Enhancing NUSNET using Two-Factor Authentication

Vaarnan Drolia, Eugene Huang

National University of Singapore
School of Computing
Singapore, 117417
vd@nus.edu.sg, eugenehuang@nus.edu.sg

Abstract. This paper describes our evaluation of the security of the National University of Singapore Network's (NUSNET) authentication system with respect to a few common attacks. It also recommends the implementation of a Two-Factor Authentication System to mitigate these vulnerabilities.

1 Introduction

The National University of Singapore uses a userid and password combination to give students and staff access to the NUS services listed in section A.1. This is essentially a single point of failure and it leaves a user susceptible to the complete compromise of identity and services if one can obtain the user's password. The full impact of the attack is discussed in Section 3 .

We discovered one such service which has an insecure channel of authentication which can lead to compromise of the user's credentials. Notwithstanding, there are several other ways that the system can be compromised such as Cross-Site Scripting vulnerabilities which have been commonly found in the NUS Integrated Virtual Learning Environment (IVLE) System (Camillus and Kwan).

A possible way to protect users from too much damage is to have Two Factor Authentication upon login and also have some areas on the website which are more privileged and secured via 2FA. These sections are especially important for administrator accounts as well as those of lecturers and instructors who might be storing details such as grades on the system.

Note, all the details about the NUS Network are based on our investigations conducted external to the system and may not fully match the details and specifications of the internal system.

2 New Attack

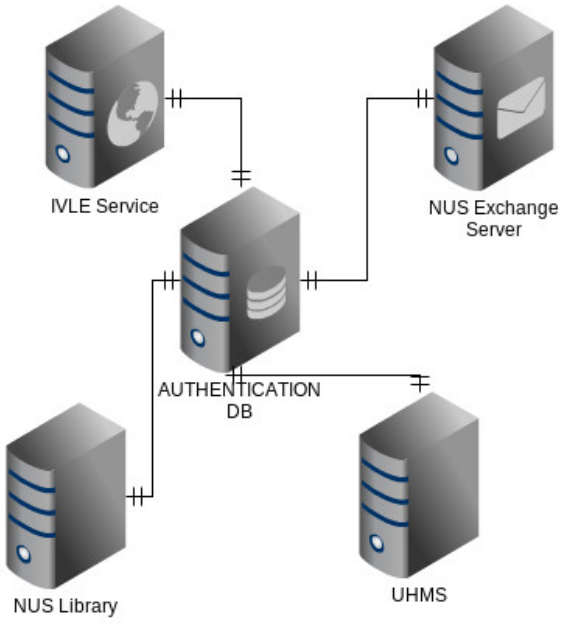
The security of a system is only as strong as its weakest link and any NUS service using a weak form of authentication will compromise all the services. An example is the UHMS website which does not use Secure Sockets Layer (SSL) and opens up a wide array of insecurities. One possible attack is a network sniffing attack

as passwords are sent in cleartext and can be retrieved by the use of a network sniffing tool such as wireshark. Appendix section B.1 demonstrates our attack showing this vulnerability.

3 Impact of the Attack

Every student in NUS is issued with one set of NUSNET ID and password and this can be used to access all the web services in NUS. Once the password is compromised, a malicious user can gain access to the personal information of the victim, gain access to all the modules, can perform denial of service by dropping the victim's module through CORS, can access the victim's email account and invade the privacy of the user and can even abuse the email account to send spam and malicious emails to the rest of NUS or perform social engineering on other users. This can effectively cause even more accounts to be compromised. Access to the SoC Sunfire and Tembusu/Angsana Cluster accounts can also be granted. Even access to the Wireless Network within NUS is open to the attacker giving rise to the possibility of hacking from internal networks. Appendix A.1 gives a small list of affected services.

4 System Design



4.1 Status Quo

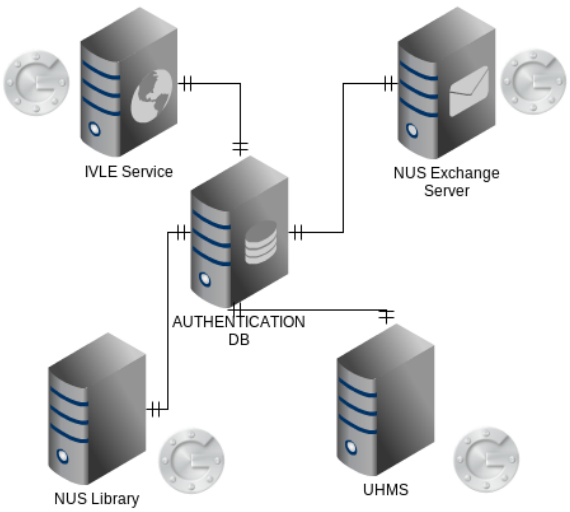
The current NUSNET system seems to have an architecture similar to the one shown in figure 4. The reason we believe this is that after sampling HTTP headers, we found most of the NUS services run autonomous of each other and have different servers and network configurations (Appendix A.2 and A.3). They also manage sessions differently and being logged in with one service does not authenticate a user with another service.

However, the credentials used by each service is common for all the services, therefore, we can conclude that there is a centralized database server/authentication server which has the NUSNET login details and each of these servers in turn communicate with this central server. The authentication server probably returns true or false as to whether the username/password combination is correct.

Session cookie From the HTTP headers (Appendix A.2), we found that NUS communicates this data over Basic Authentication but this is not as insecure but is widely accepted in the industry since the communication is done over HTTPS so it is relatively safe from snooping (StackOverflow). However, the vulnerable UHMS website also does the form submission in plain text without HTTPS which is insecure.

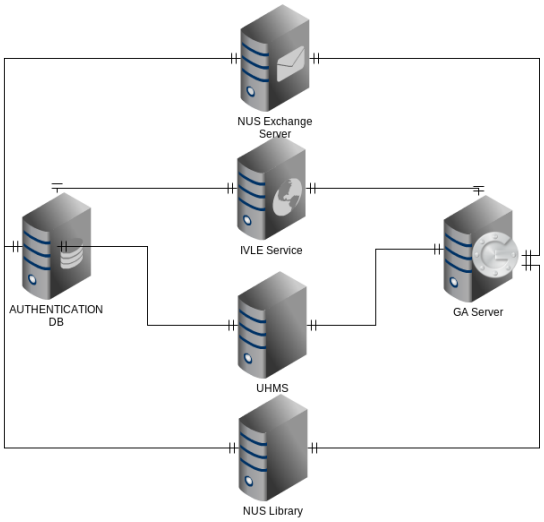
4.2 NUS-2FA Section

There are three different configurations that we can propose for the Two-Factor Authentication system and we will discuss the design and security evaluation of all three in this section choosing Google Authenticator as a representative example.



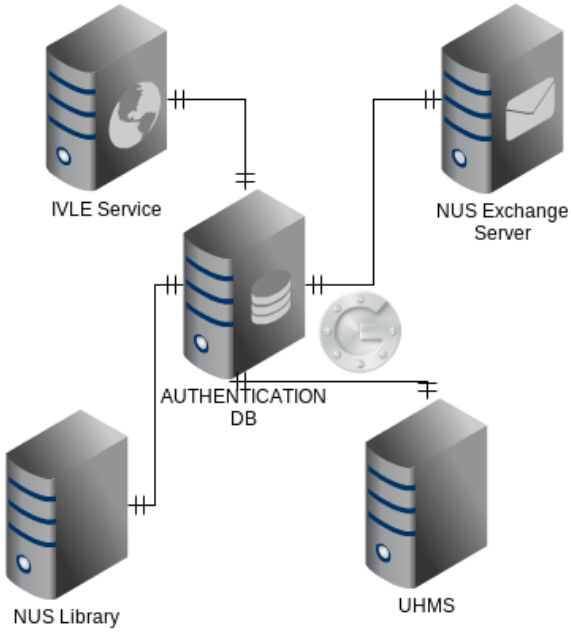
4.3 Distributed

Each server which hosts an NUS service will also host the Google Authenticator software and for each request which is passed on to the authentication DB, the secret key of the Google Authenticator will be passed back to the NUS service server for verification of the OTP.



4.4 Two Authentication Servers

In this system, there are two servers, one which has the authentication database server which verifies the username/password combination while the Google Authentication server stores the secret key of the users and verifies the OTP.



4.5 Single server for 2FA and authentication

This system will have the username/password/secret key combination for the user. The different services will redirect to a page created by this server and it will have a callback url for the service's url.

5 Two-Factor Authentication

There are many ways that a Two-Factor authentication system can be implemented and several of them can turn out to be quite poor if they do not have proper peer review and open design principles. Thus, we chose to analyse an existing open-source system which is widely being used and is based on two RFCs which have been thoroughly peer reviewed apart from the fact that the creators of the RFCs are notable security experts.

5.1 Google Authenticator

Google Authenticator is a Two-Factor Authentication system which supports the HMAC-Based One-time Password (HOTP) algorithm specified in RFC 4226 and the Time-based One-time Password (TOTP) algorithm specified in RFC 6238.

5.2 RFC4226 HMAC-Based One-time Password (HMAC-OTP)

Description HMAC-OTP uses two parameters to generate its hashes. Namely, a secret key and a counter. The secret key is known only to the user and the validation service and the counter is an increasing value. In order to generate an OTP from it, it uses the HMAC-SHA1 algorithm to generate a 160-bit output. The output is further truncate to an OTP for the user to easily enter it.

Implementation The generation of OTP can be broken down into three individual steps, namely, hashing, dynamic truncating and computing. First, a 20byte hash H is obtained by H = HMAC-SHA1(k,c), where k is the secret key and c is the current counter value. In the dynamic truncating phase, the last two bytes are used to select four bytes from H given by TH = Dynamic-Truncate(H). One bit is dropped from TH to ensure that it converts to an unsigned integer in the next phase. In the computing phase, TH is converted to a number and the number is modulus with 10^D to obtain a output in the range of 0 to 10^{D-1}. The final returned value is given by OTP = (TH mod 10^D).

5.3 RFC6238 Time-based One-time Password (TOTP)

Description TOTP is an extension to the HMAC-OTP algorithm that instead of using an increasing counter value, a time value is computed based upon the current time and an agreed upon time step value between the user and the validation service.

Implementation The general implementation of TOTP would be the same as HMAC-OTP with some key variation. Instead of using an increasing counter C in the initial hashing, a time-value T will be generated from the current time W and time-step X. It is given by T = Floor((W-T_0) / X). T_0 is usually 0 or can be set by the system.

5.4 Implementation of Google Authenticator

HMAC-OTP pseudo code

```
\usepackage{mathtools}
\DeclarePairedDelimiter{\ceil}{\lceil}{\rceil}
function GoogleAuthenticatorCode(string secret)
     key := base32decode(secret)
     counter := get_hotp_counter()
     hash := HMAC-SHA1(key, counter)
     offset := last nibble of hash
     truncatedHash := hash[offset..offset+4]  //4 bytes starting at the offset
     Set the first bit of truncatedHash to zero  //remove the most significant bit
```

```
    code := truncatedHash mod 1000000
    pad code with 0 until length of code is 6
    return code
```

TOTP pseudo code

```
function GoogleAuthenticatorCode(string secret)
    key := base32decode(secret)
    message := current Unix time  30
    hash := HMAC-SHA1(key, message)
    offset := last nibble of hash
    truncatedHash := hash[offset..offset+4]  //4 bytes starting at the offset
    Set the first bit of truncatedHash to zero  //remove the most significant bit
    code := truncatedHash mod 1000000
    pad code with 0 until length of code is 6
    return code
```

5.5 Description of NUS + 2FA

2FA for logging in Whenever the user has keyed in their username and password, a secondary prompt is displayed to the user for him/her to enter a One-Time-Password(OTP). The user will have to refer to his google authenticator application on his mobile to enter the OTP. The application will have to be configured the first time the user sets up his account for 2FA authentication.

2FA for privileged content For certain privilege content on the website, the user will again be prompted to enter a OTP before he can continue browsing the website.

Description of Current Prototype We have developed a prototype of the Google Authenticator Two Factor Authentication system using an API key of IVLE LAPI. It is developed using the Ruby on Rails web framework and uses Google Authenticator for the Two-Factor Authentication Part.

Another extension that we have added is the ability of defining "privileged services" where another OTP is required since it would be able deal with attacks related to the cookie being compromised as in Section B.2 .

An in-depth security analysis of this prototype will not reflect the true nature of the solution since this is relying on an external endpoint so the evaluation in Section 6.3 should be referred.

Prototype URL: `http://www.github.com/vellvisher/nus-otp`

6 Evaluation

6.1 Evaluation of NUS System

NUS System requires its student to only remember a set of credential for all their school needs. Although this allows ease of use and in some form adhere

to the design principle of psychological acceptability, it violates the principle of single point of failure and separation of privilege.

Single Point of Failure (SPOF) - As a single credential is reused across multiple services in NUS, it is likely that there exist a centralised server for authentication and it could present a single point of failure. A SPOF is a potential risk to Denial of Service (DOS) attack on NUS. In the event that a DOS attack was performed and the attacker managed to bring enough traffic to the server such that it is not responding to new authentication request and all the services provided by NUS that requires user authentication will be affected since it is the SPOF. This is based on the assumption that there aren't considerations for redundancy and there is no mirroring.

Separation of Privilege - NUS allows students to login with only a single set of credentials (username and password) and then they will be able to access all parts of the website regardless of the sensitivity or importance of the action performed. For example, dropping a module on cors can be very disruptive to a student if it was intentionally dropped by a malicious attacker to cause pranks after gaining access to the students account. A higher privilege level should be expected for certain kind of more important or sensitive action and thus require a second form of authentication confirmation to be able to proceed further. For example, having a 2FA authentication with mobile phone will make it harder for an attacker to do bad stuff to the student as they will need to compromise both the username/password and the users phone.

For the distributed configuration of NUS-2FA, a possible advantage for this configuration is if the NUS service only requires an OTP for privilege content but not for login, then this configuration can allow the flexibility of which NUS service is using the google authenticator server. A possible disadvantage for this configuration is that the secret key needs to be transmitted and could be subjected to possible compromises of the key.

The 2-server configuration would be useful when one of the server is compromised but not the other. A attacker will need to compromise both servers in order to gain access to the NUS systems.

For the single server configuration, it would make it more simple to implement but require a stronger security on the server to protect the two databases.

The configuration between the 2-server and single server is closely similar to each other and a decision to choose between the 2-server or single server will be dependent on the infrastructure and cost available to support the system.

6.2 Evaluation of Google 2FA

Economy of mechanism - The security algorithm used for Google 2FA is sufficiently small and simple as to be verified and implemented. It involves the use of a one way hash, a dynamic truncation algorithm and a conversion to obtain the OTP as mentioned in the description of RFC4226.

Psychological acceptability - Allows the use of QR code or enter a short code to configure the application. Does not require any complicated step to confuse the user. The application on the phone can be downloaded and configured with

the QR code and it is successfully setup and usable within a few steps. Also, The OTP generated is 6 numbers long which is easy for a user to enter while also satisfying the security of having a huge enough number such that it is not susceptible to brute force guessing within the allocated window of the OTP, which by default would be 30 seconds.

6.3 Evaluation of NUS/2FA

With the introduction of NUS/2FA, students will need to provide an OTP key they login or access a certain privilege content on the NUS services. This will prevent their account from being accessible in the event that their username and password was sniffed on an unsecured nus service (eg. the ohms website). A point to note is that even though attackers arent able to get access to their NUS services, if these students practices bad password habits by using the same password for multiple accounts (email, facebook etc), the exposed password can also compromise these accounts.

If a DDOS is performed, it could be tough to block out all connections into the server. In this case, the server can temporarily blocked out authentication for students who have repeatedly keyed in the wrong OTP in sequence. The server can check the login id against a table of blocked students before performing any authentication on the student.

For privileged content on the NUS services, a cross site request forgery attack will not be able to be performed with the introduction of the 2FA OTP for these areas. This will reduce the motivation for malicious attackers to target NUS students.

7 Conclusion

Security is often an afterthought in most organizations and it is always regarded as a good practice in hindsight.

The NUSNET system operates on the same principle and is archaic and not keeping ahead of the times by adopting newer more secure technologies but instead going on the road of disaster management rather than disaster aversion.

We have proposed a state-of-the-art system of Two-Factor Authentication which is built on open standards, is peer reviewed and highly accessible due to its availability on a large range of mobile devices.

The algorithm is simple and easy to implement on existing server systems as we have demonstrated with our prototype.

Thus, we conclude that using a Two-Factor System of Authentication will greatly enhance the security of the NUS Network.

8 Acknowledgments

We thank Dr. Hugh Anderson for his help throughout the project, being supportive of our efforts and providing us with advice as we met with obstacles one after another.

9 References

1. Camillus Gerard Cai, Kwan Yong Kang Nicholas. "Cross Site Scripting: Case Studies and Mitigation", 7 Nov 2012. http://www.comp.nus.edu.sg/~hugh/CS2107/cs2107AllProjects.pdf
2. StackOverflow. http://stackoverflow.com/questions/5511589/securing-an-api-ssl-http-basic-a
3. M'Raihi, et al. An HMAC-Based One-Time Password Algorithm. December 2005. https://tools.ietf.org/html/rfc4226
4. M'Raihi, et al. TOTP: Time-Based One-Time Password Algorithm. May 2011. https://tools.ietf.org/html/rfc6238
5. http://code.google.com/p/google-authenticator/
6. http://en.wikipedia.org/wiki/Cross-site_request_forgery
7. Patrick McDaniel. Pennsylvania State University. Authentication. September 18, 2006. http://www.patrickmcdaniel.org/pubs/mcdaniel-netauth.pdf
8. "What's a "single point of failure", and why do I need to know?". http://ask-leo.com/whats_a_single_point_of_failure_and_why_do_i_need_to_know.html
9. http://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF

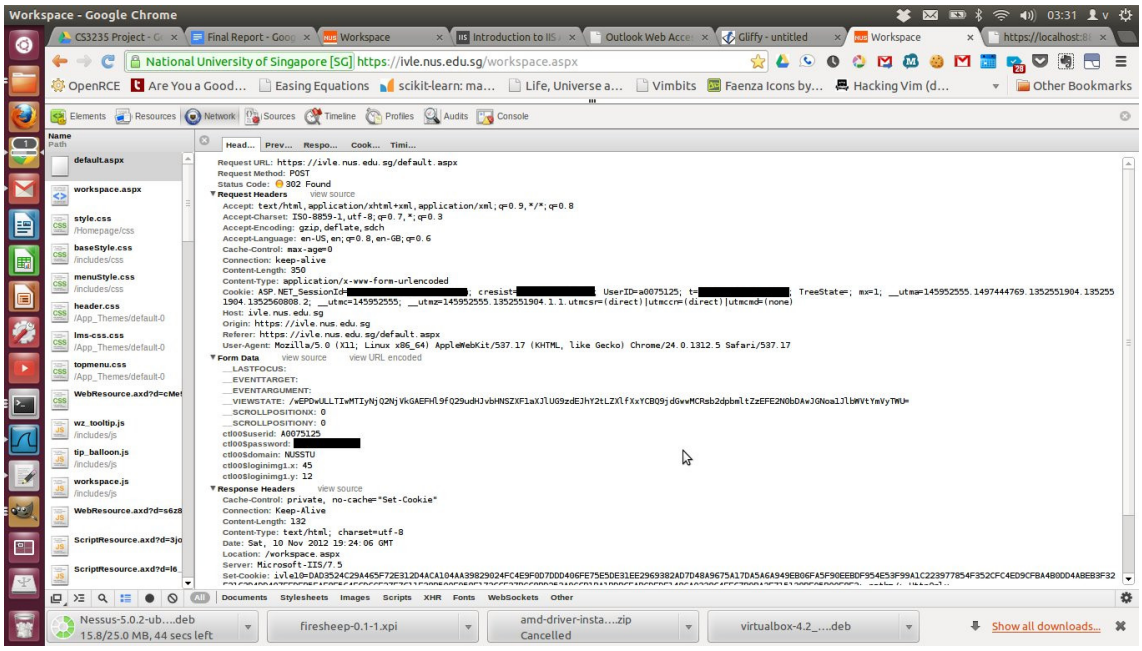
A Appendix A

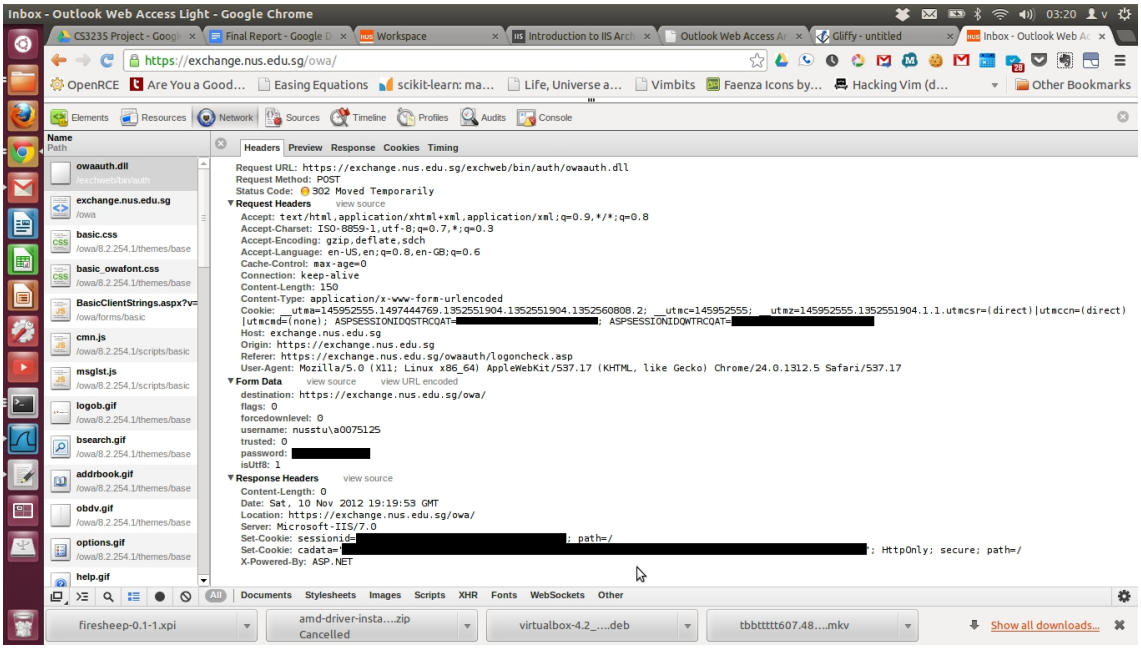
A.1 Some of the NUS Services using NUSNET Authentication

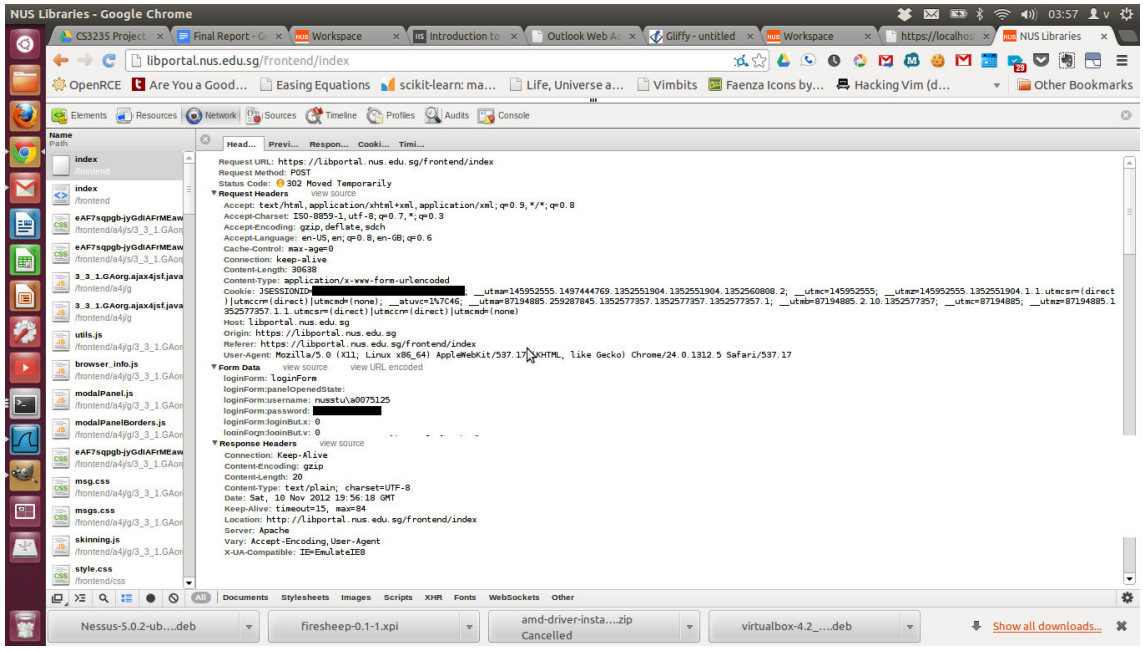
This list is by no means exhaustive and is just based on a preliminary survey.

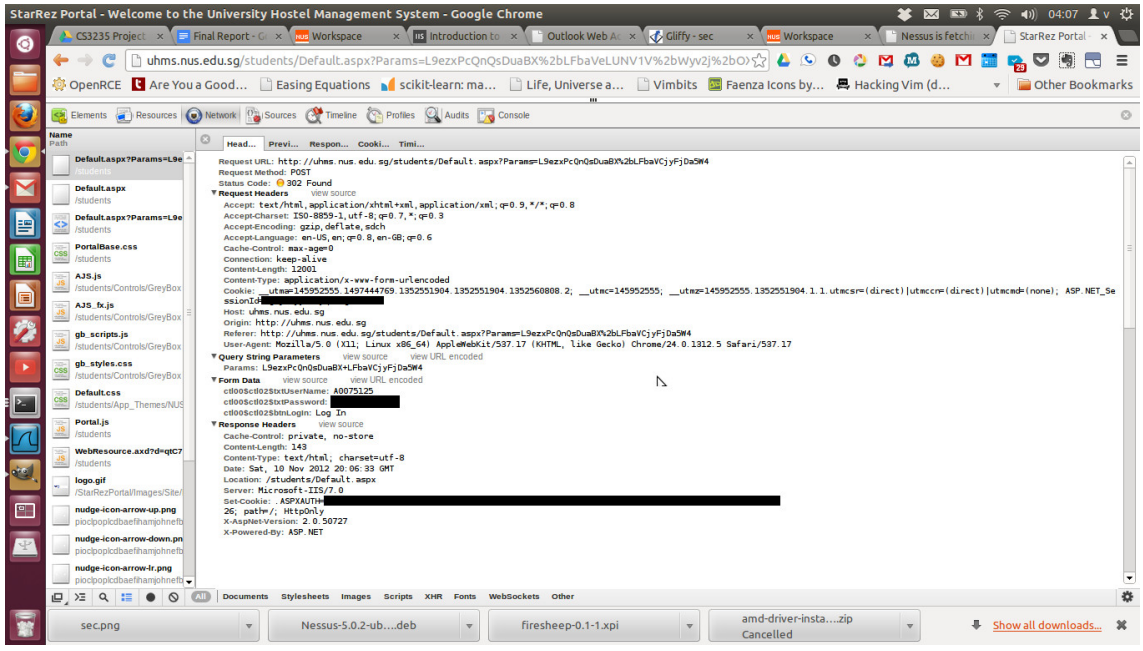
- ⋆ NUS Exchange Email
- ⋆ CORS
- ⋆ mySoC
- ⋆ SoC Account Password Reset
- ⋆ SoC Computing Clusters - Tembusu and Angsana
- ⋆ IVLE
- ⋆ NUS Wi-Fi
- ⋆ University Health Management System
- ⋆ MyISIS

A.2 HTTP Headers from NUS Services






NUS Libraries - Google Chrome
libportal.nus.edu.sg/frontend/index
Request URL: https://libportal.nus.edu.sg/frontend/index
Request Method: POST
Status Code: 302 Moved Temporarily
Request Headers
Form Data
Response Headers
Nessus-5.0.2-ub....deb
firesheep-0.1-1.xpi
amd-driver-insta....zip
Cancelled
virtualbox-4.2_....deb
Show all downloads...


StarRez Portal - Welcome to the University Hostel Management System - Google Chrome
uhms.nus.edu.sg/students/Default.aspx?Params=L9ezxPcQnQsDuaBX%2bLFbaVeLUNV1V%2bWyv2j%2bO
Request URL: http://uhms.nus.edu.sg/students/Default.aspx?Params=L9ezxPcQnQsDuaBX%2bLFbaVCjyFjDa5W4
Request Method: POST
Status Code: 302 Found
Request Headers
Query String Parameters
Form Data
Response Headers
sec.png
Nessus-5.0.2-ub....deb
firesheep-0.1-1.xpi
amd-driver-insta....zip
Cancelled
Show all downloads...

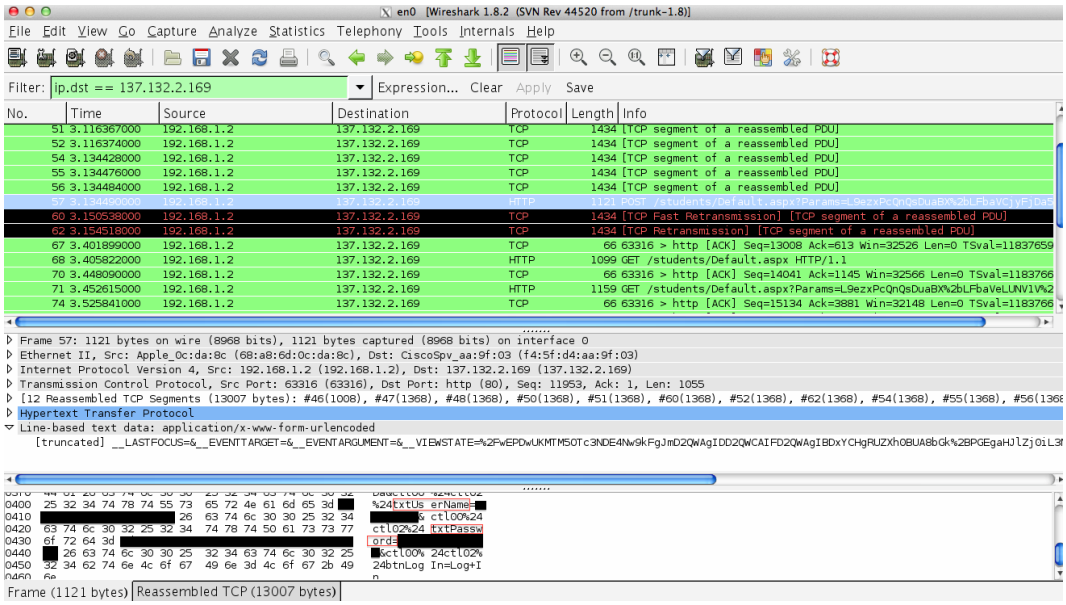
A.3 NUS Services Configuration Table

Service	Server	Version
IVLE	IIS	7.5
NUS Exchange	IIS	7.0
UHMS	IIS	7.0
Library NUS	Apache	X

B Appendix B

B.1 Login Sniffing

The wireshark screenshot below captured a packet sent to perform a login into the uhms.nus.edu.sg server. In line no 57, we can observe a POST request to /students/Default.aspx which is used to process the login. If we examine the Line-based text data in the packet, we can observe txtUsername and txtPassword to be sent in clear-text. The username and password below have been darken out for security purposes.



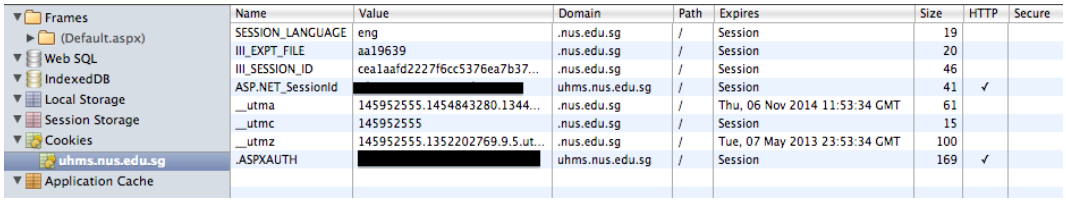
Wireshark-Login

B.2 Session Hijacking

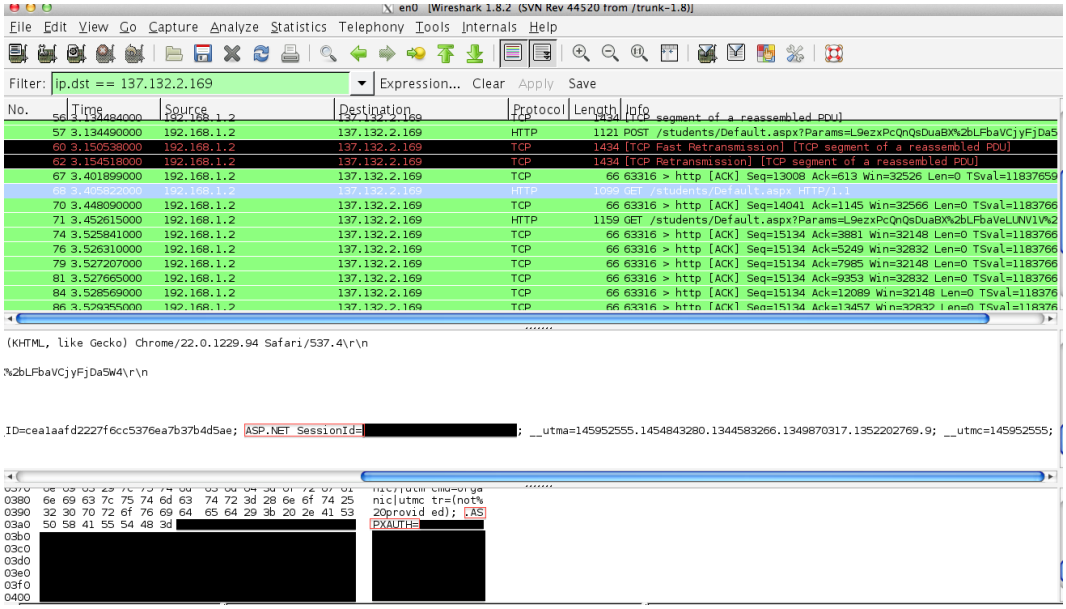
The web inspector screenshot below shows the cookie data for the uhms.nus.edu.sg server. Some of the cookie data are created with a HttpOnly flag so that the value

cannot be accessed through a client-side script. However, because the connection to the uhms.nus.edu.sg server is only via a HTTP-request instead of HTTPS, we could run wireshark to capture the cookie data that is actually sent across in plain-text. The wireshark screenshot below shows we can capture all the cookie data including the ASP.NET_SessionId and _ASPXAUTH data. We can then used a separate computer and pre-populated our cookie with the captured data and we were able to gain access to the uhms.nus.edu.sg server.

This attack cannot be performed on the other NUS services as the web requests are made over a HTTPs connection hence we were not able to capture the cookie data.



WebInspector-Cookie



Wireshark-cookie

C Appendix C

C.1 Analysis on the DBS Secure Token

We discarded our previous project idea on cracking open the DBS secure token as we ran into a dead end after we couldnt separate the e-proxy protection from the micro-controller on the chipset. However, we have made some findings along the way and this appendix will serve to cover what we have done.

Original state of the Secure Token

In order to pry open the secure token, we tried pulling the keyring out from the token but we realised that the token is sealed from on all ends and the keyring is just an external attachment to the casing. We used a screw driver and a hammer to dig deep into the sides along the seal lines. The casing itself is made up of plastic of different strength. The front part can be easily peeled off part by part with a screw driver but the bottom part is much stronger and it takes much hammering just to create a dent. We worked on the softer part around the token until we we have a hole around the token. Fortunately, the hard plastic on the rear side of the token has a slight round inwards design that exposes part of the board below it. We stuck the screw driver in between and slowly pry open the bottom side as shown in the diagram below.

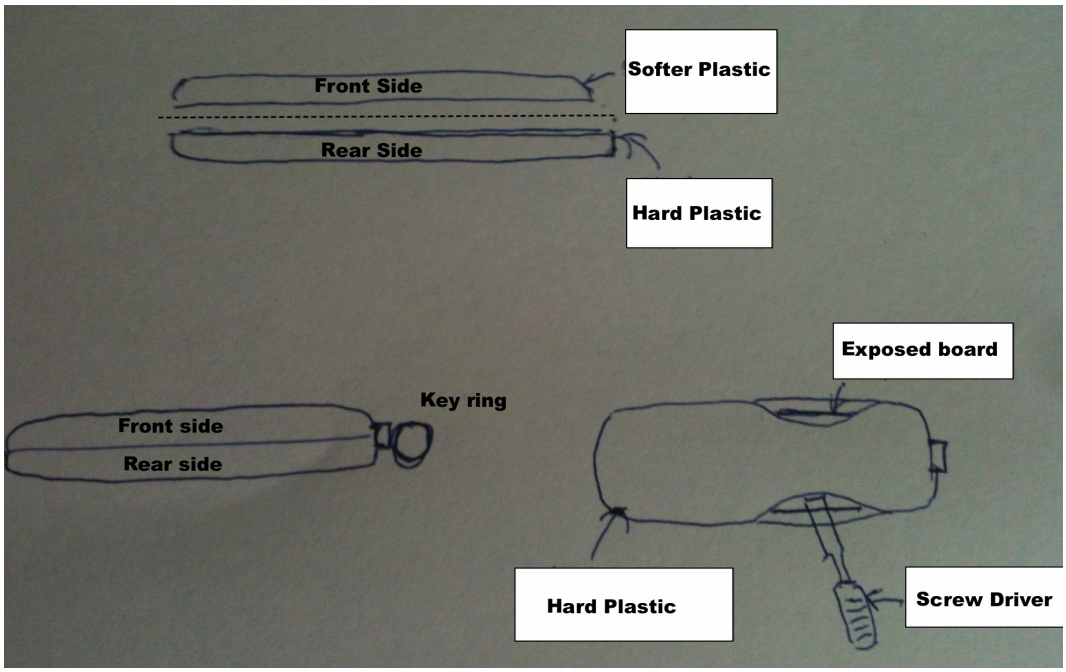
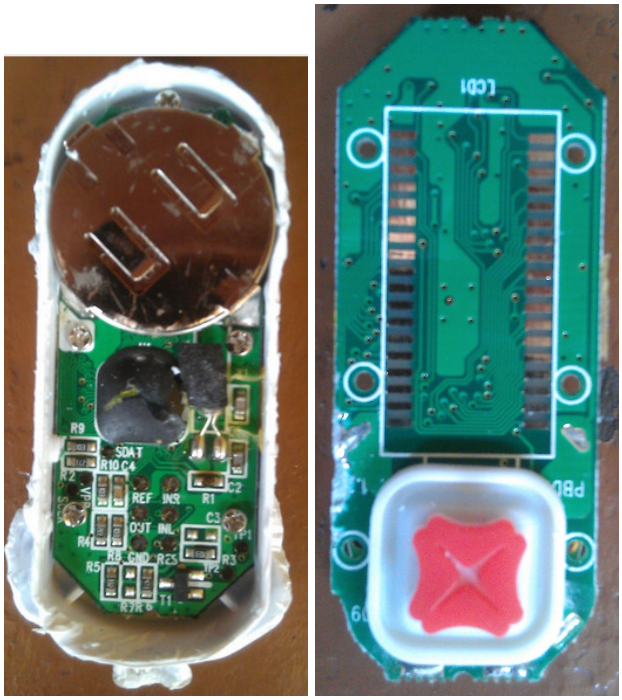
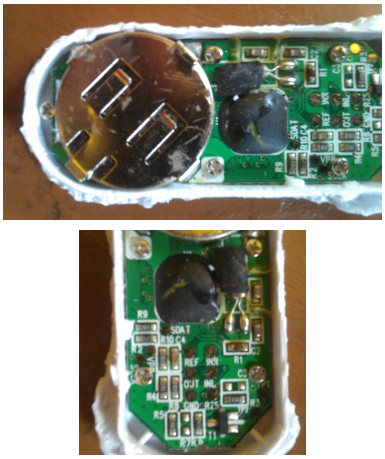


Illustration of how we pry the token open

After we managed to removed the board from the casing, the diagram below shows how the board is designed. We managed to establish that the crystal is running at a clock rate of 32khz by measurement without needing to examine beyond its protection e-proxy. Next, we wanted to remove the huge blop of e-proxy resting on the micro-controller.



Left- Rear side of the token, Right - Front side of the token

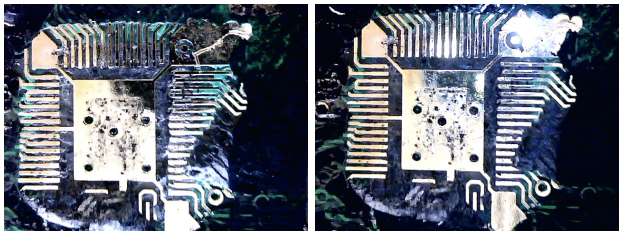


Close up view of the back end

We heated the e-proxy with a candle but instead of loosening up the structure of the e-proxy, it became very brittle and started to fall apart. We tried to salvage the small pieces but we were still unable to determine the micro-controller used in the token.

Internal overview of the Secure Token with bits of piece of the e-proxy and micro-controller

There is a total of 60 pins with some of them connected to ground. A small portion of the connectors may have been removed together with the e-proxy when we tried to rip it out. Due to the lack of information, we were unable to determine the micro-controller from the findings we have collected.



Conducting plate where the micro-controller was resting on