Some Security Stuff About Mythea

sqwishy <sqwishybon@gmail.com>

January, 2014

1 Introduction to Mythea

Mythea is an online multiplayer game that runs in a browser. It makes heavy use of JavaScript, using it for anything from DOM manipulation to sending authentication data. In the game, each player runs her own province. Each province is on a team with up to eleven other provinces, forming a kingdom. Generally, all provinces in a kingdom work together in competition against every other kingdom. There is also a chat system built into the website that allows users to communicate in real time. Players, by default, enter a global chat room, which makes the entire playerbase an audience to everybody else, and a private kingdom room, where players can communicate to those in their own kingdom and private information about kingdom actions, such as the details of attacks, are posted automatically by the system. There is also a feature for creating and joining arbitrary chat rooms, separate from the aforementioned special ones. The chat system is explained more and in section 4.[1]

2 Connection Eavesdropping

Mythea makes minimal use of encryption. HTTPS connections do work, but no effort is made to prevent or reduce unencrypted HTTP requests. HTTP requests used in authentication and the transmission of account information is not encrypted. This can be seen in figures 1 and 2. Furthermore, the password reminder feature emails users their passwords in plain text, as seen in figure 3. This indicates that passwords are stored without encryption, which is generally considered to be poor security design.

[1] See the Mythea Guide found at `https://www.mythea.com/` for more information on the game.

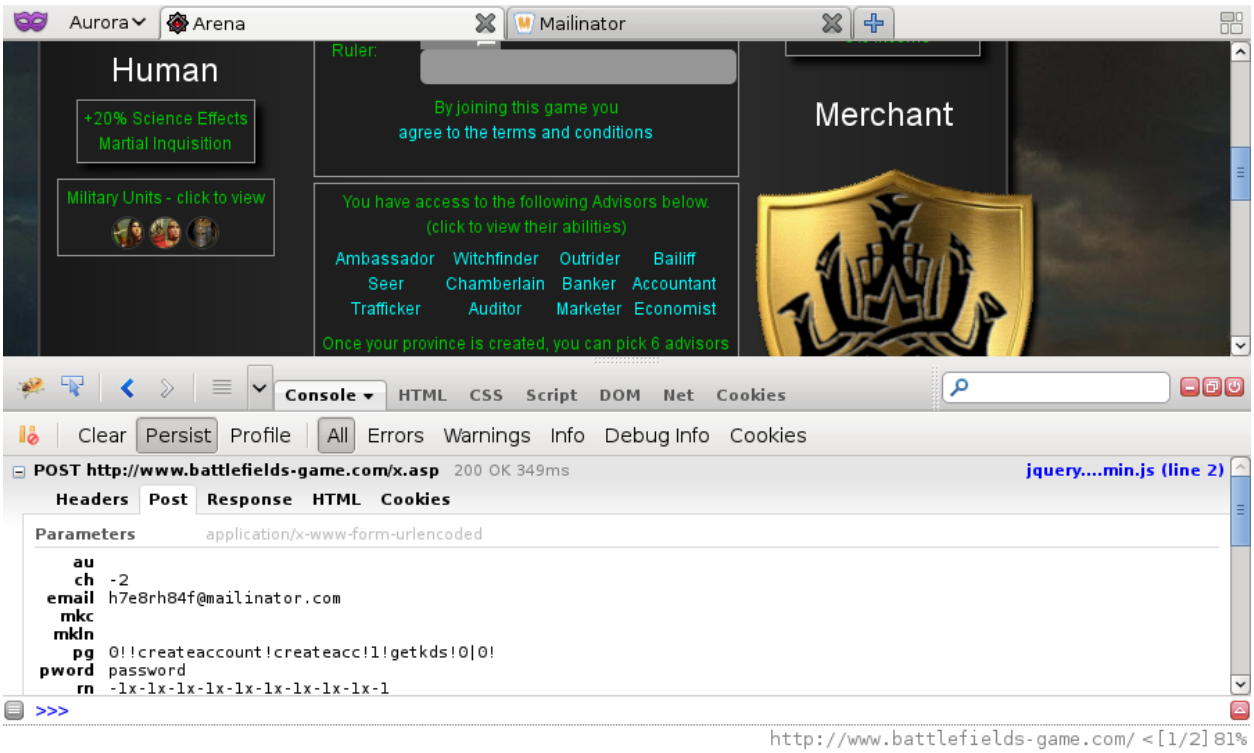

Figure 1: Registration request not sent over HTTPS

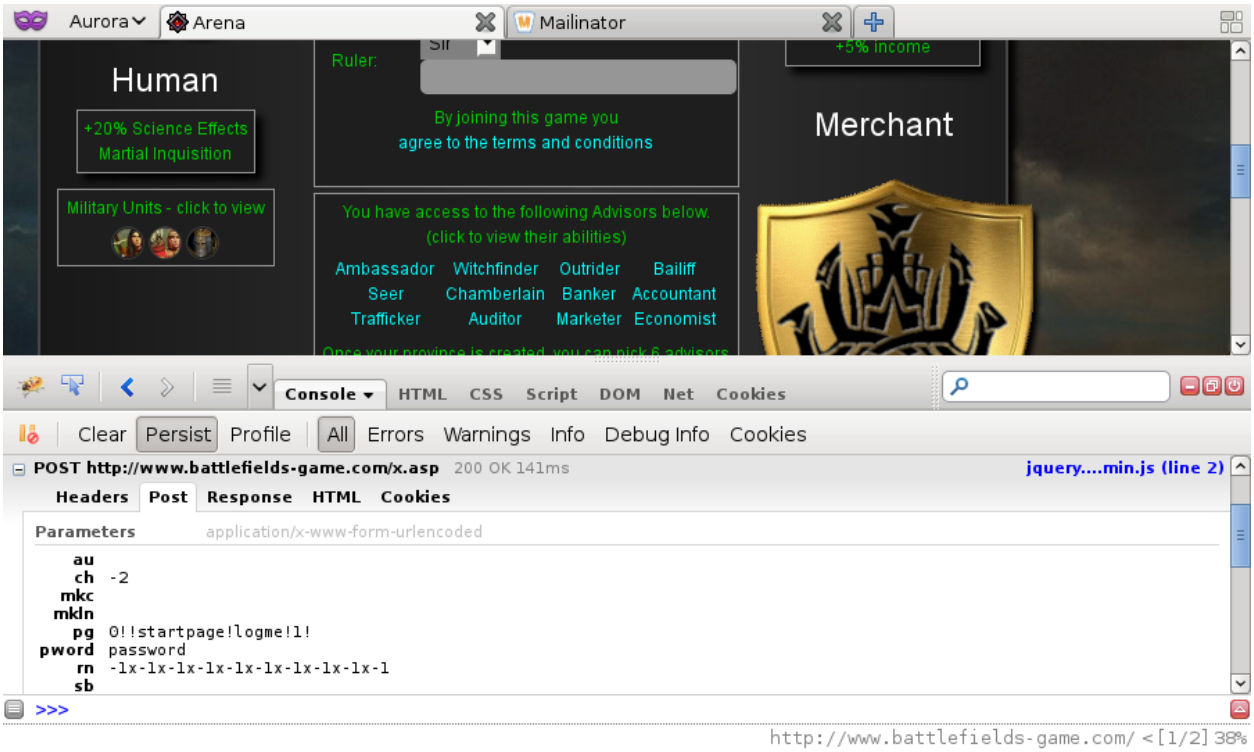

Figure 2: Login request not sent over HTTPS

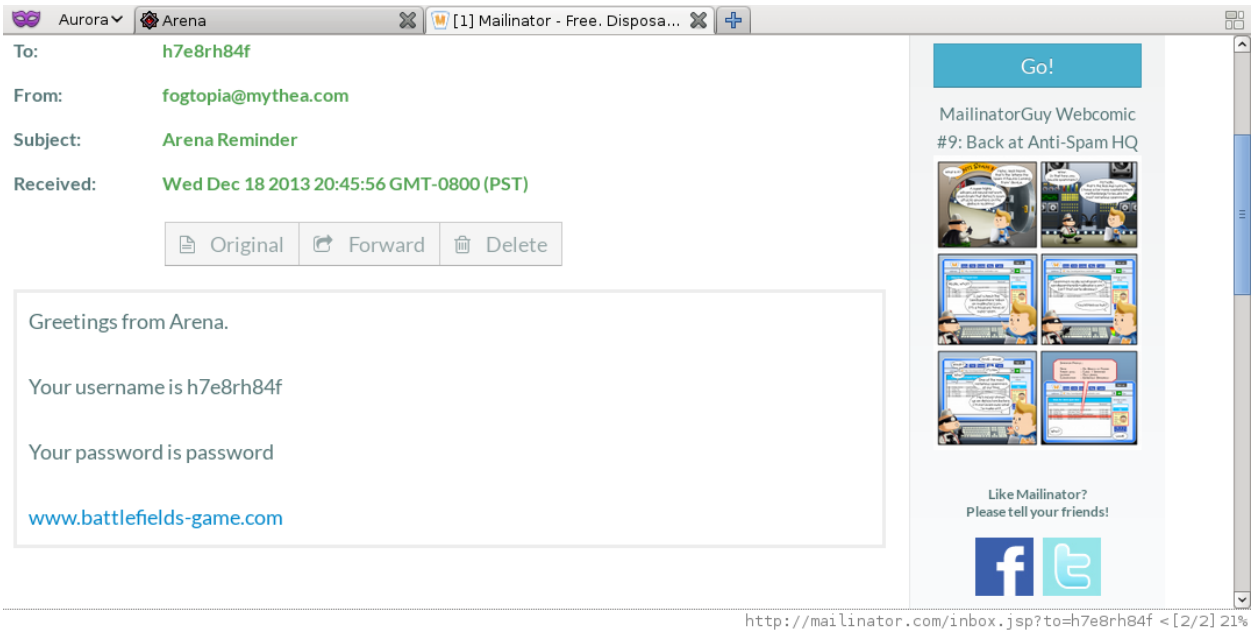

Figure 3: Password reminder email featuring my password in plain text

3 Mythea Kingdom Customization

The most simple exploit is possible because a bit of user input isn't sufficiently escaped. One feature in Mythea is one which lets players assign an image as a kingdom icon providing a URL which is then inserted into a stylesheet rule visible to other players. However, little regulation is applied to the input. Less than and greater than signs (<, >) are altered, single quotes (') are translated into backticks (‘), but the double quote character (") is not altered[2]. This information is then inserted in a style attribute on a div element created when players view the kingdom.

As a result, users can terminate the style with a double quote character and insert arbitrary data, such as JavaScript that triggers on a mouse event. They can even add further styles if they wish. For example, in order to provoke mouse events, a user might set the height and width of the element such that it will cover the entire page. There is a maxlength attribute on the input field; however, it can be removed by the clients and no length checks are done server-side. In order to work around the single apostrophe issue, I chose to produce strings as a result of the evaluation of String.fromCharCode() calls, seen in figure 4, although I'm sure there are more elegant workarounds.

Figure 5 demonstrates using the input in figure 4 to create an alert in my browser when it views my kingdom overview.

This vulnerability allows attackers to access private information, such as document.cookie, on the machines of other players. It also allows them to create and send events to elements, which can be used to make the clients of victims do actions that the victims would rather they not have done.

[2]Interestingly, the right parenthesis ()) character is replaced by a right bracket (]) in a different, yet similar, input field used to specify the URL of a kingdom banner; however, that rule isn't applied in the kingdom icon input.

```
); background:transparent; border:0; position:fixed; top:0; left:0;
    width:100%; height:100%" onmouseover="alert([72, 101, 108, 108, 111,
     44, 32, 119, 111, 114, 108, 100, 33].map(function(c){return String.
    fromCharCode(c)}).join(String()))" nothing="(
```

Figure 4: Example input for banner URL that executes JavaScript on other players' browsers

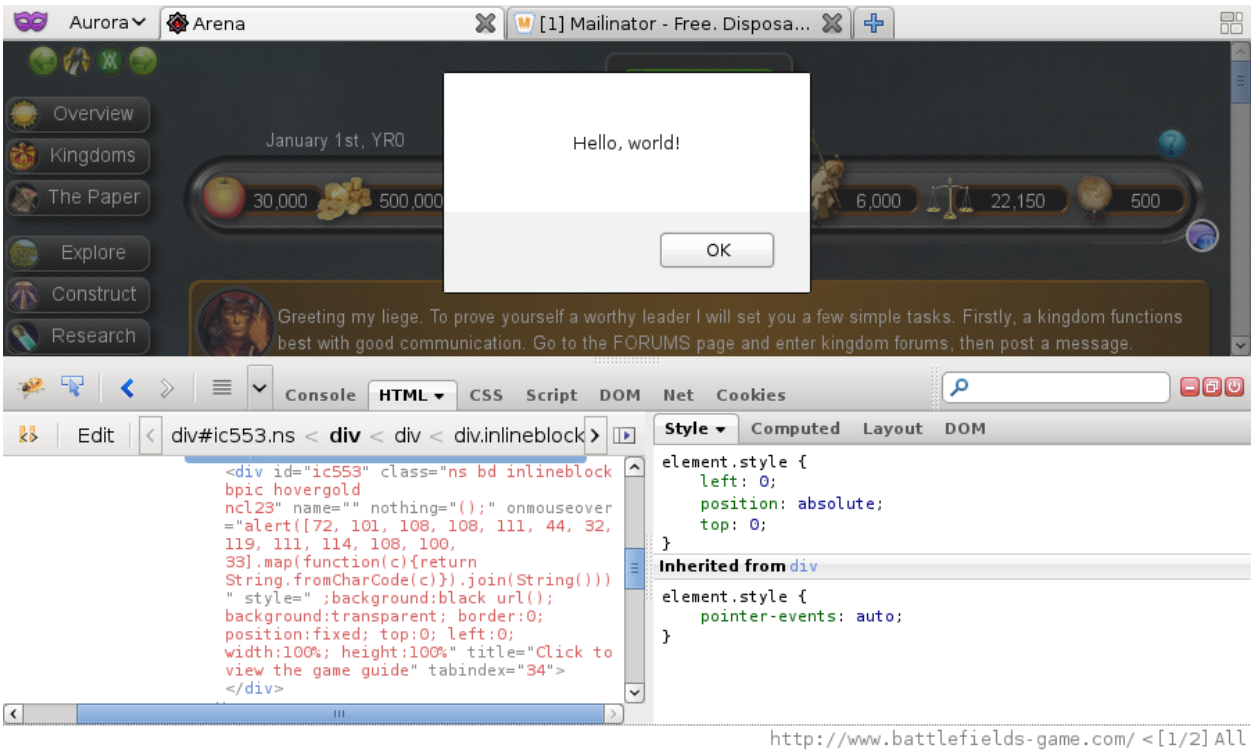

Figure 5: JavaScript being executed by clients that navigate to my kingdom page

4 Mythea Chat Service

The chat service in Mythea works by using JavaScript to send HTTP requests. The format of the messages that move between the client and the server is a bit tricky to parse, but the obfuscation contributes little to security. By identifying key parts of the syntax, malicious messages can be formed relatively easily.

4.1 Herald Exploit

The herald is a feature in the Mythea chat used for announcing game events and other news. Important messages are sent by the server to clients as responses to periodic polling that each client makes to keep itself up-to-date. The server lets users make announcements, as shown in figure 6, and, due to the way these messages are handled, arbitrary JavaScript can be executed on the clients of other players who are in chat rooms where a herald announcement is made.

Part of the problem is that `eval()` is used, and it is used fairly liberally throughout the client. This expands the set of JavaScript that can be executed on clients as a result of a message from a server. For example, if the strings were just passed as arguments to functions and written into text nodes in the DOM - never passed to `eval()` - the worst case scenario is relatively limited. Perhaps a funny message is displayed to the user. However, with `eval()`, the malicious code path is up the attacker's imagination. In general, the server should not trust the client (it could be cURL!) and the client should not trust the server.

Another problem is that the delimiters aren't escaped. Using a more popular encoding, such as JSON, lets you use the encoding libraries available on the platform. Modern web browsers come with a way to marshal JavaScript data structures and unmarshal JSON encoded strings. They also tend to be more secure and have greater features.

Taking advantage of the `eval()`s through the chat is especially interesting since they have access to the main closure that holds a lot of interesting functions and data that isn't available with `onmouseover` event handler functions and other DOM related exploits.

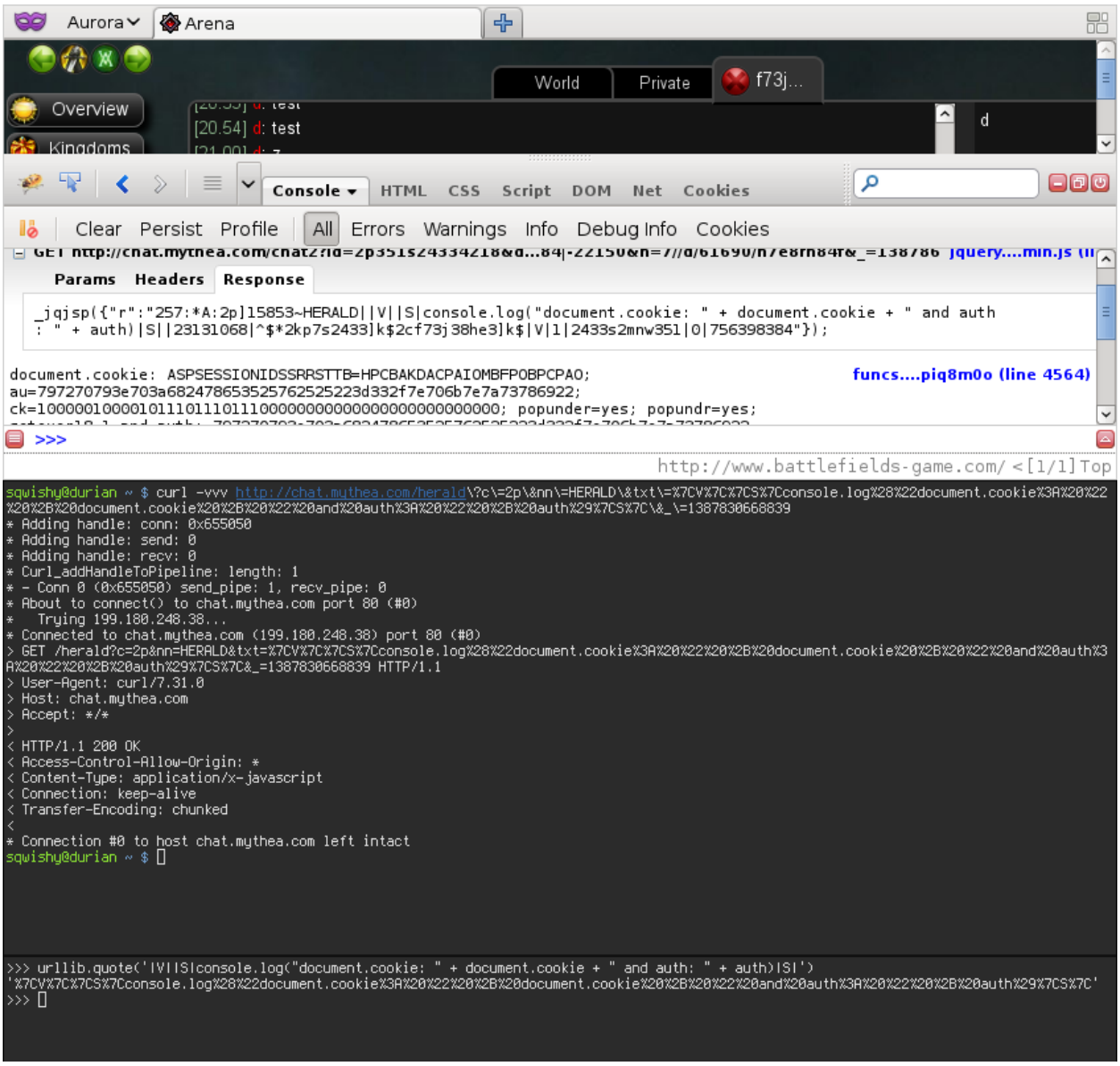

Figure 6: The '`|V|`' and '`|S|`' strings are control strings that instruct the parser to evaluate the payload. In this example, we have access to a lot of information on the client. An attacker could POST the `document.cookie` of other players to another server.

4.2 Joining Private Rooms

When sending requests to the chat server, a string is used to specify what channel the message is being sent on. For example, in the URL given to cURL shown in figure 6, the value for `c` is `2p`, indicating that the message is being sent to the public/world channel on chat server 2. For my kingdom channel, the string is `*2kp7s2433`; my kingdom id is 7. By manipulating this, I can join any kingdom channel. This is an issue since they are meant to be private and generally contain private information.

Determining the kingdom id is simple; they are not secrets. The kingdom id for the administrator is 1. Figure 7 shows that adjusting the id for the chat room in the client allows me entry to private kingdom chat rooms. The first few messages are from the herald. Messages about invasions or attacks from the herald sometimes contain sensitive information and are only available in the chat rooms for the attacking kingdom and the attacked kingdom. The information in these private rooms can be used to exploit other players.

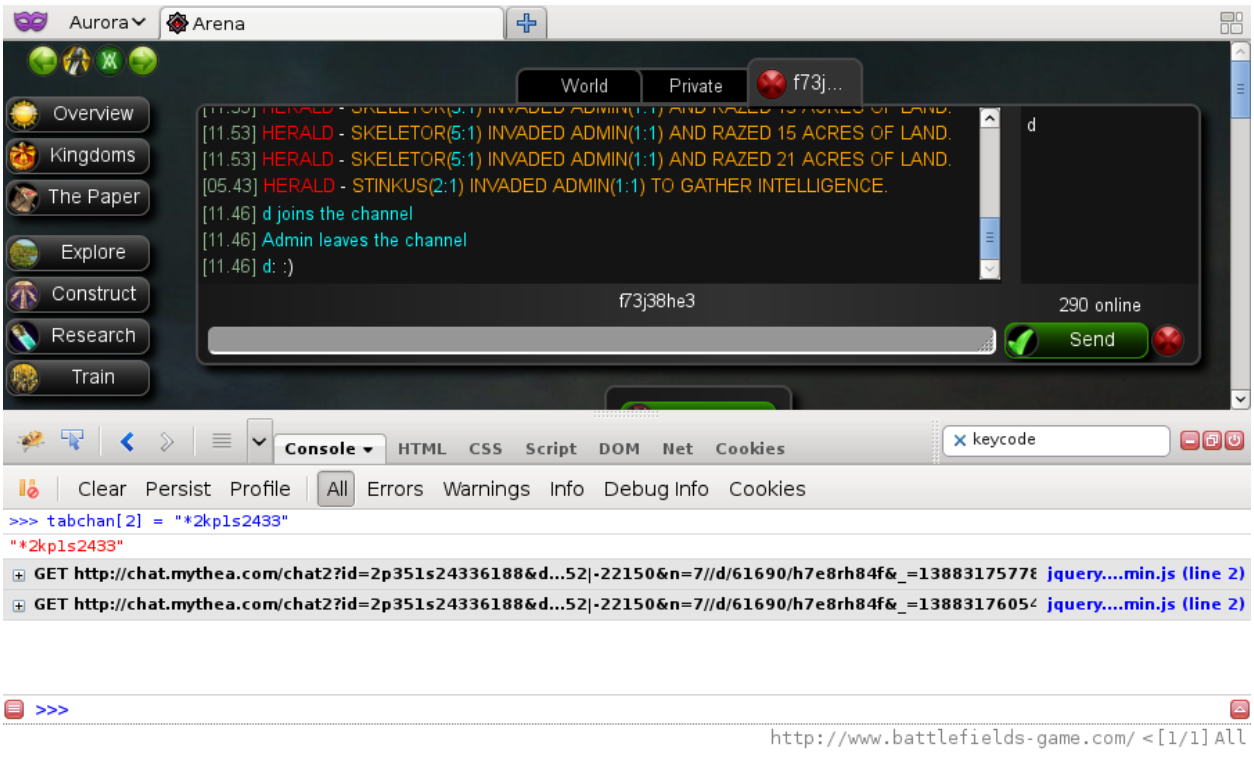

Figure 7

4.3 Nickname Exploit

Players can register different nicknames for themselves. The way this seems to work is that the server will give the user a token formed from a hash of their nickname. Making sure the token matches the hash of the nickname appears to be the extent of the authentication.

Figure 8 is an example of a request sent through the application cURL that shows up in the Mythea chat. Headers and cookies that exist in the browser are not sent in the request and the only part of the URL that needs to be modified is the very last value, which appears to be a unix timestamp.

The last value shown in firebug (`7//h7e8rh84f/34454/h7e8rh84f`) in figure 8 seems to contain the username and nickname information. Of the five parts, delimited by the "`/`", the last three are respectively the nickname, a hash formed from the nickname, and the username.

Changing the nickname in the client without changing the result of the hash causes an authentication error. The procedure for generating the hash doesn't seem to exist anywhere in the client. But, determining how nicknames are hashed was a matter of changing my username a few times in order to see if a pattern could be detected.

Figure 9 shows pairs of nicknames and their hashes. These hashes do not work for all accounts. That is, making a new account and setting the nickname to "a" and the hash to "98545" in the client would likely result in an authentication error. The implementation for the hash function in figure 10 uses magic numbers. The value for `INC` seems stable; however, the value for `START` seems to change from account to account, acting somewhat as a salt.

In figure 11, I am able to change my nickname in a way that is normally prohibited which allows me to execute arbitrary JavaScript similar to the herald exploit in section 4.1. It is left up the client to ensure that all characters are alphanumeric and that the length is less than twenty characters long. The server does prevent you from registering a nickname that is in use and from changing your nickname too quickly. Both of those are bypassed here since this method simply puts the client into the state it would be in if the authentication had worked.

This could probably be solved by using a less predictable hashing function. But integrating the chat authentication with the authentication for Mythea is probably a better and more reliable solution. And again, as figure 11 and the herald exploit shows, the client should be modified so that parsing is safer and `eval()` is not used.

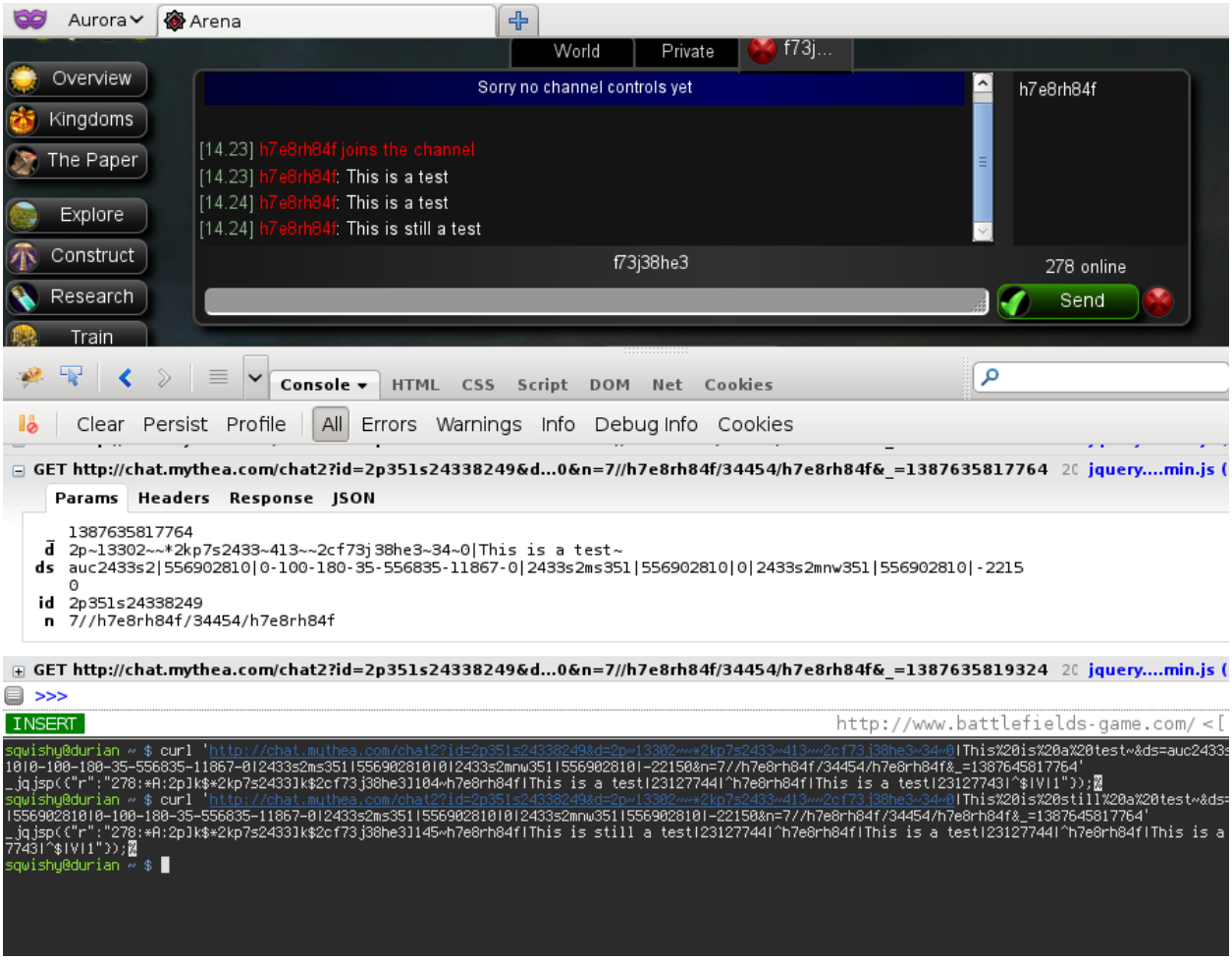

Figure 8: Using cURL to send the messages "This is a test" and "This is still a test" to the Mythea chat system.

a	98545
b	19776
c	41007
0	58226
00	77314
01	98545
A	19153
A0	38241
B	40384
9	49305

Figure 9: Nicknames and their respective hashes produced by Mythea. This was used as data to discover and verify the hash function. These hashes correspond to a `START` value of 39138.

```
START    = 38590
INC      = 21231
MAX      = 100000

def nickhash(s):
    if not s:
        raise RuntimeError, "cannot hash an empty string"
    return (START + sum(map(ord, s)) * INC) % MAX
```

Figure 10: Hash function implemented in Python. The value of START seems to work to generate names for the account with the username "h7e8rh84f" that I have been using.

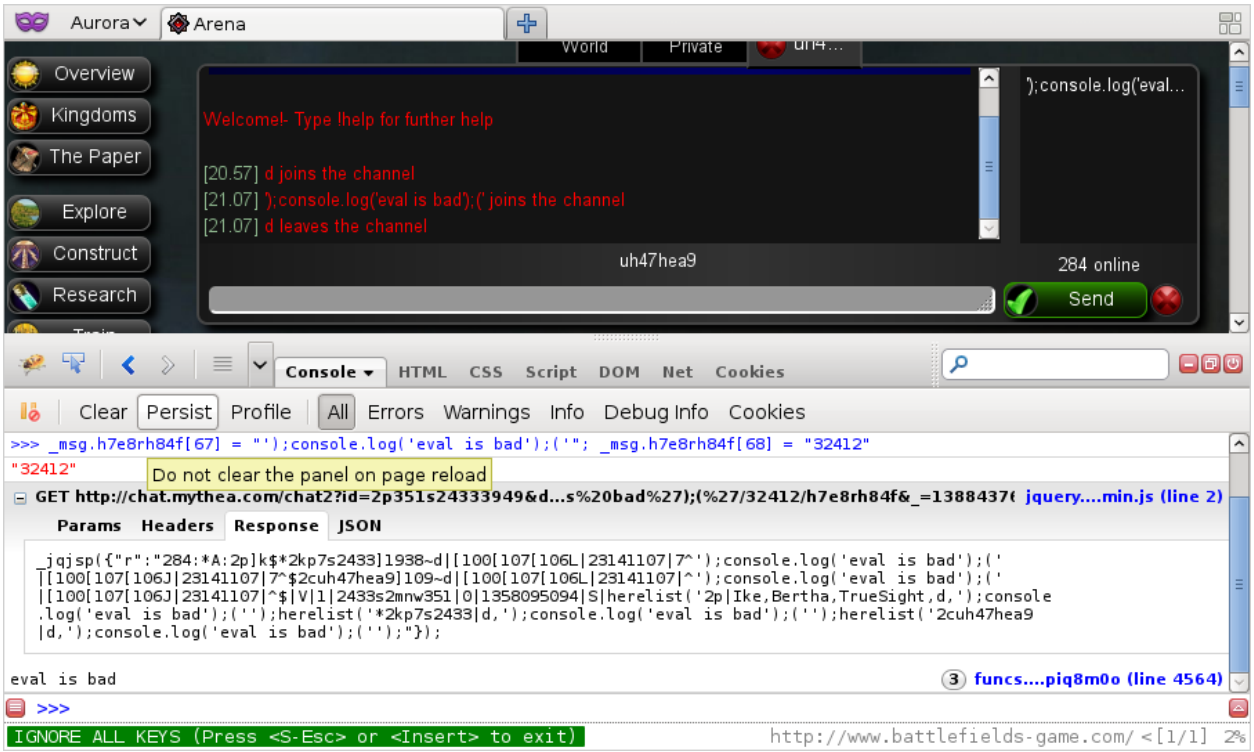

Figure 11: Executing arbitrary code as a result of using eval to parse server data in the client. (I probably should have logged in with another account at the same time to demonstrate/ensure that the effect propagates to other clients. I guess you'll just have to use your imagination.)