White Paper

Recent Advances in Web Application Security

Author:
Neelay S Shah
Principal Security Consultant
Foundstone Professional Services

Table of Contents

Introduction

Over the past decade, cross site scripting[1] (XSS) has become one of the most ubiquitous vulnerabilities afflicting web applications. More recently "ClickJacking"[2] was discovered which probably is even more prevalent than XSS in modern web applications. The ease with which these vulnerabilities can be identified and exploited along with the substantial benefits to be had (for e.g. compromising the user's session to impersonate the victim user to the application, tricking the user into submitting sensitive credential information, performing a privileged action on behalf of the user etc.) by exploiting these vulnerabilities make them a perfect target for the attackers to look for and exploit in web applications.

In this article, we will survey some of the techniques that have been introduced by the browser makers that are designed to prevent exploitation of these widespread vulnerabilities. These techniques are not dependent on HTML5 but instead are standalone techniques. We will NOT be looking at purely client side techniques such as "Cross Site Scripting (XSS) filters" that are completely implemented and enforced client side. We will be focusing on the techniques that include a server side component and allow the web developer to control and tweak the level of protection enforced. Also note that there have been solutions presented to remediate these vulnerabilities; however these new techniques present the web developer and administrators an elegant and efficient way to eliminate these vulnerabilities as compared to the more involved techniques. All of these new mechanisms are enforced by the end user's browser. Further, they are also backward compatible and as such a browser that does not understand these techniques continues to interpret and render the response as if they did not exist.

This article is intended for web developers and web application administrators.

Content Security Policy

Content Security Policy (CSP)[3] was primarily developed to eliminate exploitation of existing XSS vulnerabilities. It does so without requiring the developer to substantially modify the application source code. Additionally, CSP also offers support for mitigating ClickJacking vulnerabilities as well as information disclosure through network sniffing and man-in-the-middle attacks.

Exercising CSP in practice consists of 2 parts:

- Defining a policy—Web application developers and administrators control this part and notify the policy to the end user's browser through the HTTP response
- Enforcing the policy—The browser parses the policy (sent by the web application) and enforces it appropriately on the client side

CSP is extremely granular and allows setting of a different policy per resource (e.g. individual web page/image/script etc.). As a result, the web application needs to send a CSP for every resource in order for the browser to enforce that policy for the resource. Also, CSP is a completely stateless mechanism and the browser does not remember the CSP directives that the web application sent for a particular resource once the request is over. Thus, the web application needs to send the CSP directives with every response in every session. The web application can include the CSP directives as part of the HTTP response headers by using the X-Content-Security-Policy header or as part of the HTML response by using the <meta http-equiv=" X-Content-Security-Policy"> tag.

Following are the core elements that are part of the CSP. Note if the web developer chooses to not define any CSP policy then the browser interprets and renders the response as it did in the pre-CSP days.

1. Cross Site Scripting—https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
2. ClickJacking—https://www.owasp.org/index.php/Clickjacking
3. Content Security Policy—https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html

Directive	Description
default-src / allow	• Allows the web developer to define the whitelist of sources from which the browser is allowed to fetch content. • Serves as the default i.e. defines the policy for all resources that are not explicitly restricted by any other directives
script-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to download and execute JavaScript • Disables run time construction and execution of code from strings • Supports an unsafe-inline attribute which when enabled, enables inline JavaScript execution • Supports an unsafe-eval attribute which enables converting strings into code and executing the same
object-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to download resources while constructing the <applet>, <object> and <embed> tags • The following attributes are subject to the restrictions imposed by the object-src directive ◦ data attribute on the <object> element ◦ src attribute on the <embed> element ◦ code and archive attributes on the <applet> element
img-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to download images • The following attributes are subject to the restrictions imposed by the img-src directive ◦ src attribute on the <img> element ◦ url() and image() values on any CSS
media-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to download media • The following attributes are subject to the restrictions imposed by the media-src directive ◦ src attribute on the <video> element ◦ src attribute on the <audio> element
style-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to download stylesheets • The following attributes are subject to the restrictions imposed by the style-src directive ◦ href attribute on the <link rel=stylesheet"> element ◦ href attribute on the <?xml-stylesheet?> element
font-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to download fonts • The following attributes are subject to the restrictions imposed by the font-src directive ◦ url value of the src descriptor of the @font-face CSS rule
frame-src	• Allows the web developer to define the whitelist of sources from which the browser is allowed to fetch frames • The following attributes are subject to the restrictions imposed by the font-src directive ◦ src attribute on the <iframe> element
frame-ancestors	• Allows the web developer to define the whitelist of sources which are allowed to frame the current resource
options	• Allows the web developer to modify the base restrictions of the CSP ◦ inline-script—Enables inline execution of JavaScript ◦ eval-script—Enables run time construction and execution of code from strings

Best Practices

1. Do not send CSP directives over clear text HTTP channel since those could be removed all together or alternatively abused by an attacker to their own advantage.
2. Do NOT modify the defaults and enable inline script execution, runtime construction of code from strings, or loading images from non-trusted domains. This will help prevent XSS attacks.
3. Modify the default "Frame-ancestor" to enable only trusted web sites to be able to load the web application within frames to prevent ClickJacking attacks.

Browser Support[4]

Firefox	Chrome	Internet Explorer	Opera	Safari
4+	13+	Not supported as yet	Not supported as yet	Not supported as yet

Example

Foundstone HacmeBank[5] is a training web application that is vulnerable to cross-site scripting attacks. We will see how we can remediate the exploitation of the same using a suitable CSP.

The following figure shows Foundstone HacmeBank redirecting the user to a malicious login page when the victim user browses to the "Posted Messages". This is because a malicious user has posted a message with a specially crafted JavaScript:

```
<script>window.location =
http://www.evilattacker.com/malicious_login_page.htm <script>
```

When the victim user clicks the "Posted Messages" menu link, they are redirected to the attacker's login page and tricked into submitting their login credentials to the attacker.

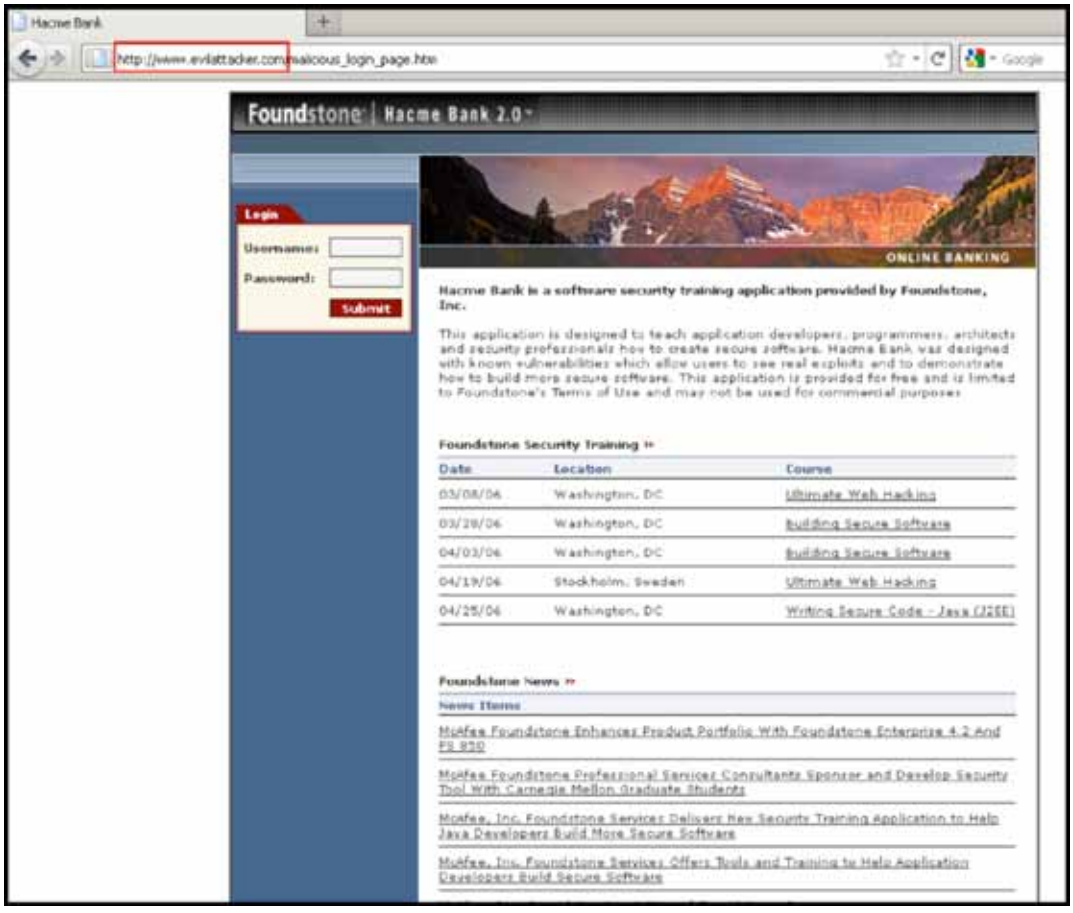

Figure 1. Figure shows the victim user being redirected automatically to the attacker controlled website.

4. Browser support for CSP
Firefox—http://blog.mozilla.com/security/2011/03/22/creating-a-safer-web-with-content-security-policy/
Chrome (Experimental support)—adds experimental support for CSP—http://blog.chromium.org/2011/06/new-chromium-security-features-june.html
5. http://www.mcafee.com/us/downloads/free-tools/hacme-bank.aspx

We now modify the "PostedMessages" web page server side source code so that it returns the following policy as part of the response headers:

```
X-Content-Security-Policy: default-src 'self'
```

This time around, when the victim user navigates to the "Posted Messages" section, the browser enforces the CSP directives sent by the page and does not execute any inline script. As such the user is not redirected to the attacker controlled web server. The following figure(s) shows the same:

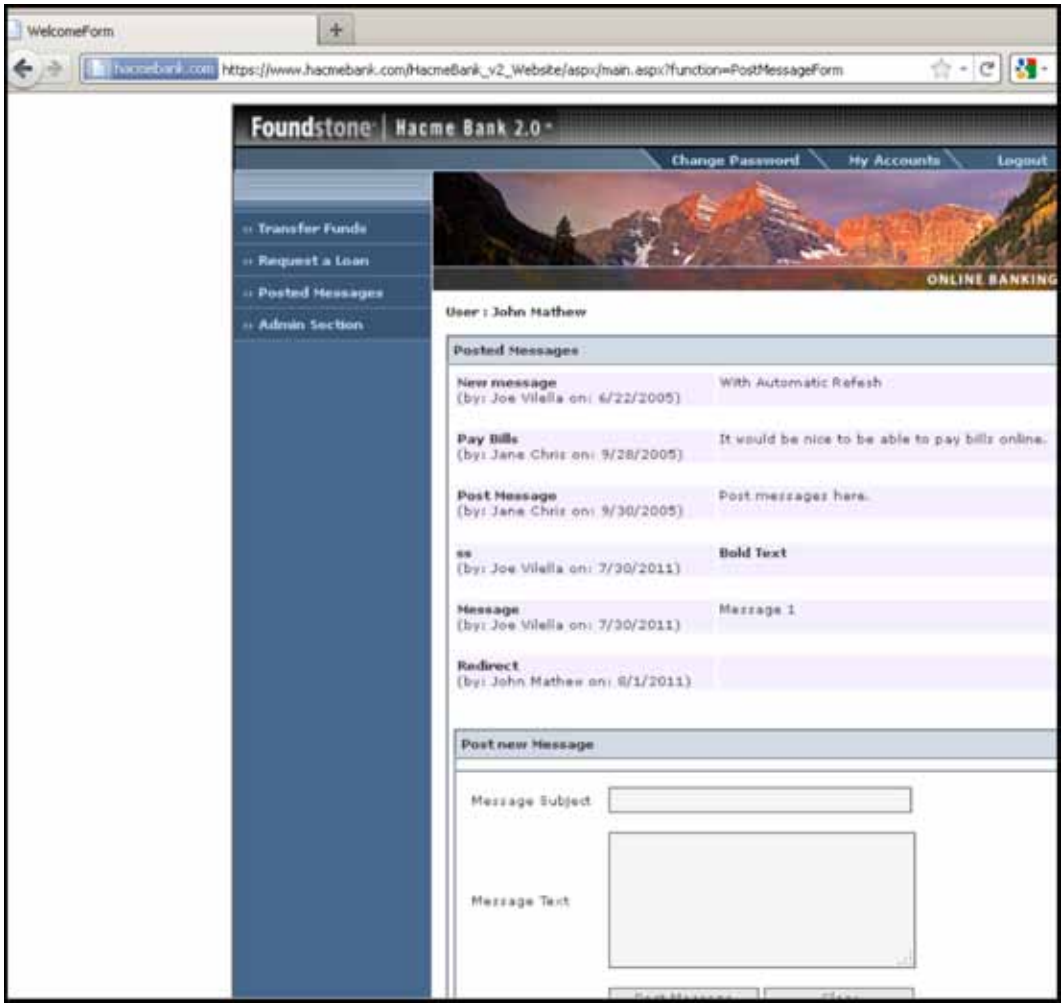

Figure 2. Figure shows the application not redirecting the victim user but instead displaying the posted messages.

X-Frame-Options

X-Frame-Options mechanism was designed to offer protection against "framing" attacks such as ClickJacking attacks. Exercising X-Frame-Options in practice consists of 2 parts:

- Defining a policy—Web application developers and administrators control this part and notify the policy to the end user's browser through the HTTP response
- Enforcing the policy—The browser parses the policy (sent by the web application) and enforces it appropriately on the client side

The policy is conveyed to the browser through HTTP response headers. Specifically the "X-Frame-Options" response header is used for this purpose. As with CSP, the "X-Frame-Options" technique is resource specific and completely stateless. Thus the web application needs to send the "X-Frame-Options" header for every resource and moreover for every resource response in every session in order for the browser to enforce that policy for the resource.

The X-Frame-Options policy can support the following directives. Note if the web developer chooses to not define any X-Frame-Options policy then the browser interprets and renders the response as it did in the pre-X-Frame-Options days.

Directive	Description
DENY	• The resource cannot be enclosed within a frame by any webpage
SAMEORIGIN	• The resource can be enclosed within a frame by a webpage with the same **top-level** Origin as the resource itself • The Origin comprises of the following primitives ◦ Scheme e.g. http or https ◦ Hostname e.g. www.foundstone.com ◦ Port e.g. 80 and 443
ALLOW-FROM[6]	• The resource can be enclosed within a frame by a webpage originating from the same Origin as specified in the directive • Allows the developer to specify only a single origin. Wildcards are NOT supported

Best Practices

1. Do not send X-Frame-Options header over clear text HTTP channel since those could be removed all together or alternatively abused by an attacker to their own advantage
2. As far as possible, set the X-Frame-Options to DENY to prevent the page from being framed by any other page
3. If the SAMEORIGIN directive is used then make sure there isn't another web page on the domain that allows embedding an arbitrary page within a frame
4. If DENY and SAMEORIGIN are not granular enough and / or if you legitimately want 3rd party sites (partner, vendor etc.) to be able to frame the page then use the following approach:
   a. Include the origin of the requesting page as an URL parameter
   b. Validate the same on the server to ensure that a trusted page is requesting to frame the resource
   c. Respond with either DENY or ALLOWFROM with the value set to the origin of the requesting page
   d. Note if an un-trusted requesting page did spoof the origin URL parameter, the browser will not frame the resource since the browser will validate the top-level origin against the specified policy before it frames the resource

6. Not supported by all browsers. Supported in Internet Explorer 8 and later—http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx

Browser Support[7]

Firefox	Chrome	Internet Explorer	Opera	Safari
3.6.9+	4.1.249.104+	8.0+	10.50+ (Presto 2.6)	4.0+

Example

The Foundstone HacmeBank web application is vulnerable to ClickJacking. The following figures show how the "Delete User" functionality could be framed and invoked without the user's knowledge

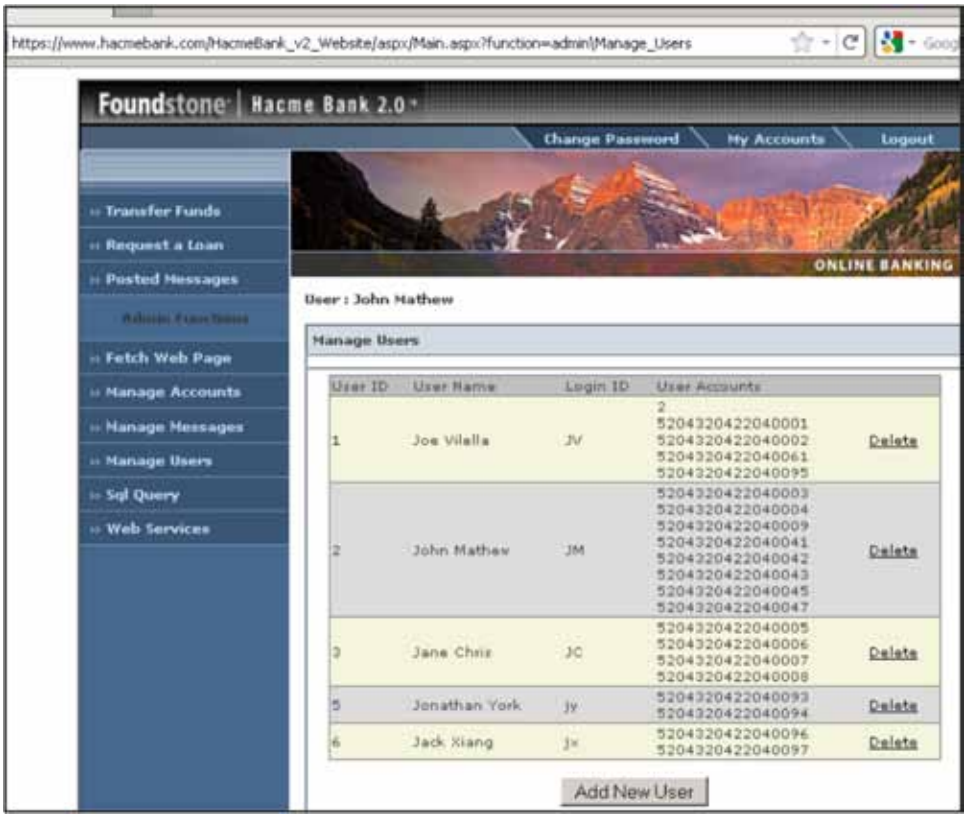

Figure 3. Figure shows the "Manage Users" page which allows deleting existing users.

7. Browser support for X-Frame-Options—
Firefox—https://developer.mozilla.org/en/The_X-FRAME-OPTIONS_response_header
Chrome—http://blog.chromium.org/2010/01/security-in-depth-new-security-features.html
IE—http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
Opera—http://www.opera.com/docs/specs/presto26/#network
Safari—http://support.apple.com/kb/ht3613

Figure 4. Figure shows the HacmeBank "Manage User" page being framed within a malicious page. The frame opacity has not been completely turned off for sake of clarity.

When the attacker tricks the user into visiting the malicious page the user would see the "Click Here to Win an Ipad" and when the user clicks the link, the request would end up going to the HacmeBank web application which in turn would duly delete the user. The HacmeBank web application can be modified to use the following policy to enforce X-Frame-Options which would prevent "framing" of the web page:

```
X-Frame-Options: DENY
```

This time around when the attacker tricks the user into visiting the malicious page the browser will only load the attacker's page. It will not load the HacmeBank resource because of the X-Frame-Options response header sent by HacmeBank to the browser. The following figure shows the same:

Figure 5: Figure shows that the browser prevents loading the HacmeBank resource within a frame.

HTTP Strict Transport Security (HSTS)

HSTS was designed to securely setup the browsing session. For instance, consider the typical situation wherein a user types www.mybank.com in the browser address bar and clicks Go. The browser then tries to establish a connection over clear text HTTP to www.mybank.com on port 80. At this point, the website www.mybank.com has the option to redirect the browser to https://www.mybank.com which then results in the browser initiating an SSL connection. If you observe this sequence closely, the SSL connection setup is introduced too late. The initial request is over clear text HTTP and an attacker could choose to launch a man-in-the-middle (MiTM) attack right there and completely negate the enforcement of SSL to a large extent. A slight extension of the above scenario is if an attacker managed to trick the user into clicking http://www.mybank.com. Even if the web application does not serve any content over clear text HTTP, the user's session cookies will be potentially sent over the clear text HTTP protocol and an attacker on the same network could sniff the same and compromise the user's session.

HSTS allows the web application developer and/or web application administrator to indicate to the browser that it should always use a secure protocol (HTTPS) to initiate a connection to the web application. This closes the window of opportunity wherein HTTP would be used for the initial connection and when a MiTM attack could be performed. Note that this relies on the end user to connect securely to the application the first time. The will ensure that the HSTS policy is obtained securely and is not subject to MiTM attacks.

Exercising HSTS in practice consists of 2 parts:

- Defining a policy—Web application developers and administrators control this part and notify the policy to the end user's browser through the HTTP response
- Enforcing the policy—The browser parses the policy (sent by the web application) and enforces it appropriately on the client side

The policy is conveyed to the browser through HTTP response header "Strict-Transport-Security". Unlike CSP and X-Frame-Options, HSTS is applicable to the host and not to a specific resource. Thus if a particular resource is served with an HSTS policy then that policy is in effect for the server as a whole. Additionally, HSTS is persistent and the web application developer / administrator control the duration for which the browser should enforce the policy.

HSTS comprises of the following core directives. Note if the web developer chooses to not define any HSTS policy then the browser interprets and renders the response as well as make future requests as it did in the pre-HSTS days.

#	Directive	Description
1	max-age	• Allows the web developer to indicate the time in seconds when the policy should remain in effect
2	includeSubdomains	• Allows the web developer to indicate that the browser should apply the policy to the website subdomains as well

Once HSTS has been configured, and the browser encounters an error while setting up the SSL session (e.g. an attacker trying to perform a MiTM attack or even an expired / self-signed certificate), the browser will not present the user the ability to override the SSL certificate error and proceed with the connection. The SSL handshake is terminated and there is no way to browse to the website until the policy has been manually cleared, expired or the certificate error has been remediated.

Best Practices

1. Send the Strict-Transport-Security header over a securely established HTTPS channel
   a. The browser does not respect and enforce the Strict-Transport-Security header when it is received over a clear text HTTP channel
   b. The browser does not respect and enforce the Strict-Transport-Security header when it is received over an insecurely established SSL session. This implies that in situations where the user manually overrides the SSL certificate error presented by the browser, the browser will not respect and enforce the Strict-Transport-Security header
2. If you do not have a certificate issued by a trusted root certificate authority (CA), have an internal CA issue a certificate and modify the end user browser settings to trust the internal CA
3. Include the Strict-Transport-Security header (over a securely established HTTPS channel) as part of the response for every resource request so that the browser continuously updates the HSTS policy for the web application

Browser Support[8]

Firefox	Chrome	Internet Explorer	Opera	Safari
4+	4.1.249.104+	Not supported as yet	Not supported as yet	Not supported as yet

Example

In this case, the Foundstone HacmeBank web application can be modified to use the following policy as part of the HTTP response headers to enforce HSTS. The policy instructs the browser to use HTTPS when accessing the HacmeBank application for the next 90 days expressed as the number of seconds.

```
Strict-Transport-Security: max-age= 7776000
```

8. Browser support for HSTS
Firefox—http://hacks.mozilla.org/2010/08/firefox-4-http-strict-transport-security-force-https/
Chrome—http://blog.chromium.org/2010/01/security-in-depth-new-security-features.html

The following figure shows HacmeBank including the policy as part of the response header

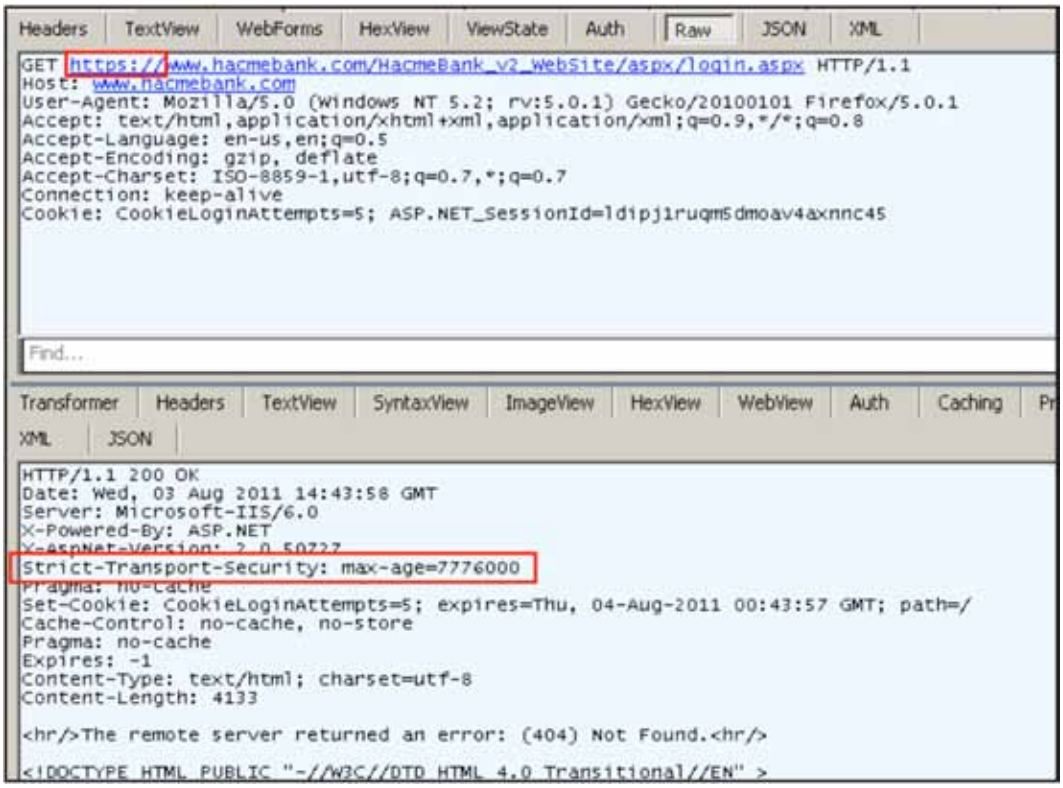

Figure 6. Figure shows HacmeBank including the Strict-Transport-Security header in the response.

Once the policy is in effect, attempting to access any HacmeBank resource over clear text HTTP channel automatically causes the browser to modify the request such that the access is made over HTTPS. The following figures show the same

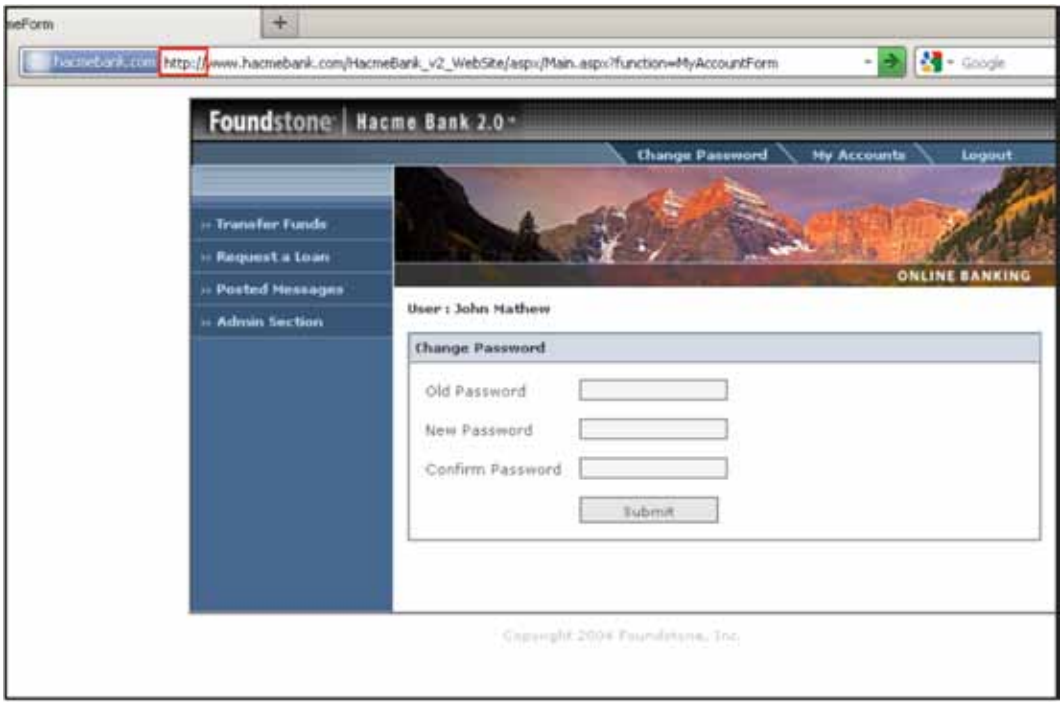

Figure 7. Figure shows the user manually trying to access a HacmeBank resource over clear text HTTP.

Figure 8. Figure shows the browser automatically modifies the request so that the connection attempt is made over HTTPS.

Conclusion

The web application security advancements that we have discussed in this article without a doubt raise the bar in securing web applications by allowing web developers to establish an additional layer of defense as well as remediate the common web application vulnerabilities efficiently. While web developers are strongly recommended to investigate and leverage these techniques within their applications it should also be noted that the modern browser is a key component in the enforcement of these techniques. Hence these new techniques don't serve as a complete replacement to the existing remediation techniques since users not using the modern browsers would have no protection against the vulnerabilities that they remediate.

About Foundstone Professional Services

Foundstone® Professional Services, a division of McAfee. Inc., offers expert services and education to help organizations continuously and measurably protect their most important assets from the most critical threats. Through a strategic approach to security, Foundstone identifies and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively. The company's professional services team consists of recognized security experts and authors with broad security experience with multinational corporations, the public sector, and the US military.

About McAfee

McAfee, a wholly owned subsidiary of Intel Corporation (NASDAQ:INTC), is the world's largest dedicated security technology company. McAfee delivers proactive and proven solutions and services that help secure systems, networks, and mobile devices around the world, allowing users to safely connect to the Internet, browse, and shop the web more securely. Backed by its unrivaled global threat intelligence, McAfee creates innovative products that empower home users, businesses, the public sector, and service providers by enabling them to prove compliance with regulations, protect data, prevent disruptions, identify vulnerabilities, and continuously monitor and improve their security. McAfee is relentlessly focused on constantly finding new ways to keep our customers safe. http://www.mcafee.com.

McAfee
2821 Mission College Boulevard
Santa Clara, CA 95054
888 847 8766
www.mcafee.com

34200wp_web-app-security_0811_fnl_ETMG