6.857 Final Project

Craig B. Cheney
Tugsbayasgalan Manlaibaatar
Matthew J. Mucklo
Elijah Rivera

May 2018

1 Introduction

Our final project was the security analysis of the patient portal web application of a major healthcare company in the area (which has requested to remain anonymous for the purposes of this project). We used automated testing tools tools and industry-standard practices to perform an exhaustive scan of the gateway to the extent that time permitted. We managed to uncover multiple vulnerabilities, ranging from innocuous to moderately severe. Following the responsible disclosure process, we have already made the company aware of the entire collection of vulnerabilities found.

2 Background

A healthcare company reached out to our class via Professor Rivest. They requested a team to scan several different online services they had been developing for vulnerabilities. They listed a web application, an API, and a mobile application as proposed areas to examine. With input from the CISO and the company we jointly decided that due to time constraints we would only focus on the web application. Previous assessments of the company had been performed, and we were able to build upon that work.

2.1 Initial Project Proposal

In our initial project proposal, we indicated that we intended to do both black-box and white-box testing. For black-box testing, we planned on doing automated penetration testing, man-in-the-middle attacks (using the WiFi Pineapple), and denial-of-service attacks. For white-box testing, we intended to analyze the user data flow and source code looking for specific vulnerabilities, especially cross-site scripting. We also indicated our intent to put together a security questionnaire, a set of questions meant to help us understand their security policies. This questionnaire was to include questions about internal passwords, roles of different users, and administration questions.

2.2 Rules of Engagement

After forming the group, writing our proposal, and making initial contact with the company, we ended up having a brief conference call defining the "rules of engagement" with the company. Present on the call (apart from our team) was the CISO of the company and several others who were either developers or were on the security team for the company.

After some amount of discussion, we all agreed that our team would begin with the web application, as it was the the highest priority for the company and the quickest area to get our team set up in. We also were given some fairly explicit rules of engagement:

- All testing (both black-box and white-box) would be done only on-site at the physical location of the company.
- All automated test tools and other testing methods would be disclosed beforehand, and could be vetoed.
- All tests could only be run against a staging server, and NOT against the production server.
  - However, through a series of negotiations, we were granted permission to perform a small amount of non-invasive tests on the production server, namely DNS, SSL, and traceroute checks, as well as viewing the source code.

With these rules in mind, we organized 1-2 weekends in late April/early May to visit the company and perform our tests.

3 Preparation and Division of Labor

Before going on-site, we made a lot of preparations. The prep-work was divided into four main areas, with each of us taking the lead role in one of these areas.

3.1 Production testing

Per the established rules, all production testing was purely non-invasive, and performed off-site. As we were not permitted to do any invasive testing against production, it limited the information we were able to find. Nonetheless we did come across a couple of things.

3.1.1 SSL certificate verification

We checked their SSL certificate using a well-known testing service called "SSL Labs" by Qualys. It is standard to aim for at *least* an A grade on the certificate, but the company's certificate scored a maximum of B.

This result also depends on whether one tests www.REDACTED or REDACTED. The former tests correctly, but the latter fails due to "trust issues". For more details, please see 11.6.

3.1.2 View Source

Viewing the source of their production website immediately revealed one script failing to be loaded, and several best-practice violations. For more details, please see, 11.4.

3.1.3 Traceroute

We performed a traceroute to their server, and cross-referenced the result against ARIN's whois database (www.arin.net) to see if there were any peers along the way which were known to be weak or exploitable. Testing from MIT's network revealed a series of safe medical networks along the way, when cross-referenced against the ARIN database. For more details, please see 11.7.

3.1.4 DNS

We checked DNS hosting to make sure that the company's host was not a weak third party. Testing showed that DNS was hosted internally and within other medically-owned networks. For more details, please see 11.8.

3.2 Pen-testing tools

Utilizing internet resources, we compiled a list of pen-testing tools including Metasploit[4] and BurpSuite[7], among many others. For a large portion of the tools, we were not able to obtain a license within a reasonable budget constraint for only four days of use. We also contacted IS&T to ask if they had educational licenses for security tools that MIT members could use, but unfortunately they did not have anything. Therefore, we chose tools that have either free trial or open source version. We focused mainly on Metasploit[4], BurpSuite[7], and IronWASP[3].

3.3 Pineapple

A WiFi Pineapple is a network auditing tool that facilitates man-in-the-middle (MITM) attacks. We took a WiFi Pineapple on-site to see if there was any data from the website that could be leaked.

3.4 Security Questionnaire

We didn't want to limit our security analysis to only the website for the web application itself, but also the security of the infrastructure supporting it. To that end, we put together a security questionnaire containing 40-50 questions on the company's security policies. This questionnaire contained topics ranging from password rotation policies and update frequencies to patching for known exploits like Spectre and Meltdown. Our hope was that the information provided through the answers to those questions might uncover indirect vulnerabilities that we could investigate further. When we arrived on-site, we attempted to interview our contact at the company using the security questionnaire. It was waived off as "secondary" as they already had something similar internally, and then was not revisited during our time on-site. We have included the full list of questions in the appendix, see 11.9.

3.5 Previous Reports

In addition to the 4 tasks above, we also spent time reviewing the previous assessments of the company, making sure that we understood all of the tactics used to produce the vulnerabilities that were previously found, in order to make sure that these things were fixed.

4 Rules of Engagement pt. 2

Once on-site, we received further instruction on how to approach the testing of the service, given WiFi credentials, the staging URL, and login credentials to the staging site.

We were also constrained in the following manner:

- We were permitted to test known usernames/emails and passwords (e.g. from Pwned Passwords), but limited to 500 total at a rate of at most 1 per second, and only against the staging server.
- We were asked to limit the load against the staging server so as not to crash it.

These constraints further restricted our testing strategies. The larger automated test suites also seemed to require lengthy setup procedures in order to tune them to the environment, and the load constraint limited how much automated scanning we could really perform in the short time we had on-site.

Due to these factors, both the brute force login attacks proposed in our initial proposal, and *most* of the automated testing suites were largely ineffective for us. Instead, we relied on a series of manual sleuthing and investigation techniques such as: using the browser console window, crafting exploits by hand, and testing certain areas via terminal *curl* calls.

5 Major vulnerabilities

5.1 User session hijacking

5.1.1 Cross-site scripting (XSS) vulnerability

A cross-site scripting attack can occur when a website does not perform any checks on input presented to it, allowing a malicious third-party to inject and run arbitrary JavaScript code in the client's browser by presenting a specific type of malformed input.

The company's Contact Us page exhibited such a vulnerability, which allowed us (and any such malicious third-party) to create and run JavaScript exploits using this site, as well as importing and running JavaScript from other websites. This vulnerability was first exposed in a previous report, with the ability to include such JavaScript directly in the URL parameters in a GET request. Upon re-testing the attack from the previous report, we discovered that it had indeed been protected against. However, a scan of the staging server with the IronWASP utility showed that such a vulnerability did still exist when performing a POST request to the server, where data was not being properly encoded and/or sanitized before being embedded into the page.

5.1.2 Missing cookie flags

There are multiple flags that can be set for cookies on a site. Two of these flags are the "HttpOnly" flag and the "Secure" flag. "HttpOnly" means that any JavaScript on the client side is prevented from reading the value of the cookie. "Secure" ensures that the cookie is only transmitted via HTTPS, not HTTP.

We found two vulnerable cookies with missing flags:

- The Session ID cookie was not set to "Secure", which meant it was accessible without encryption over HTTP traffic, and therefore possibly susceptible to MITM attacks. We were able to discover a user's specific Session ID by using the WiFi Pineapple.
- The User Session Token cookie was not set to "HttpOnly", which meant it was accessible to any JavaScript running on the page. This discovery was made primarily through trial and error.

5.1.3 Our Attack

Taking advantage of both of these finds, we were able to put together an exploit which would allow a third party to hijack a user's session, gaining full control over that user's account. A full video demonstration of this was presented in class.

Let Alice be a valid user, and Eve be any malicious third party. The exploit works as follows:

1. Alice logs in to the web application as normal.
2. Somewhere else on the machine, Alice is presented with a malicious link to the Contact Us page. This link uses the XSS vulnerability mentioned above to inject JavaScript code into the page.
3. The injected JavaScript runs, and is able to access the User Session Token cookie, which is not "HttpOnly" as mentioned above.
4. The injected JavaScript sends that User Session Token to Eve. For our specific exploit, we set up a server which would accept these tokens and make a list of them for easy use.
5. Eve goes to the patient website, and then inputs the cookie into her browser using standard built-in developer tools. This process could also easily be automated.
6. Without logging in, Eve now has full access to Alice's account.

5.1.4 Possible Solution

To prevent this specific attack, the company need only either patch the XSS vulnerabilities or add the missing cookie flag(s). In the interest of preventing similar attacks, we (of course) recommend doing both.

The XSS vulnerabilities can be fixed by ensuring that all input data is sanitized and properly encoded before embedding it into the website, regardless of origin or type of HTTP

request.

The missing cookie flags are fixed by simply adding the correct flags to the corresponding cookies.

Another step which should be taken is the inclusion of a Content Security Policy (CSP). CSP is a relatively new development over the past few years. It adds an HTTP header which helps protect against maliciously inserted JavaScript. [9][5] In addition to the same-origin-policy of the browser, it can help guard against XSS attacks by enforcing which scripts are permitted to run. It can even require a nonce for scripts executed directly within HTML (as opposed to being loaded as a separate file).

5.2 Buffer Overflow

5.2.1 Overview

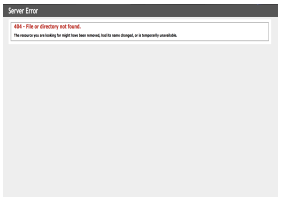
A buffer overflow attack is an attack where data is written to a buffer such that it "overflows" and corrupts data values in memory addresses adjacent to the destination buffer. Buffer overflow often occurs due to insufficient bounds checking. Although this suspected vulnerability was discovered in a previous assessment, it was not fully remediated and is still present in the web application.

5.2.2 Our Attack

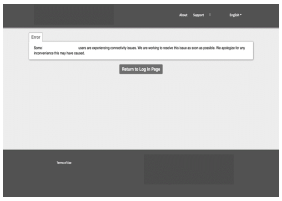
We were able to insert random number of characters into the URL that allowed us to see an internal error message that is different from typical "404 Not Found" error. Specifically, when we inserted more than 4068 characters into their contact page, the server displayed something different from usual (see Figure 1). In the previous report, the previous team found similar result by inserting 2048 characters. Note that we were unable to exploit this further in the given timeframe.

5.2.3 Possible Solution

There are a number of techniques to prevent this class of attacks. Prevention techniques are often specific to the programming languages used in building the system. The most direct solution would be to trim any user input when it reaches the maximum allotted buffer size.



Less than 4067 characters



More than 4067 characters

Figure 1: Different Error Messages

6 Other on-site vulnerabilities

We found other minor vulnerabilities, which we will list here for conciseness.

- Login flow exposes whether email address is registered or not. *See appendix 11.1*
- Bootstrap and jQuery with vulnerabilities (as reported by retire.js and Chrome console) *See appendix 11.4*
- Error pages showing stack traces and specific library versions used. Achieved through the following process:

  1. Log in
  2. Delete User Session Token (which will log you out)
  3. Log back in

  See appendix 11.2
- Developer's name and date visible in comments in non-minified JavaScript. *See appendix 11.3*

7 Potential dangers

In addition to the vulnerabilities discussed, we also found a number of issues which we saw as "yellow flags". These are things which are concerning or out of line with best practices, but are not necessarily vulnerabilities.

- All login cookies start with the same AAAAAgABA prefix
  - It seems that the company may be padding their cookies with something which may encrypt to the same thing every time. This implies that perhaps their cookie-encryption algorithm is not CCA2-secure (i.e. it does not appear random each time). [8].
- "Blocked loading mixed active content"
  - This was a warning we saw in the browser console. As a policy, secure pages (HTTPS) should never serve up insecure (HTTP) content. [6]
- Site is fingerprintable
  - We were able to use the IronWASP tool to detect certain parts of the site were served by an IIS 8.5 web server.[3]
  - In addition, in other parts of the site the headers that it was served via IIS 7.5. *See appendix 11.4*
- HTTP/1.1 being used, but loading many individual JavaScript files. These should be combined, or a switch should be made to HTTP/2.

8 Good security practices we observed

In addition to all of the issues we've listed, we'd also like to highlight things which we believe the company did well.

- Physical security/appearance
  - Their on-site security provided a well-guarded environment. Having a full NOC, SOC / VSOC also gave credibility to the fact that they take things seriously concerning the security of their site.
- Automated scans.
  - The company runs a large suite of automated scans on their own systems, which we were able to use in directing our vulnerability search.
- Cross-site Request Forgery (CSRF) protection
  - Their consent page appears to use a CSRF nonce token every time, so that the same submission can't be replayed with different inputs. This makes automating a SQL-injection-style attack more difficult.
- Two-Factor Authentication (2FA)
  - Two of our team members encountered 2FA when first trying to login to the site, which is a good sign. However, we are still unsure how it gets triggered. After security questions were properly filled in, it stopped prompting us and we were never prompted further about it.
  - The CISO has further let us know that they are actually using RSA Risk-based authentication to detect abnormal behaviours, which beyond a certain threshold triggers the 2FA.
- Click-jacking
  - According to the IronWASP utility, the site is well-protected against click-jacking attacks.
- No CDNs
  - Normally this is a negative (for website performance), but since they are pretty much hosting all their own content, they effectively can't be subject to a CDN poisoning attack.[1]
- Existing security policy
  - We have yet to receive any details of the company's security policy, but the fact that they have one is an indicator that they are aware and prepared for a variety of attack vectors.
- DDOS protection
  - From our cursory investigation (keep in mind we did no extensive production testing, per requirements) we did not detect any presence of Cloudflare[2], however the CISO has informed us that they are using Radware with Netscalar load balancing as preventive measures.

9 Conclusion

In summary, even through a controlled and constrained environment, we were able to successfully find and document multiple vulnerabilities with the company's patient portal web application. In the time allotted, we were also able to build a working exploit of these vulnerabilities, which under certain conditions allowed for a user's session to be hijacked. We believe this exploit is serious enough to require immediate attention, and through the responsible disclosure process, we have made the company aware of this exploit, as well as all of the other vulnerabilities and dangers which we uncovered. We believe that the company will have ample time to correct these issues before this paper is released.

Even though there were flaws found, the company seems to be reasonably well prepared and seems to take security seriously and attentively.

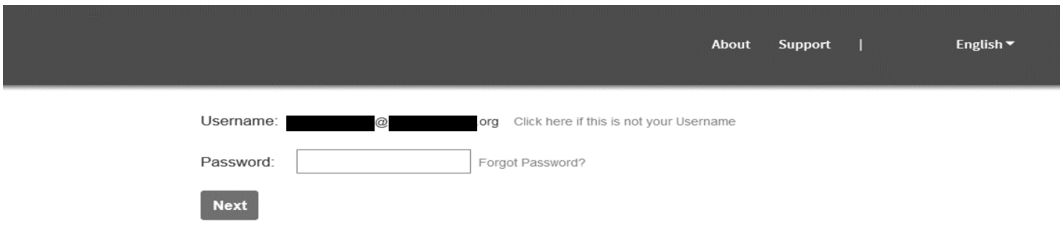
10 Acknowledgements

We'd like to thank the following for their support:

- Major Healthcare Company
  - Thank you for the opportunity, and hospitality while on-site.
- CISO of the Healthcare Company
  - Thank you for your support and assistance, especially while we were on site.
- Professor Ron Rivest
  - Thank you for connecting us with the company.
- Cheng Chen, TA for 6.857
  - Thank you for your support.

11 Appendix

11.1 User Flow

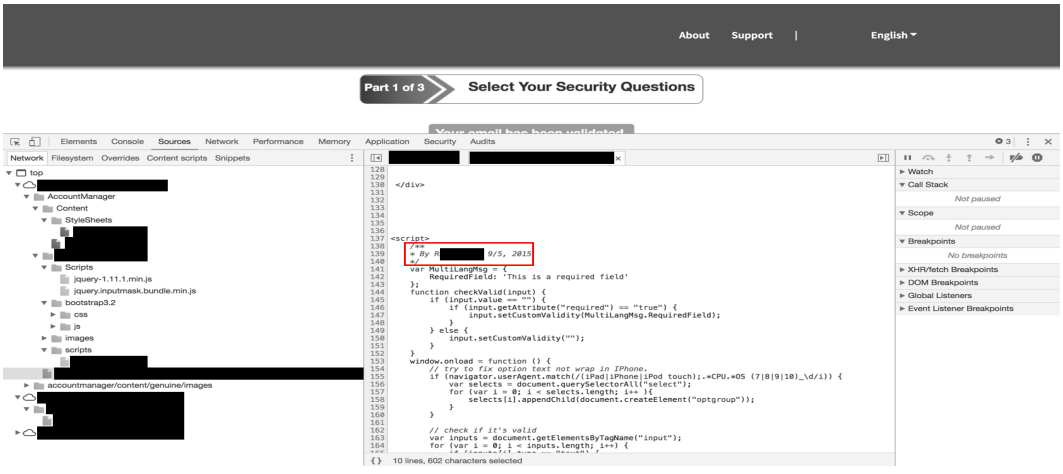


Login flow showing invalid account

11.2 Stack Traces

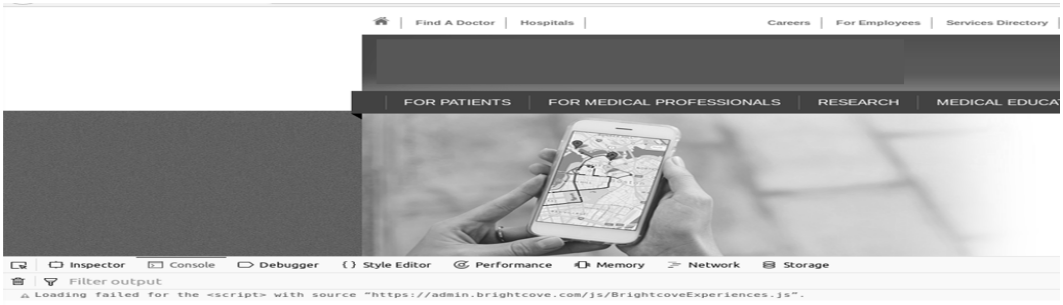
Error stack trace

11.3 Unminified Javascript

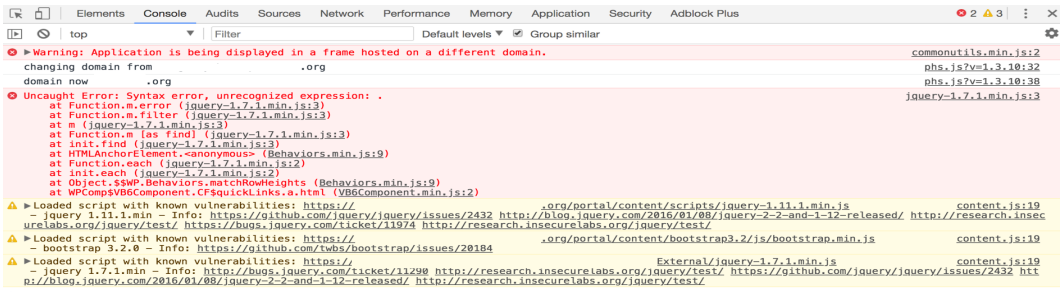


Visible comments

11.4 Vulnerable Scripts

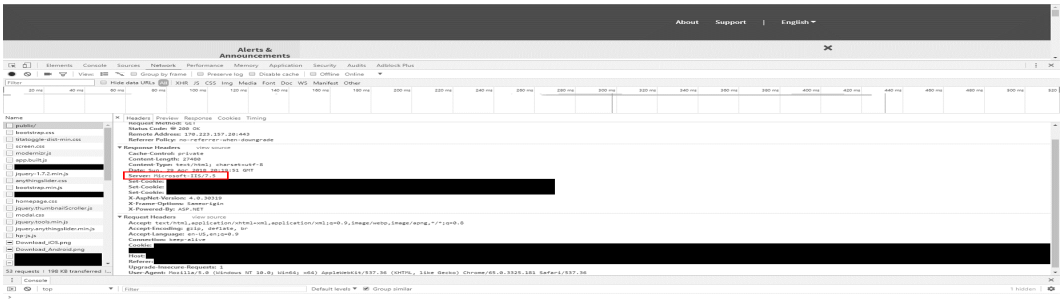


Loading Failed Internal Script



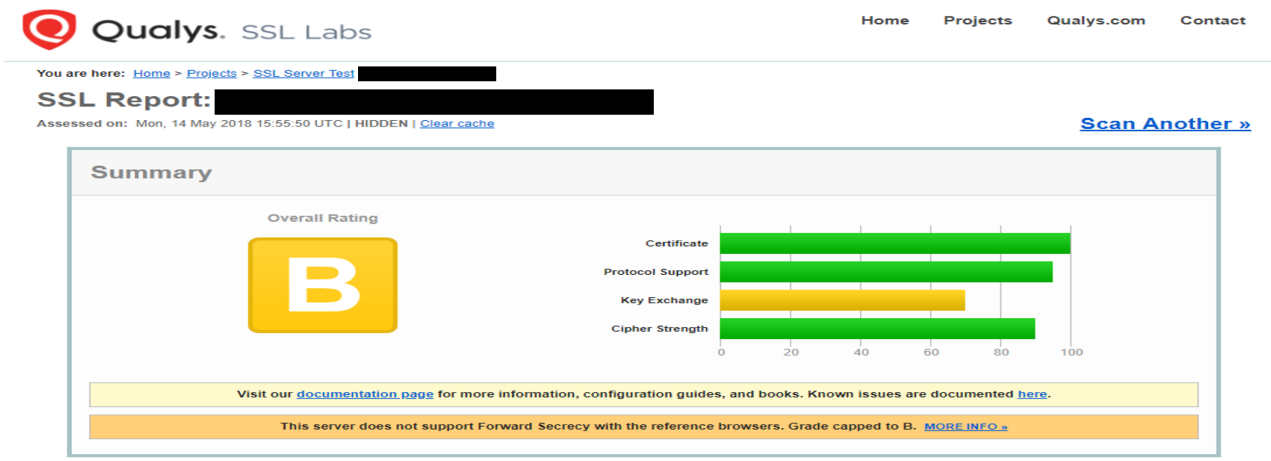
Bootstrap & jQuery Scripts

11.5 Internet Information Server

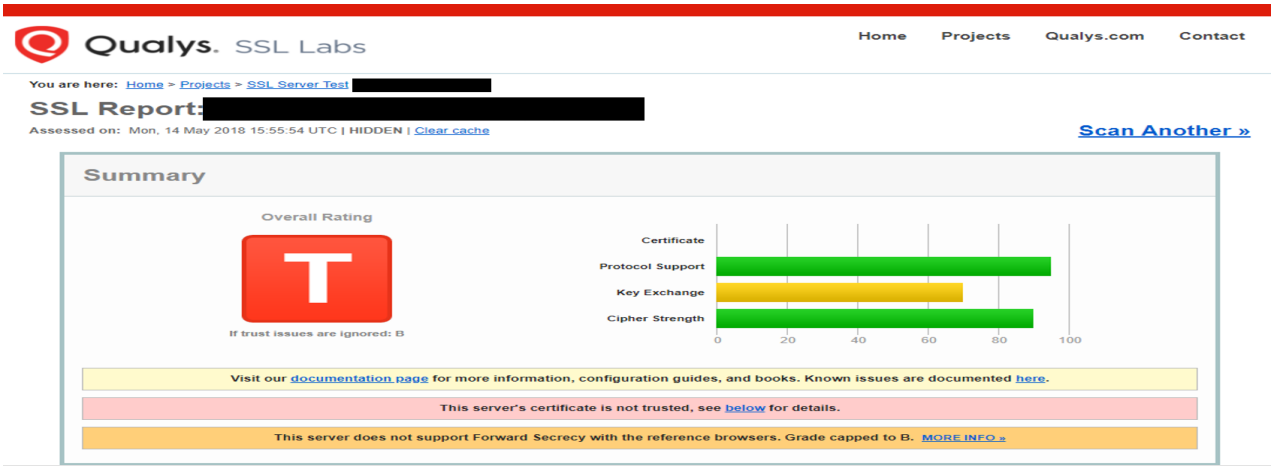


Loading Failed Internal Script

11.6 SSL



SSL test of www.REDACTED



SSL test of site with out www (REDACTED)

11.7 Tracert

```
tracert REDACTED

Tracing route to REDACTED [REDACTED]
over a maximum of 30 hops:

  1     6 ms     3 ms     2 ms  18.189.64.1
  2    15 ms     4 ms     5 ms  BACKBONE-RTR-1-OC11-WRTR-1.MIT.EDU [18.123.73.1]
  3     5 ms     5 ms     5 ms  DMZ-RTR-1-BACKBONE-RTR-1.MIT.EDU [18.69.1.1]
  4     5 ms     4 ms     4 ms  EXTERNAL-RTR-3-DMZ-RTR-1.MIT.EDU [18.69.7.2]
  5     4 ms     4 ms     4 ms  NOX-EXTERNAL-RTR-3.MIT.EDU [18.32.4.109]
  6     8 ms     5 ms     4 ms  18.32.4.6
```

```
  7    12 ms     6 ms     5 ms  REDACTED
  8     7 ms     6 ms     7 ms  REDACTED
  9     *        *        *     Request timed out.
 10     *        *        *     Request timed out.
 11     *        *        *     Request timed out.
 12     *        *        *     Request timed out.
 13     *        *        *     Request timed out.
 14     *        *        *     Request timed out.
```

11.8 DNS

```
    Default Server:  STRAWB.MIT.EDU
Address:  18.71.0.151

> q=any
Server:  STRAWB.MIT.EDU
Address:  18.71.0.151

*** STRAWB.MIT.EDU can't find q=any: Non-existent domain
> set q=any
> REDACTED
Server:  STRAWB.MIT.EDU
Address:  18.71.0.151

Non-authoritative answer:
REDACTED
        primary name server = REDACTED
        responsible mail addr = REDACTED
        serial  = 23894245
        refresh = 3600 (1 hour)
        retry   = 3600 (1 hour)
        expire  = 2592000 (30 days)
        default TTL = 300 (5 mins)
REDACTED    text =

        "teamviewer-sso-verification=a044c4da7dfb46859fac28671076a433"
REDACTED    text =

        "6go9j21nf4bgvjn16u1j2cf7vq"
REDACTED    text =

        "dropbox-domain-verification=owl9rzuulzsq"
REDACTED    text =

        "lYA94FJAcmIGnd274tfgtCdvLjSUuXn/5H4Y70Z5NT0gWel/pIXh1N+4P3qythYWuZSJg6HkA+di7aaJULFqKw==
REDACTED    text =

        "MS=ms64884954"
```

```
REDACTED    MX preference = 20, mail exchanger = REDACTED
REDACTED    internet address = REDACTED
REDACTED    nameserver = REDACTEDns1.REDACTED
REDACTED    nameserver = REDACTEDns2.REDACTED
REDACTED    nameserver = REDACTEDns1.REDACTED
REDACTED    nameserver = REDACTEDns2.REDACTED

REDACTED    nameserver = REDACTEDns1.REDACTED
REDACTED    nameserver = REDACTEDns2.REDACTED
REDACTED    nameserver = REDACTEDns1.REDACTED
REDACTED    nameserver = REDACTEDns2.REDACTED
REDACTEDns1.REDACTED  internet address = REDACTED
REDACTEDns2.REDACTED  internet address = REDACTED
REDACTEDns1.REDACTED  internet address = REDACTED
REDACTEDns2.REDACTED  internet address = REDACTED
```

11.9 Security Questionnaire

General

- Is there a company-wide security policy? If so:
  - Where is it documented?
  - Are all security / IT Professionals aware of it?
  - Is anyone else given access?
- Does the company use an independent security auditor or an independent security team? Are the operators of the site solely responsible for the security of the site?
- Is the company employing differential privacy methods of protecting anonymized data?
- Does the company have a security training? If so:
  - Who is required to participate?
  - What does it include?

Passwords

- What is the current password rotation policy (if any)?
  - Does this policy apply to database passwords as well? If not, what is the database password rotation policy (if any)?
- What are the requirements on passwords (i.e. how "strong" are passwords required to be)?
- Are passwords hashed before being stored? If so:
  - Are they salted before being hashed?
  - What hashing algorithm is being used?

- How is credential storage handled?
  - Is a credential manager (e.g. Hashicorp's Vault) in place?
  - Is a credential encryption scheme in place?
  - Are any application/database passwords stored in plaintext anywhere?

Administration

- Who patches/updates servers as new vulnerabilities are discovered?
  - How long does this process take?
  - How often is this checked?
- Who takes note of and response to CVE (Common Vulnerabilities and Exposures) alerts?
  - How are responses processed, and how long does this take?
  - Are the systems currently patched to mitigate the effects of Spectre and/or Meltdown?
- What is the process for when an IT employee and/or administrator leaves?
  - Is account (and/or VPN) access automatically revoked?
- Are servers accessible via VPN?
  - If so, are there any known weaknesses caused by this?
    - If so, how are those weaknesses mitigated?
- Is data encrypted "at rest"? That is, is data encrypted while static on an individual machine?
  - Is BitLocker/FileVault (or an equivalent) enabled on all Windows / Mac machines?
- Does the company have a current antivirus subscription installed on all Desktops / Laptops?
- Does the company have any email filtering for Phishing attacks, and/or malicious attachments?

External Network

- What firewalls are currently in place protecting the system?
- Is a NAT (Network Address Translation) protocol in place?
- Are necessary precautions taken to prevent port forwarding to individual machines?
- Is there a DMZ?
- Who patches/updates network devices?
  - How long does this process take?
  - How often does this occur?

Application Development

- Who audits applications for security vulnerabilities after they're created?
- Do the auditors do standard pen-testing on all new applications?
- Do the auditors do standard pen-testing on all third party apps?
- How does the company handle 3rd party application authentication - is there a company wide SSO, or is it fragmented by vendor?

11.10

References

[1] Benjamin Brown. *On Web Cache Deception Attacks*. URL: `https://blogs.akamai.com/2017/03/on-web-cache-deception-attacks.html`. (accessed: 5.15.2018).

[2] cloudflare.com. *Cloudfare*. URL: `https://www.cloudflare.com`. (accessed: 5.15.2018).

[3] ironwasp.org. *IronWASP*. URL: `https://ironwasp.org`. (accessed: 5.14.2018).

[4] metasploit. *metasploit*. URL: `https://www.metasploit.com/`. (accessed: 5.15.2018).

[5] mozilla.org. *Content Security Policy (CSP)*. URL: `https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP`. (accessed: 5.14.2018).

[6] mozilla.org. *How to fix a website with blocked mixed content*. URL: `https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content/How_to_fix_website_with_mixed_content`. (accessed: 5.14.2018).

[7] portswigger. *Burp Suite*. URL: `https://portswigger.net/burp`. (accessed: 5.15.2018).

[8] Various. *Ciphertext indistinguishability*. URL: `https://en.wikipedia.org/wiki/Ciphertext_indistinguishability`. (accessed: 5.14.2018).

[9] Various. *Content Security Policy*. URL: `https://en.wikipedia.org/wiki/Content_Security_Policy`. (accessed: 5.14.2018).