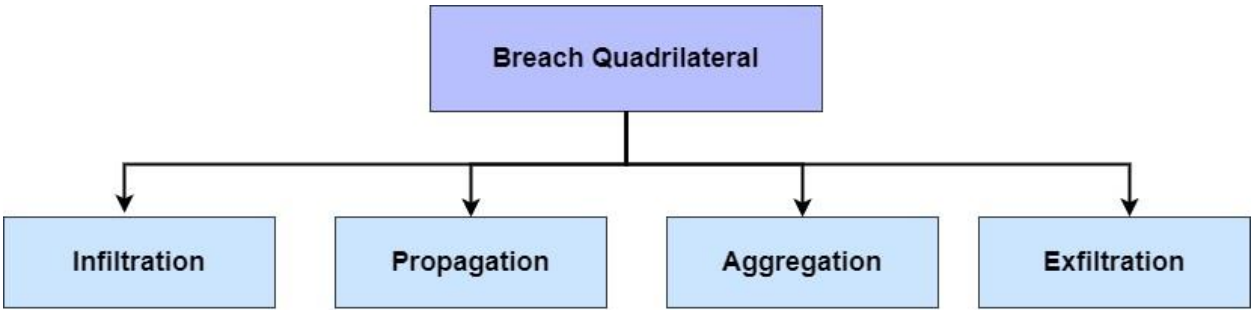
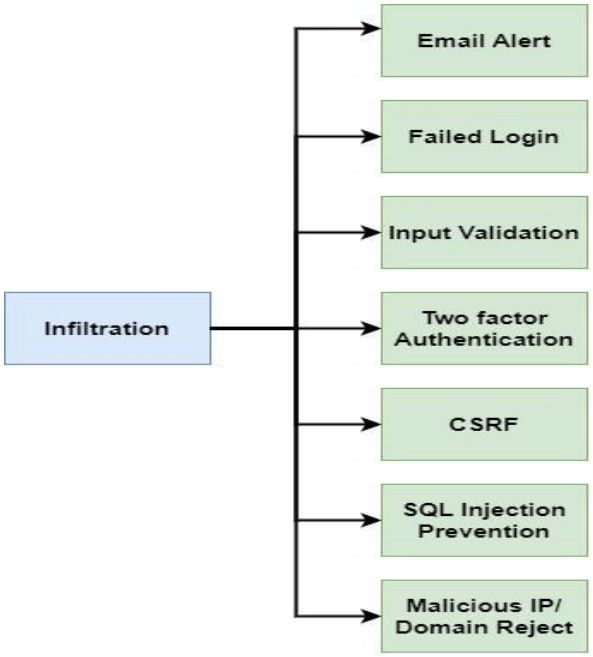
Methodology

For Security measures, we have maintained Breach Quadrilateral. And, try to touch every quadrilateral with at least one feature.



Infiltration:



Email Alert: Here, we have implemented that if a user entered his account, then he will be notified by email.

```
send_mail(subject, message, sender_email, recipient_list)
```

```
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587 #465
EMAIL_HOST_USER = 'shossain201214@bscse.uiu.ac.bd'
EMAIL_HOST_PASSWORD = 'lolsjvdpvbnexknv'
EMAIL_USE_TLS = True
```

Failed Login: Here, we have set 5 time wrong tried for a user at a time. After trying 5 time, the user will be blocked for some time.

```
@ratelimit(key='ip', rate='5/m', method='POST', block=True)
```

Input Validation: Here, we have tried to take accurate input for every form. For example: when a user tries to register an account then he should place a correct format of email. Otherwise, it will show input invalidation.

- Django auto generated Input validation for Email password and other elements

```
password1 = forms.CharField(label='Password',
            widget=forms.PasswordInput(attrs={'class':'form-control', 'placeholder':'Password'}))
 password2 = forms.CharField(label="Confirm Password",
    widget=forms.PasswordInput(attrs={'class':'form-control', 'placeholder':'Confirm Password'}))
email = forms.CharField(required=True, widget=forms.EmailInput(attrs={'class':'form-control',
        'placeholder':'Email Address'}))
```

Two-factor Authentication: Here, we have decided to add authentication for admin panel. And use Google Authenticator for the OTP code which is continuously updated after every 30 second.

```
from django_otp.admin import
    OTPAdminSite
admin.site.__class__ =  OTPAdminSite
```

```
'django_otp',
'django_otp.plugins.otp_totp',
'django_otp.plugins.otp_static',
```

```
'django_otp.middleware.OTPMiddleware',
```

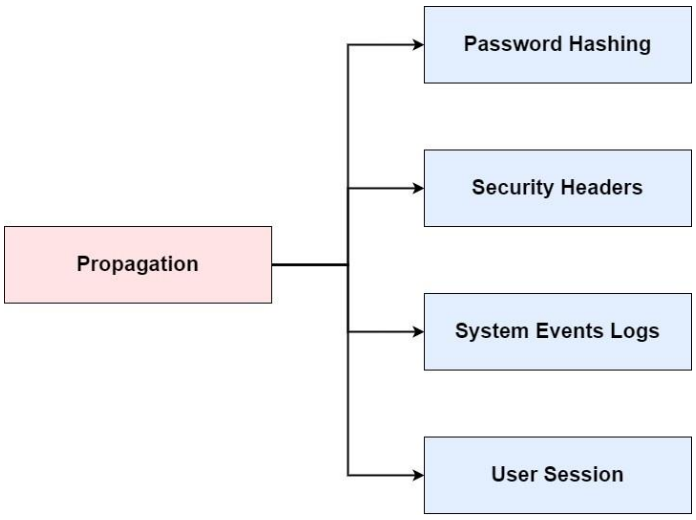
CSRF: A CSRF token is a secure random token that is used to prevent CSRF attacks. And it is used before every form in the system.

SQL Injection: For preventing SQL Injection, we utilize parameterized database queries with bound, typed parameters and careful use of parameterized stored procedures in the database.

Malicious IP Rejection: Set some Malicious IP and domain whose couldn't enter to the system.

```
# prevent malicious ip or domain
MALICIOUS_IPS = ['1.2.3.4', '5.6.7.8']
MALICIOUS_DOMAINS = ['malicious.com', 'evil.org']
```

Propagation:



Password Hashing: Tries to hide user password to the admin panel. Method was**: PBKDF2** algorithm with **SHA256** Hash which is inbuilt library in Django.

Security Headers: Added some security headers for **Click jacking, Content Security Policy (CSP), X-Content-Type-Options, X-Frame-Options, and X-XSS-Protection.**

```
SECURITY_HEADERS = {
   'Content-Security-Policy': "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-
                              src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self'; connect-src
                              'self';",
   'X-Content-Type-Options': 'nosniff',
   'X-Frame-Options': 'DENY',
   'X-XSS-Protection': '1; mode=block',
   'Strict-Transport-Security': 'max-age=31536000; includeSubDomains',

}
```

System Event Logs: For every activity of the system, we try to save a log in the code so that if any error or vulnerability shows then it shows by the developers.

```
LOGGING = {
   'version': 1,
   'disable_existing_loggers': False,
   'handlers': {
      'file': {
         'level': 'INFO',
         'class': 'logging.FileHandler',
         'filename': os.getcwd() + '/log/logfile.log',
         'filters': ['security'],
      },
   },
   'filters': {
      'security': {
         '()': 'django.utils.log.CallbackFilter',
         'callback': lambda record: record.levelname in ['CRITICAL', 'ERROR', 'WARNING'],  # Customize as needed
      },
   },
```

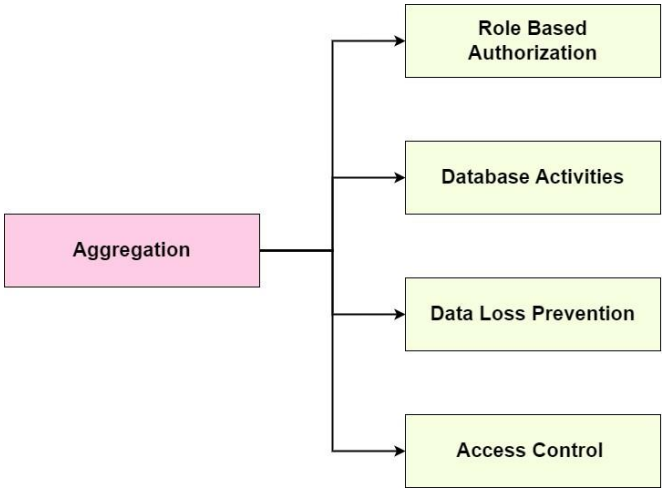
```
    'loggers': {                                    'propagate': True,
        'django': {                                 },
            'handlers': ['file'],                   },
            'level': 'INFO',                    }
```

User Session: When a user has no activity for 5 minutes in the system then it will automatically logout the user.

SESSION_EXPIRE_AT_BROWSER_CLOSE = False

SESSION_COOKIE_AGE = 5 * 60

Aggregation:



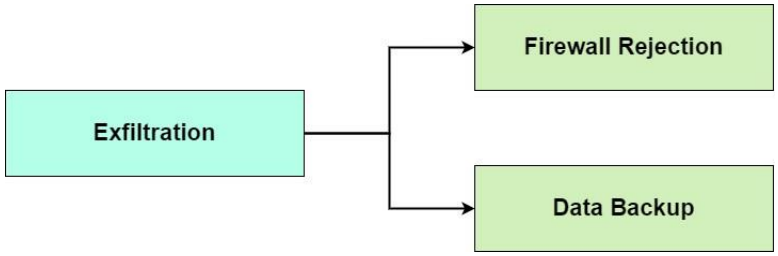
Role Based Authorization: Admin, user and super stuff have their own dashboard for own accessibility. Noone can get access of others.

Database Activity: We try to save an admin database activity or changes in the system corner.

Data Loss Prevention: If any hacker success to take the system over him, then other Developer can Roll back to the system initial stage after saving the Backup data.

Access Control: Admin can manage the user. If any user has any malicious activity, then admin can control them.

Exfiltration:



Network Firewall: We try to reject all foreign country access of our system

```
class CountryAccessMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        self.reader = geoip2.database.Reader('GeoIP2-Country-Test.mmdb')
        self.logger = logging.getLogger(__name__)

    def __call__(self, request):
        client_ip, _ = get_client_ip(request)
        if client_ip and client_ip != '127.0.0.1':
            try:
                response = self.reader.country(client_ip)
                country_code = response.country.iso_code
                self.logger.info(f"Request from IP {client_ip} is from country {country_code}")
                if country_code != 'BD':
```

```
                self.logger.warning(f"Access from IP {client_ip} (country: {country_code}) is restricted.")
                return HttpResponseForbidden("Access from your country is restricted.")
            except Exception as e:
                self.logger.error(f"Error while processing request from IP {client_ip}: {e}")
        return self.get_response(request)
```

Data Backup: Here, we can save the data in pdf after a day or a week.

```
@login_required
def data_backup(request):
    if request.user.is_superuser:
        database_file = 'db.sqlite3'
        if os.path.exists(database_file):
            email = EmailMessage(
                subject='Database Backup',
                body='Attached is the backup of the database.',
                to=[request.user.email],
            )
            email.attach_file(database_file)
            email.send()
            print("Backup sent")
        else:
            print('Database not found')
        return redirect('store:home')
    else:
        return redirect('store:home')
```

Result & Discussion:

- We added more security steps like email alerts, watching for failed logins, and making sure user input is safe. We also made the system secure like admins need two ways to prove who they are when logging in, stopped certain types of attacks, and blocked harmful IP addresses. These changes helped keep our site safer.
- We made passwords harder to steal, added extra security to our web pages, and kept a close eye on what's happening on our system. These things made it tougher for bad guys to mess with our site.
- Also, we made sure only the right people could get into sensitive parts of our site, kept a close watch on our database to catch any weird activity, and controlled who could see what. These steps helped keep our data safe and our site secure.

Limitations & Conclusion:

- Even with all these security improvements, we can't forget that no system is perfect. Skilled attackers might still find ways to break in.
- We need to stay alert and update our defenses regularly to stay safe. In short, while our added security measures are helpful, we always need to be on the lookout for new threats online.
- Some security feature like input validation, two-step verification recent activities are only for one section either for admins or users. We look forward to add them in both sections.
- In the end, while we've made our site much safer, we have to keep working hard to keep it that way. Regular checks and staying up-to-date are key to keeping our business and our customers safe from online dangers.