RAPID7
UNIFIED VULNERABILITY MANAGEMENT

WHITE PAPER

Securing Web 2.0 Applications

CLOSING THE DOOR TO DANGEROUS VISITORS

RAPID7 Corporate Headquarters 545 Boylston Street Boston, MA 02116 617.247.1717 www.rapid7.com

INTRODUCTION

It wasn't long ago that many doubted the heavy use of JavaScript in corporate applications. Real enterprise applications were built using programming languages and client/server configurations. Web technologies have enabled the Internet to develop into an application platform, becoming the platform of choice for both internal and external corporate applications. Today the set of web technologies and programs known as Web 2.0 is cultivating a social trend of new behaviors enabling real time communication and information sharing.

Web 2.0 sites make it easy and convenient for web users to collaborate and interact with web applications by providing asymmetric communications and superior user interface design. Web 2.0 sites use powerful new technologies that put more intelligence into the browser to expand the range of what can be delivered to the end user. The most important of these technologies is *AJAX*, short for *Asynchronous JavaScript a*nd *XML*.

This new framework once considered social and collaborative at the personal level has entered the corporation, enabling enterprises to develop relationships and collaborate with their customers. However, these applications present risks, opening the corporation to data theft, information leakage and liability for information misuse by employees. While Web 2.0 tools are simple and facilitate conversations, they weren't necessarily built for enterprise use.

Despite having the same business drivers, the landscape of corporate networks has changed significantly since the days of static web pages. There are a myriad of regulations that govern the responsibilities of network operators driving the impetus to secure corporate networks, but do nothing to inform the owners or their security teams of the new threats which have emerged.

Web 2.0 technologies have created a windfall of utility, making web applications extremely versatile. But this new opportunity opens up exposure to vulnerabilities which attackers can find easily through manual methods. This paper will make the argument that Web 2.0 applications are easier to attack through flaws that are harder to automatically detect.

Background

While this paper will argue that the web has been radically altered, it should be noted that business has not. Three main tenets of business from the perspective of the website owners are largely the same as they were in the 1990's.

- The application must maintain availability of revenue generating functionalities and accessibility to the customer. This directive ensures that the application can be touched from anywhere on the Internet without sacrificing the functionality that affects profitability.
- The application must ensure corporate and customer data such as credit card, social security numbers and account information is secure from exploits. Nothing is more embarrassing or costly to a company then a data breach containing sensitive information. The number on how much each lost record costs varies but generally speaking the multiplier is so high (usually thousands to millions of records) that a huge loss is guaranteed.
- Finally, the entire website needs to adhere to compliance regulations like HIPAA, PCI, SOX, and GLBA. These frameworks won't necessarily prevent a data loss incident, but the spirit of their guidance is supposed to prevent or deter just that. Staying in step with the regulations can be a daunting process.

Business owners may be responsible for the compliance of the web application but IT security administrators are responsible for protecting their networks and everything that runs on them, including web applications. When there is a breach or an exploit, the security team (or person) becomes the center of attention. It is imperative that companies have a computer intrusion response team (CIRT) built and ready to respond. This means putting together comprehensive plans for various intrusion scenarios and simulating these events to test preparedness. The first step towards that preparedness is to perform a thorough investigation of the network and the applications which live on it.

Vulnerability Scanners

Vulnerability management solutions have traditionally focused on the network or operating system, providing the analysis using both the traditional manual penetration testing as well as automated security testing tools. Automated tools have mostly focused on either the network or the application, attempting to go deeper into the environment to find exposures that would put an organization at risk. Web applications are typically known to be "rabbit holes" in testing because they are difficult to scan using automated tools due to their potentially infinite depth to explore.

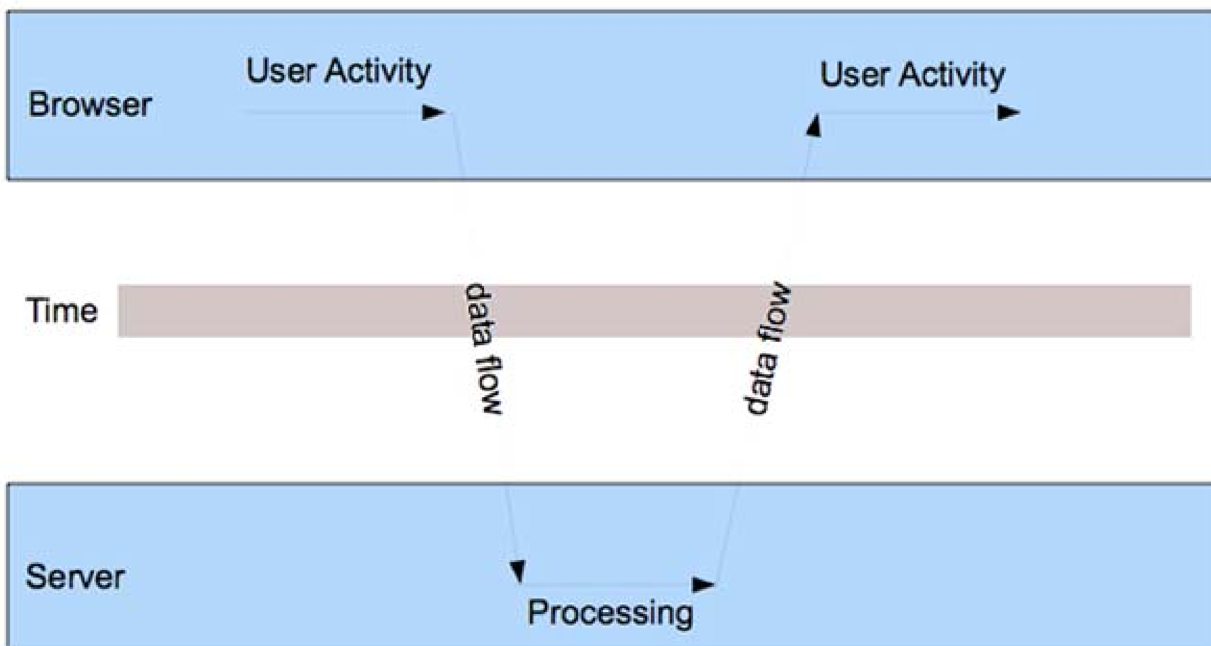
Trends are leaning towards software that merges both the ability to scan for network vulnerabilities and application-level vulnerabilities. This is a win for the consumers of detection technology since they will not have to purchase separate product licenses, training, and service contracts. A unified scanning application also allows for easier reporting as well as comprehension.

Web 2.0 Application Scanning

To understand what is so special about the Web 2.0 technologies, we first need to look at how classic web applications worked. The stateless nature of the HTTP transaction made user activity a punctuated event. Whenever a user selected some piece of data from a drop down box or other form elements, the web server would need to process that selection and redraw the entire page. This took a long time and provided the user with a fragmented experience. The classic web architecture also made certain features, such as auto-completing search boxes, impractical.

Figure 1: Classic Web Application Model from "Subverting AJAX" by Stefano Di Paola and Giorgio Fedon

Web 2.0 has provided a way to interact with a page asynchronously. A user could begin typing into a search box and the page will be able to respond with choices from a database without needing to redraw the entire page.



AJAX Web Application Architecture

The main technology framework behind Web 2.0 is Asynchronous JavaScript and XML (AJAX). Using this framework, a developer can produce web pages which seem to act similar to a local application. The interface is no longer bound by punctuated HTTP transactions which require the page to be redrawn. As illustrated in Figure 2, the processing occurs asynchronously on the server while the AJAX framework continues to the user. This is the technology that allows for features such as auto-complete in search engines. The web interface can poll the text field in real time and send the characters back to the server asking for any partial matches to what the user has entered. As partial matches are found the server returns them to the interface to be displayed to the user. All of the returned results represent known entries in the database. However, as one of the older axioms of security predicts, the new ease of usability brings with it an excess of security issues.

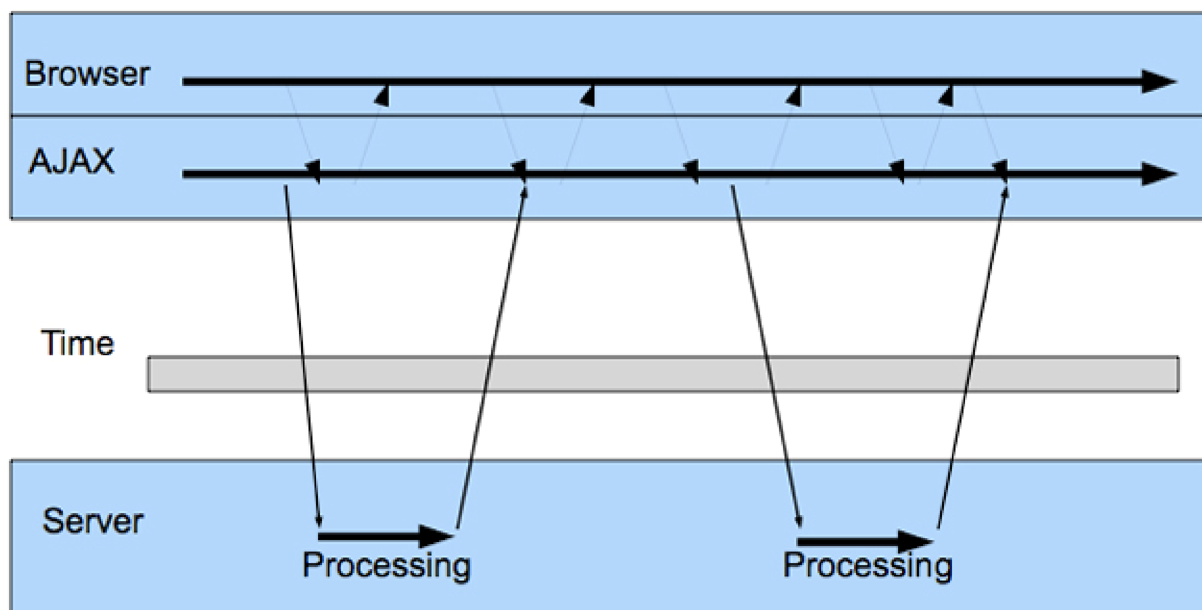


Figure 2: Asynchronous Web Application Model from "Subverting AJAX" by Stefano Di Paola and Giorgio Fedon

AJAX Security Issues

The most important thing to understand about automated web application assessments is that you have to treat every form field, every URL query parameter, every web services method argument, every cookie and HTTP header as a possible injection point. The security community began documenting issues in AJAX technologies as far back as 2006. In December of that year, Di Paola and Fedon presented a seminal paper on this topic to the 23rd Chaos Communication Congress entitled "**Subverting AJAX**". Billy Hoffman also notes in a **2006 Blackhat presentation** that the real ingenuity of the AJAX attack class is the ability to follow the DOM model yet still produce results that are malicious to the end user. Di Paola and Fedon describe several scenarios in their paper which show the ability to bypass authentication, sniff sensitive transactions, and even bypass perimeters designed to stop web attacks against internal users.

Prototype Hijacking

Prototype is an open source JavaScript library designed to help make developing dynamic web applications a whole lot easier. One of the greatest strengths of prototype is the ability to override any function, allowing for an extensible set of frameworks to be written, distributed and customized. This feature is also the greatest weakness of prototype. It is trivial once code has been injected into a working user's session to override the getter and setter functions. This would allow a JavaScript sniffer to be created which would remain invisible to the user. The example from Di Paola shows the simplicity of this attack using the XMLHttpRequest.prototype object:

```
XMLHttpRequest.prototype.__defineSetter__(

/* code from the Di Paola example */
     "multipart",function (h){ // Hijacked multipart
     this.xml.multipart=h

       /***************************/

       /****** adding a sniffer ***/
     sniff("multipart: "+" "+h);

       /***************************/
     return h;
});
```

This attack technique can be used to maliciously modify or inject code to sniff out bank transactions and various other sensitive web-based transactions totally transparent to the user. In this example, the user sees a dialog box that a bank transfer is about to happen and the bank further notifies the customer via SMS for every bank transfer accomplished by an authenticated user. If this AJAX code is injected with JavaScript, both the request to transfer and receipt of the transaction will be forwarded to both the attacker and the legitimate user.

This level of attack makes Cross Site Scripting (XSS) attacks incredibly powerful when leveraged properly. The nature of the injected XSS attack allows the attacker to wait for the user to properly authenticate and bypass even two factor authentication. This attack is also interesting in that it does not depend on a particular framework.

SQL Injection

Now let us turn to another example; the classic SQL injection attack. In this example we will show pseudo code which is not protecting against SQL injection. However because it is using AJAX, the injection vector would not be found by traditional scanning methods. It is important to note that when you are testing injection points, you have to deliver your attack payload deep into the application logic if you want to really test for SQL injection and XSS. Most web applications have request processing code which performs what we can call "trivial validation" on requests before doing anything interesting. Take the following Java servlet code for example:

```
/* Not protected against SQL injection */
public void doGet(HttpServletRequest req, HttpServletReseponse resp)
        throws ServletException, IOException
{
        String email = req.getParameter("email");
        if (!isValidEmail(email))
                throw new ServletException("invalid email address");
        String fname = req.getParameter("fname");
        String lname = req.getParameter("lname");
        String sql = "INSERT INTO users VALUES ( '" + fname + "', '" +
                lname + "', '" + email + "')";
        executeSql(sql);
}
```

In order to find AJAX related vulnerabilities, the most important first step is to identify the AJAX endpoints on the server. These are the URLs which, when requested, will perform the AJAX functionality and respond with XML or JSON for example. For example:

```
<script type="text/javascript">

/* This is vulnerable */
       var req = new XMLHttpRequest();
       var url = '/ajax/' + 'list_users.php?' + document.location.search;
```

```
        req.open("POST", url, true);
</script>
```

Now let us assume that the automated scanner finds the form values and attempts to inject via the first name parameter:

```
email=testvalue&fname='&lname=test
```

Automated attacks often do not understand how to forge authentic looking data values. In this case the code above would fail to produce a valid email address and not detect the true injection vectors. You have to not only test each injection point; you also have to construct requests that look somewhat valid. It is easy to see why many automated attempts would throw data at the endpoint and they keep getting rejected by the back end before any SQL code even gets executed. However developers typically do not spend as much time securing the AJAX endpoints on the server (especially from XSS) because these endpoints are regarded as the "plumbing" of the web application. This makes them a good target for attackers and manual investigations would make short work of these injection points.

Conclusion

New attack vectors are opening in the web application space due to architectural changes brought about by technologies like AJAX. The rush to implement these new features has resulted in a lot of insecure code and improper defensive practices. Even companies who employ traditional vulnerability scanners may be at risk due to the difficulty in finding flaws in an automated fashion. Specialized web application scanners do exist but they may not be purchased as regularly as traditional vulnerability scanners. While there is movement towards unified scanners, the current gap in capabilities will leave a lot of web applications unprotected. Developers should move towards **Security Lifecycle Development (SDL)** like models to look for flaws before the code is moved to production. Vulnerability scanning (both network and web based) should be conducted regularly during the development process and consider every input vector as a potential source of attack.