FlexFM

Best Practices for Authenticating FileMaker® / Adobe® Flex™ Apps Using a PHP Proxy

Contents

FlexFM

Abstract

FMProxy is part of FlexFM,
an Open Source Project
Sponsored by Soliant Consulting

Web applications come with their own special set of security challenges. Any good implementation takes these challenges into consideration and accounts for the associated risks.

Responsible handling of user names and passwords by Web application developers is probably less common than it should be. In this paper we'll review some Web security fundamentals, and look at some advantages of implementing a proxy server. The proxy example we will look at is built for FileMaker® Server and is tailored for Adobe® Flex™ applications, although the principles can easily be applied to any Web client/server technology pairing.

The FileMaker suite of products has been a cornerstone of Soliant's custom software consulting practice. To further extend the capabilities of FileMaker's products, we have been working on ways to implement Adobe Flex as a FileMaker Custom Web Publishing client technology. In planning our approach to this endeavor, we decided to start with a light-weight implementation that allows Flex apps to connect directly to the FileMaker Web Publishing Engine[1], which is a REST-like RPC[2] implementation. Thus, our FlexFM package is able to parse the Web Publishing Engine's XML output (fmresultset.xml) and can be used to make a direct connection to FileMaker Server.

We'll look at some of the practical limitations with a direct Flex/FileMaker connection, and when it makes sense to implement a proxy to overcome these limitations. PHP is a natural choice for building a proxy because FileMaker Server provides built-in support for PHP[3].

SOLIANT CONSULTING®

1 http://www.filemaker.com/support/technologies/xml.html
2 http://en.wikipedia.org/wiki/Representational_State_Transfer#REST_versus_RPC
3 http://www.filemaker.com/support/technologies/php.html

The Security Landscape

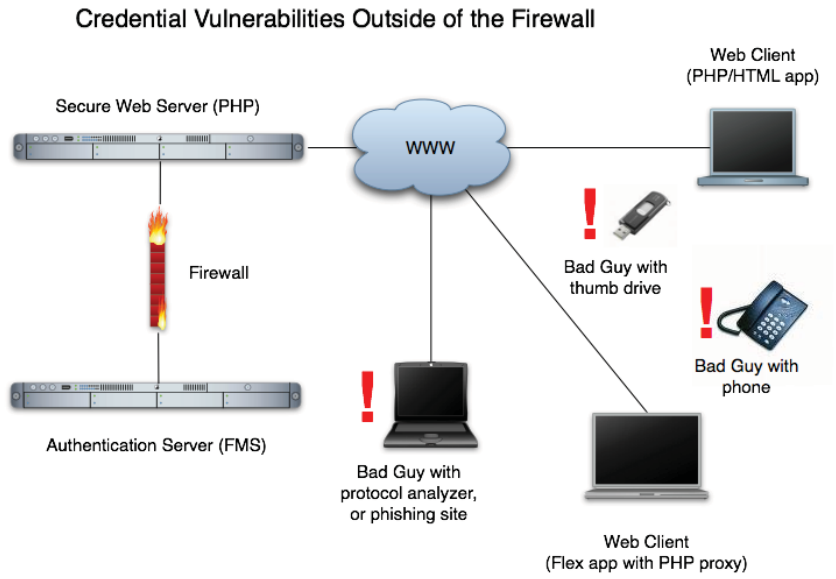
When engineering a Flex or HTML application to interact with the FileMaker Custom Web Publishing API, you need to provide a secure means by which a client computer can connect to a Web server over the public Internet, and you must be able to verify the identity of the user before granting access to create, read, update, and delete information in your database.

A Web application must be capable of securely handling user credentials. It is very important that developers resist the temptation to write custom encryption schemes or, worse yet, simply store user credentials in a clear text table. This is summed up well in a post by Tom Moertel on his blog where he warns, "If you are storing passwords in a database, you are almost certainly making a mistake."[4]

The FileMaker suite of products offers two built-in ways to authenticate users. The first is to authenticate users against external domain or local host accounts, absolving you (as application architect) of the responsibility for storing user credentials, and ensuring that they are as secure as your enterprise domain security.

The second involves the built-in FileMaker user management capability that automatically manages an industry-standard salted hash of user passwords for you. According to FileMaker security expert Steven H. Blackwell[5], FileMaker's key derivation function (PBKDF2) is based on a RSA Public-Key Cryptography Standard (PKCS #5)[6]. As summarized in Wikipedia, "PBKDF2 applies a pseudorandom function, such as a cryptographic hash, cipher, or HMAC to the input password or passphrase along with a salt value and repeats the process many times [...] to produce a derived key, which can then be used as a cryptographic key in subsequent operations. The added computational work makes password cracking much more difficult, and is known as key strengthening." This level of encryption means that even for a database administrator with developer access, there is no practical way to retrieve user passwords. If an attacker manages to gain physical access to a FileMaker file, it is conceivable he could hack the file and replace password hashes for existing accounts with new hashes known to the attacker, but decrypting the stored hashes to reveal the clear text of the passwords is beyond practical means.[7]

In other words, even if your server is physically breached, or backup files end up in the wrong hands, user passwords stored in



Even with well-secured servers, it is crucial to guard against interception of Web traffic and to avoid leaving anything of use to an attacker in case of physical attack on client machines. It is nearly impossible to create technological measures to protect users from frauds like phishing and social engineering, but these are also very real vulnerabilities to credential security, and where this kind of attack is deemed to be a threat, measures such as user training and appropriate Human-computer Interface design should be employed to mitigate them.

4 http://blog.moertel.com/articles/2006/12/15/never-store-passwords-in-a-database
5 http://www.filemakersecurity.com
6 http://www.rsa.com/rsalabs/node.asp?id=2127
7 http://www.google.com/search?q=crack+filemaker

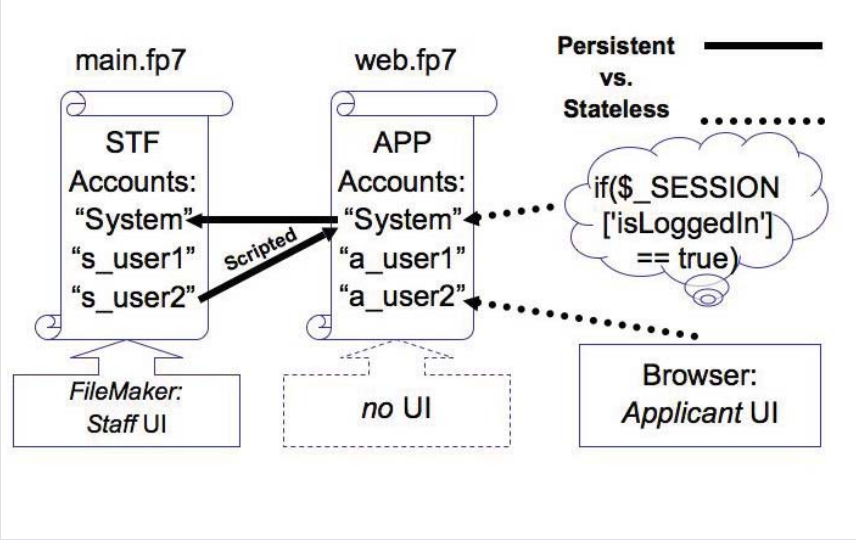
FileMaker's internal user hash are not as vulnerable as they would be if stored in clear text. FileMaker's built-in user management thus absolves the application developer of responsibility for encryption and security of user passwords. This is not to say that such a breach would be fruitless for the attacker, as they might gain access to data by cracking a file if they could breach server or backup security. But while this is an important subject, the focus of this paper is on the secure handling of user credentials, not of data.

A cornerstone of the case made here is that you should not try to roll your own user management system; rather, leverage what FileMaker provides. As an application architect, spend your security dollar on securely handling credentials during the authentication exchange (and the processes surrounding password creation and recovery, should that become necessary). See the sidebar on this page for a brief discussion about options for managing FileMaker user accounts. It's unlikely that you can or need to write a more secure user management system than what FileMaker has provided, which you can leverage with relatively little effort. The same advice applies regardless of what backend is being used for a web application.

Given that FileMaker provides a built-in secure means of storing account credentials plus an API that defines the means by which these credentials may be submitted, and subsequently by which the data may be fetched and manipulated via a Web server, we will assume for the purpose of this paper that by taking advantage of this, everything inside the firewall is secured sufficiently. There are still several distinct programming challenges with regard to handling user credentials outside the firewall. The primary security focus of this paper is on proper handling of credentials in this vulnerable window between the time the user types into a login form and when the Web server receives login information.

In August of 2006 we created a demo app that demonstrates a self-proxying PHP/FileMaker implementation. Web users may each create a self-serve login account stored in FileMaker's built-in account hash. These user accounts do not have access to the main data file, so the PHP session in combination with stored system credentials is used to give proxy-access to valid users.

You can download the demo files from flexfm.us/source-code/ and inspect the way it has been designed to take advantage of session proxy credentials such that the user accounts are only valid in the web file, yet valid users may access data from the staff file. Look for the link under Sample applications.



This is of particular importance if your application is using FileMaker's external authentication with domain accounts, because in this case, interception of the web application credentials by definition reveals user domain credentials. Even when not using domain authentication, credential security must be taken seriously. Human nature dictates that users are prone to use the same credentials across multiple applications, and hence, discovery of credentials for one application will often yield access to others with which the user interacts.

In the most basic sense, there are three ways that credentials might be stolen by an attacker outside the firewall:

Interception

Between the client's machine and the Web server, each request that travels on the open Internet might be copied to a log by an attacker or careless administrator. It is important that any stream containing sensitive data, such as a clear text password be encrypted so that it is of no use (for practical purposes) to anyone listening. We'll look at setting up an encrypted connection between client and host, and how to guard against accidental user circumvention.

Breach of Client Machine

If a user walks away from a computer while it is logged in, or otherwise fails to secure a client machine, an attacker can easily

gain access to the HTML cookies or Flash shared objects stored on the machine's hard drive. A well-written Web application will not store any sensitive information, such as a password, in clear text on the client machine. This ensures that even if a machine is compromised, the attacker will not be able to retrieve the user's credentials for the Web application. Of course, there is not much we can do if a client machine becomes severely compromised. For example, a keylogger might be installed. This level of security is beyond the control of a Web application architect.

Social Engineering and Phishing

In his 2002 book, *The Art of Deception,* Kevin Mitnick[8], one of the world's most well-known convicted hackers, describes how he used "social engineering" — persuading individuals to release sensitive information to him — as opposed to technology, to breach the security of the networks he attacked. Phishing[9] is the similar practice of illegally luring people into disclosing sensitive information, such as usernames and passwords, by spoofing a legitimate Web site. Aside from good Human-computer Interaction design that helps users steer clear of social engineering and phishing attacks (which is beyond the scope of this paper), there is very little we can do to prevent them with technology alone.

While server-side code is more resistant to external attacks than client-side code, Steven H. Blackwell, author of *FileMaker Security: The Book,* reminds us, "Insider threats constitute the greatest danger to the Confidentiality, Integrity, and Availability of digital assets. Insider attacks are the hardest to defend against; they cause the most damage; and, they cost the most to remediate. That's why *FileMaker Security: The Book* starts right off in Chapter One talking about the environment where these insiders work." [10] While physical security of servers is beyond the scope of this paper, it is important to remember that trusted individuals can do great mischief from the inside. For example, it would be trivially easy for someone with access to the Web server file system to add a password logger or some other exploit, and it is also important to remember that vulnerabilities can be introduced by error, lack of understanding, or lack of management mandate. Make sure to consider how important password security is in your application, and carefully audit your internal vulnerabilities accordingly.

In the following sections we'll look more closely at fundamental security considerations, such as enforcing SSL for sensitive connections, and the risks inherent in HTML cookies and Flash shared objects. We'll consider the inherent problems with basic access HTML authentication, then we'll look at PHP sessions, differences in security considerations for programming in PHP vs Flex (server-side script interpreter vs client-side compiled binaries), and how to set up a PHP proxy for handling Flex authentication.

8 http://en.wikipedia.org/wiki/Kevin_Mitnick
9 http://en.wikipedia.org/wiki/Phishing
10 http://www.fmpug.com/members_download.php?filename=credentialexposureincwp.pdf&free=true

Preparing for a Secure Web App

We've already touched on some of the inherent security challenges one faces when designing a public facing Web login system for any application. These are the guiding principles we use when designing login systems for FileMaker or any other back-end system.

1. *Don't "roll your own" authentication scheme; instead leverage FileMaker's.*
Many developers and security experts concur that building a custom password system is a bad idea.

> "If you are storing passwords in a database, you are almost certainly making a mistake."
>
> "Do you think you have a good reason for storing passwords in your database? If so, you're probably wrong."
>
> *- Tom Moertel[11], software developer*

> "No, really. Use someone else's password system. Don't build your own."
>
> "Most of the industry's worst security problems (like the famously bad LANMAN hash) happened because smart developers approached security code the same way they did the rest of their code. The difference between security code and application code is, when application code fails, you find out right away. When security code fails, you find out 4 years from now, when a DVD with all your customer's credit card and CVV2 information starts circulating in Estonia."
>
> *- Thomas Ptacek[12], Principal at Matasano Security*

2. *Never store or embed credentials, even "temporarily" on the client side, e.g. cookies.*
Don't store passwords in anything that is cached (cookies, Flash shared objects, or client-side code)

It is a serious security risk to store credentials in HTTP cookies[13], Flash shared objects[14], or in any code that will be delivered to the client machine, such as a compiled Flex app. Any SWF file can easily be decompiled, and therefore, any stored credentials inside the code could easily be discovered[15].

> "Storing authentication credentials in cookies is not a good idea, as cookies can be stolen through cross-site scripting attacks or local access to the hard drive. Once cookies have been stolen, an attacker can gain access to the vulnerable site and masquerade as a legitimate user. This vulnerability is enhanced when authentication credentials are stored in clear text. In this situation, the username and password can be obtained merely by viewing the cookie contents."
>
> *- iDefense Labs PUBLIC ADVISORY: 06.10.02, Michael Sutton[16]*

---

11 http://blog.moertel.com/articles/2006/12/15/never-store-passwords-in-a-database
12 http://www.matasano.com/log/958/enough-with-the-rainbow-tables-what-you-need-to-know-about-secure-password-schemes/
13 http://en.wikipedia.org/wiki/HTTP_cookie
14 http://en.wikipedia.org/wiki/Local_Shared_Object
15 http://www.google.com/search?q=flash+decompiler
16 http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=61

3. *Never transmit credentials or sensitive information in the clear; enforce SSL at the Web server level.*

There is a lot of available documentation about SSL enforcement. It's not that hard to implement. The first thing you will need to do is get a certificate (SSL will work fine without it, but a third party certificate keeps browsers from displaying a certificate warning to the user). Next you, or your Web server admin, will have to configure SSL on your Web server and install the certificate. Finally, you will need to create a rule or rules that rewrite any non-secure requests for URLs that will be transacting secure information as SSL. This is so that users who type the URL manually without explicitly specifying HTTPS as the protocol will have a seamless experience as their request is gracefully rewritten for the secure port on your server.

Note that it is particularly important to enforce SSL-only for your login page or pages, but you might consider it for your whole Web app. In some ways it is easier to make a blanket rule that all traffic on your Web server will be redirected to the secure port. There is a general assumption that this will negatively impact throughput, as all data is being encrypted and decrypted. But there is a popular study done at NYU in 1998 that shows there is very little penalty for using encryption[17]. The study concludes: "We find that secure Web servers perform well in comparison to non-secure servers. In particular, measurements show that on typical PCs encrypted Web communications using SSL and RC4 can transfer data at speeds similar to non-encrypted HTTP. This bodes well for electronic commerce." In other words, there's probably very little reason NOT to use SSL, although you must assess each case on its own merits.

Depending on what type of Web server you are using, there are various options for converting requests on the default, non-secure port to your secure one. You might employ Apache's mod_rewrite or ISAPI_Rewrite for IIS.

You generally will be adding a rule or rules so that either the whole site or certain URLs trigger the rewrite. In Apache's httpd.conf file, a redirect that effects everything would look something like this:

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
```

There is plenty on the Web to help you with your configuration needs. Just keep three things in mind before you start making changes to your Web server config files. Backup, backup, and backup.

Suggested search terms:

http://google.com/search?q=ssl+certificate

http://google.com/search?q=redirect+http+to+https

http://google.com/search?q=mod_rewrite

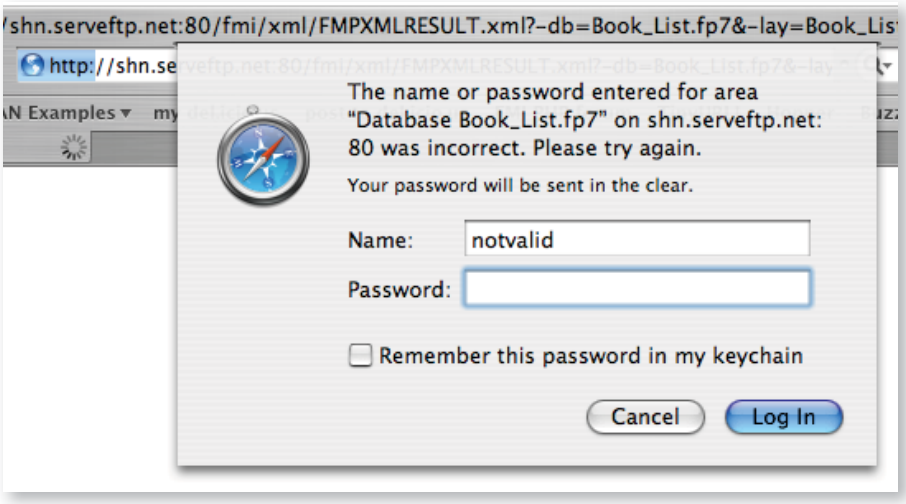
http://google.com/search?q=ISAPI_Rewrite

17 http://www.cs.nyu.edu/artg/research/comparison/comparison.html

Basic Access HTTP Authentication

FileMaker's Web Publishing Engine (WPE) uses basic access authentication to validate HTTP request. For our purposes, this means that Flex applications cannot effectively connect directly to a FileMaker WPE, unless it is configured to allow anonymous access (hence requiring no authentication). This is not because it is impossible to do so, but because it is very difficult to do so securely and seamlessly. Let's look at why.

In the Wikipedia entry for basic access authentication[18], there is a summary of advantages and disadvantages. Among the advantages, the article points out that basic access authentication "is supported by all popular Web browsers." There is a caveat to this, however[19]; in short this is not completely true of Internet Explorer. With the combination of security restrictions that the Adobe® Flash® Player framework has imposed, coupled with the limited IE implementation of basic access authentication, it is not possible to authenticate directly to a FileMaker WPE with a Flex application running inside IE.



Furthermore, even inside of browsers that still support the single built-in means of passing basic access credentials from Flex, if the server responds to the authentication request with "Unauthorized" (status-code 401), there is no way to prevent the browser from presenting the user with its own login dialog superimposed your Flex application (see picture).

One other important disadvantage Wikipedia mentions is, "existing browsers retain authentication information indefinitely. HTTP does not provide a method for a server to direct clients to discard these cached credentials. This is a significant defect that requires further extensions to HTTP." This means that apart from quitting the browser, there is no way to disconnect a browser agent connected directly with basic access authentication.

18 http://en.wikipedia.org/wiki/Basic_access_authentication

19 The current version of Internet Explorer will not pass through credentials as part of the URL. FireFox and Safari both continue to support this technique, despite the risk that is poses to credential security if misused.

FlexFM

Clearly these add up to a series of challenges which call for a different technique in most cases. We'll get into that shortly, but first a bit of background.

When Soliant originally built the "simple" classes in the FlexFM package[20] for connecting Flex applications directly to FileMaker's WPE, we implemented the request part of the HTTP transaction via POST, the same way most PHP classes do it, namely, by inserting base64 encoded credentials into the HTTP header. This is what that can look like in a PHP class:

```
if (!(empty($this->_username) && empty($this->_password)) ) {
        $post .= "Authorization: Basic ".
            base64_encode($this->_username.":".$this->_password)."\r\n";
    }
```

This is how that used to look (note the past tense) in FlexFM's ActionScript:

```
    public function basicAuthentication(username:String,password:String):void {
        var encoder:Base64Encoder = new Base64Encoder();
        encoder.encode(username + ":" + password);
        var value:String = encoder.drain();
        this.addHeader("Authorization","Basic " + value);
    }
```

However, as of this writing, this is what how code now looks in the FlexFM sources:

```
    // Flash Player 9 took away our ability to modify the Authorization header here,
    // so we have created a PHP proxy.
    /*
    public function basicAuthentication(username:String,password:String):void {
        var encoder:Base64Encoder = new Base64Encoder();
        encoder.encode(username + ":" + password);
        var value:String = encoder.drain();
        this.addHeader("Authorization","Basic " + value);
    }
    */
```

As you may infer from the fact that the second version of the FlexFM code is completely commented out, our original light-weight approach of connecting Flex apps directly to FileMaker hosts was disabled by the Flash Player team[21], most likely to "protect us from ourselves". In any case, since there is no longer any built-in way to modify the Authorization header of a POST request, we discussed rewriting the HTTPService class in ActionScript as Abdul Qabiz has[22]. The trouble is, we assumed there was a risk that future changes to the Flash Player platform might make this fruitless; and furthermore, we saw some advantages to handing off the entire payload to a server-side application and re-POSTing it from there. A proxy affords advantages in terms of both security and performance optimizations on the server side.

The solution we settled on was to build a simple but very flexible PHP class called FMProxy that understands FlexFM's WPE

20 http://flexfm.us
21 https://bugs.adobe.com/jira/browse/SDK-14481
22 http://www.abdulqabiz.com/blog/archives/flash_and_actionscript/http_authentica.php

requests. Then we added a proxy property to FlexFM's FMServer class. The PHP proxy can simply receive FlexFM's form-based credentials and add them to the header on the server side before passing the request through to the WPE, but as already mentioned, it also presents some powerful additional possibilities which we'll discuss in detail later.

FlexFM can still connect directly to a specified FileMaker host without a proxy, and if you provide credentials, it will attempt to transmit them to the host by adding them to the URL. This is not recommended for several reasons as mentioned in the previous section, such as, it simply won't work with credentials on Internet Explorer and if credentials passed as part of the URL are not valid, there is no way to prevent a 401 response from the host from being presented on top of your Flex app. Using FlexFM without FMProxy remains an option, though, for those cases in which it is useful. FMProxy is covered in more detail later.

Where is your code executed?

Flex apps are .swf binaries which are downloaded and then executed in Flash Player on the client side. As mentioned earlier, this means anyone can decompile a SWF and examine its sources in clear text. So as previously mentioned in regard to cookies and Flash shared objects, we wouldn't want to store credentials in the source code for a bad guy to find. You should embed in the Flex source only those things that pose no security threat if discovered by an attacker. This is known as "security by design" and stands in contrast to "security by obscurity." A system relying on security by obscurity may have theoretical or actual security vulnerabilities, but its designers believe that the flaws are not likely to be discovered, and therefore attackers are unlikely to exploit them. Security by design usually means that everyone is allowed to know and understand the design, *because it is secure*. In real world applications, it is likely that elements of both strategies will be employed, but particularly in parts of the code in which it is trivially easy for anonymous users to explore. It would be irrational to imagine that the code is truly obscure. It is critical that security by design strategies be employed for client-side code.

The main difference between PHP and ActionScript has nothing to do with structure and syntax of the script languages themselves. The difference lies in where the code is executed. While it is possible for PHP to be run from a command line interface or in standalone graphical applications, it is most common for PHP to be deployed on a Web server. When PHP scripts are requested via the Web server, they are run through a parser and return only the results. As described in Wikipedia, "PHP primarily acts as a filter, taking input from a file or stream containing text and/or PHP instructions and outputs another stream of data."[23]

The fact that a properly configured Web server running PHP serves as a filter is very important. Since the PHP code itself is never sent to the client for execution, unlike in a Flex app, it is possible to define constants which cannot be decompiled by an attacker, unless the attacker manages to breach the server in such a way as to be able to retrieve the PHP files without them being filtered by the parser. Generally speaking, this is far more difficult than simply gaining a user's credentials in some manner (limited to social engineering or some other fraud, if you have designed your Web application securely). This being the case, it is arguably advisable to restrict the user login account privileges to the bare minimum needed to prove that the user is authorized to use the system but not access data. This is commonly done in sophisticated PHP applications by self-proxying an authenticated user session.

After the user has established an authenticated session, subsequent requests from that valid session can be made using stored system credentials that are defined to provide the elevated level of access an authenticated user needs to interact with the system. When this technique is employed, if someone trying to hack the application finds a way in other than the application interface (for example, via a FileMaker Pro client or via direct WPE queries), the only data the hacker might be able to access using stolen user credentials are user-specific preferences which the login account is permitted to read and write during the login transaction. Direct queries into tables the user would ordinarily interact with when logged into the application will fail because the user account is not directly authorized to access those, only the proxy system account is.

23 http://en.wikipedia.org/wiki/PHP

Handling credentials during a user session

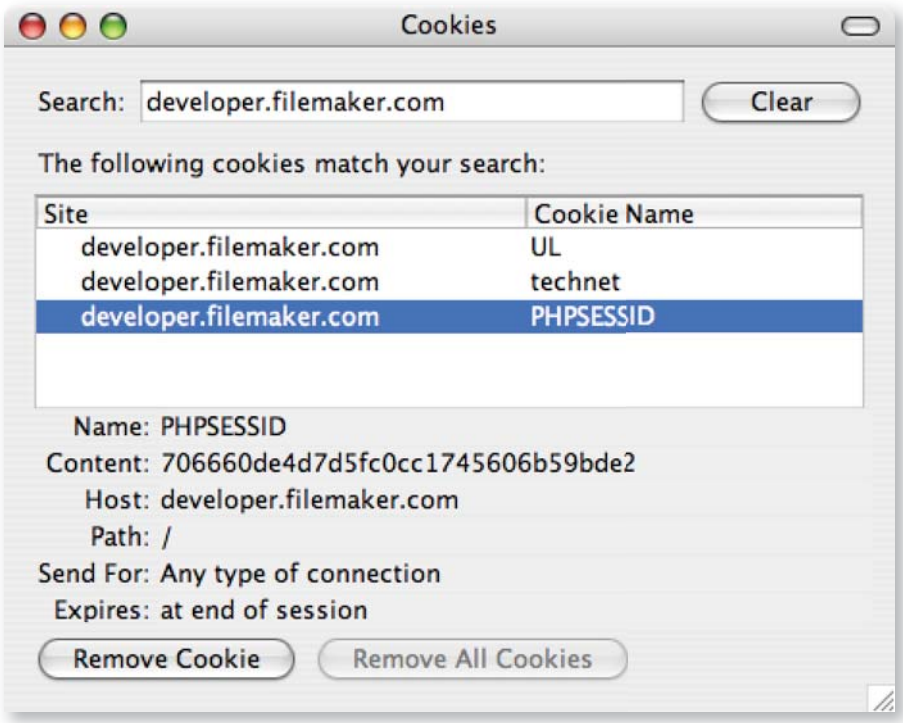
Like other client-side technologies, Flex permits objects to be instantiated, hold strings (and other data types) in memory for the duration of the application session, which are then destroyed when the session is ended. There is no technical reason that we couldn't take advantage of this fact in order to hold the user's login credentials for the duration of the session and transmit login credentials as part of every HTTP transaction.

Assuming that the transactions are all conducted over SSL, this strategy doesn't present an inherent threat; but, doing so means that you are continually passing the user credentials with every transaction, which leaves open the possibility that an error or bug may expose the credentials. Furthermore, user accounts would need to be granted enough access to perform whatever actions that may be needed in the course of an authenticated user working with the application. As such, if some user credentials were discovered or guessed by a bad guy, those credentials might be of some use to him in looking for ways to circumvent the Web application and make direct calls into the database, perhaps finding an unforeseen vulnerability.

As alluded to earlier, we'll look at a strategy, in which the application architect allows the user account to serve only as a means of proving that the person logging in is permitted to interact with your Web application in the prescribed manner, but is of no use to an attacker who may try to use the credentials for direct access to the data tables. Limiting the user credentials to proving they are who they claim to be, but with access restricted to stored account preferences, means that a great number of possible direct attacks are eliminated, as opposed to a design that uses the same credentials to interact with the database. Using proxy system credentials reduces the chance of accidental mishandling of user credentials and reduces the usefulness of those credentials should they be obtained by an attacker.

PHP Sessions

PHP implements a special kind of global variable in the server's memory space called `$_SESSION`. When a session is initially created, it is assigned a unique session ID by the host. Session variables may be set and retrieved only by scripts running in the context of that host when called by a connection which provides the session ID. The session ID is usually stored in a cookie but will be propagated in the URL if cookies are disabled. A script running on the server that looks for or tries to set a value in `$_SESSION` will be able to access anything that may exist under the session ID it provides for that host. Any page (PHP script) on that same host has access to those values associated with the session ID. This overcomes the stateless nature of Web pages by allowing distinct calls to the Web server to cohesively share variable data in a secure way. Since the host holds the values, it does not have to trust the client with them. The client will know the session ID, but if the scripts don't need to return the values in the session variables, those remain completely hidden. From the user side, it would be impossible to even know what the variable names are, much less the contents of them, unless of course the PHP application reveals something by design.[24]



As suggested in the previous section, PHP sessions provide an application architect with a means of only handling user credentials during the authentication transaction and then dropping them like a hot potato. Rather than holding the credentials in memory on the client side, or even in PHP session variable (memory on the server side), and resubmitting them each time there is a database transaction, it is easy to create a login process that receives the username and password from an encrypted form submission. When the login script receives credentials from the submitting form, it will in turn attempt to log into FileMaker's WPE with them. If the response is affirmative, the login script next requests a brand new session ID from the server, and sets a session variable (such as `$_SESSION['loggedin']`) which any other script in the application will require to be true before considering any other instructions. The application developer might also consider loading some user preferences and the user name into session variables (e.g. `$_SESSION['username']`) based on the affirmative response.

What this means is that from the point that the authentication transaction is completed affirmatively, the application can trust that session and act on behalf of that user, and it can do so without needing that user's literal account credentials, which were simply discarded at the completion of the authentication transaction. For each database interaction, PHP can present FileMaker Server with system credentials that have been stored as constants in a config file.

Note: Since the FileMaker will mark record the creation and modification account as "System", for example, creation and modification events triggered by the logged-in user can be stored by PHP in FileMaker fields as record attributes.

While there are a few important precautions to consider when employing PHP sessions, it is possible to use them in a secure manner. The two fundamental session vulnerabilities are known as "session fixation" and "session hijacking". Session security is a

24 http://www.w3schools.com/PHP/php_sessions.asp

sophisticated topic, and there are many good resources available for understanding how to keep sessions secure, so we'll just summarize the concepts briefly.

Fixation is the simplest method of fraudulently obtaining a valid session identifier by tricking a user into visiting the target server from a rigged third party Web server. While it's not very difficult to defend against, if your session mechanism consists of nothing more than `session_start()`, you are vulnerable. The attacker can attempt to dictate the victim's session ID by appending a name value pair to the URL with a session value determined by the attacker. If the session fixation is successful, the attacker then knows the victim's session ID for the target domain, since the attacker himself set it. When the victim authenticates to the target application, the attacker hijacks that session and interacts with the target application just as if the attacker were sitting at the victim's computer. To defend against fixation, it is important to regenerate the session identifier whenever there is any change in privilege level (for example, after verifying a username and password). Doing so will practically eliminate the risk of a successful session fixation attack. Other kinds of hijacking exploits are more difficult to construct and more difficult to defend against. Make sure you study and understand the risks. More information can be found in many places. Two excellent places to start are the PHP Security Consortium[25] and The PHP Group itself[26].

With the preceding credential-handling guideposts and security considerations in mind, we are ready to look at some implementation examples.

25 http://phpsec.org/projects/guide/4.html
26 http://us.php.net/manual/en/session.security.php

FM Proxy

If it hasn't become apparent by now, the trick to securely authenticating a FileMaker Flex app is to authenticate a proxy PHP app. This principle can be employed regardless of what your presentation technology is. Even if you are serving up pure HTML with PHP, it's still good to think of the authentication transaction as a one-time event which validates the user session, allowing the application to subsequently interact with the database using self-proxying system credentials.

To facilitate this for FlexFM, Soliant has created a PHP class called FMProxy[27]. This class is very simple but very powerful. Strictly speaking, it is agnostic as to what technology addresses it, as it accepts a POST payload. Later we'll look at an HTML form that is constructed to POST payloads in the format FMProxy expects. In its most basic implementation, an FMProxy looks like this:

```
<?php

        require_once('FMProxy.php'); // require the file
        $myProxy = new FMProxyRequest(); // the constructor parses $_POST
        $myProxy->execute(); // re-submit the request to the FileMaker host
        echo $myProxy->result; // echo the results to the requesting agent
?>
```

A proxy as simple as this does not employ sessions. In fact, it doesn't really do anything except faithfully pass through any request it receives and return the raw reply. Nevertheless, there is a dramatic difference between submitting a form request directly to FileMaker's WPE and a pass-through implementation of FMProxy. In particular, the proxy suppresses basic access authentication from flowing back to the user (see figure 2 on page 6). Even if authentication is successful, the user's browser (or Flash Player) will not become authenticated via basic access. Furthermore, this works with Internet Explorer. A FlexFM app pointed at the above proxy would need to submit credentials each time, but this is a big step in the right direction and is a perfectly valid use in some situations. The FlexFM objects might look something like this:

```
<soliant:FMServer id="myServer"
    host="myfmhost.com"
    proxy="http://mywebhost.com/proxy.php" />
<soliant:FMRequest id="myQuery"
    server="{myServer}"
    db="theDB" layout="webLayout"
    username="{myUsername.text}"
    password="{myPassword.text}"
    result="myQueryHandler();"/>
```

Notice that myQuery is configured to always pass the string in the myUsername and myPassword objects, which we'll imagine are TextInput objects from the login screen. Unless cleared or modified by the application, the clear text values in these TextInput objects will persist in memory on the client side until the SWF is closed. Each time a myQuery is sent, the credentials will be sent in clear text (thus this app should always enforce SSL); however, it is entirely within the control of the application developer to clear them, in effect logging out the user. For example, we could give the users a Log Out button. Furthermore, when the browser window containing the Flex app is closed (as opposed to exiting the whole browser), the user is automatically logged out. As previously mentioned, with basic access authentication directly from the browser, this is not possible.

While this is interesting and very useful, it doesn't begin to tap the potential of the proxy. Here is some pseudo code that begins to show the possibilities:

27 http://flexfm.us/source-code/

```
<?php

session _ start();

if (@$ _ GET['logout']==1){
        unset($ _ SESSION['isLoggedIn']);
        session _ destroy();
        die();
}

require _ once('includes/FMProxy.php');
require _ once('includes/FileMaker.php');
$systemUsername = 'System';
$systemPassword = 'f7d#qKm8';

$myProxy = new FMProxyRequest();

if (@$ _ SESSION['isLoggedIn']=== true ){

        $myProxy->username = $systemUsername;
        $myProxy->password = $systemPassword;
        $myProxy->execute();
        echo $myProxy->result;

}else {

        // If the session is not recognized, we turn the payload into a FileMaker.php request
        // and attempt to use the submitted credentials to pass an authentication challange.

        unset($myProxy->commandArray); // first we destroy any submitted commands
        $myFM = new FileMaker($targetDB,$myProxy->host,$myProxy->username,$myProxy->password);
        $findReq =& $myFM->newFindCommand($targetLay);
        $findReq->addFindCriterion($targetField, '=='.$myProxy->username);
        $result = $findReq->execute();

        if (FileMaker::isError($result)){
```

```
            // Login Failed

            // echo error xml

        } else {

            // Login Success (NOTE: elevating privileges so regenerate the session ID)

            session_regenerate_id();

            $_SESSION['isLoggedIn'] = true;
            $_SESSION['username'] = $record[0]->getField('_ka_username');
            // etc.
            // echo success xml

        }

}

// when using a html form for testing we can add a parameter to tell the proxy

// to echo out some extra debug info

if (@$_GET['verbose']===1){

        echo '<pre>'; print_r($myProxy->commandArray); echo '</pre>';

        echo 'Proxy Command Override: '.$command.'<br/><br/>';

}


// execute the request with any new, overridden or removed commands and echo the result.

$myProxy->execute();

echo $myProxy->result;


?>
```

In this proxy implementation, there are several new features. Perhaps most obvious is the use of the PHP session. By invoking any session that may exist on the client for our host, and then before doing anything else, we allow for active management of the logged-in state of the proxy, testing to see if the proxy is being called with a parameter that indicates a logout request.

When the logout parameter is not present, the next step is to include both FMProxy.php and FileMaker.php so that we'll have access to the classes in them.

For the purpose of this example, we are defining the system credentials as script variables inside the proxy rather than inside a config file. Instantiating FMProxy automatically unpacks the POST payload into the FMProxy object.

Next we test to see if we have an existing logged-in state for the current session. If so, we insert the system credentials into the FMProxy object, execute it, and echo the results. Simple as that.

However, if the session is not logged in, we repack the submitted payload into a completely different FileMaker.php request object and submit a pre-configured authentication challenge instead. This begins to show the significance of this technique.

Proxy Screening

By design, a proxy is intercepting every request, allowing us to read and potentially censor its mail, so to speak. There is no reason to expose the FileMaker WPE outside the firewall at all. As long as the proxy server can connect to it, there is no reason to allow a direct connection from other sources. This allows you to tailor the proxy very pessimistically, allowing only those methods you explicitly need. For example, you might decide that there is never a reason for your proxy to access anything other than the find, new, and edit commands. You don't want to allow WPE queries to execute other commands, particularly findany or findall. You could easily build an array of disallowed commands and strip any of those out of a submitted request.

```
$disallowedWPECommands = array (
    '-dbnames'=>null,
    '-delete'=>null,
    '-dup'=>null,
    '-findall'=>null,
    '-findany'=>null,
    '-findquery'=>null,
    '-layoutnames'=>null,
    '-scriptnames'=>null,
    '-view'=>null);

foreach ($disallowedWPECommands as $name=>$value){
    if (array_key_exists($name, $theCommandVars)){
        unset($theCommandVars[$name]);
    }
}
```

This is an example that illustrates the concept of using screening to strip queries. It could certainly be argued that there are better ways to restrict certain behaviors, such as defining the database privilege set properly in the first place. Furthermore, it is relatively unlikely that someone would be motivated enough to try to hack the proxy in this way (build his or her own POST payload that conforms to the FMProxy signature). To count on the second point would be to fall back on security by obscurity, and in our initial security considerations we set out to rely on security by design, especially for the public facing components of the system. Following a security by design strategy, it follows that where possible we should disarm any methods that are not required and make it as difficult as possible even for an attacker armed with the complete specs for the application.

Let's have a look at a much more practical screening example in which we override queries. Imagine that you are designing a system that will employ the single-challenge session model we proposed earlier. In this design, the user will authenticate with his or her credentials once, and queries will subsequently be done with proxy system credentials. Our requirements dictate that we restrict users to only those records that they created. With proxy screening, we can quite easily accommodate this.

To do so, we would need to know the name of the restricted table and the field which contains the username. During the initial login process, we will have already validated the username and stored it in a session variable. Every record created by the user will be marked with this attribute, and our requirements dictate that records in this table must always have this correlation, regardless of whether a record is being created, edited, deleted, or searched for.

Using simple parsing techniques, rather than stripping commands as in the previous example, we can force additional name-value pairs onto the command string. FMProxy automatically explodes the command string into an array. If we simply add a new element to that array, and then repack that array as a command string, any submitted command that omitted the user name limiter would have it added, and any command that attempted to pass an alternate value would simply be overwritten with the legal one.

```
$theCommandVars['_ka_username']= $_SESSION['username'];
```

These examples merely touch the surface of what can be achieved by using a proxy to enhance security. When designing your next Web app for FileMaker Server, consider employing a proxy and a pessimistic approach. This will put you explicitly in control of the legal methods to address your database and reduce the opportunities of unforeseen exploits.

Examples

As alluded to previously, FMProxyRequest's constructor unpacks whatever $_POST variable the form submits, looking for the specific signature defined in FlexFM. Here is the constructor for FMProxy:

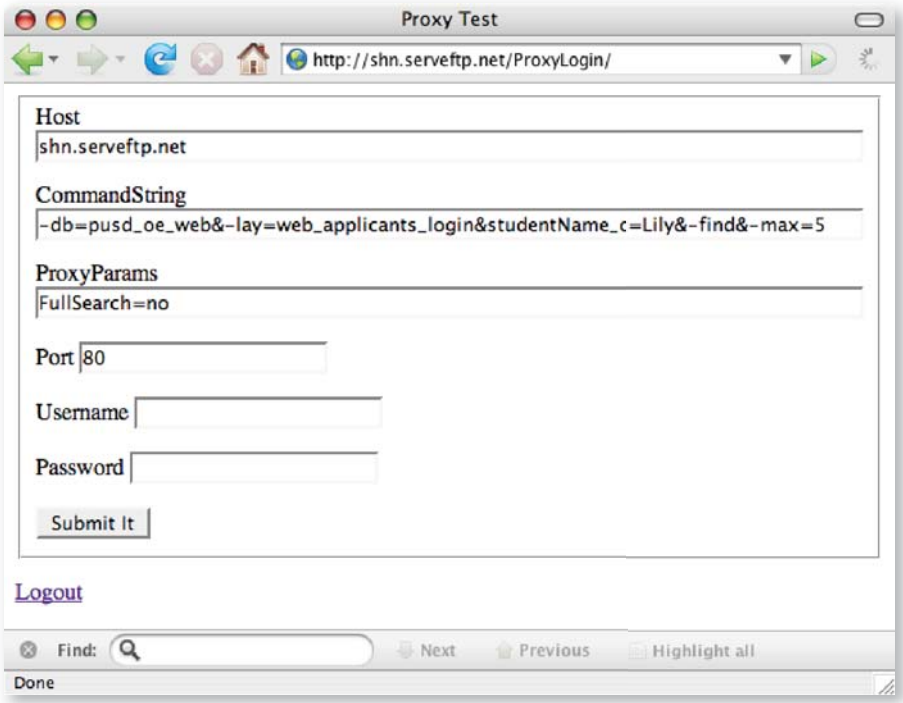
```
/**
 * Constructor
 * @return void
 */
function _ _construct(){
	if (@$_POST['Host']!=null){
		$this->host = $_POST['Host'];
	}
	if (@$_POST['Protocol']!=null){
		$this->protocol = $_POST['Protocol'];
	}
	if (@$_POST['Port']!=null){
		$this->port = $_POST['Port'];
	}
	if (@$_POST['FmiUri']!=null){
		$this->fmiUri = $_POST['FmiUri'];
	}
	if (@$_POST['Username']!=null){
		$this->username = $_POST['Username'];
	}
	if (@$_POST['Password']!=null){
		$this->password = $_POST['Password'];
	}
	if (@$_POST['CommandString']!=null){
		$this->commandString = $_POST['CommandString'];
		$this->commandArray = $this->explodeNameValueString($_POST['CommandString']);
	}
	if (@$_POST['ProxyConfig']!=null){
		$this->proxyConfigArray = $this->explodeNameValueString($_POST['ProxyConfig']);
	}
	if (@$_POST['ProxyParams']!=null){
		$this->proxyParamsArray = $this->explodeNameValueString($_POST['ProxyParams']);
	}
}
```

HTML Form Demo

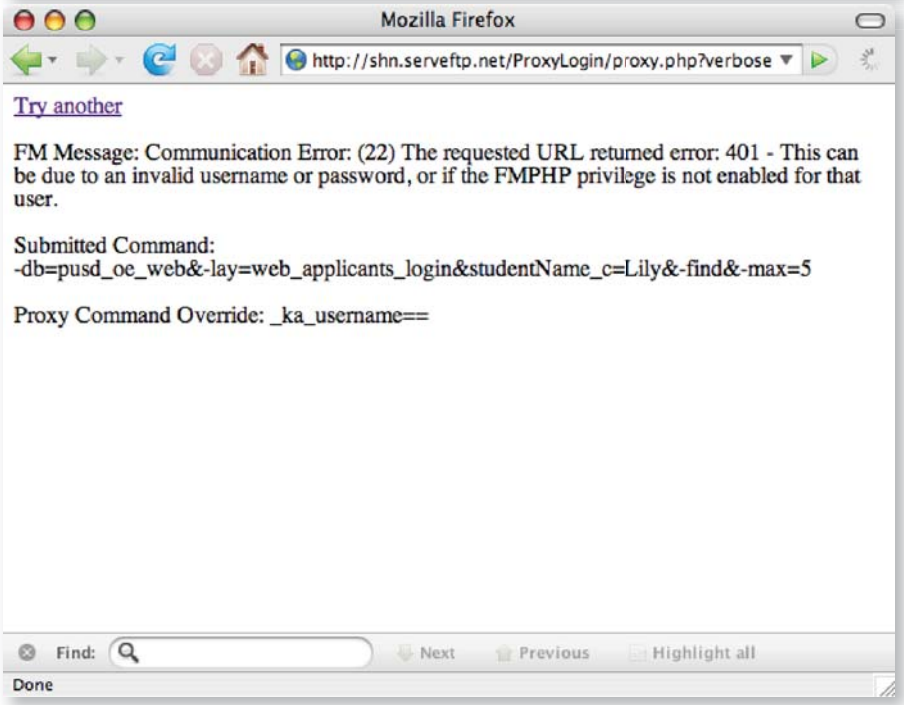
The following example demonstrates a proxy being addressed by a simple HTML form which conforms to the FMProxy signature. The proxy is defined to initially ignore the submitted commands if the session has not been authenticated. Once authenticated, the proxy will look for a specific argument in the optional ProxyParams variable. If it finds FullSearch=yes, it will not override the submitted query with a username limiter. Otherwise it will actively force all commands to include the username as part of the constraint.

1 The default configuration of the form.

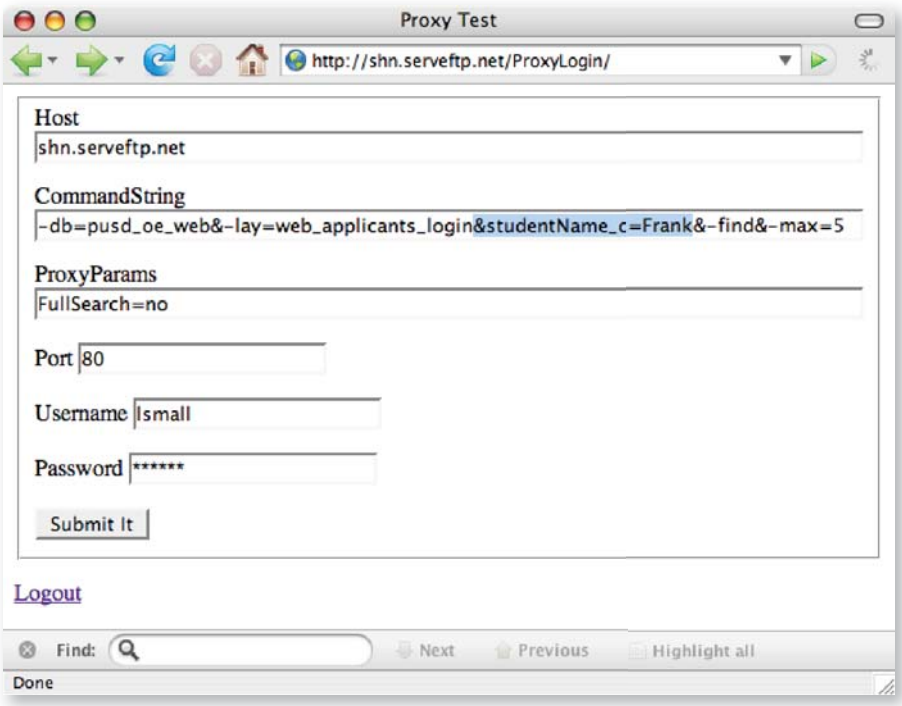
Note: The ProxyParam is included in FlexFM for convenience and flexibility, but it is not advisable to use it as the basis for passing anything other than options which are not tied to security. The ProxyParams are subject to being overridden, as we are doing in this example.



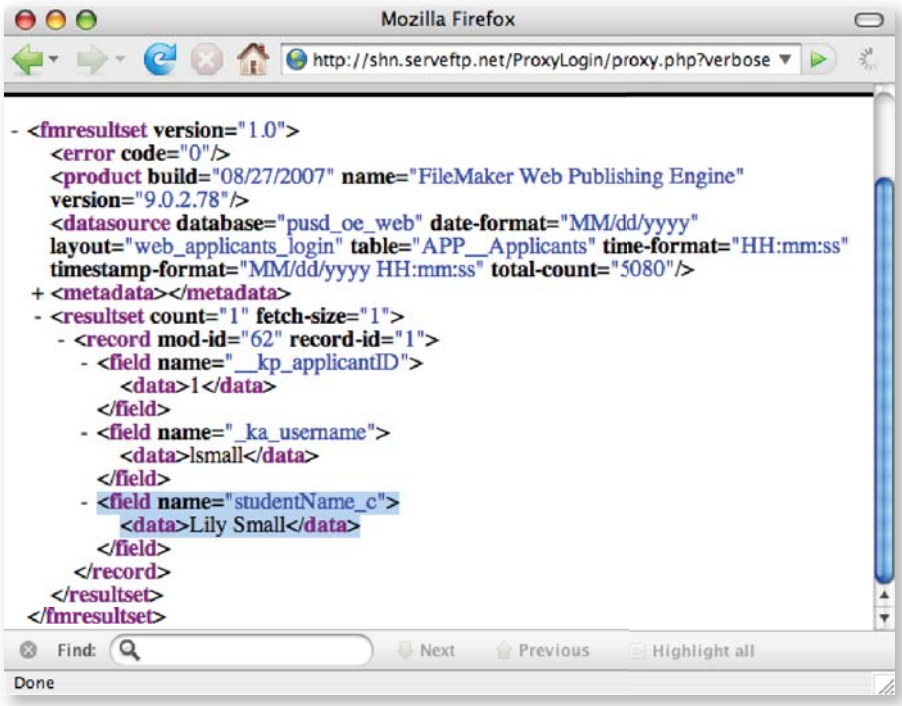
2 After clicking Submit with blank credentials, we get an HTTP 401: Forbidden error (not a FileMaker 401: No records found).



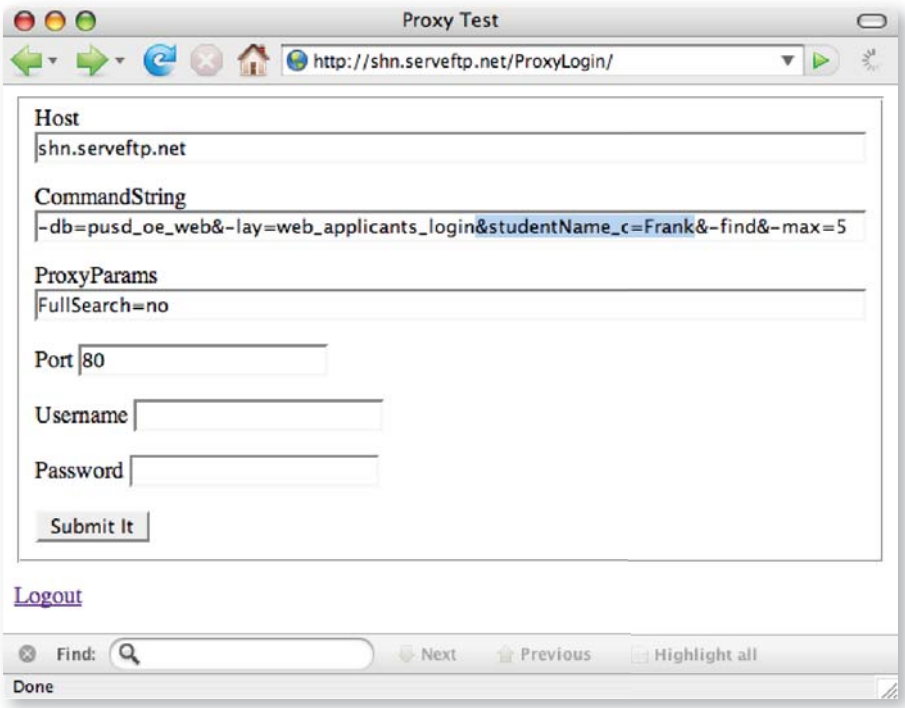
3 Fill in the form with valid credentials for "Lily Small" and a command string which calls for records containing "Frank" in the name



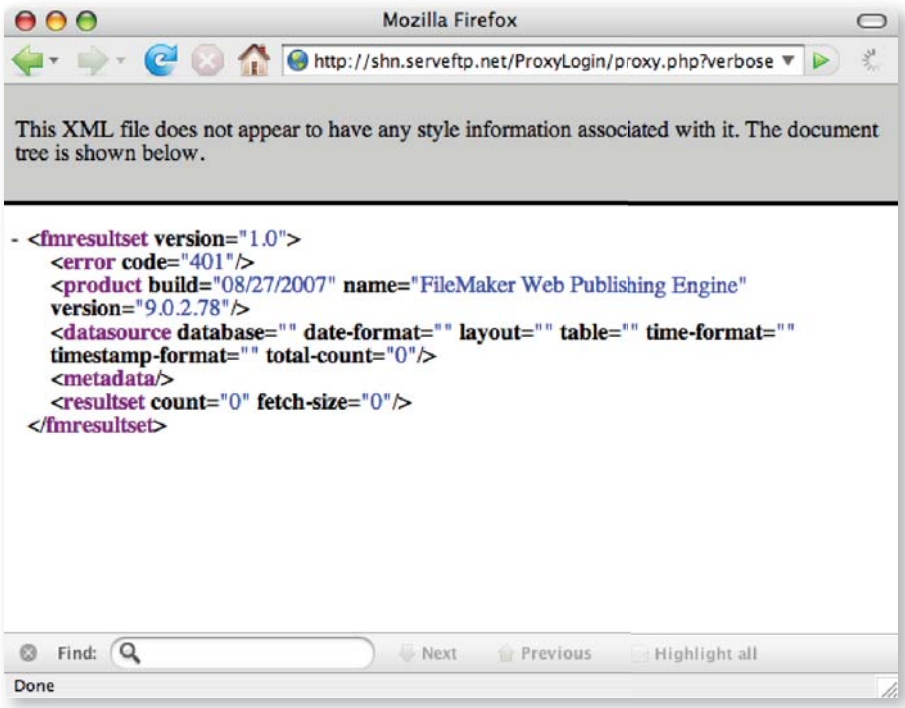
4 Credentials are valid, but since the session was new, the proxy ignored the submitted query and replaced it with a predefined search for the authenticated username.



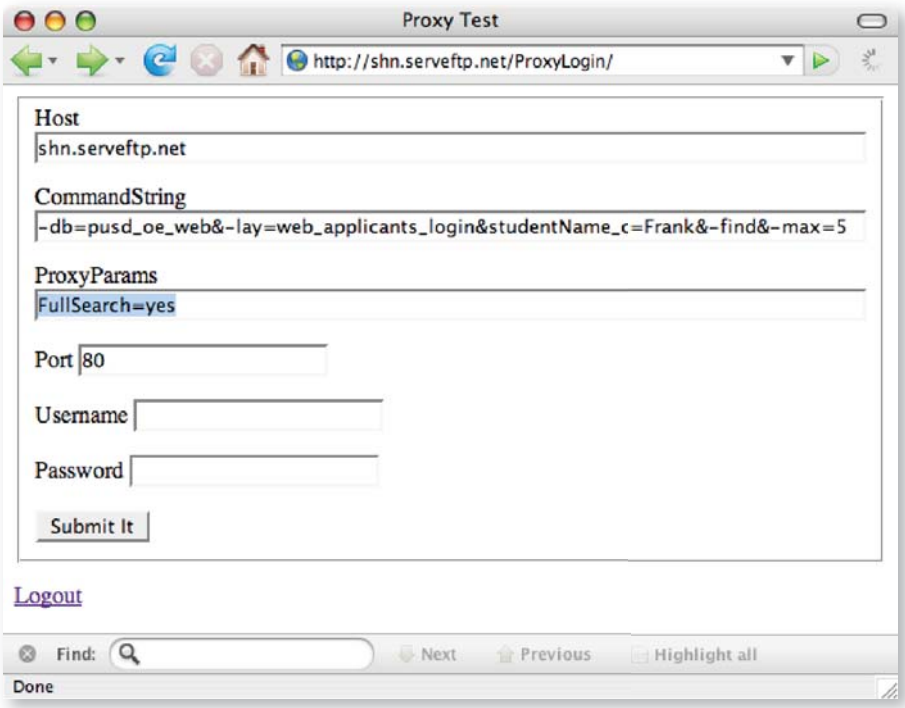
5 Now that we have an authenticated session, we'll go back and resubmit the query for "Frank" but leave the credentials blank. Note the ProxyParam FullSearch=no.



6 Our query is answered, but the FileMaker Web Publishing Engine error code is 401. No records match this query. This is because the proxy's default search is overriding the command string with a username limiter, and there are no records containing "Frank" in the name owned by "lsmall".



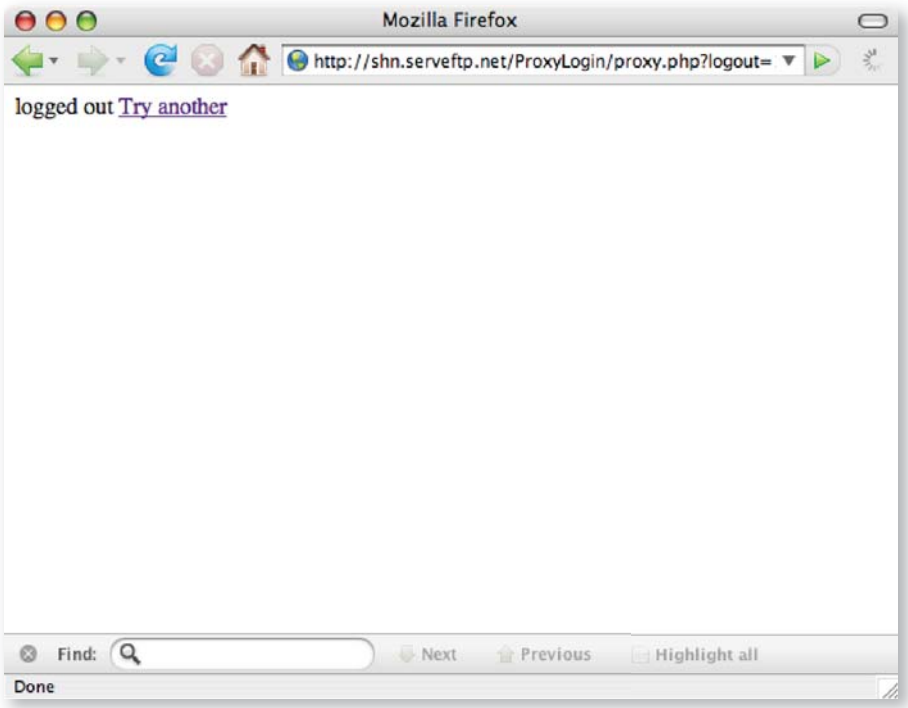
7 Switch the ProxyParam to FullSearch=yes and submit the same query, again with no credentials.



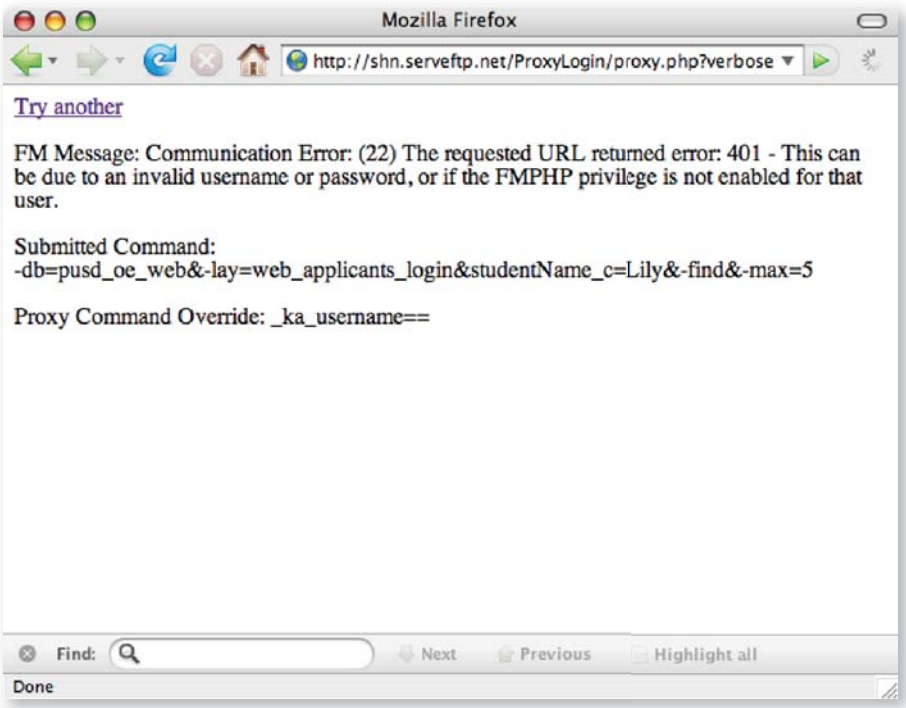
8 The query comes back with 11 records matching "Frank" this time because the search was passed through unaltered.

Remember: The ProxyParam was included in FlexFM for convenience and flexibility, but it is not advisable to use it as the basis for passing anything other than options which are not tied to security. The ProxyParams are subject to being overridden, as we are doing in this example.

9 Finally, let's look at what happens when we tell the proxy to log us out. Clicking the logout link calls the proxy with an extra argument which invalidates the session.



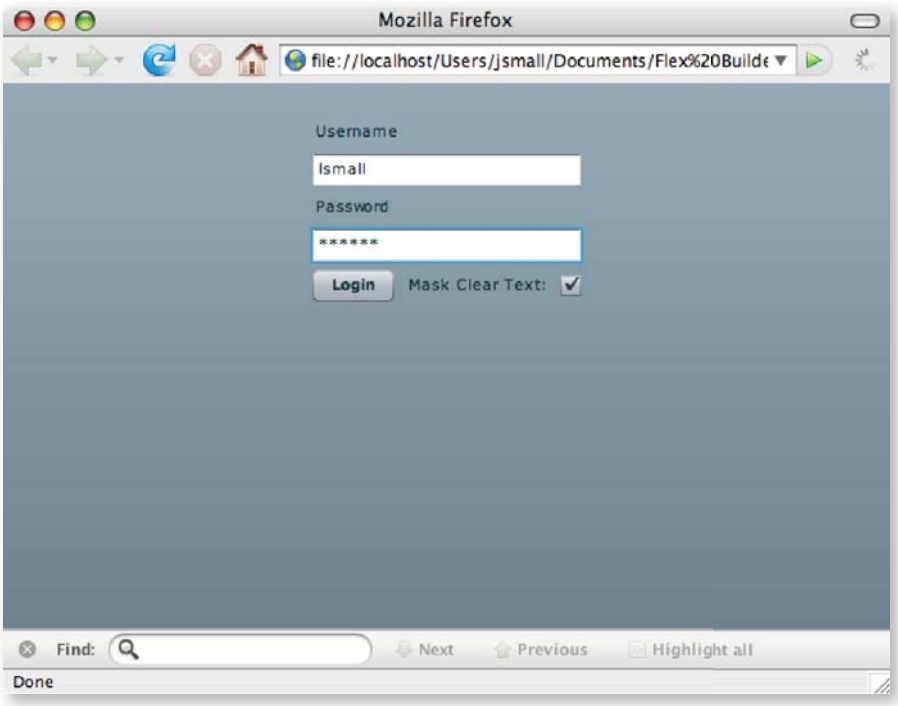
10 Going back to the form and submitting it without credentials results in the HTTP 401 error again.



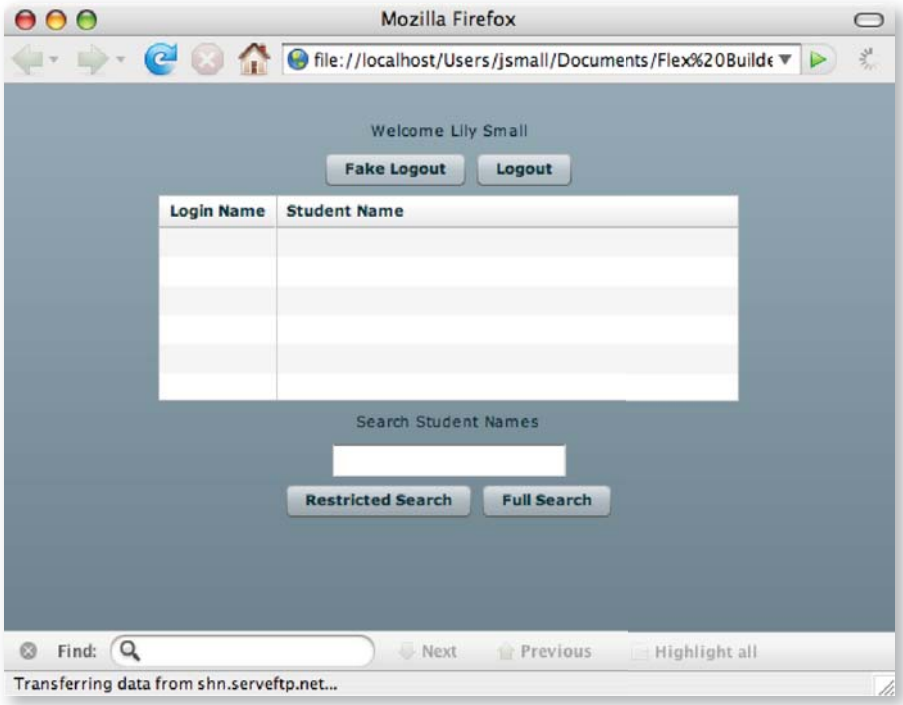
Flex Demo

Here is a sample Flex login application built for the same proxy used by the HTML form in the previous demo.

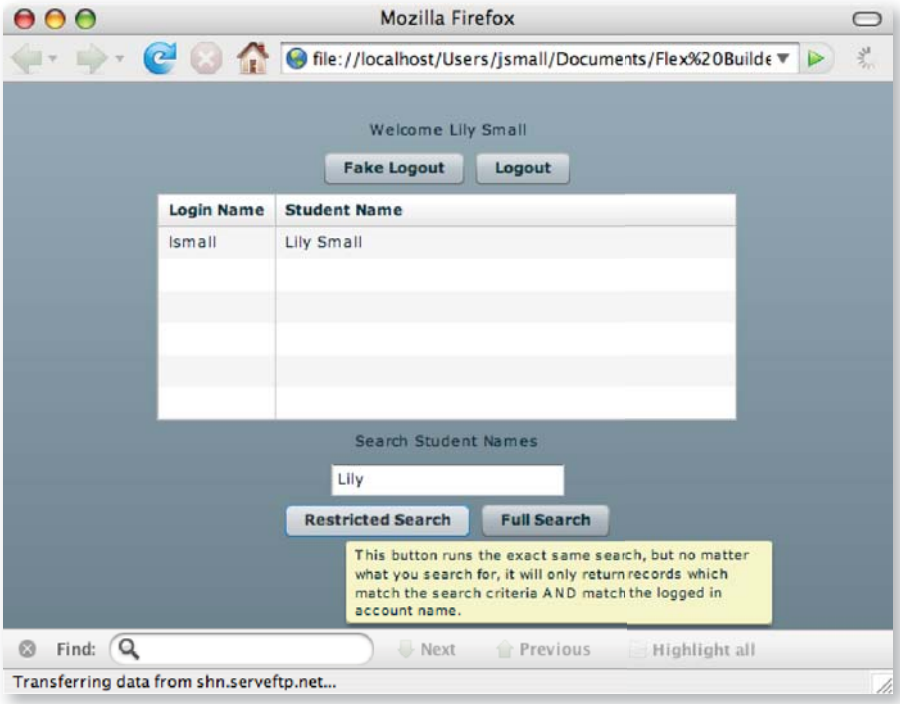
1 The initial screen presents a logged out state with a form that will submit credentials to the proxy.



2 Upon submitting valid credentials, the user record is returned, which includes details such as user full name and is displayed in a welcome message.

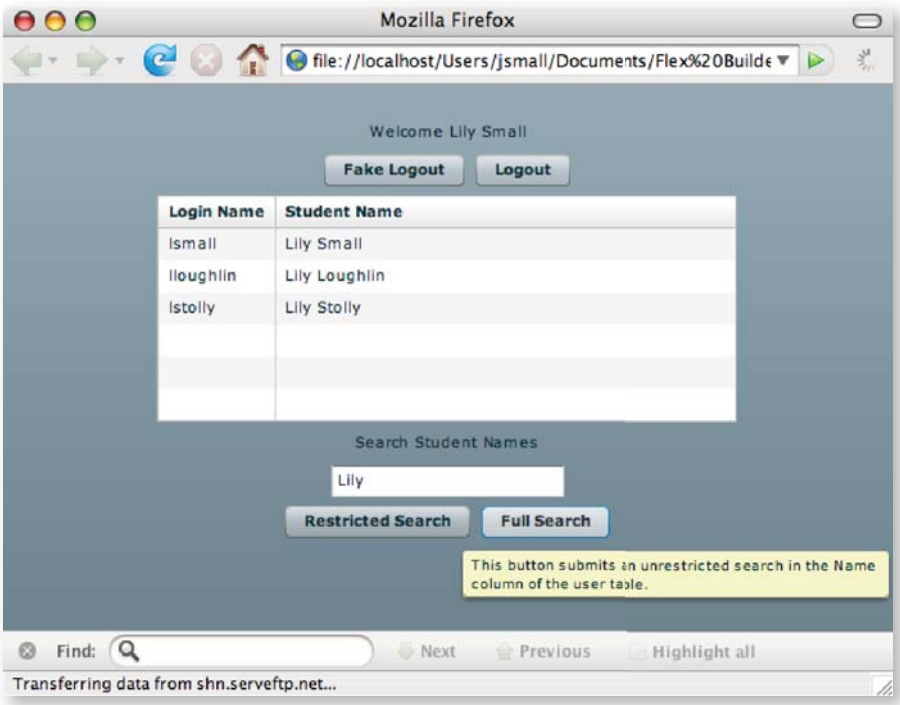


3 The same two search functions options are presented here as were shown in the previous demo. The restricted search will only return records that the logged-in user owns. Obviously you wouldn't really use a client-side criteria to base this restriction on. A proxyParam is used here to illustrate the concept. The ActionScript function for the restricted search button looks like this:



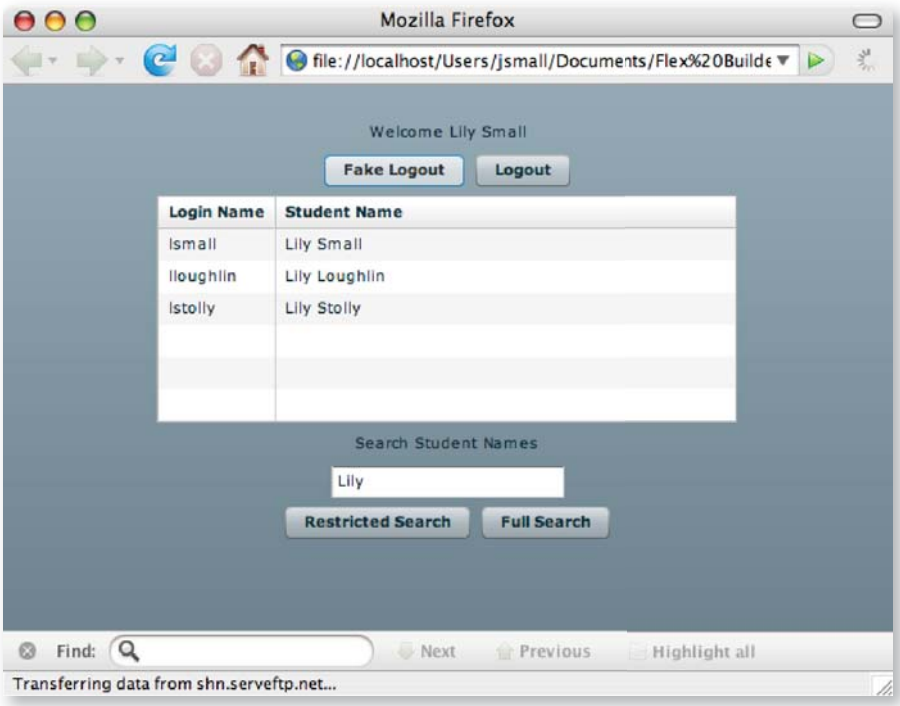

```
public function doFind(user:String):
void {
       mySearch.proxyParams = "FullSearch=no";
       mySearch.query = "studentName_c=" + user + "&-find&-max=5";
       mySearch.execute();
}
```

4 Running the identical search with the other button returns three results instead of one, because the proxy has been instructed not to add the search limiter. The ActionScript function for the full search button looks like this:




```
public function
doFullFind(user:String): void {
       mySearch.proxyParams = "FullSearch=yes";
       mySearch.query = "studentName_c=" + user + "&-find&-max=5";
       mySearch.execute();
 }
```

5 To illustrate the developer control over the PHP session, the Flex demo has two logout buttons which respectively run these ActionScript functions:



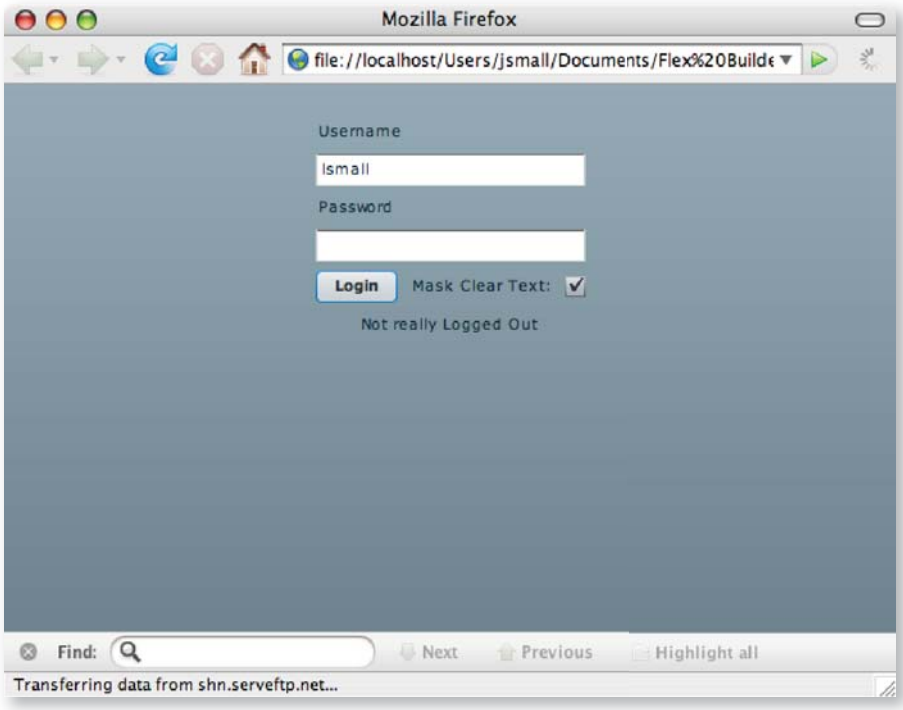

```
public function doLogout():void {
       myLogout.send();
       currentState = "";
       message.text = "Logged Out"
 }

 public function doFakeLogout():void {
       currentState = "";
       message.text = "Not really Logged Out"
 }
```

myLogout is defined in the MXML like this:

```
<mx:HTTPService id="myLogout" url="http://shn.serveftp.net/ProxyLogin/proxy.php?logout=1" />
```

6 When you run the fake logout, it clears the logged in display state, and the password object can plainly be seen to have been cleared. But if you click login again, it takes you right back into the app with no password. This is because the PHP session is still active. To destroy it, the real logout button tells the proxy to destroy the session.



Here is the complete source for the demo app:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    xmlns:soliant="com.soliantconsulting.flexfm.simple.*">

    <soliant:FMServer id="myServer" host="shn.serveftp.net"
        proxy="http://shn.serveftp.net/ProxyLogin/proxy.php" />
    <soliant:FMRequest id="myLogin" server="{myServer}" db="pusd _ oe _ web" layout="web _ applicants _ login"
       username="{myUsername.text}" password="{myPassword.text}" query="-findany" result="loginHandler();"/>
    <soliant:FMRequest id="mySearch" server="{myServer}"  db="pusd _ oe _ web" layout="web _ applicants _ login" />
    <mx:HTTPService id="myLogout" URL="http://shn.serveftp.net/ProxyLogin/proxy.php?logout=1" />

    <mx:VBox id="logoutState">
        <mx:Text text="Username" id="text2"/>
        <mx:TextInput id="myUsername" width="185"/>
        <mx:Text text="Password" id="text1"/>
        <mx:TextInput id="myPassword" displayAsPassword="true" enter="doLogin();" width="185"/>
        <mx:HBox>
            <mx:Button id="login" label="Login" click="doLogin();"/>
            <mx:CheckBox id="checkBox" label="Mask Clear Text:" labelPlacement="left" selected="true"
                change="myPassword.displayAsPassword = checkBox.selected;" />
        </mx:HBox>
    </mx:VBox>

    <mx:Text id="message"/>

    <mx:states>
        <mx:State name="LoggedIn">
            <mx:RemoveChild target="{logoutState}"/>
            <mx:AddChild position="lastChild">
                <mx:VBox horizontalAlign="center" id="loginState">
                    <mx:HBox>
                        <mx:Button id="fakelogout" label="Fake Logout" click="doFakeLogout();"/>
                        <mx:Button id="logout" label="Logout" click="doLogout();"/>
                    </mx:HBox>
                    <mx:DataGrid id="myResult" dataProvider="{mySearch.lastResult.records}"  width="400">
                        <mx:columns>
                            <mx:DataGridColumn dataField=" _ ka _ username" headerText="Login Name" width="80" />
                            <mx:DataGridColumn dataField="studentName _ c" headerText="Student Name" />
                        </mx:columns>
                    </mx:DataGrid>
                    <mx:Text text="Search Student Names"/>
                    <mx:TextInput id="searchTerm" enter="doFullFind(searchTerm.text);"/>
                    <mx:HBox>
                        <mx:Button id="search" label="Restricted Search" click="doFind(searchTerm.text);"
                            toolTip="This button runs the exact same search, but no matter what you
                              search for, it will only return records which match the search criteria
                              AND match the logged in account name."/>
                        <mx:Button id="searchFull" label="Full Search" click="doFullFind(searchTerm.text);"
                            toolTip="This button submits an unrestricted search in the Name column
                              of the user table."/>
                    </mx:HBox>
                </mx:VBox>
            </mx:AddChild>
        </mx:State>
    </mx:states>

<mx:Script>
    <![CDATA[

        public function doLogin():void {
            message.text = "Logging you in...";
            myLogin.execute()
        }

        public function loginHandler():void{
            var errorCode:int = myLogin.lastResult.error;
            myPassword.text = "";
            if (errorCode==0){
                currentState = "LoggedIn";
```

```
                message.text = "Welcome "+myLogin.lastResult.records[0]['studentName_c'];
            } else {
                message.text = "Error: "+myLogin.lastResult.error;
            }
        }

        public function doFind(user:String): void {
            // Obviously you wouldn't really use a client-side criterion to base this restriction on.
            // A proxyParam is used here to illustrate the concept.
            mySearch.proxyParams = "FullSearch=no";
            mySearch.query = "studentName_c=" + user + "&-find&-max=5";
            mySearch.execute();
        }

        public function doFullFind(user:String): void {
            mySearch.proxyParams = "FullSearch=yes";
            mySearch.query = "studentName_c=" + user + "&-find&-max=5";
            mySearch.execute();
        }

        public function doLogout():void {
            myLogout.send();
            currentState = "";
            message.text = "Logged Out"
        }

        public function doFakeLogout():void {
            currentState = "";
            message.text = "Not really Logged Out"
        }
    ]]>
</mx:Script>

</mx:Application>
```

Conclusion

While no Web application is 100% secure, it is possible, with a little forethought, to apply some best practices and significantly reduce the vulnerability of your Web login systems. Resist the temptation to write custom user authentication rubrics. Rarely handle user credentials in clear text, and never store them in cookies or shared objects on the client side. Transmit all sensitive information, especially user credentials, in encrypted streams. Take advantage of PHP's session management and server-side filtering nature to employ a proxy and reduce the number of potential vulnerabilities in your system. Proxying of some form is a widely used best practice for permitting web apps access to internal databases, or for permitting external access to internal web apps.[28]

Password.
Invisible in my stream,
bound for server's salted hash.

28 http://en.wikipedia.org/wiki/Reverse_proxy

Glossary of Terms

Term	Definition	URL
401 (FileMaker)	When returned by FileMaker, this error code means, "No records match the request"	http://www.filemaker.com/help/21a-FMP_error%20codes.html
401 (HTTP)	When returned in an HTTP transaction, this error code indicates the request contains bad syntax or cannot be fulfilled. The 4xx class of status code is intended for cases in which the client seems to have erred. Similar to 403 Forbidden, but specifically for use when authentication is possible but has failed or not yet been provided. See Basic access authentication.	http://en.wikipedia.org/wiki/List_of_HTTP_status_codes#4xx_Client_Error
ActionScript	In June 2006, ActionScript 3.0 debuted with Adobe Flex 2.0 and its corresponding player, Flash Player 9. ActionScript 3.0 was a fundamental restructuring of the language, so much so that it uses an entirely different virtual machine than its predecessor.	http://en.wikipedia.org/wiki/ActionScript
API	An application programming interface (API) is a source code interface that an operating system, library or service provides to support requests made by computer programs.	http://en.wikipedia.org/wiki/API
Basic Access	In the context of an HTTP transaction, the basic access authentication is a method designed to allow a web browser, or other client program, to provide credentials – in the form of a user name and password – when making a request.	http://en.wikipedia.org/wiki/Basic_access_authentication
Class	In object-oriented programming, a class is a programming language construct used to group related attributes and methods.	http://en.wikipedia.org/wiki/Class_%28computer_science%29
Clear Text	In data communications, cleartext is the form of a message or data which is in a form that is immediately comprehensible to a human being without additional processing.	http://en.wikipedia.org/wiki/Clear_text
Cookie	HTTP cookies, or more commonly referred to as Web cookies, tracking cookies or just cookies, are parcels of text sent by a server to a web client (usually a browser) and then sent back unchanged by the client each time it accesses that server.	http://en.wikipedia.org/wiki/HTTP_cookie
Credentials	Credentials in information systems are widely used to control access to information or other resources. The classic combination of a user account number or name and a secret password is a widely-used example of IT credentials.	http://en.wikipedia.org/wiki/Credentials#Information_technology

FileMaker Pro	FileMaker Pro is a cross-platform database application from FileMaker Inc. (a subsidiary of Apple Inc.), known for its combination of power and ease of use. It is also noted for the integration of the database engine with its GUI-based interface, which allows users to modify the database by dragging new elements into the layouts/screens/forms that provide the user interface. This results in a "quasi-object" development environment of a kind that is still largely unique in the "industrial strength" database world.	http://en.wikipedia.org/wiki/Filemaker_Pro
Flash Player	The Adobe Flash Player is a widely distributed proprietary multimedia and application player created and distributed by Macromedia (a division of Adobe Systems). Flash Player runs SWF files that can be created by the Adobe Flash authoring tool, by Adobe Flex or by a number of other Macromedia and third party tools.	http://en.wikipedia.org/wiki/Flash_player
Flex	Adobe Flex is a collection of technologies released by Adobe Systems for the development and deployment of cross platform, rich Internet applications based on the proprietary Adobe Flash platform.	http://en.wikipedia.org/wiki/Adobe_Flex
Form	A webform on a web page allows a user to enter data that is, typically, sent to a server for processing and to mimic the usage of paper forms. Forms can be used to submit data to save on a server (e.g., ordering a product) or can be used to retrieve data (e.g., searching on a search engine).	http://en.wikipedia.org/wiki/Html_form
Framework	In computer programming, an application framework is a software framework that is used to implement the standard structure of an application for a specific operating system. Object-oriented programming techniques are usually used to implement frameworks such that the unique parts of an application can simply inherit from pre-existing classes in the framework.	http://en.wikipedia.org/wiki/Application_framework
Hash	In cryptography, a cryptographic hash function is a transformation that takes an input and returns a fixed-size string, which is called the hash value.	http://en.wikipedia.org/wiki/Cryptographic_hash_function
HTTP	Hypertext Transfer Protocol (HTTP) is a communications protocol for the transfer of information on the intranet and the World Wide Web.	http://en.wikipedia.org/wiki/HTTP
HTTP Basic Access	See Basic Access.	na
HTTP Header	HTTP Headers form the core of an HTTP request, and are very important in an HTTP response. They define various characteristics of the data that is requested or the data that has been provided.	http://en.wikipedia.org/wiki/HTTP_header
MXML	MXML is an XML-based user interface markup language first introduced by Macromedia in March 2004. Adobe (who acquired Macromedia in December 2005) gives no official meaning for the acronym, but some developers suggest it should stand for "Magic eXtensible Markup Language" (which is a backronym). It's likely that the name comes from the MX suffix given to Macromedia Studio products released in 2002 and 2004. Application developers use MXML in combination with ActionScript to develop Rich Internet applications.	http://en.wikipedia.org/wiki/MXML

Old No 7	Jack Daniel's Tennessee whiskey (not to be confused with bourbon) known for its square bottles and black label.	http://en.wikipedia.org/wiki/Jack_Daniel's#Product_name
PBKDF2	PBKDF2 (Password-Based Key Derivation Function) is a key derivation function that is part of RSA Laboratories' Public-Key Cryptography Standards (PKCS) series.	http://en.wikipedia.org/wiki/PBKDF2
Phishing	is an attempt to criminally and fraudulently acquire sensitive information, such as usernames, passwords and credit card details, by masquerading as a trustworthy entity in an electronic communication.	http://en.wikipedia.org/wiki/Phishing
PHP	PHP is a widely used general-purpose scripting language that is especially suited for web development and can be embedded into HTML. It generally runs on a web server, taking PHP code as its input and creating web pages as output. It can be deployed on most web servers and on almost every operating system and platform free of charge.	http://en.wikipedia.org/wiki/Php
PKCS#5	In cryptography, PKCS refers to a group of Public Key Cryptography Standards devised and published by RSA Security.	http://en.wikipedia.org/wiki/PKCS
POST	POST is an HTTP request method. POST submits data to be processed (e.g. from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.	http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods
Proxy	In computer networks, a proxy server is a server (a computer system or an application program) which services the requests of its clients by forwarding requests to other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource, available from a different server. The proxy server provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy server may optionally alter the client's request or the server's response, and sometimes it may serve the request without contacting the specified server. In this case, it would 'cache' the first request to the remote server, so it could save the information for later, and make everything as fast as possible.	http://en.wikipedia.org/wiki/Reverse_proxy
Proxy Screening	A term used in the context of this paper, referring to the practice of inspecting and manipulating the proxy payload before passing it along. See the article on Reverse Proxy.	http://en.wikipedia.org/wiki/Reverse_proxy
REST	REST strictly refers to a collection of network architecture principles which outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. It is also possible to design simple XML+HTTP interfaces that do not conform to REST principles, and instead follow a model of remote procedure call. FileMaker's Web Publishing Engine is not RESTful in the strict sense. See RPC.	http://en.wikipedia.org/wiki/REST#REST_versus_RPC

RPC	Remote procedure call (RPC) is a technology that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. That is, the programmer would write essentially the same code whether the subroutine is local to the executing program, or remote. When the software in question is written using object-oriented principles, RPC may be referred to as remote invocation or remote method invocation.	http://en.wikipedia.org/wiki/Remote_procedure_call
RSA	In cryptography, RSA is an algorithm for public-key cryptography.	http://en.wikipedia.org/wiki/Rsa
Salt	In cryptography, a salt comprises random bits that are used as one of the inputs to a key derivation function. The other input is usually a password or passphrase. The output of the key derivation function is stored as the encrypted version of the password. A salt can also be used as a key in a cipher or other cryptographic algorithm. The key derivation function typically uses a hash function.	http://en.wikipedia.org/wiki/Salt_%28cryptography%29
Security by Design	Secure by design, in software engineering, means that the software has been designed from the ground up to be secure. Malicious practices are taken for granted and care is taken to minimize impact when a security vulnerability is discovered.	http://en.wikipedia.org/wiki/Security_by_design
Security by Obscurity	In cryptography and computer security, security through obscurity (sometimes security by obscurity) is a controversial principle in security engineering, which attempts to use secrecy (of design, implementation, etc.) to provide security. A system relying on security through obscurity may have theoretical or actual security vulnerabilities, but its owners or designers believe that the flaws are not known, and that attackers are unlikely to find them. The technique stands in contrast with security by design, although many real-world projects include elements of both strategies.	http://en.wikipedia.org/wiki/Security_through_obscurity
Self proxy	The term self proxy is used in this paper in reference to PHP applications which accept credentials from a user for the purposes of establishing authentication, and then which subsequently rely on that established session validity to subsequently act on that user's behalf with stored system credentials.	na
Session (general)	In computer science, in particular networking, a session is a semi-permanent interactive information exchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user (see Login session). A session is set up or established at a certain point in time, and torn down at a later point in time. An established communication session may involve more than one message in each direction. A session is typically, but not always, stateful, meaning that at least one of the communicating parts need to save information about the session history in order to be able to communicate, as opposed to stateless communication, where the communication consists of independent requests with responses.	http://en.wikipedia.org/wiki/Session_%28computer_science%29

Session (PHP)	Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site. A visitor accessing your web site is assigned a unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.	http://us3.php.net/manual/en/intro.session.php
Session Fixation (PHP)	Session fixation attacks attempt to exploit the vulnerability of a system which allows one person to fixate (set) another person's session identifier (SID). Most session fixation attacks are web based, and most rely on session identifiers being accepted from URLs (query string) or POST data.	http://en.wikipedia.org/wiki/Session_fixation
Session Functions (PHP)	See Session (PHP)	na
Session Hijacking (PHP)	The term session hijacking refers to the exploitation of a valid computer session - sometimes also called a session key - to gain unauthorized access to information or services in a computer system. In particular, it is used to refer to the theft of a magic cookie used to authenticate a user to a remote server. It has particular relevance to web developers, as the HTTP cookies used to maintain a session on many web sites can be easily stolen by an attacker using an intermediary computer or with access to the saved cookies on the victim's computer (see HTTP cookie theft).	http://en.wikipedia.org/wiki/Session_hijacking
Session ID (PHP)	See Session (PHP)	na
Shared Object	Local Shared Object (LSO), sometimes known as flash cookies, is a cookie-like data entity used by Adobe Flash Player. The player allows web content to read and write LSO data to the computer's local drive on a per-domain basis. The Local Shared Objects are available in Flash Players starting from version 6. This technology permits web sites to preserve session state and record user data and behavior.	http://en.wikipedia.org/wiki/Local_Shared_Object
Signature	Type signature is a term that is used in computer programming. A type signature defines the inputs and outputs for a function or method. A type signature includes at least the function name and the number of its parameters. In some programming languages, it may also specify the function's return type or the types of its parameters.	http://en.wikipedia.org/wiki/Type_signature
Social Engineering	Social engineering is a collection of techniques used to manipulate people into performing actions or divulging confidential information. While similar to a confidence trick or simple fraud, the term typically applies to trickery for information gathering or computer system access and in most cases the attacker never comes face-to-face with the victim.	http://en.wikipedia.org/wiki/Social_engineering_%28security%29
SSL	Hypertext Transfer Protocol over Secure Socket Layer or https is a URI scheme used to indicate a secure HTTP connection. It is syntactically identical to the http:// scheme normally used for accessing resources using HTTP. Using an https: URL indicates that HTTP is to be used, but with a different default TCP port (443) and an additional encryption/authentication layer between the HTTP and TCP. This system was designed by Netscape Communications Corporation to provide authentication and encrypted communication and is widely used on the World Wide Web for security-sensitive communication such as payment transactions and corporate logons.	http://en.wikipedia.org/wiki/Https

SWF	The SWF file format delivers vector graphics, text, video, and sound over the Internet and is supported by Adobe® Flash® Player and Adobe AIR™ software. The SWF file format is available as an open specification to create products and technology that implement the specification. SWF 9 introduces the ActionScript™ 3.0 language and virtual machine.	http://www.adobe.com/devnet/swf/
WPE	The Web Publishing Engine is a component of FileMaker Server that provides the Custom Web Publishing services for databases hosted by FileMaker Server.	http://filemaker.com/downloads/pdf/fms9_getting_started_en.pdf