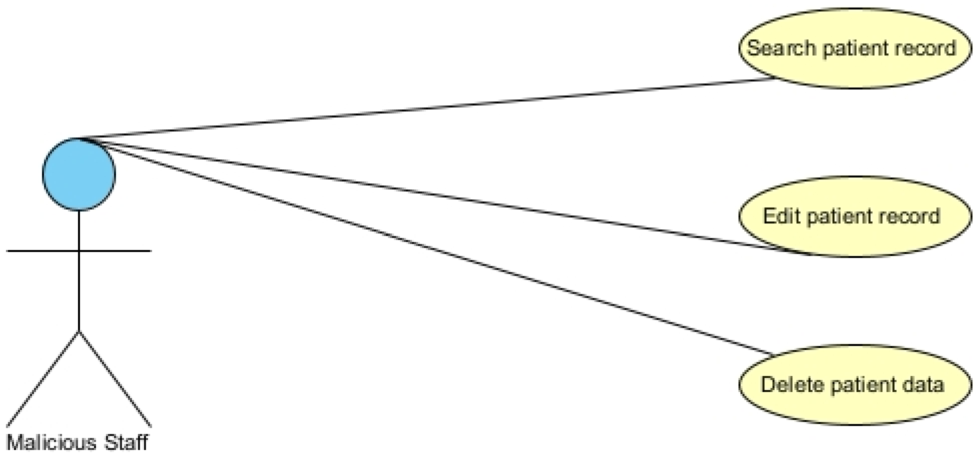
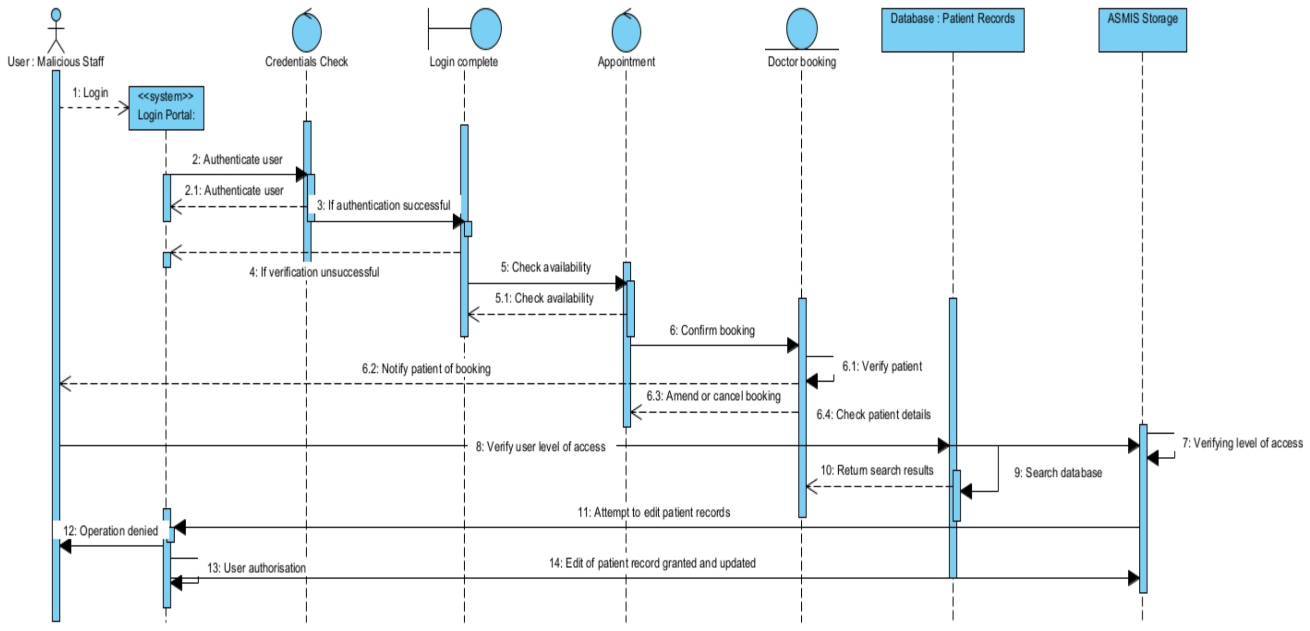
Unit 10 Programming Outline:

The threat is a staff member may act maliciously to tamper, disclose information or delete data from the storage system. To prevent this, authenticated usernames and passwords are set with access controls at the login portal stage. This will allow authorisation of the correct level of access for the user, and malicious attempts to gain elevated privileges would be nullified.





User : Malicious Staff
Credentials Check
Login complete
Appointment
Doctor booking
Database : Patient Records
ASMIS Storage
<<system>> Login Portal:
1: Login
2: Authenticate user
2.1: Authenticate user
3: If authentication successful
4: If verification unsuccessful
5: Check availability
5.1: Check availability
6: Confirm booking
6.1: Verify patient
6.2: Notify patient of booking
6.3: Amend or cancel booking
6.4: Check patient details
7: Verifying level of access
8: Verify user level of access
9: Search database
10: Return search results
11: Attempt to edit patient records
12: Operation denied
13: User authorisation
14: Edit of patient record granted and updated

Python Code (Phillips, 2015)

```
import hashlib
class User: #stores the username and an encrypted password
 def __init__(self, username, password):
    '''Create a new user object. The password will be encrypted before storing.'''
    self.username = username
    self.password = self._encrypt_pw(password)
    self.is_logged_in = False
 def _encrypt_pw(self, password): #password will be stored encrypted to reduce the
chances of its being stolen
    '''Encrypt the password with the username and return the sha digest.'''
    hash_string = (self.username + password)
    hash_string = hash_string.encode("utf8")
    return hashlib.sha256(hash_string).hexdigest()
 def check_password(self, password): #test whether a supplied password is the
correct one
    '''Return True if the password is valid for this user, false otherwise.'''
    encrypted = self._encrypt_pw(password)
    return encrypted == self.password


class AuthException(Exception):
 def __init__(self, username, user=None):
    super().__init__(username, user)
    self.username = username
    self.user = user
```

```
class UsernameAlreadyExists(AuthException): #check if a username that already
exists
    pass

class PasswordTooShort(AuthException): #raise an exception if the password is too
short
    pass

class Authenticator:
 def __init__(self):
    '''Construct an authenticator to manage users logging in and out.'''
    self.users = {}
 def add_user(self, username, password):
    if username in self.users:
        raise UsernameAlreadyExists(username)
    if len(password) < 6:
        raise PasswordTooShort(username)
    self.users[username] = User(username, password)

class InvalidUsername(AuthException):
    pass

class InvalidPassword(AuthException):
    pass
 def login(self, username, password):
   try:
      user = self.users[username]
   except KeyError:
      raise InvalidUsername(username)
   if not user.check_password(password):
      raise InvalidPassword(username, user)
   user.is_logged_in = True
   return True
authenticator = Authenticator()
class Authorizor:
 def __init__(self, authenticator):
   self.authenticator = authenticator
   self.permissions = {}
 def add_permission(self, perm_name):
   '''Create a new permission that users can be added to'''
   try:
      perm_set = self.permissions[perm_name]
   except KeyError:
      self.permissions[perm_name] = set()
   else:
      raise PermissionError("Permission Exists")
 def permit_user(self, perm_name, username):
   '''Grant the given permission to the user'''
   try:
        perm_set = self.permissions[perm_name]
    except KeyError:
        raise PermissionError("Permission does not exist")
    else:
        if username not in self.authenticator.users:
            raise InvalidUsername(username)
        perm_set.add(username)
class PermissionError(Exception):
    pass
 def check_permission(self, perm_name, username):
    if not self.authenticator.is_logged_in(username):
        raise NotLoggedInError(username)
    try:
        perm_set = self.permissions[perm_name]
    except KeyError:
        raise PermissionError("Permission does not exist")
    else:
        if username not in perm_set:
            raise NotPermittedError(username)
        else:
            return True
class NotLoggedInError(AuthException):
    pass
class NotPermittedError(AuthException):
    pass
```

```
authorizor = Authorizor(authenticator)


import auth
# Set up a test user and permission
auth.authenticator.add_user("joe", "joepassword")
auth.authorizor.add_permission("test program")
auth.authorizor.add_permission("change program")
auth.authorizor.permit_user("test program", "joe")
class Editor:
 def __init__(self):
    self.username = None
    self.menu_map = {
        "login": self.login,
        "test": self.test,
        "change": self.change,
        "quit": self.quit}
 def login(self):
    logged_in = False
    while not logged_in:
        username = input("username: ")
        password = input("password: ")
        try:
            logged_in = auth.authenticator.login(
                username, password)
        except auth.InvalidUsername:
            print("Sorry, that username does not exist")
        except auth.InvalidPassword:
            print("Sorry, incorrect password")
        else:
            self.username = username
def is_permitted(self, permission):
    try:
        auth.authorizor.check_permission(
            permission, self.username)
    except auth.NotLoggedInError as e:
        print("{} is not logged in".format(e.username))
        return False
    except auth.NotPermittedError as e:
        print("{} cannot {}".format(
            e.username, permission))
        return False
    else:
        return True
def test(self):
    if self.is_permitted("test program"):
        print("Testing program now...")
def change(self):
    if self.is_permitted("change program"):
        print("Changing program now...")
def quit(self):
    raise SystemExit()
 def menu(self):
    try:
        answer = ""
        while True:
            print("""
Please enter a command:
\tlogin\tLogin
\ttest\tTest the program
\tchange\tChange the program
\tquit\tQuit
""")
            answer = input("enter a command: ").lower()
            try:
                func = self.menu_map[answer]
            except KeyError:
                print("{} is not a valid option".format(answer))
            else:
                func()
    finally:
        print("Thank you for testing the auth module")
Editor().menu()
```

References

Phillips, D. 2015. *Python 3 Object Oriented Programming*, Packt Publishing.