user wishes to access.

To help users reach their secure site, many site operators use the insecure HTTP protocol to redirect their users to an HTTPS-enabled URL. However, this approach leaves users vulnerable to man-in-the-middle attacks.

Users cannot *prevent* man-in-the-middle attacks when they rely upon insecure HTTP to redirect them to a secure site. Instead, they are expected to *verify* that they have securely reached the correct web site. This verification requires two steps:

1. **A secure channel has been established**
   The user must verify either the presence of the `https` scheme in their browser's address bar or the presence of the closed lock icon in the browser's window frame. The user must not confuse these indicators with similar looking content within the displayed web page, like the lock icons frequently displayed next to login forms on popular financial sites.

   A user who fails to verify that the channel has been secured will not detect a man-in-the-middle attack that prevents redirection to a secure URL.

2. **The channel leads to the correct domain**
   The user must also verify that the domain name of the server that has been authenticated by the browser is indeed that which they have requested. If redirection changes the domain name, the user must be able to recognize whether the new name belongs to the same organization. For example, the user must know that `ebay.reallysecuresite.com` is not necessarily controlled by the same entity as `ebay.com`.

   A user who fails to recognize such changes to the domain name will not detect a man-in-the-middle attack that redirects him to an HTTPS site in a portion of the name space controlled by the attacker.

Few users verify that they are at the correct domain and using a secure channel—indeed, users may not even know how to verify these requirements [**?**].

Why are users burdened with the responsibility of requesting or verifying the use of security? If web site operators must choose whether or not their site will support HTTPS, why can't they also choose whether or not browsers use HTTPS to contact the site? The problem lies in the absence of a mechanism through which browsers can learn whether a site requires HTTPS. Paradoxically, communicating whether or not a secure channel should be used is a process that itself requires a secure channel from the site to the browser. Universal deployment of such a mechanism might, at first, appear more difficult to achieve than universal deployment of HTTPS.

However, the pieces required to construct a secure channel for the transmission of site security requirements are almost in place. The domain name system (DNS) already provides a mechanism through which a web site's operators can publish information, such as the IP addresses of the site's servers, to client software. The deployment of the DNS Security (DNSSEC) standard [**?**, **?**, **?**] will enable client browsers to verify the integrity of any information published by a domain name holder. We propose the creation of the final piece to complete the puzzle: a DNS resource record in which to store the security requirements for each of the site's services. We call this the Service Security Requirement (SSR) record [**?**].

The SSR resource record will specify the protocols with which a site is willing to communicate. For example, the operator of a web site could specify that the site will only accept HTTPS connections, and will never communicate over plaintext HTTP. Furthermore, the web site could specify that it requires those HTTPS connections to use SSL version 3 or higher.

The SSR resource record will be stored in the site's DNS zone, alongside the resource records that provide the IP address of the web server. The zone must be signed using the DNSSEC standard to ensure the integrity of these records.

To retrieve the resource record matching a DNS name, a client traverses the domain name hierarchy from the top (the root zone) down to the named zone. For example, to access `w3c.org` the client would first access the root zone, then the `.org` zone, and finally the `w3c.org` zone. Starting with only a key for the DNS root zone, a client can use DNSSEC to establish the authenticity of keys at each step down the hierarchy. This verification is performed by induction, starting from the root. The authenticity of each child zone's key can be established by verifying the signature of its parent. Signatures also indicate whether or not each child zone is signed, thwarting attempts to substitute an unsigned zone for a signed zone. DNSSEC also provides a means to establish which records are not stored in a zone. By verifying proofs of nonexistence, clients can thwart attempts to make security-critical records appear to be absent. This ensures that attempts to block SSR records from being read by clients will be detected.

Our SSR proposal does not face the chicken and egg bootstrapping problems that often face new technologies. The prerequisites to deployment are support for SSR in the user's client software and the deployment of DNSSEC in the domain name hierarchy above the site's zone. For sites under `.com`, this means only the root zone and the operator of the `.com` zone (VeriSign) must have deployed DNSSEC.

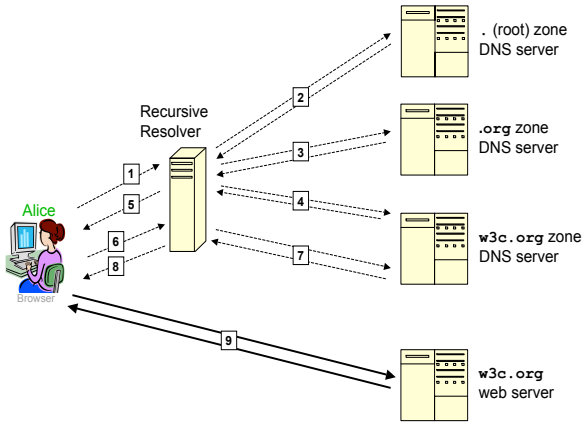


Figure 1: The steps through which Alice's browser establishes a trusted connection to `w3c.org`.

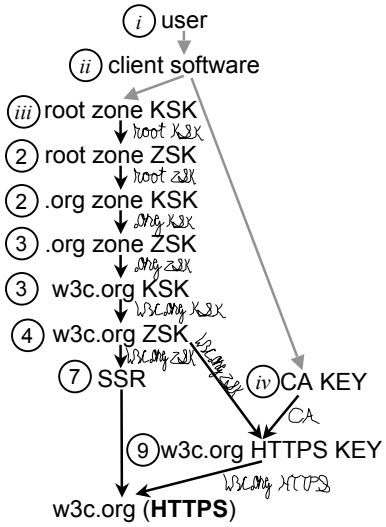


Figure 2: The chain of trust from Alice to `w3c.org`.

## 2 How SSR Works: An Example

Suppose Alice wants to access an account hosted at `w3c.org` by typing its domain name into her browser's address bar. Figure 1 illustrates the steps through which Alice's browser uses DNSSEC and the SSR resource record to establish a trusted connection to `w3c.org`.

The chain of trust that is used to establish the authenticity of the `w3c.org` site is illustrated in Figure 2. The roman numerals and gray arrows indicate trusted components and relationships. The arabic numerals indicate information that can be verified via the trusted components. For example, the arrow from the user Alice (bubble *i* in the figure) to her computer's client software (*ii*) shows that she trusts this client software. If a node that is trusted is not trustworthy, the connection will not be secure.

We also see that the client software must trust the owners of keys with which it has been pre-configured. These include the public key signing key of the DNS root zone (*iii*). They may also include the keys of certificate authorities (*iv*), who certify the public keys used by web servers.

For this example we will assume that Alice's computer has a DNS configuration common to most client hosts: a stub resolver on Alice's computer issues its requests through a recursive resolver at her ISP or on her corporate network. The recursive resolver retrieves the requested records and returns the results to Alice's stub resolver.

We will now proceed through each numbered step in Figure 1, describing the means by which the `w3c.org` domain name is resolved and the chain of trust to the server is established (shown in Figure 2). The arabic numbers in Figures 1 and 2 correspond to the numbered steps below.

1. Alice types '`w3c.org`' in her browser's address bar. The stub resolver in Alice's browser uses issues a DNS request to the recursive resolver located at her ISP.

2. The recursive resolver passes the request to the server for the DNS root zone, which returns a set of resource records. One record holds the root zone's public Zone Signing Key (ZSK). This key is signed with the root zone's Key Signing Key (KSK), thereby delegating to the keyholder the ability to sign records within the root zone. Recall that the browser is configured to trust the key signing key (*iii*) at the source of this delegation.

   The root server also returns resource records delegating the `.org` zone as the next step in the chain. Some of these records contain the addresses of the `.org` zone's servers. Others contain the key signing keys that are delegated the authority to sign within the `.org` zone. These delegation records are themselves signed by the root server's zone signing key. Thus the keys of the root zone can be used to authenticate the addresses and keys of its child zone, `.org`.

3. The recursive resolver now obtains resource records from a DNS server for `.org`. Once again the key signing key for the zone is used to establish the authenticity of a zone signing key. The resource records obtained from `.org` provide the address of the `w3c.org` zone's servers and the key signing key. Again, all resource records are signed.

4. The recursive resolver now queries the DNS server for `w3c.org`. It receives a list of resource record types supported in the zone. It also receives the address of the web server. Once again, all records are signed.

5. The recursive resolver returns to Alice's stub resolver the resource records needed to establish the chain of trust to `w3c.org`, as well as the requested records

within that zone. Alice's stub resolver uses its pre-installed root zone key to verify the chain.

6. Detecting the presence of an SSR record, Alice's browser requests that the recursive resolver retrieve this record.

7. The recursive resolver, having cached the resource records needed to identify the `w3c.org` zone, routes the DNS request directly to that zone's servers. It then verifies that the result is signed with the correct zone signing key.

8. The recursive resolver returns the signed SSR record back to Alice's browser, which also authenticates the record.

9. In our example the SSR record indicates that `w3c.org` only accepts HTTPS connections. It could also indicate other security requirements: e.g. that HTTPS connections must use SSL version 3 or higher. The SSR record may also specify which mechanism is required to establish the validity of the HTTPS authentication key: it may require that the HTTPS authentication key be signed by one of the Certificate Authority keys stored in the browser. Alternatively, it may require that the HTTPS authentication key be stored in the DNS zone [**?**] and be signed with the zone signing key for `w3c.org`.

   Once the authenticity of the HTTPS key is established, the secure HTTPS connection can be established.

# 3 Architecture & Design Decisions

We have developed a draft Service Security Requirement (SSR) specification [**?**] in collaboration with the IETF's DNS Extensions Working Group.

While working on that draft, we have encountered a number of difficult design decisions: which options should be encompassed within each SSR record, which options need not be part of the record, and how can new options be added when new security technologies become available? The SSR record should be applicable to services beyond the web, so we must also decide how an SSR record can indicate the services to which it should be applied.

Sections 3.1 and 3.2 discuss some of these design decisions and elaborate on our current approach.

## 3.1 Security requirement options

The realm of possible options for securing connections goes well beyond the choices of 'secure channel' or 'insecure channel'. Web-based services may support a combination of the SSLv2, SSLv3, or TLS protocols. Other services may be secured through a different set of protocols, such as IPsec or SSH.

*Should SSR records explicitly list those protocols it supports or explicitly forbid those protocols that do not provide the required level of security?*

Suppose a web site operator wants to specify that the site should only be accessed via the SSLv3 or TLS protocols. Should the SSR record for that web site state that SSLv3 and TLS are required, or should it state that regular HTTP and SSLv2 are forbidden? We have chosen the former approach. Requiring a service provider to explicitly forbid services is impractical: new services may be introduced over time, and a service provider may not be aware of each service that it does not support. Furthermore, in order to support the majority of Internet services, clients must continue to default to insecure protocols when no SSR record is present. As a result, it is acceptable to assume that the presence of an SSR record indicates that the operator will list all supported protocols. If operators are not even aware of which protocols they support, they certainly cannot establish that their services are secure.

*At what level of detail should SSR requirements be specified?*

The standard should support as fine a granularity as is required by each underlying security protocol (TLS, IPsec, etc.).

The SSR specification will support protocol-specific options for each allowable protocol. We will use a hierarchical name space to prevent collisions. For example, an option for TLS should be preceded by the protocol name and a delimiter: e.g., "`tls.selfsigning='never'`" or "`tls.selfsigning.never`".

*How can future protocols and options be supported?*

An IANA registry will be created for security protocol names and the names of the services that use these protocols. When proposing new standards, authors must specify how the protocols or options they introduce will be integrated into the SSR record.

*Will the SSR record support redirection?*

Secure redirection is an important component of any proposal to link domain names to secure services. It is necessary because many secure web services are provided on servers that fall under a subdomain of the organizations zone (e.g. `secure.w3c.org`). The SSR specification could add this feature by incorporating the IETF's S-NAPTR standard [**?**], which provides for secure redirection from one zone to another. However, incorporating S-NAPTR would increase the

number of standards that must be supported by the developers of client software. Operators of secure services would also need to learn how to use the S-NAPTR standard to protect their zones.

On the other hand, adding support for redirection directly into the SSR standard will make it more complex. As we work with browser developers to complete the SSR specification, we will determine whether secure redirection should be a feature of SSR or whether it should be implemented via S-NAPTR.

### 3.2 Scope for applying requirements

Once we have a mechanism for specifying the security requirements of a service, we need to step back and ask an important question.

*What constitutes a service?*

This seemingly simple question adds a surprising amount of complexity to the design decisions for SSR.

Do HTTP and HTTPS provide the same service, or does the use of a secure channel and a different IP port make these services different? If they are different services, multiple SSR records will need to be stored. Services that do not want to be reached via HTTP will need a separate record to indicate that the HTTP service should not be used. A means of redirecting the browser to a secure service may also become a necessity. Treating HTTP and HTTPS as different services also sets a precedent that site operators must include SSR records for services that they do not support. If an operator of a service offered over a secure protocol is unaware of the existence of a corresponding insecure protocol, he will not include the SSR record.

If we assume that HTTP and HTTPS are the same service, we might then need to ask if SOAP over HTTP is a different service than a request for an HTML page? If so, where do we draw the line between services?

Our current specification considers HTTP and HTTPS to provide the same service. New services can be created as necessary, and can provide finer granularity if a site wishes to distinguish between different services built on top of protocols such as HTTP.

## 4 Conclusion

Securely connecting to a web site is more difficult than it should be. DNSSEC was not available to the authors of the first browsers to support HTTPS. Without DNSSEC, browsers could not retrieve the security requirements for a site. Instead, users are stuck with the responsibility of specifying whether a requested site should be accessed via a secure channel.

As a result, today's users must either type a URL prefixed by the HTTPS scheme before connecting to a site, or they must verify the security of the connection after the fact. This verification requires that the user understand the structure of domain name addresses and the meaning of browser security indicators. These requirements make it unnecessarily arduous for users to detect attacks on sites they believe to be secure.

Now that the underlying technology is ready, it's time to use DNSSEC to free users from this burden. We hope to use our presentation at this workshop to discuss the decisions we face in designing the SSR record and to solicit assistance in developing this new standard.