Chapter 6. Authentication

Table of Contents

What is Authentication?

- Types of Authentication
- Browser Limitations
- HTTP Basic
- HTTP Digest
- Forms Based Authentication
- Digital Certificates (SSL and TLS)
- Entity Authentication
- Infrastructure Authentication
- Password Based Authentication Systems

What is Authentication?

Authentication is the process of determining if a user or entity is who he/she claims to be.

In a web application it is easy to confuse authentication and session management (dealt with in a later section). Users are typically authenticated by a username and password or similar mechanism. When authenticated, a session token is usually placed into the user's browser (stored in a cookie). This allows the browser to send a token each time a request is being made, thus performing entity authentication on the browser. The act of user authentication usually takes place only once per session, but entity authentication takes place with every request.

Types of Authentication

As mentioned there are principally two types of authentication and it is worth understanding the two types and determining which you really need to be doing.

User Authentication is the process of determining that a user is who he/she claims to be.

Entity authentication is the process of determining if an entity is who it claims to be.

Imagine a scenario where an Internet bank authenticates a user initially (user authentication) and then manages sessions with session cookies (entity authentication). If the user now wishes to transfer a large sum of money to another account 2 hours after logging on, it may be reasonable to expect the system to re-authenticate the user!

Browser Limitations

When reading the following sections on the possible means of providing authentication mechanisms, it should be firmly in the mind of the reader that ALL data sent to clients over public links should be considered "tainted" and all input should be rigorously checked. SSL will not solve problems of authentication nor will it protect data once it has reached the client. Consider all input hostile until proven otherwise and code accordingly.

HTTP Basic

There are several ways to do user authentication over HTTP. The simplest is referred to as HTTP Basic authentication. When a request is made to a URI, the web server returns a HTTP 401 unauthorized status code to the client:

HTTP/1.1 401 Authorization Required

This tells the client to supply a username and password. Included in the 401 status code is the authentication header. The client requests the username and password from the user, typically in a dialog box. The client browser concatenates the username and password using a ":" separator and base 64 encodes the string. A second request is then made for the same resource including the encoded username password string in the authorization headers.

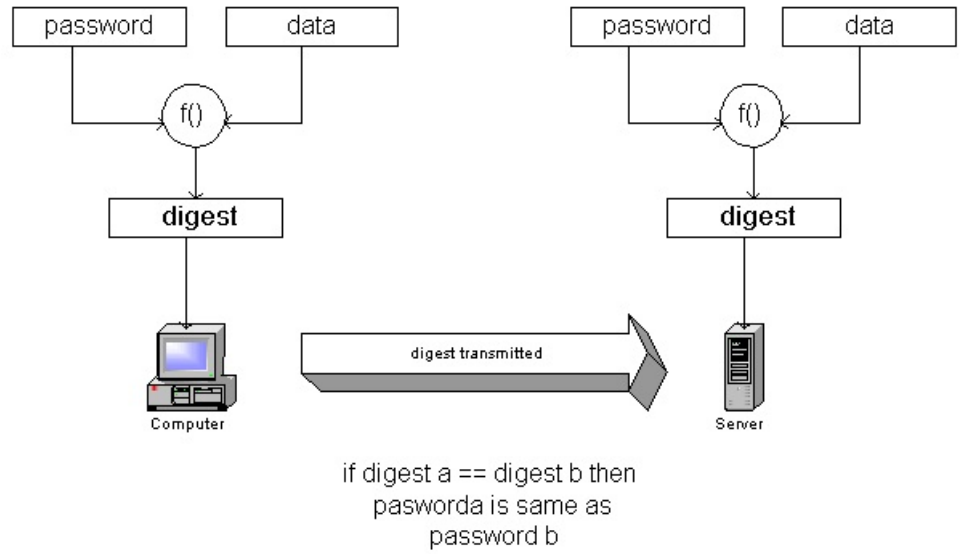
HTTP authentication has a problem in that there is no mechanism available to the server to cause the browser to 'logout'; that is, to discard its stored credentials for the user. This presents a problem for any web application that may be used from a shared user agent.

The username and password of course travel in effective clear-text in this process and the system designers need

to provide transport security to protect it in transit. SSL or TLS are the most common ways of providing confidentiality and integrity in transit for web applications.

HTTP Digest

There are two forms of HTTP Digest authentication that were designed to prevent the problem of username and password being interceptable. The original digest specification was developed as an extension to HTTP 1.0, with an improved scheme defined for HTTP 1.1. Given that the original digest scheme can work over HTTP 1.0 and HTTP 1.1 we will describe both for completeness. The purpose of digest authentication schemes is to allow users to prove they know a password without disclosing the actual password. The Digest Authentication Mechanism was originally developed to provide a general use, simple implementation, authentication mechanism that could be used over unencrypted channels.



As can be seen by the figure above, an important part of ensuring security is the addition of the data sent by the server when setting up digest authentication. If no unique data were supplied for request, an attacker would simply be able to replay the digest or hash.

The authentication process begins with a 401 Unauthorized response as with basic authentication. An additional header WWW-Authenticate header is added that explicitly requests digest authentication. A nonce is generated (the data) and the digest computed. The actual calculation is as follows:

1. String "A1" consists of username, realm, password concatenated with colons.

   owasp:users@owasp.org:password

2. Calculate MD5 hash of this string and represent the 128 bit output in hex
3. String "A2" consists of method and URI

   GET:/guide/index.shtml

4. Calculate MD5 of "A2" and represent output in ASCII.
5. Concatenate A1 with nonce and A2 using colons
6. Compute MD5 of this string and represent it in ASCII

This is the final digest value sent.

As mentioned HTTP 1.1 specified an improved digest scheme that has additional protection for

- Replay attacks
- Mutual authentication
- Integrity protection

The digest scheme in HTTP 1.0 is susceptible to replay attacks. This occurs because an attacker can replay the correctly calculated digest for the same resource. In effect the attacker sends the same request to the server. The improved digest scheme of HTTP 1.1 includes a NC parameter or a nonce count into the authorization header. This eight digit number represented in hex increments each time the client makes a request with the same nonce. The server must check to ensure the nc is greater than the last nc value it received and thus not honor replayed requests.

Other significant improvements of the HTTP 1.1 scheme are mutual authentication, enabling clients to also

authenticate servers as well as allowing servers to authenticate clients and integrity protection.

Forms Based Authentication

Rather than relying on authentication at the protocol level, web based applications can use code embedded in the web pages themselves. Specifically, developers have previously used HTML FORMs to request the authentication credentials (this is supported by the TYPE=PASSWORD input element). This allows a designer to present the request for credentials (Username and Password) as a normal part of the application and with all the HTML capabilities for internationalization and accessibility.

While dealt with in more detail in a later section it is essential that authentication forms are submitted using a POST request. GET requests show up in the user's browser history and therefore the username and password may be visible to other users of the same browser.

Of course schemes using forms-based authentication need to implement their own protection against the classic protocol attacks described here and build suitable secure storage of the encrypted password repository.

A common scheme with Web applications is to prefill form fields for users whenever possible. A user returning to an application may wish to confirm his profile information, for example. Most applications will prefill a form with the current information and then simply require the user to alter the data where it is inaccurate. Password fields, however, should never be prefilled for a user. The best approach is to have a blank password field asking the user to confirm his current password and then two password fields to enter and confirm a new password. Most often, the ability to change a password should be on a page separate from that for changing other profile information.

This approach offers two advantages. Users may carelessly leave a prefilled form on their screen allowing someone with physical access to see the password by viewing the source of the page. Also, should the application allow (through some other security failure) another user to see a page with a prefilled password for an account other than his own, a "View Source" would again reveal the password in plain text. Security in depth means protecting a page as best you can, assuming other protections will fail.

Note: Forms based authentication requires the system designers to create an authentication protocol taking into account the same problems that HTTP Digest authentication was created to deal with. Specifically, the designer should remember that forms submitted using GET or POST will send the username and password in effective clear-text, unless SSL is used.

Digital Certificates (SSL and TLS)

Both SSL and TLS can provide client, server and mutual entity authentication. Detailed descriptions of the mechanisms can be found in the SSL and TLS sections of this document. Digital certificates are a mechanism to authenticate the providing system and also provide a mechanism for distributing public keys for use in cryptographic exchanges (including user authentication if necessary). Various certificate formats are in use. By far the most widely accepted is the International Telecommunication Union's X509 v3 certificate (refer to RFC 2459). Another common cryptographic messaging protocol is PGP. Although parts of the commercial PGP product (no longer available from Network Associates) are proprietary, the OpenPGP Alliance (http://www.openPGP.org) represents groups who implement the OpenPGP standard (refer to RFC 2440).

The most common usage for digital certificates on web systems is for entity authentication when attempting to connect to a secure web site (SSL). Most web sites work purely on the premise of server side authentication even though client side authentication is available. This is due to the scarcity of client side certificates and in the current web deployment model this relies on users to obtain their own personal certificates from a trusted vendor; and this hasn't really happened on any kind of large scale.

For high security systems, client side authentication is a must and as such a certificate issuance scheme (PKI) might need to be deployed. Further, if individual user level authentication is required, then 2-factor authentication will be necessary.

There is a range of issues concerned with the use of digital certificates that should be addressed:

- Where is the root of trust? That is, at some point the digital certificate must be signed; who is trusted to sign the certificate? Commercial organizations provide such a service identifying degrees of rigor in identification of the providing parties, permissible trust and liability accepted by the third party. For many uses this may be acceptable, but for high-risk systems it may be necessary to define an in-house Public Key Infrastructure.
- Certificate management: who can generate the key pairs and send them to the signing authority?
- What is the Naming convention for the distinguished name tied to the certificate?
- What is the revocation/suspension process?
- What is the key recovery infrastructure process?

Many other issues in the use of certificates must be addressed, but the architecture of a PKI is beyond the scope of this document.

Entity Authentication

Using Cookies

Cookies are often used to authenticate the user's browser as part of session management mechanisms. This is discussed in detail in the session management section of this document.

A Note About the Referer

The referer [sic] header is sent with a client request to show where the client obtained the URI. On the face of it, this may appear to be a convenient way to determine that a user has followed a path through an application or been referred from a trusted domain. However, the referer is implemented by the user's browser and is therefore chosen by the user. Referers can be changed at will and therefore should never be used for authentication purposes.

Infrastructure Authentication

DNS Names

There are many times when applications need to authenticate other hosts or applications. IP addresses or DNS names may appear like a convenient way to do this. However the inherent insecurities of DNS mean that this should be used as a cursory check only, and as a last resort.

IP Address Spoofing

IP address spoofing is also possible in certain circumstances and the designer may wish to consider the appropriateness. In general use gethostbyaddr() as opposed to gethostbyname(). For stronger authentication you may consider using X.509 certificates or implementing SSL.

Password Based Authentication Systems

Usernames and passwords are the most common form of authentication in use today. Despite the improved mechanisms over which authentication information can be carried (like HTTP Digest and client side certificates), most systems usually require a password as the token against which initial authorization is performed. Due to the conflicting goals that good password maintenance schemes must meet, passwords are often the weakest link in an authentication architecture. More often than not, this is due to human and policy factors and can be only partially addressed by technical remedies. Some best practices are outlined here, as well as risks and benefits for each countermeasure. As always, those implementing authentication systems should measure risks and benefits against an appropriate threat model and protection target.

Usernames

While usernames have few requirements for security, a system implementor may wish to place some basic restriction on the username. Usernames that are derivations of a real name or actual real names can clearly give personal detail clues to an attacker. Other usernames like social security numbers or tax ID's may have legal implications. Email addresses are not good usernames for the reason stated in the Password Lockout section.

Storing Usernames and Passwords

In all password schemes the system must maintain storage of usernames and corresponding passwords to be used in the authentication process. This is still true for web applications that use the built in data store of operating systems like Windows NT. This store should be secure. By secure we mean the passwords should be stored in such a way that the application can compute and compare passwords presented to it as part of an authentication scheme, but the database should not be able to be used or read by administrative users or by an adversary who manages to compromise the system. Hashing the passwords with a simple hash algorithm like SHA-1 is a commonly used technique.

Ensuring Password Quality

Password quality refers to the entropy of a password and is clearly essential to ensure the security of the users' accounts. A password of "password" is obviously a bad thing. A good password is one that is impossible to guess. That typically is a password of at least 8 characters, one alphanumeric, one mixed case and at least one special character (not A-Z or 0-9). In web applications special care needs to be taken with meta-characters.

Password Lockout

If an attacker is able to guess passwords without the account becoming disabled, then eventually he will probably be able to guess at least one password. Automating password checking across the web is very simple! Password lockout mechanisms should be employed that lock out an account if more than a preset number of unsuccessful

login attempts are made. A suitable number would be five.

Password lockout mechanisms do have a drawback, however. It is conceivable that an adversary can try a large number of random passwords on known account names, thus locking out entire systems of users. Given that the intent of a password lockout system is to protect from brute-force attacks, a sensible strategy is to lockout accounts for a number of hours. This significantly slows down attackers, while allowing the accounts to be open for legitimate users.

Password Aging and Password History

Rotating passwords is generally good practice. This gives valid passwords a limited life cycle. Of course, if a compromised account is asked to refresh its password then there is no advantage.

Automated Password Reset Systems

Automated password reset systems are common. They allow users to reset their own passwords without the latency of calling a support organization. They clearly pose some security risks in that a password needs to be issued to a user who cannot authenticate himself.

There are several strategies for doing this. One is to ask a set of questions during registration that can be asked of someone claiming to be a specific user. These questions should be free form, i.e., the application should allow the user to choose his own question and the corresponding answer rather than selecting from a set of predetermined questions. This typically generates significantly more entropy.

Care should be taken to never render the questions and answers in the same session for confirmation; i.e., during registration either the question or answer may be echoed back to the client, but never both.

If a system utilizes a registered email address to distribute new passwords, the password should be set to change the first time the new user logs on with the changed password.

It is usually good practice to confirm all password management changes to the registered email address. While email is inherently insecure and this is certainly no guarantee of notification, it is significantly harder for an adversary to be able to intercept the email consistently.

Sending Out Passwords

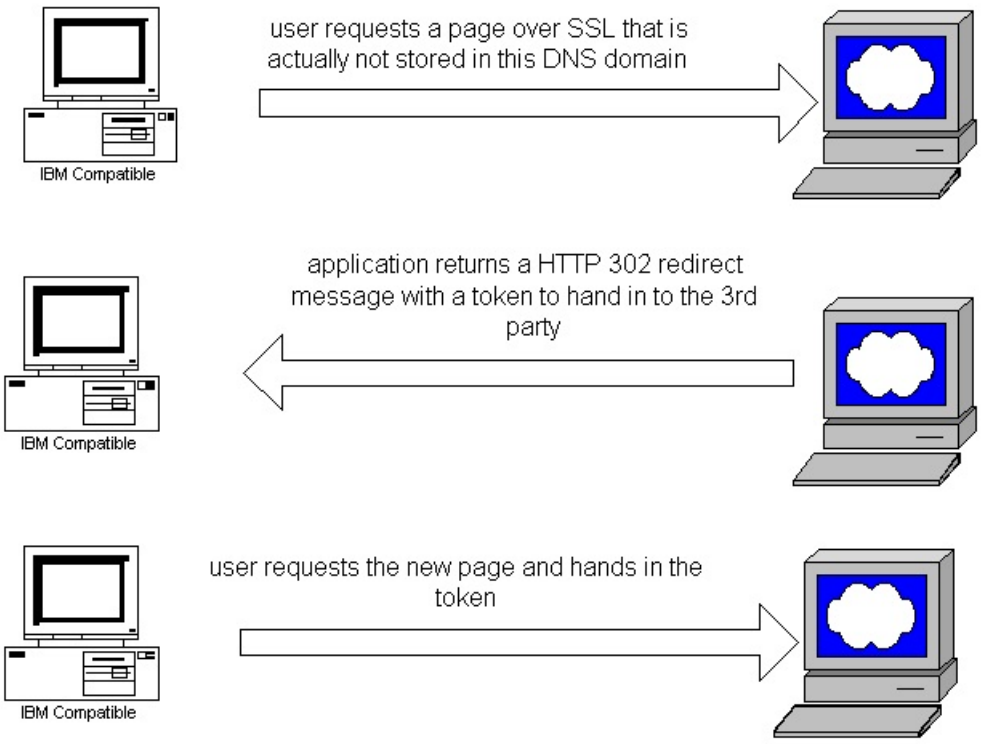
In highly secure systems passwords should only be sent via a courier mechanism or reset with solid proof of identity. Processes such as requiring valid government ID to be presented to an account administrator are common.

Single Sign-On Across Multiple DNS Domains

With outsourcing, hosting and ASP models becoming more prevalent, facilitating a single sign-on experience to users is becoming more desirable. The Microsoft Passport and Project Liberty schemes will be discussed in future revisions of this document.

Many web applications have relied on SSL as providing sufficient authentication for two servers to communicate and exchange trusted user information to provide a single sign on experience. On the face of it this would appear sensible. SSL provides both authentication and protection of the data in transit.

However, poorly implemented schemes are often susceptible to man in the middle attacks. A common scenario is as follows:



The common problem here is that the designers typically rely on the fact that SSL will protect the payload in transit and assumes that it will not be modified. He of course forgets about the malicious user. If the token consists of a simple username then the attacker can intercept the HTTP 302 redirect in a Man-in-the-Middle attack, modify the username and send the new request. To do secure single sign-on the token must be protected outside of SSL. This would typically be done by using symmetric algorithms and with a pre-exchanged key and including a time-stamp in the token to prevent replay attacks.
