Appointment & Scheduling Management Information System

Application Prototype control testing guide

Executive Summary

The Appointment and Scheduling Management Information System (ASMIS) application stores confidential patient information. The ASMIS application is intended to resolve two somewhat opposing goals; government regulation (Information Commisioner's Office, 2020) requires health information be protected from harm or loss while authorized users expect the application is internet accessible and easy to use. Consequently, multiple layers of security controls must be implemented within the application design.

ASMIS is anticipated to be a multitiered web application utilizing multiple protection layers which increases the complexity of deploying a suitable small-scale model for assessment. Therefore, a prototype of the authorized user login process, (fig. 1), which implements many of the of key security controls recommended for the application has been created. The limitations of a single Python script and command line interface dictate communications to external systems are simulated through screen messages so testers and reviewers understand what data would be transmitted outside the system.

The prototype also stores both telephone numbers and passwords within the source code. In addition to resolving a technical requirement, storage of this information is representative of the need to store confidential personal data somewhere within the ASMIS system making it suitable for demonstration within the prototype as this is a key ASMIS requirement. At an abstract level the steps required to encrypt, store, access and decrypt data do not change on larger distributed systems, although the technical implementation will be significantly more complex. The General Data Protection Regulation (GDPR) definition of personal data (Intersoft Consulting, n.d.) includes telephone numbers justifying the need for a sound technical protection measure such as encryption even with the limited prototype data set.

In addition to accepting user input, the ASMIS application prototype provides response messages that would be typically presented as web page content to the user interacting with the portal. The prototype provides indications of which application controls are currently being assessed as well as security log entries related to the user interaction that will be forwarded to a security monitoring service. A slight delay has been intentionally included in each control step indicator to simulate waiting periods user would normally experience interacting with the application. The test cases and inputs identified in the following analysis should be incorporated into the testing activity needed for commissioning the fully functional application as they align to abuse cases identified in the original ASMIS report.

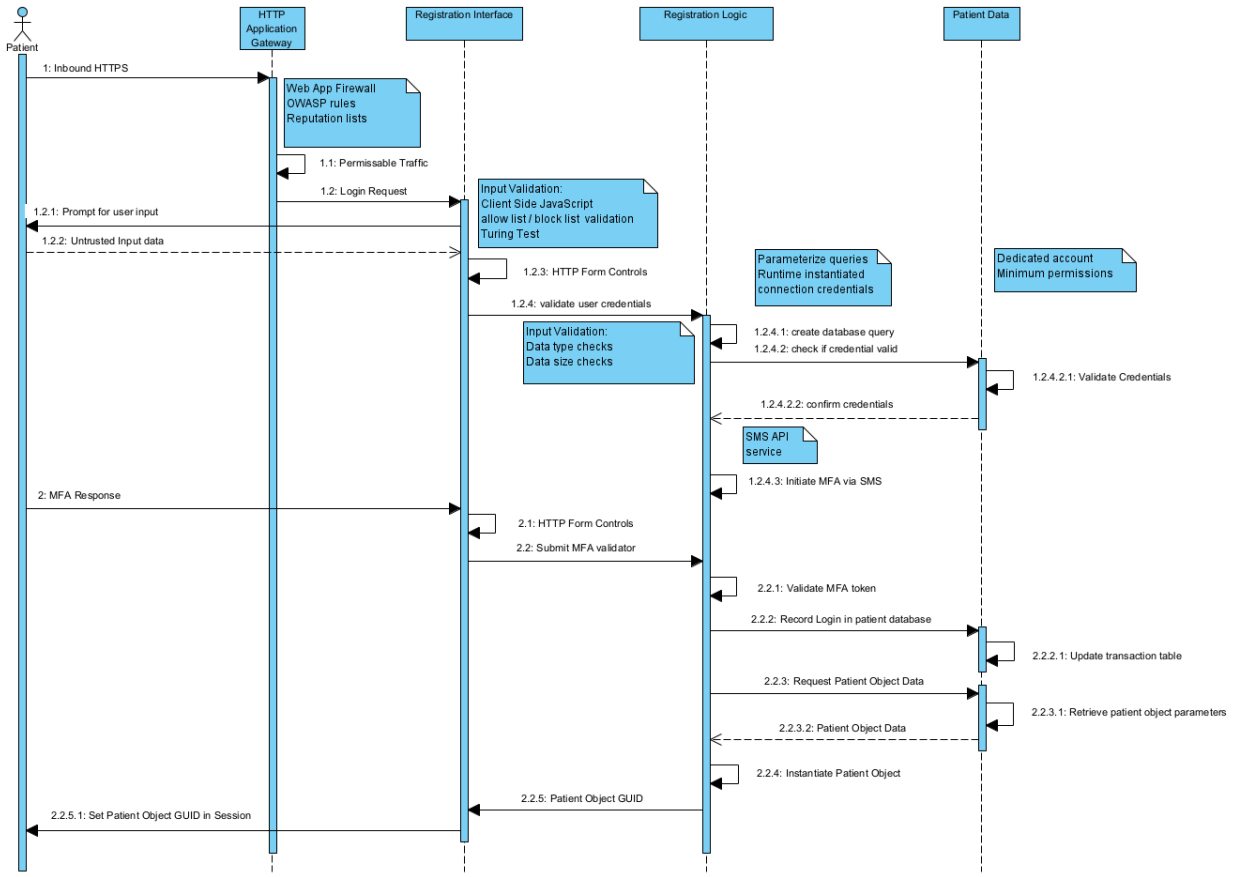

Figure 1Authorized User Login Process

Control Analysis and Test plan

Control 0: Web Application Firewall IP reputation list check

A WAF can use blacklists to block traffic from specific IP addresses but maintaining a list of known malicious addresses is not feasible for an individual organization (Proofpoint, 2021). Subscribing to one or more services that provide regularly updated blocking lists for security devices will allow Queens to disrupt communications from known malicious sources almost immediately with no additional effort required by the Queens' Information Technology (IT) staff.

The prototype function *testsourceip* determines if the IP address connecting to the application is included in the current list of blacklisted IP addresses. If an entry is found the connection is immediately terminated and a security log entry is created. If no entry is found the connection is permitted to proceed to the next control.

Testing Procedure:

Preparation:

Update the *ipreputationlist* values to include the IP address of the computer running the prototype, (line 51 of program).

Testing Action:

Initiate the ASMIS_Control_Prototype.py program using a python3 interpreter.

Anticipated Outcome:

The program should exit almost immediately after the control is applied. No response would be provided to the connecting user although the event would be captured in the security event log. The event log would be generated by the WAF since the connection would never have been forwarded to a webserver hosting the ASMIS application.

Testing Result:

The program output shows the initial control testing IP address 10.222.14.230, the Codio virtual machine, which was added to the *ipreputationlist* data structure. No response message is presented to the connecting party and the required information is included in a formatted entry which would be forwarded to a security event monitoring service in the actual implementation.

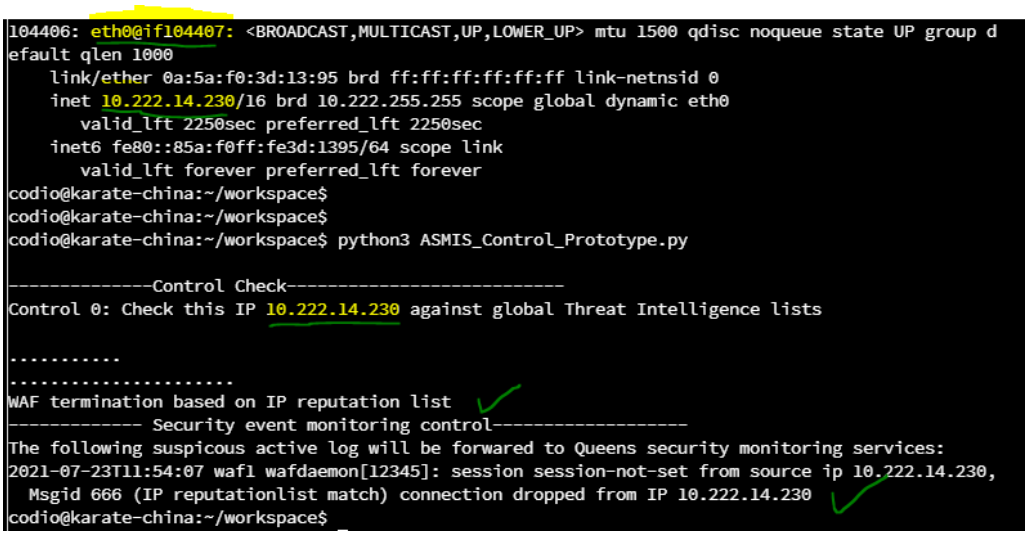

Figure 2 IP Connection blocked by reputation list

Control 1: Allow List / Block List validation

The data collected from the username input field is tested against a specific list of characters that are allowed, an approach commonly referred to as whitelisting, in alignment with OWASP Proactive Control guidance regarding input validation (OWASP Foundation, 2019) . Rejecting all data that does not conform to the format permitted for a username greatly restricts a threat actor's ability to manipulate the authentication database tables, bypass authentication or disclose sensitive information from that database.

Testing Procedure:

Preparation:

Identify the syntax for a common injection attack that could be typed into the username input field such as an additional SQL statement, commonly called 'SQL injection' (PortSwigger, 2021).

Testing Action:

Initiate the ASMIS_Control_Prototype.py program using a python3 interpreter and append an SQL statement such as "OR 1=1 - -'" to some data that would be similar to a username.

Anticipated Outcome:

The program should reject the login request before creating a query to the authentication database to determine if the password supplied is valid. Note, whitelist checking is not performed on the password field because special characters are preferred in that data.

The user should not be permitted to login and will be informed there was a problem with either the username or password.

Testing Result:

The program output (FIG 3) shows the SQL OR statement appended to a valid username is rejected.

A response message is presented to the connecting party indicating the login attempt was not successful.

```
codio@karate-china:~/workspace$
codio@karate-china:~/workspace$
codio@karate-china:~/workspace$ python3 ASMIS_Control_Prototype.py

--------------Control Check---------------------------
Control 0: Check this IP 10.222.14.230 against global Threat Intelligence lists

...........
.......................
Welcome to ASMIS, please enter your username and password
Username: bsmith+OR+1=1--
Password: ************

--------------Control Check---------------------------
Control 1: Confirm only whitelisted characters are in the following username bsmith+OR+1=1--

...........
.......................
------------------ user message -----------------
Invalid user name or password
You have 5 attempts remaining
```

Figure 3 Input Validation blocking SQL injection attempt

Reviewing the program source code (FIG 4) confirms credential checking is performed in two stages. The first stage validates the data provided in the username input field does not contain any potentially malicious input prior to creating the query against an authentication source.

```
def getcredentials(thissession):
    # only works via command line, warn & exit
    if sys.stdin.isatty():
        print("Welcome to ASMIS, please enter your username and password")
        username = input("Username: ")
        password = stdiomask.getpass(prompt='Password: ', mask='*')
        validusername=testusername(username,thissession)      You, 5 days ago • cl
    else:
        print("This does not appear to be a command line interface")
    # Check the username for malicious content prior to testing for password
    if validusername:
        message='Control 2: checking if password is valid'
        controldisplay(message)
        validpassword = testpassword(username,password)
        if validpassword:
            return [True,username]
        else:
```

Figure 4 Two stage credential check function

Control 2: Confirm password credential is valid

Password based authentication is familiar to users and therefore easy to implement but requires choosing a password that is not easily guessed and keeping the password secret. Most users require several different online accounts, consequently password reuse and variation expose the user to multiple forms of password guessing attacks (Vinberg & Overson, 2021; Center for Internet Security, 2021).

As recommended by NIST (Grassi, et al., 2020), rate limiting of unsuccessful login attempts will be implemented within the application software to severely restrict the number of connections allowed. Unique session identifiers generated by the application, rather than network source IP addresses will be used calculate rate limiting thresholds. The application-based approach enables the collection of additional information for security event log messages and avoids false positive blocking conditions due to multiple users accessing the internet from behind a single public IP address.

The ASMIS application must provide the same failed authentication error message regardless of failure is due to an incorrect username or password (OWASP Foudnation, 2021).

Multiple government and private industry security standards require the logging of failed authentication attempts (Government of Canada, 2020). To reduce the number of event logs collected and increase the information detail within those log messages, application security logs for failed authentication will only be generated after six failed attempts. While threshold numbers are adjustable within application parameters, six attempts in a single session is a strong indicator of malicious intent and should not interfere with an authorized user mistyping their password two or three times.

Testing Procedure:

Preparation:

Identify a common password attack technique to be tested such as password spraying. In a password spraying attack the adversary uses a large list of possible usernames and tests a commonly used password against all possible usernames. This step is then repeated using the next common password and over several dozen iterations the attacker is likely to find one or two accounts with these well-known weak passwords (Vinberg, et al., 2021).

Testing Action:

Initiate the ASMIS_Control_Prototype.py program using a python3 interpreter and repeat the login sequence six times, modifying the username while leaving the password the same.

Anticipated Outcome:

The program should provide an indication to the user that the authentication has failed but no indication of which component was incorrect. The program should also track the number of failed attempts by application session identifier and terminate the user's session after six failed attempts. A terminated session due to multiple authentication failures should generate a security event log that includes details of the user activity and forensic artifacts such as timestamps.

Testing Result:

The program output (FIG 5) shows the login attempts decrementing as the attacker tries each new username in the alphabetical sequence, (asmith, bsmith, …, fsmith).

The user error message is generic and will not assist an attacker performing an enumeration attack. The username *bsmith* is valid within the test data set but provided the same authentication error message.

Upon six failed authentication attempts the session identifier is destroyed, the user's connection to the application terminated and a custom formatted security event log generated for forwarding to the organization's security monitoring service.

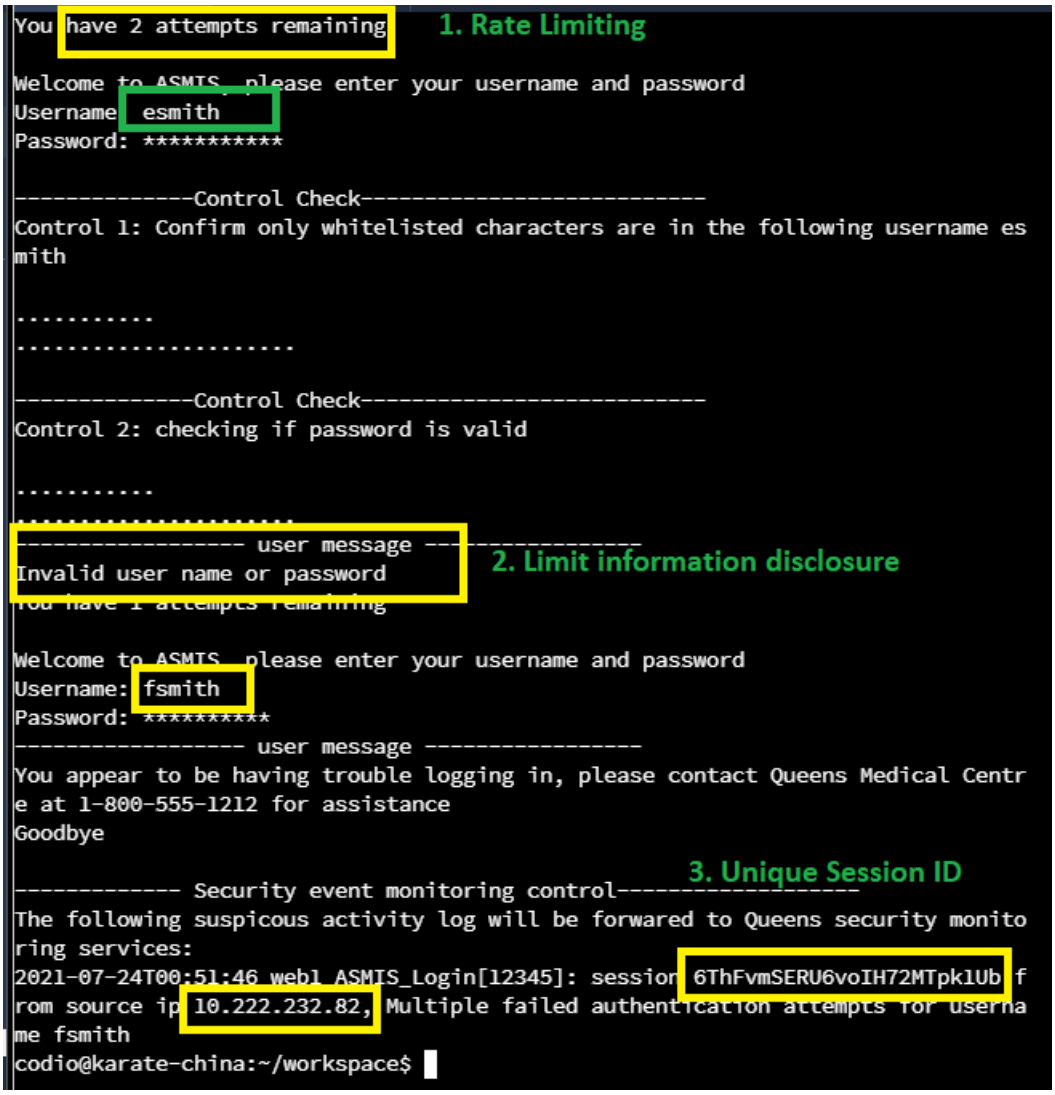

Figure 5 Session termination due to multiple authentication failures

Control 3: Data Encryption and Secrets Management

To mitigate the impact of an adversary gaining direct access to the data residing on one or more application servers, credentials and encryption keys used by the application must be retrieved from an external system such as a credential vault application rather than hardcoded into program source code (Amazon Web Services, 2021). Data at rest such as confidential user information stored in a database must also be protected from privileged information technology insiders and intruders gaining access to the application infrastructure.

To enable functionality within the limitations of a prototype implemented within a single Python file the user account data is stored within the application and instantiated at runtime. Although the confidential user data within the prototype code is encrypted, the decryption key is also stored within this application. Such practices are insecure and should never be implemented beyond example programs such as this prototype. Although the decryption key could be input during the application startup process, the high likelihood of mistyping the forty-seven-character key would only serve to frustrate the application tester. Consequently, the key was left in the prototype source code and the impact of information disclosure was mitigated by creating a fictitious dataset based on deceased people.

Testing Procedure:

Preparation:

The decryption of personal data is not a process the tester can directly influence but is able to validate the results of the decryption action.

Testing Action:

Review the data used to instantiate the Python dictionary containing user information to confirm the password and personal information fields are not stored in clear text or easily reversable ciphers.

Initiate the ASMIS_Control_Prototype.py program using a python3 interpreter and login to the application using a valid account within the data set. Confirm the confidential user information, represented by the SMS contact number, is viewable in the application and matches the data set value associated with the account tested.

Anticipated Outcome:

The confidential data should be encrypted at rest and decrypted within the application once certain preconditions are met such as a successful user authentication event.

Testing Result:

Reviewing the prototype source code shows the dictionary source data includes two cryptographically protected strings associated with each username (FIG 6), the user's confidential data is symmetrically encrypted, and the password utilizes a salted hash function. The Python cryptography Fernet class is used for the symmetric encryption (Cryptography.io, 2020). for the confidential user information stored in the prototype application. The Python implementation of the Bcrypt library is used for creating and validating the hashed password (ZetCode, 2020).

```
# User account database data
recordsdictionary={\
    "bsmith": ["$2b$12$Me7cJkoqN0ya8Jh6NQOqHeDWlZ55xzffxQu5VOUPiNZ1OqekBKKUi", "gAAAAABg81k4KZcGxf4waRbGonV0QDcRwXVRba-QDP-5Y205sm2zfg23GPfnecLmyzLQeOiypP0MgsY5CBpqThIDanS
        "wchandy": ["$2b$12$caHEQ4FGG41DGYcT8z5NYe9deowfJhmeCdhqFYiMVLMGb5BbHsWFq", "gAAAAABg81l3ILWRjuPd7zPY48A9qT_k86VKAV0A7GKIIjsCOLKKailUhH_2tExtm_9h51agyreIE02bTSqLGf
            "svaughan": ["$2b$12$S1w.O3swC4Ipaab2EnEOS.UFknDqMrVD2DnuTY4abiwrrM39j8Hai", "gAAAAABg81maMxJOfackasxS7YJUM7XaqGWyHLIrU1Arub1IiRDaNCH29ihm01eeC4eNNCZtAb_622flP
                "mrebennack": ["$2b$12$hBiwTstfHRD17Cc.aJstGeJmLxpiUyOdAk59lAbOGCicuUGNKzy9K", "gAAAAABg81nZhyKUcJa2DsSNo1DqRLRdddOMtVA3fVL-KN0w4iCGGakffDBpT36qzKhiesKN8kE
                    "bbking": ["$2b$12$hdkOAoFWZEIXR.wCHeT7h.8BJD139Z8Uc2LYCXlfylNznu4LZvyvO", "gAAAAABg81n9oHQL5SQwU1tXpPjikVC6qKgm0ol7Y_DQ6GHbQdS-1N5YVeDodEXk1OzRdVu9CcR
                        }
```

Figure 6 Data encryption at test

Reviewing the prototype source code (FIG 7) also shows the confidential data is only decrypted once the application has safely validated the username against the salted hash representation of the account's password.

```
def getsmscontact(username,key):
    message="Control 3.1: Decrypting user contact information"
    controldisplay(message)
    enccontact=recordsdictionary[username][1]
    smscontact=decryptdata(key,enccontact)
    return smscontact
```

Figure 7 User data encryption function

The program output confirms the confidential data associated with the *svaughan* account is the same as the data set, (Fig 8 & Fig 9).

```
--------------Control Check-----------------------------
Control 3: Retrieving application data decryption key from Secrets Manager appl
ication

..........
......................

--------------Control Check-----------------------------
Control 3.1: Decrypting user contact information

..........
......................

--------------Control Check-----------------------------
Control 4: Multifactor authentication using stored data for user account svaugh
an

..........
......................

--------------Control Check-----------------------------
Sending six digit code 580171 to contact number 1-403-829-6016 via SMS

..........
```

Figure 8 Confidential user information decrypted within the application

User's full name	User Account ID	Password	SMS contact number
Elizabeth Smith	bsmith	qqww1122	1-423-266-8658
William Handy	wchandy	Passw0rd1	1-256-275-3128
Steven Vaughan	svaughan	password1	1-403-829-6016

Figure 9 Clear text values from the test data set used for validation

Control 4: Multifactor Authentication

Multifactor Authentication (MFA) is required by the ASMIS application to reduce the likelihood of account compromise (Thomas & Moscicki, 2019)in the event a patient, medical office staff member, doctor or technology support person unknowingly loses control of their password. While an attacker can steal or guess the ASMIS user's password and access the application from anywhere on the internet there is no ability for the attacker to tamper with the telephone number that receives the MFA code via short message service (SMS).

To simulate an MFA solution the prototype generates a random 6-digit code and identifies the telephone number that would be receiving the code. The prototype then waits for user input and only permits further access to the application if the correct code is applied. The input process is further protected through rate limiting to prevent bulk guessing of random 6-digit codes and definable timeout value to disconnect the session if the correct code is not input within the time allocated.

Finally, the occurrence of successful account validation through username and password, followed by an MFA failure or timeout can be an indicator of an account compromise. Consequently, this specific condition does trigger a customized security monitoring event. While intrusion detection and incident response are out of scope for this prototype capturing high fidelity inputs such as the MFA behaviour in the software security design can be beneficial in the ongoing security operations for a system.

Testing Procedure:

Preparation:

Select an account and corresponding password from the test data set implemented in the prototype which is available in appendix B.

Testing Action:

Initiate the ASMIS_Control_Prototype.py program using a python3 interpreter and login to the application using a valid account within the data set.

Three separate tests will be required, use of the valid MFA code, brute force guessing of the MFA code and MFA time out.

Anticipated Outcome:

Inputting the correct MFA code should open a menu with options associated with the user's role. Inputting six incorrect MFA codes should terminate the session. Failing to provide any MFA code should case the session to time out and terminate.

Testing Results:

Typing in the correct code within the timeout period results in the user advancing to the correct menu as shown in figure 10.

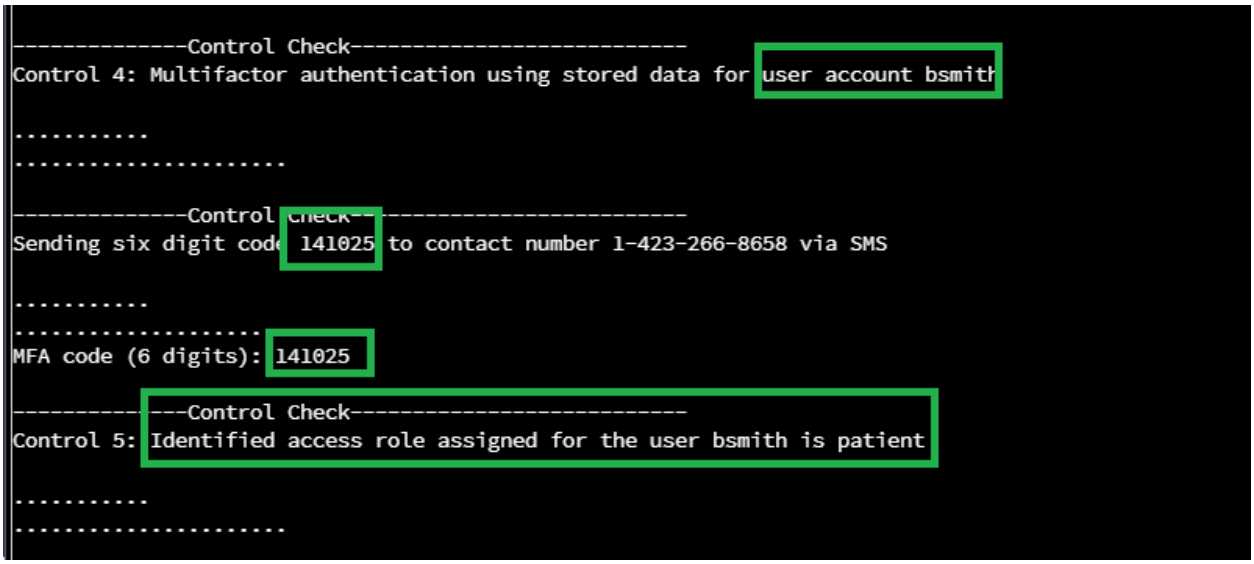

Figure 10 Successful MFA event

The *bsmith* account was reused a few minutes later to confirm the MFA code generated for each session is unique.

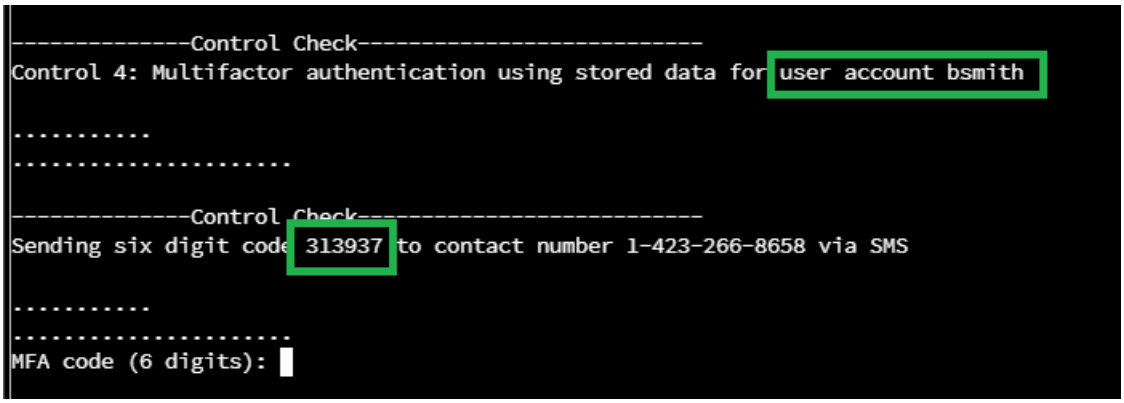

Figure 11 Confirming MFA code randomness

Typing in six incorrect MFA codes within the timeout period results in the session being terminated (Fig 12). The user informed that the MFA code is incorrect to encourage them to recheck. In this scenario the more detailed error message does not increase the risk of enumeration because the attacker would already know the username and password.

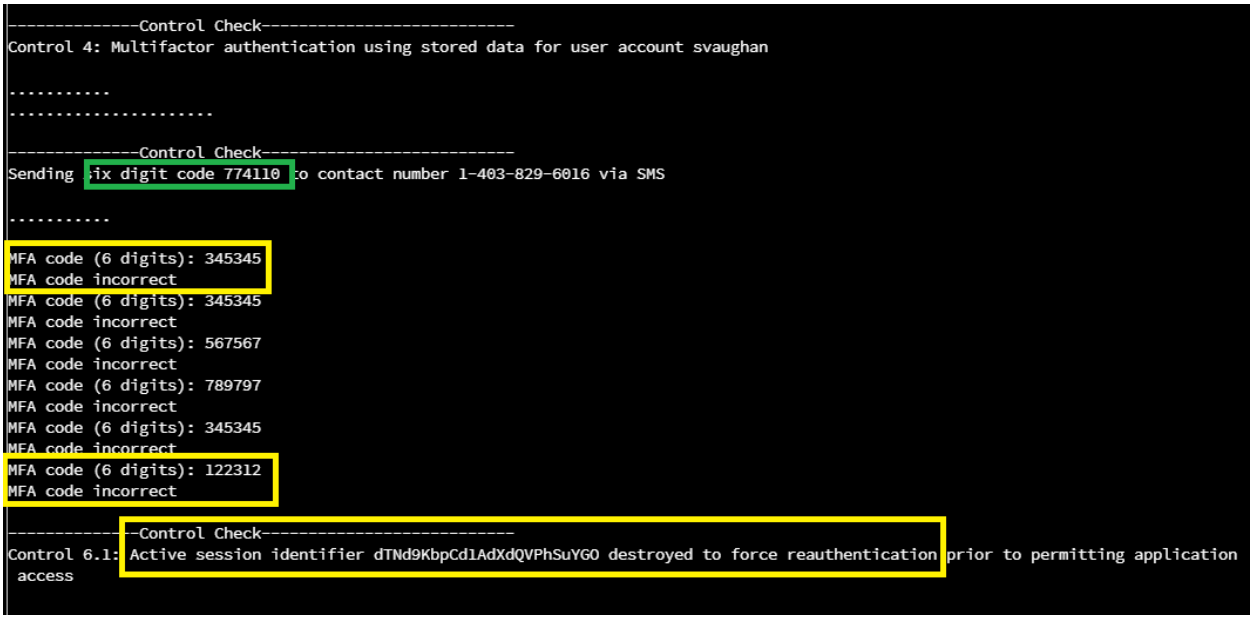

Figure 12 Multiple failed MFA attempts

Failing to input the correct MFA code within the defined timeout period results in the session terminating (FIG 13). Both termination events, multiple failed attempts and timeout generated customed security events (FIG 13).

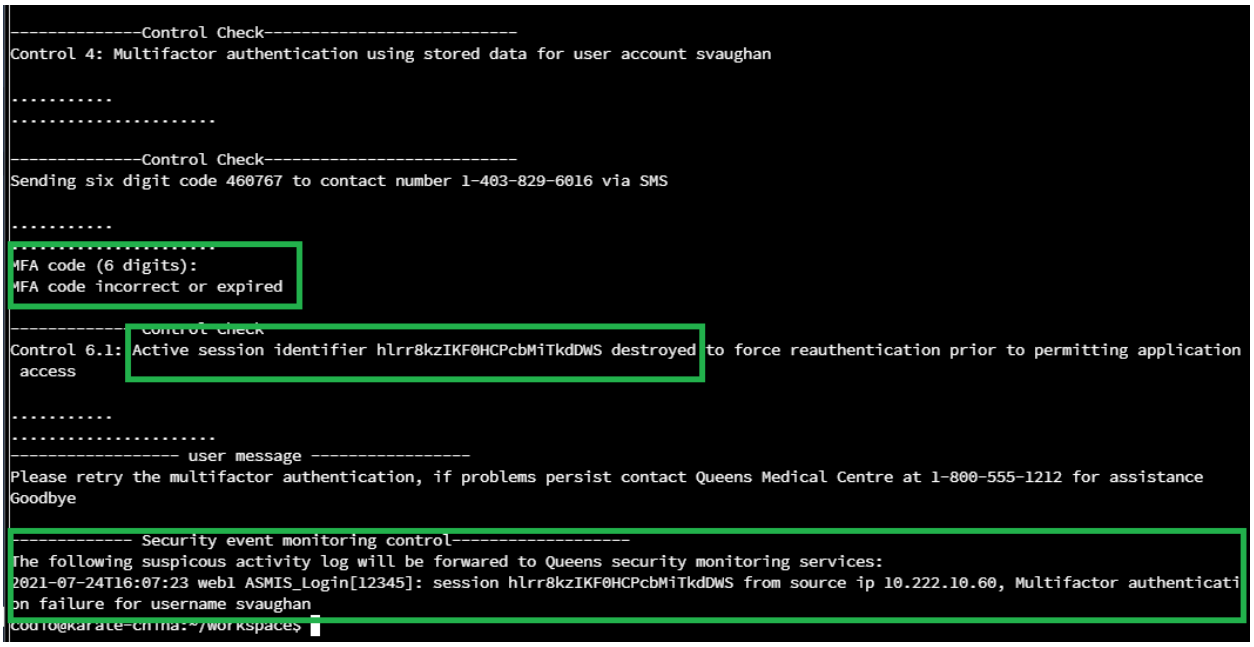

Figure 13 MFA timeout event

Control 5: Role Based Access Control

The ASMIS application will make different features available to a user depending on what they require of the application. Fully implemented role-based access control is beyond the scope of this prototype so

the required capability has been demonstrated through a mockup menu that is presented to the user after successful MFA validation. In a complete application each menu function would also restrict the data accessible to the person using the function based on business rules. A medical specialist may be able to review patient details from previous clinic visits if and only if the specialist is now treating the patient. Conversely, a technology support person may be able to create a new account for a medical specialist but would never be able to access patient information.

Testing Procedure:

Preparation:

Select an account and corresponding password from the test data set implemented in the prototype for each role implemented. (See appendix B)

Testing Action:

Initiate the ASMIS_Control_Prototype.py program using a python3 interpreter and login to the application using the valid account for the role being tested. The four roles implemented in the prototype are included in the following table:

User account	Role
bsmith	patient
wchandy	medical office staff
mrebennack	medical specialist
svaughan	information technology

Anticipated Outcome:

Each user account with a different role should have features limited to how they need to interact with the application and potentially features that are common to all users such as logging out of the application.

Testing Results:

The user account *bsmith* has the patient role and the correct menu was displayed (FIG 14.).

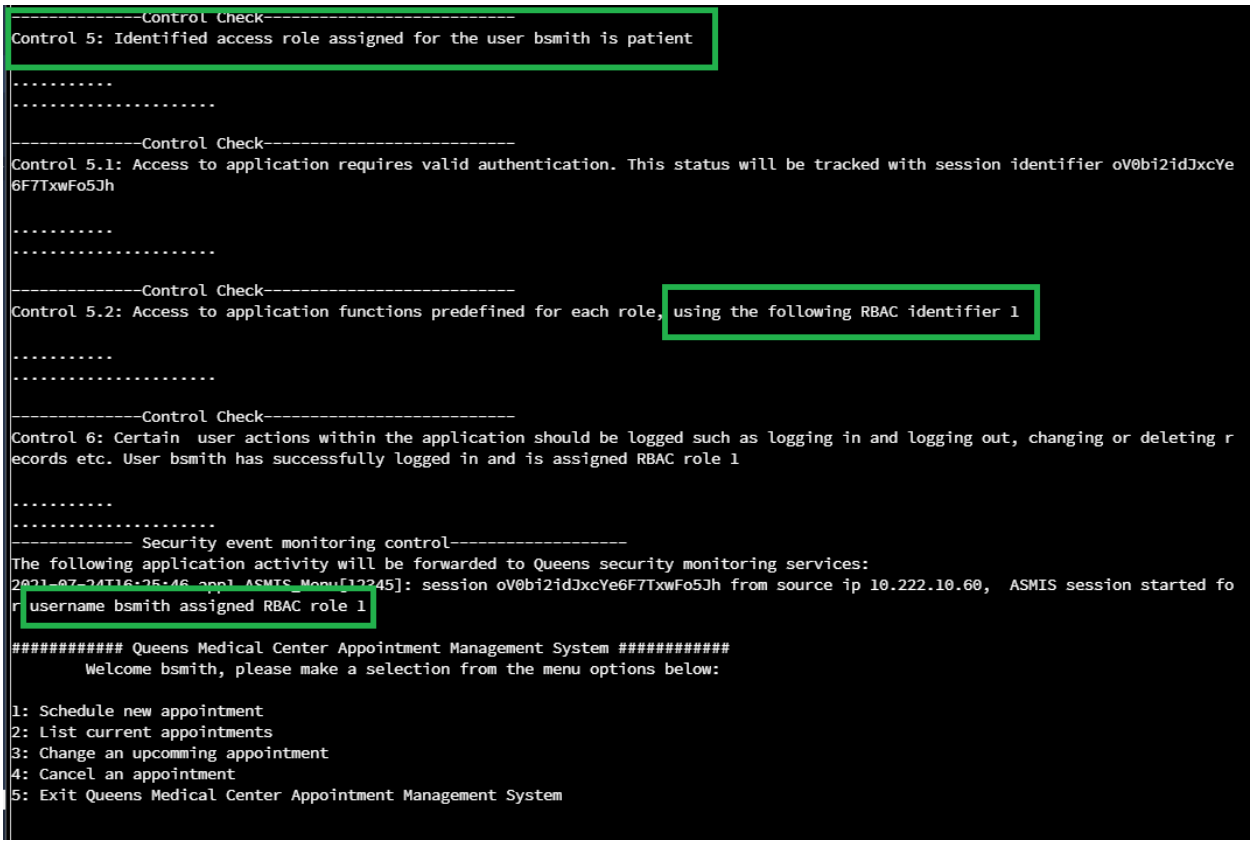

Figure 14 RBAC confirmation, patient menu

The user account *wchandy* is able to view the correct ASMIS menu for medical office staff (FIG 15.)

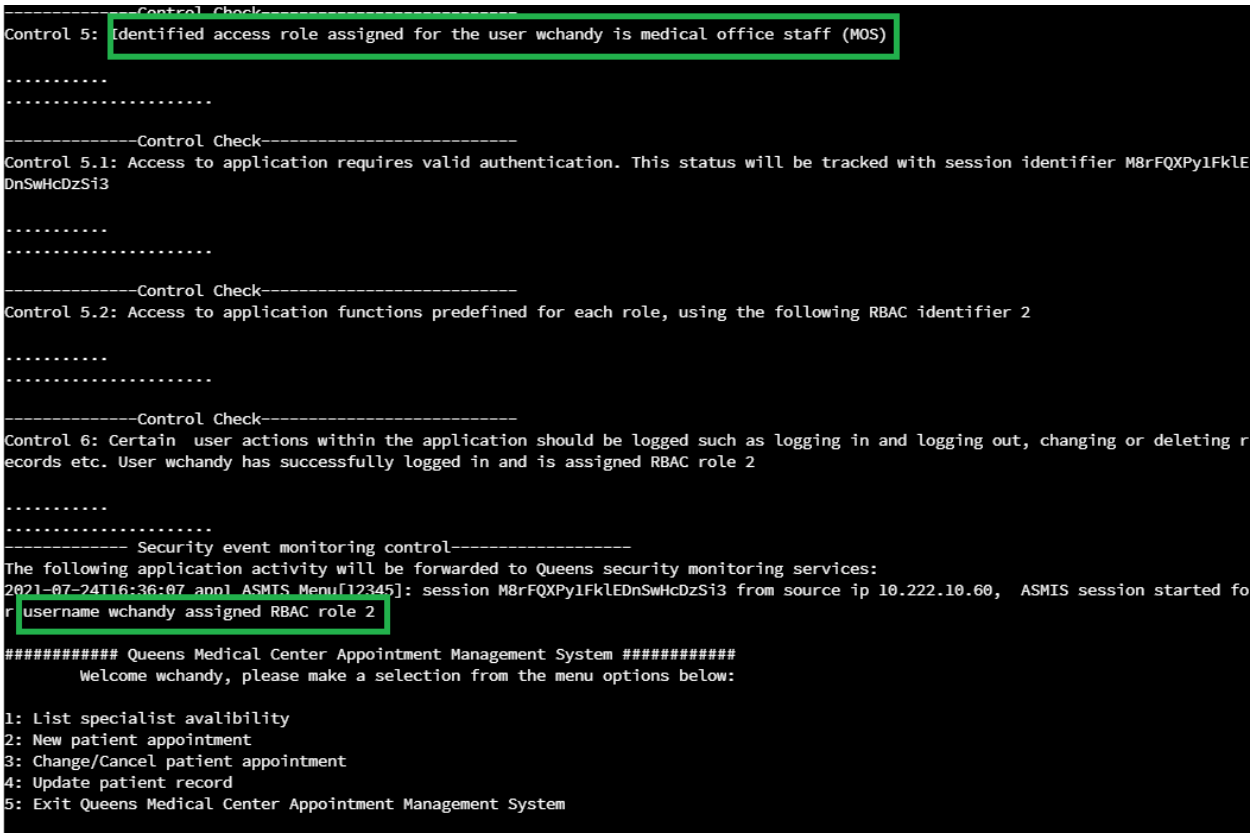

Figure 15 RBAC confirmation Medical Office Staff

The user account *mrebennack* is able to view the correct ASMIS menu for medical specialists (FIG. 16).

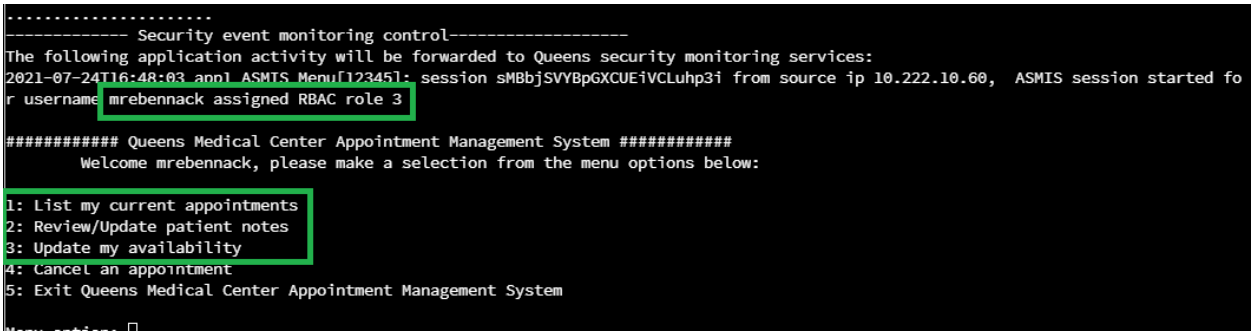

Figure 16 RBAC confirmation Medical Specialist

The user account *svaughan* is able to view the correct ASMIS menu for information technology staff (FIG. 17).

Figure 17 RBAC Confirmation Information Technology

Control 6: Session Management

A unique session identifier will need to be created for each allowed connection to the ASMIS application, both anonymous and authenticated. The session identifier or session cookie is stored by the application user's browser and presented on each new HTTP connection to uniquely identify this user through all interactions within the application (SecureCoding, 2021).

The prototype creates a random session identifier each time the application is run and passes the IP reputation control check. The unique session identifier is used for tracking all other activity within the application and is deleted when the session ends. Due to the limitations imposed by the prototype, key modifications to the session identifier noted with screen messages.

Testing Procedure:

Preparation:

This security control can only be tested through source code review. Open the program source in suitable analysis tool such as an integrated development environment.

Testing Action:

Review the functions *newsessionid, updatesessiontracker* and *deletesession* which were designed to enable management the session token through various other activities such as new connections, failed authentication attempts and logging out of the application.

Post code review, interact with the application and note the messages that include session management. Reauthenticate to the application with the same user account, repeat the action that generated the session token message and confirm the new session token is different from the previous.

Anticipated Outcome:

Session tokens will different despite using the same account and performing the same activity.

Testing Results:

Reviewing the source code confirms use of the *deletesession* function will generate a message to the screen that includes the active session identifier.

```python
# Security monitoring functions      You, 6 days ago • cleaned up unused lines, logging
def newsessionid():
    sessionid=''.join(random.choices(string.ascii_letters + string.digits,k=24))
    return sessionid

def updatesessiontracker(sessionid):
    # track login attempts by session
    if sessionid in sessiontracker.keys():
        previousfailures = sessiontracker[sessionid]
        sessiontracker[sessionid] = previousfailures + 1
    else:
        sessiontracker[sessionid]=1
    return

def deletesession(sessionid):
    message="Control 6.1: Active session identifier {} destroyed to force reauthenticat
    try:
        sessiontracker.pop(sessionid)
        controldisplay(message)
    except KeyError:
        pass
    return

def getfailedlogincount(sessionid):
    return sessiontracker[sessionid]
```

Figure 18 Session Management source code review

Logging into and out of the application records the session identifier in two log messages and the *deletesession* function. Figure 19 shows the session identifier is LziRBIFiDZwJgfTFFQUM96YQ, figure 20 confirms the second event generated a different identifier, YzvGJNodAa3a9KJ5QxTlssg4.

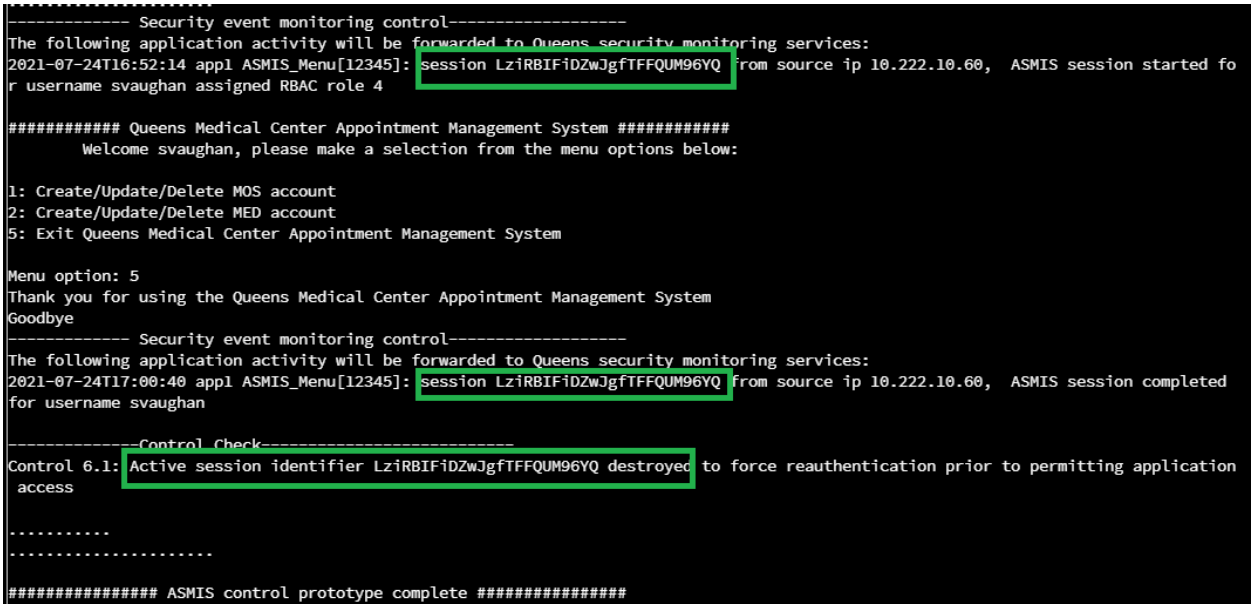

Figure 19 Session Management Review part 1

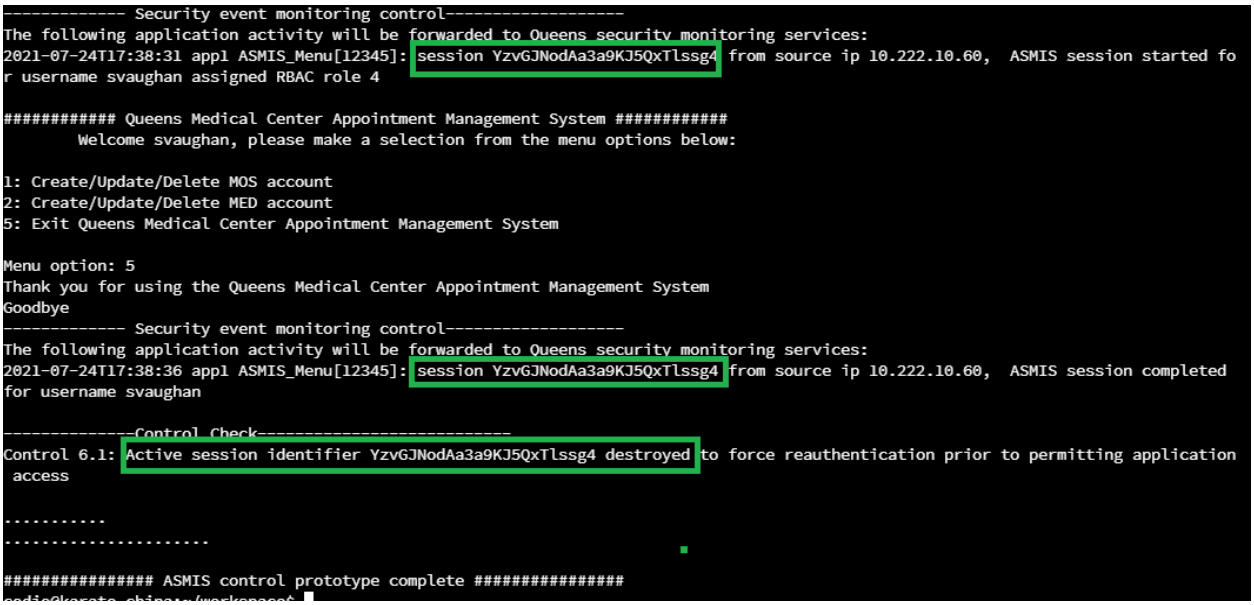

Figure 20 Session Management Review part 2

Appendix A: Python Module Dependencies

The prototype program requires three additional Python modules to be installed for certain critical features to work properly. The rationale behind each module's use it included below for reference, highlighting certain requirements that must be met for either security or user experience that must be implemented regardless of software programing language choice.

Stdmaskio:

The stdmaskio module (Sweigart, 2020) replaces user typed characters with a star (*) character. In addition to protecting the user from exposing their password to someone looking at the computer screen the star characters allow the user to visually confirm the correct number of characters have been input.

Bcrypt:

This module (pythonmodulebcrpt) is a Python based implementation of the bcrypt hashing function designed by Niels Provos and David Mazieres. Commonly referred to as salted passwords, some additional random characters, unique to the application, should be combined with the user supplied password prior to calculating the hash value. In the event the password hashes are exposed to an attacker those hash values will not be comparable to a table of precalculated hashes based on common passwords. The bcrypt hashing algorithm creates this secure storage value in a single step, simplifying the implementation of this security requirement (ZetCode, 2020).

Pytimedinput:

The security of an application can benefit from limiting the amount of time a transition state is permitted (Schwartau, 2018), such as waiting for additional data input from a user. The Python input function will wait indefinitely for user input which can tie up application resources and lead to an inconclusive state, but timeout events can also be used as an indicator of compromise enabling proactive security response.

Consider the ASMIS application, once a user has been authenticated with a username and password they must also provide a random six digit code that has been sent to their cellular phone via short message service (SMS). Failure to input the code within a reasonable time could be an indicator that an external attacker somehow has knowledge of the username and password. Implementing a timeout check on the SMS requirement allows the application to record both the user account and timeout event in a log. The lack of successful SMS based validation following a successful password-based authentication can indicate account compromise. Detection of this scenario can permit Queens to inform the owner of the account or forcibly resetting the password, assuming such measures are within the application's terms of service.

Appendix B: User test details

The following data set can be used to validate implementation of some security controls within the ASMIS application prototype.

User's full name	User Account ID	Password	SMS contact number	Application user role
Elizabeth Smith	bsmith	qqww1122	1-423-266-8658	patient
William Handy	wchandy	Passw0rd1	1-256-275-3128	medical office staff
Steven Vaughan	svaughan	password1	1-403-829-6016	Information Technology
Malcolm Rebennack	mrebennack	r1gthPL@CE	1-888-664-0933	medical specialist
Riley King	bbking	Luc!lle335	1-901-524-5464	patient

The prototype stores the user account information in a dictionary data structure, each username account is a unique key, the key value is a three-element list containing the account password, SMS contact number an application role assigned to the user account. The salted password and encrypted SMS contact information are stored as string data types which will require conversion to byte array datatypes at runtime. In the following example the password *qqww1122* is stored as the string beginning with *$2b$12* and the SMS contact begins with "*gAAAAAB*".

"bsmith": ["$2b$12$Me7cJkoqN0ya8Jh6NQOqHeDWlZ55xzffxQu5VOUPiNZ1OqekBKKUi", "gAAAAABg81k4KZcGxf4waRbGonV0QDcRwXVRba-QDP-5Y205sm2zfg23GPfnecLmyzLQeOiypP0MgsY5CBpqThIDanSVdo9BAA==", "1"]

A small Python program called, *createUserData.py*, available from https://github.com/DLeece-Essex/asmis-testing, can be used to create user account information for use in the prototype application should retesting with new credentials be required. The *newpassword* and *encryptdata* functions convert the user input values for password and SMS contact information into byte array data types which are then processed using cryptographic functions. The program outputs a file called *user_data.txt* which contains preformatted data that can be used for instantiating the Python dictionary data structure *recordsdictionary* as illustrated in the screen shot below.

```
# User account database data
recordsdictionary={\
    "bsmith": ["$2b$12$Me7cJkoqN0ya8Jh6NQOqHe
        "wchandy": ["$2b$12$caHEQ4FGG41DGYcT8
            "svaughan": ["$2b$12$S1w.O3swC4Ip
                "mrebennack": ["$2b$12$hBiwTs
                    "bbking": ["$2b$12$hdkOAo
                        }
```

References

Amazon Web Services. (2021) AWS Secrets Manager. Available from: https://aws.amazon.com/secrets-manager/ [Accessed 16 July 2021].

Center for Internet Securiy. (2021) Election Security Spotlight - Password Attacks. Available from: https://www.cisecurity.org/spotlight/ei-isac-cybersecurity-spotlight-password-attacks/ [Accessed 23 July 2021].

Cryptography.io. (2020) Fernet (symetric encryption). Available from: https://cryptography.io/en/latest/fernet/ [Accessed 24 July 2021].

Engel, D. (2020) Connection strings and configuration files. Available from: https://docs.microsoft.com/en-us/sql/connect/ado-net/connection-strings-and-configuration-files [Accessed 3 July 2021].

Government of Canada. (2020) Event Logging Guidance. Available from: https://www.canada.ca/en/government/system/digital-government/online-security-privacy/event-logging-guidance.html [Accessed 23 July 2021].

Grassi, P. et al. (2020) *NIST Special Publication (SP) 800-63B Digital Identity Guidelines: Authentication and Lifecycle Management.* Washington, D.C.: National Insititute of Standards and Technology.

Information Commisioner's Office. (2020) Guidance - Brexit FAQs -ICO. Available from: https://ico.org.uk/media/for-organisations/documents/brexit/2617110/information-rights-and-brexit-faqs-v2_3.pdf [Accessed 2 July 2021].

Intersoft Consulting. (n.d) GDPR Personal Data. Available from: https://gdpr-info.eu/issues/personal-data/ [Accessed 24 July 2021].

OWASP Foudnation. (2021) Authentication Cheat Sheet. Available from: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html [Accessed 23 July 2021].

OWASP Foundation. (2019) C5: Validate All Inputs. Available at: https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c5-validate-all-inputs.html [Accessed 23 July 2021].

PortSwigger. (2021) SQL Injection. Available from: https://portswigger.net/web-security/sql-injection [Accessed 22 July 2021].

Proofpoint. (2021) ET Pro Ruleset. Available from: https://www.proofpoint.com/us/threat-insight/et-pro-ruleset [Accessed 22 July 2021].

Schwartau, W. (2018), *Analogue Network Security*. 1st ed. Nashville: Winn Schwartau LLC.

SecureCoding. (2021) Session Management: An Overview. Available from: https://www.securecoding.com/blog/session-management-an-overview/ [Accessed 24 July 2021].

Sweigart, A. (2020) stdiomask 0.0.6. Available from: https://pypi.org/project/stdiomask/ [Accessed 5th July 2021].

Thomas, K. & Moscicki, A. (2019) New research: How effective is basic account hygiene at preventing hijacking. Available from: https://security.googleblog.com/2019/05/new-research-how-effective-is-basic.html [Accessed 25 June 2021].

Vinberg, S. & Overson, J. (2021) 2021 Credential Stuffing Report. Available from: https://www.f5.com/labs/articles/threat-intelligence/2021-credential-stuffing-report [Accessed 23 July 2021].

ZetCode. (2020) Python bcrypt. Available from: https://zetcode.com/python/bcrypt/ [Accessed 5th July 2021].