JavaScript and Flash Pose a Serious Threat to System Security

M J Morshed Chowdhury

One of the most amazing and striking application of technology in this century is the impact of Internet on the human society. During this period web applications have tremendous growth rate and touches almost all walks of life. Introduction of Web 2.0 has facilitates and allows increased user-creator interaction, content syndication, advancements in web-based user interfaces, which ultimately lead to the creation of an entirely new application platform.

Like any other technology, web application also has its weakness. It inherits the security vulnerabilities of open Internet architecture. According to Pete Lindstrom, Director of Security Strategies with the Hurwitz Group, Web applications are the most vulnerable elements of an organization's IT infrastructure today. An increasing number of organizations depend on Internet-based applications that leverage the power of dynamic and rich content mechanism (e.g., AJAX and Flash). As this group of technologies becomes more complex to allow the depth and functionality discussed, and, if organizations do not secure their web applications, then security risks will only increase. The most striking features of web 2.0 are its ability in harnessing collective intelligence and bringing rich users participation. The web 2.0 has rich applications with features such as user interaction, collaboration and real time communication. To support the synchronous communication AJAX is widely used. Another popular technology for motion picture of video in web space is Flash. Flash also posses few critical security vulnerabilities. If action script in flash is not implemented properly, it can compromise any web application.

A careful analysis of potential attacks against Web services as carried out e.g. by Jensen et al. immediately shows that Web services are very vulnerable especially against DoS attacks. The security issues which are inherent to the Ajax programming model and which especially affect cooperative application have been extensively documented by Michael Sonntag. In recent research it has been shown that scripting vulnerability is higher than any other web application vulnerabilities. It is getting sophisticated day by day and should be addressed from the early development cycles.

Types of Vulnerabilities and Attacks

Current attacks come through many means such as Server-side attacks (Traditional), Browser & Plugin Flaws and Client-side attacks (XSS, CSRF). Many of the current vulnerability countering mechanisms address one or few specific issues.

Browser cache and history are intended to be private in the normal stream, yet it's not difficult for malicious Web sites to "sniff" cache entries on visitors' computers and then use that information to more accurately deceive them. This leads to pose a major un-resolving issue to the research community.

On the Web, scripts embedded in multiple browser windows containing documents from the same Web site (same domain name) are allowed to access data in each other, in order to support multi windowed user interfaces. In an analysis it has been revealed that browser windows could be tricked into trusting at-tack scripts from rogue sites, thus allowing them to access their data. A rogue site could be set up to track all Web-related activity of visitors even after they had left the site, using a Trojan-horse attack.

The tracking provided access to all data typed into forms, including password fields, cookies, and visited URLs. The data was extracted right in the browser, so using a secure encrypted connection to retrieve documents didn't accord the user any extra protection. This browser vulnerability has a serious implication for Web users. Once infected by the Trojan horse, the user's Web interaction is fully exposed to the attacker - every URL retrieved, all data typed into forms - including credit card numbers and passwords, all cookies set by servers accessed etc.

The HTTP protocol supports a facility for authenticating Web users. Many Web-based services however use alternate methods of authorization that provide more flexibility. These methods involve the use of dynamically generated, opaque "session keys" embedded in URLs, in hidden fields of forms or in cookies. The ability of the attack to access such information in an HTML document makes all of these authentication mechanisms susceptible to compromise.

This browser vulnerability also has a serious implication for intranets. Most users use the same browser to access information on the intranet as well as the Internet. A user who has been 'attacked' using this vulnerability has essentially compromised the renewal for the duration of the browsing session the Trojan horse is able to extract data from subsequently loaded intranet documents and transmit it to an external entity. Any data that the user enters into forms - ID numbers, vendors and prices, bug reports, passwords and other proprietary information can be relayed to the outside.

ActionScript vulnerabilities are due to various program flow calculating errors in the verification/generation process. ActionScript code is typically compiled into bytecode format called ActionScript Byte Code (ABC). The bytecode verifier is responsible for safety check, making sure there is no type-unsafe operations, stack underflow/overflow, improper array accesses, etc.

Type confusion vulnerability exists in Adobe Flash Player ActionScript Virtual Machine. Specifically, the flaw exists in the implementation of callMethod bytecode command. The bytecode verifier fails to detect the stack misalignment under certain circumstances. An attacker can exploit this vulnerability by enticing a user to visit a crafted web page, open a crafted PDF file or open a crafted Office document; all of which may contain malicious Adobe Flash content. Successful exploitation would allow for arbitrary code execution with the privileges of the currently logged in user. কজ

Suggestion

One of the solutions to combat this security vulnerability is to use HTML encoding. It can be used either on user submitted data in the view or it can be used on user submitted data in the controller.

Another solution could be a safe interpreter. A safe interpreter has the task of isolating scripts from executing any unsafe commands (those that could result in security compromises if misused), thus implementing what is called a padded cell. The interpreter has to implement access control with respect to objects within the script's own context. A safe interpreter has to implement access control, independence of contexts, and management of trust among different contexts. Provision for these components does not realize a particular security policy. Rather, it gives a framework in which a variety of security policies can be easily implemented.

Web 2.0 applications have moved the Internet forward and help fulfill the promise of more interactive functionality and community building. The open nature of Web 2.0 presents significant challenges to the traditional enterprise approach to controlling intellectual property and proprietary content. However, security is not usually considered. The increase in functionality and interactivity has increased the ways in which an application can be attacked successfully