# Secure Programming

## Week 11

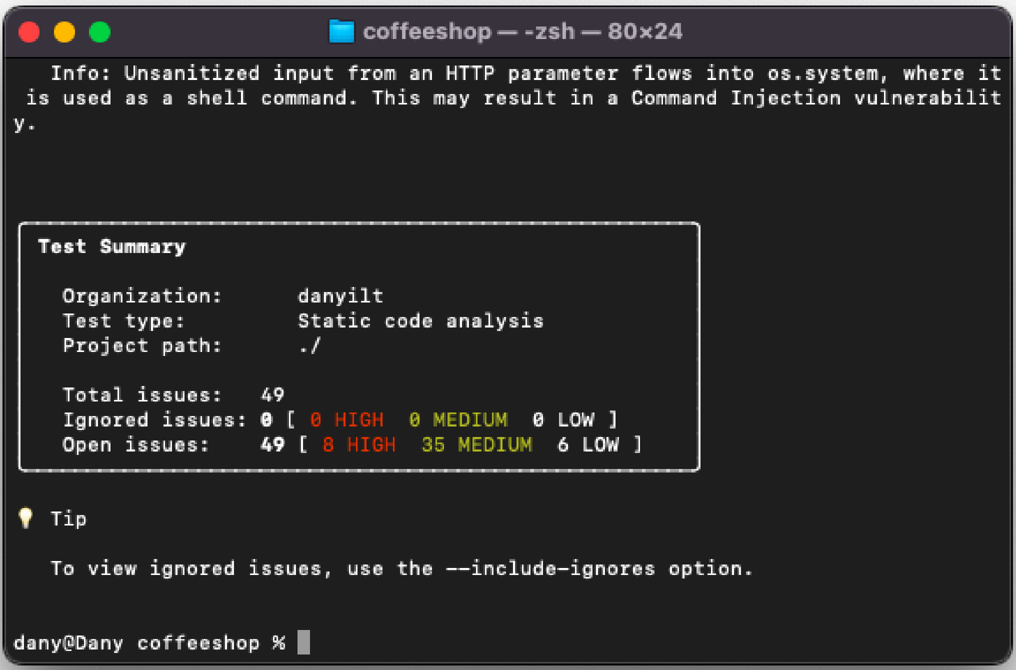
## Static Code Analysis

Using **Snyk CLI**

### Scanning the Coffeeshop Site

Run the following command:

```
$ snyk code test <path-to-your-django-coffeeshop> >>
snyk_vulns.txt
$ snyk code test ./
```
(inside coffeeshop directory)

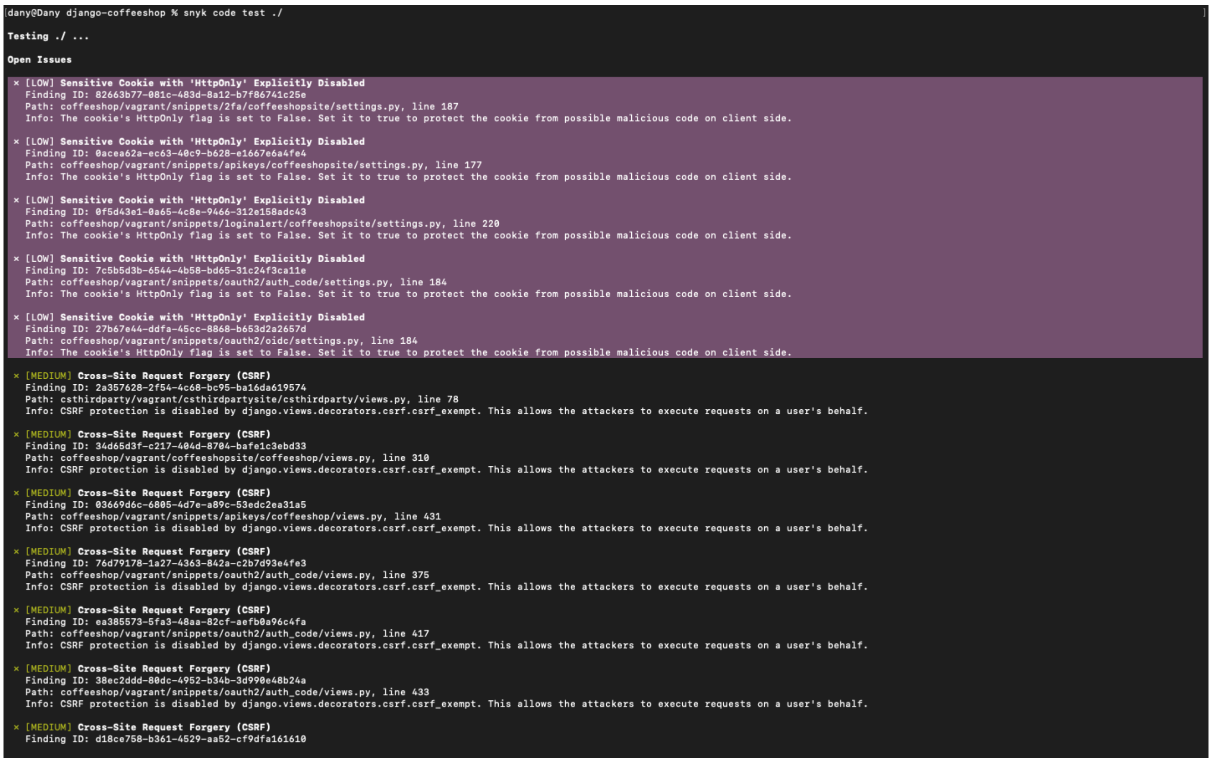


```
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it
 is used as a shell command. This may result in a Command Injection vulnerabilit
y.



┌────────────────────────────────────────────────────────┐
│ Test Summary                                           │
│                                                        │
│   Organization:      danyilt                           │
│   Test type:         Static code analysis              │
│   Project path:      ./                                │
│                                                        │
│   Total issues:   49                                   │
│   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]        │
│   Open issues:    49 [ 8 HIGH  35 MEDIUM  6 LOW ]      │
└────────────────────────────────────────────────────────┘

💡 Tip

   To view ignored issues, use the --include-ignores option.


dany@Dany coffeeshop %
```

**Found 8 High, 35 Medium, 6 Low**

## Fixing 3 vulnerabilities (one low, one medium, one critical) that we have not covered in the labs

### [LOW] Level Issue — Sensitive Cookie in HTTPS Session Without 'Secure' Attribute



```
dany@Dany django-coffeeshop % snyk code test ./

Testing ./ ...

Open Issues

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 82663b77-081c-483d-8a12-b7f86741c25e
   Path: coffeeshop/vagrant/snippets/2fa/coffeeshopsite/settings.py, line 187
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 0acea62a-ec63-40c9-b628-e1667e6a4fe4
   Path: coffeeshop/vagrant/snippets/apikeys/coffeeshopsite/settings.py, line 177
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 0f5d43e1-0a65-4c8e-9466-312e158adc43
   Path: coffeeshop/vagrant/snippets/loginalert/coffeeshopsite/settings.py, line 220
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 7c5b5d3b-6544-4b58-bd65-31c24f3ca11e
   Path: coffeeshop/vagrant/snippets/oauth2/auth_code/settings.py, line 184
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 27b67e44-ddfa-45cc-8868-b653d2a2657d
   Path: coffeeshop/vagrant/snippets/oauth2/oidc/settings.py, line 184
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 2a357628-2f54-4c68-bc95-ba16da619574
   Path: csthirdparty/vagrant/csthirdpartysite/csthirdparty/views.py, line 78
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 34d65d3f-c217-404d-8704-bafe1c3ebd33
   Path: coffeeshop/vagrant/coffeeshopsite/coffeeshop/views.py, line 310
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 03669d6c-6805-4d7e-a89c-53edc2ea31a5
   Path: coffeeshop/vagrant/snippets/apikeys/coffeeshop/views.py, line 431
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 76d79178-1a27-4363-842a-c2b7d93e4fe3
   Path: coffeeshop/vagrant/snippets/oauth2/auth_code/views.py, line 375
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: ea385573-5fa3-48aa-82cf-aefb0a96c4fa
   Path: coffeeshop/vagrant/snippets/oauth2/auth_code/views.py, line 417
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 38ec2ddd-80dc-4952-b34b-3d990e48b24a
   Path: coffeeshop/vagrant/snippets/oauth2/auth_code/views.py, line 433
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: d18ce758-b361-4529-aa52-cf9dfa161610
```

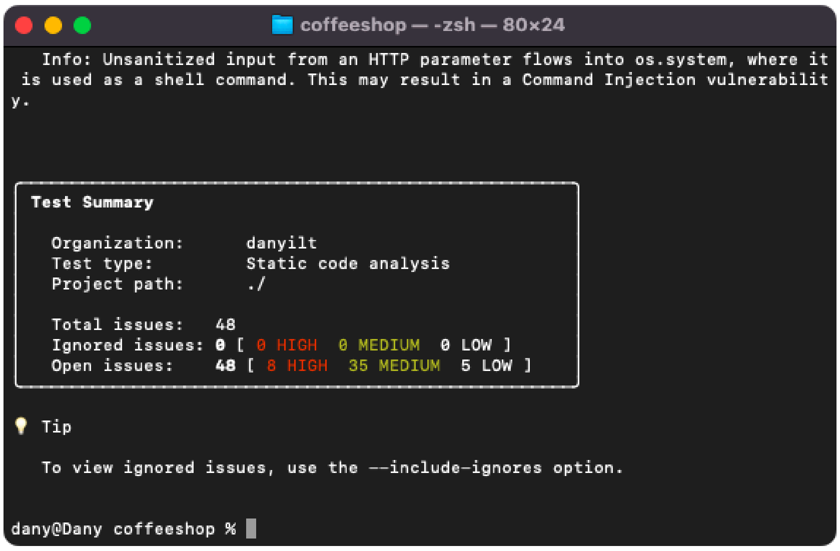
Fix: `vagrant/coffeeshopsite/coffeeshop/static/rest_framework/js/default.js`,
line `21`

```
$('a[data-toggle="tab"]').click(function() {
  document.cookie = "tabstyle=" + this.name + "; path=/";
});
```

↓

```
$('a[data-toggle="tab"]').click(function() {
  // Always set Secure to satisfy security scanners; on HTTP it may be ignored
  var cookie = "tabstyle=" + this.name + "; path=/; SameSite=Lax; Secure";
  document.cookie = cookie;
});
```

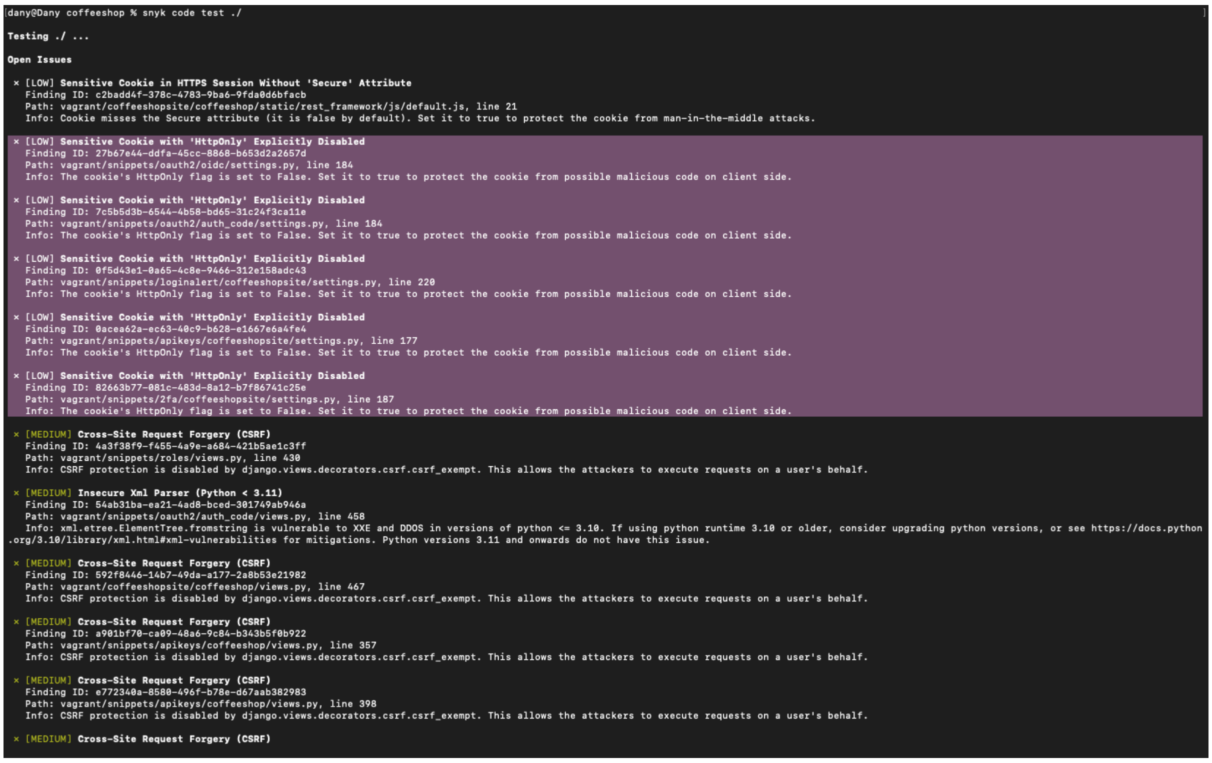
After fixing these vulnerabilities, run Snyk check again



```
  Info: Unsanitized input from an HTTP parameter flows into os.system, where it
 is used as a shell command. This may result in a Command Injection vulnerabilit
y.


 Test Summary

   Organization:      danyilt
   Test type:         Static code analysis
   Project path:      ./

   Total issues:   48
   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]
   Open issues:    48 [ 8 HIGH  35 MEDIUM  5 LOW ]

💡 Tip

   To view ignored issues, use the --include-ignores option.

dany@Dany coffeeshop %
```

These one now not detecting as vulnerability

## [LOW] Level Issue — Sensitive Cookie with 'HttpOnly' Explicitly Disabled



```
dany@Dany coffeeshop % snyk code test ./

Testing ./ ...

Open Issues

 × [LOW] Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
   Finding ID: c2badd4f-378c-4783-9ba6-9fda0d6bfacb
   Path: vagrant/coffeeshopsite/coffeeshop/static/rest_framework/js/default.js, line 21
   Info: Cookie misses the Secure attribute (it is false by default). Set it to true to protect the cookie from man-in-the-middle attacks.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 27b67e44-ddfa-45cc-8868-b653d2a2657d
   Path: vagrant/snippets/oauth2/oidc/settings.py, line 184
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 7c5b5d3b-6544-4b58-bd65-31c24f3ca11e
   Path: vagrant/snippets/oauth2/auth_code/settings.py, line 184
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 0f5d43e1-0a65-4c8e-9466-312e158adc43
   Path: vagrant/snippets/loginalert/coffeeshopsite/settings.py, line 220
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 0acea62a-ec63-40c9-b628-e1667e6a4fe4
   Path: vagrant/snippets/apikeys/coffeeshopsite/settings.py, line 177
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [LOW] Sensitive Cookie with 'HttpOnly' Explicitly Disabled
   Finding ID: 82663b77-081c-483d-8a12-b7f86741c25e
   Path: vagrant/snippets/2fa/coffeeshopsite/settings.py, line 187
   Info: The cookie's HttpOnly flag is set to False. Set it to true to protect the cookie from possible malicious code on client side.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 4a3f38f9-f455-4a9e-a684-421b5ae1c3ff
   Path: vagrant/snippets/roles/views.py, line 430
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Insecure Xml Parser (Python < 3.11)
   Finding ID: 54ab31ba-ea21-4ad8-bced-301749ab946a
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 458
   Info: xml.etree.ElementTree.fromstring is vulnerable to XXE and DDOS in versions of python <= 3.10. If using python runtime 3.10 or older, consider upgrading python versions, or see https://docs.python
.org/3.10/library/xml.html#xml-vulnerabilities for mitigations. Python versions 3.11 and onwards do not have this issue.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 592f8446-14b7-49da-a177-2a8b53e21982
   Path: vagrant/coffeeshopsite/coffeeshop/views.py, line 467
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: a901bf70-ca09-48a6-9c84-b343b5f0b922
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 357
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: e772340a-8580-496f-b78e-d67aab382983
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 398
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
```

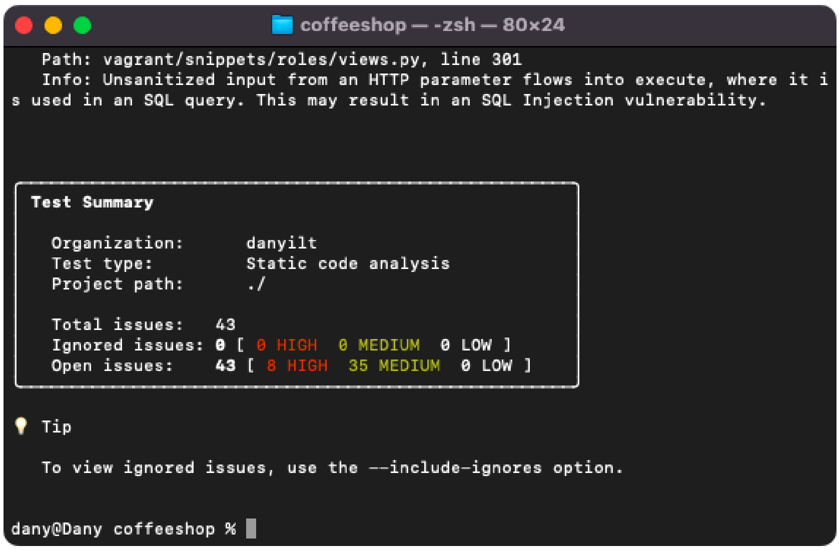
This 4 issues are the same, but in the different files:

```
SESSION_COOKIE_HTTPONLY = False
```

but should be:

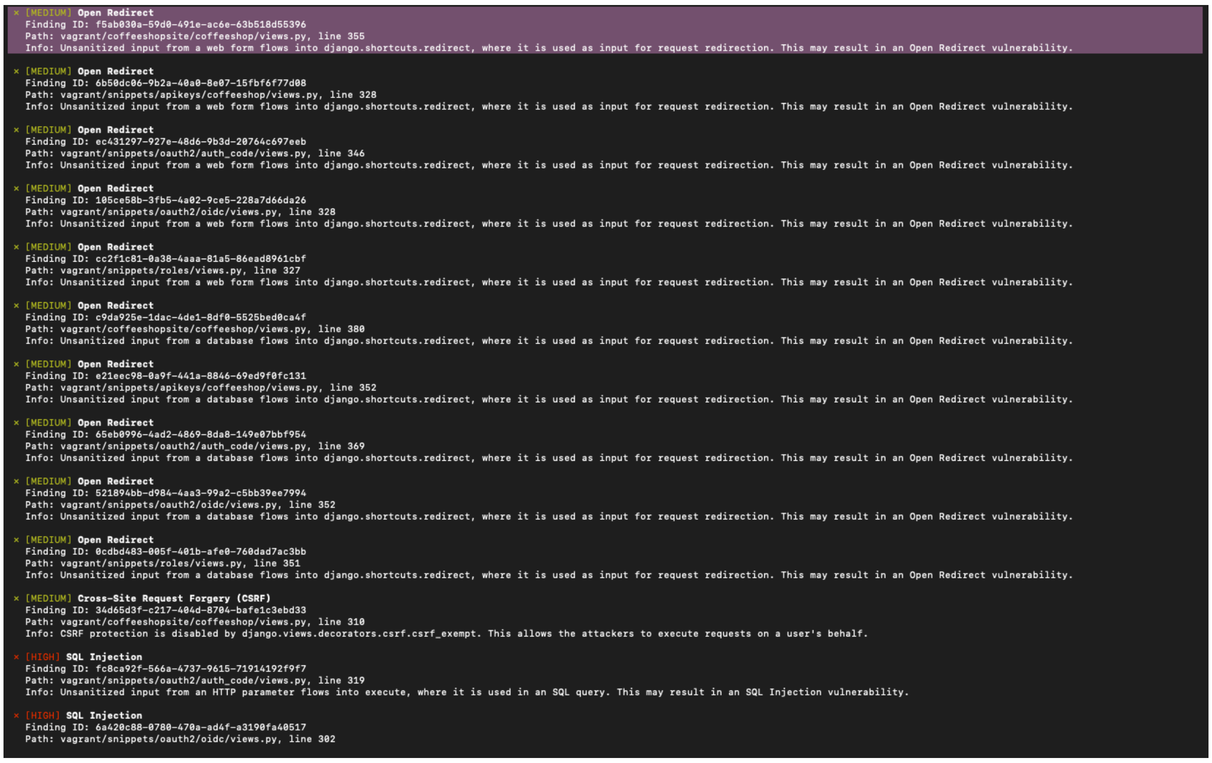
```
SESSION_COOKIE_HTTPONLY = True
```

## After fixing these vulnerabilities, run Snyk check again


```
coffeeshop — -zsh — 80×24
   Path: vagrant/snippets/roles/views.py, line 301
   Info: Unsanitized input from an HTTP parameter flows into execute, where it i
s used in an SQL query. This may result in an SQL Injection vulnerability.


 ┌──────────────────────────────────────────────────────┐
 │ Test Summary                                          │
 │                                                       │
 │   Organization:      danyilt                          │
 │   Test type:         Static code analysis             │
 │   Project path:      ./                               │
 │                                                       │
 │   Total issues:   43                                  │
 │   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]       │
 │   Open issues:    43 [ 8 HIGH  35 MEDIUM  0 LOW ]     │
 └──────────────────────────────────────────────────────┘

💡 Tip

   To view ignored issues, use the --include-ignores option.

dany@Dany coffeeshop % 
```

We can see that Snyk is not detect any Low level vulnerabilities

## [MEDIUM] Level Issue — Open Redirect


```
 × [MEDIUM] Open Redirect
   Finding ID: f5ab030a-59d0-491e-ac6e-63b518d55396
   Path: vagrant/coffeeshopsite/coffeeshop/views.py, line 355
   Info: Unsanitized input from a web form flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: 6b50dc06-9b2a-40a0-8e07-15fbf6f77d08
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 328
   Info: Unsanitized input from a web form flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: ec431297-927e-48d6-9b3d-20764c697eeb
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 346
   Info: Unsanitized input from a web form flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: 105ce58b-3fb5-4a02-9ce5-228a7d66da26
   Path: vagrant/snippets/oauth2/oidc/views.py, line 328
   Info: Unsanitized input from a web form flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: cc2f1c81-0a38-4aaa-81a5-86ead8961cbf
   Path: vagrant/snippets/roles/views.py, line 327
   Info: Unsanitized input from a web form flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: c9da925e-1dac-4de1-8df0-5525bed0ca4f
   Path: vagrant/coffeeshopsite/coffeeshop/views.py, line 380
   Info: Unsanitized input from a database flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: e21eec98-0a9f-441a-8846-69ed9f0fc131
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 352
   Info: Unsanitized input from a database flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: 65eb0996-4ad2-4869-8da8-149e07bbf954
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 369
   Info: Unsanitized input from a database flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: 521894bb-d984-4aa3-99a2-c5bb39ee7994
   Path: vagrant/snippets/oauth2/oidc/views.py, line 352
   Info: Unsanitized input from a database flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Open Redirect
   Finding ID: 0cdbd483-005f-401b-afe0-760dad7ac3bb
   Path: vagrant/snippets/roles/views.py, line 351
   Info: Unsanitized input from a database flows into django.shortcuts.redirect, where it is used as input for request redirection. This may result in an Open Redirect vulnerability.

 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 34d65d3f-c217-404d-8704-bafe1c3ebd33
   Path: vagrant/coffeeshopsite/coffeeshop/views.py, line 310
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [HIGH] SQL Injection
   Finding ID: fc8ca92f-566a-4737-9615-71914192f9f7
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 319
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 × [HIGH] SQL Injection
   Finding ID: 6a420c88-0780-470a-ad4f-a3190fa40517
   Path: vagrant/snippets/oauth2/oidc/views.py, line 302
```

Fixing the first file: `coffeeshop/vagrant/coffeeshopsite/coffeeshop/views.py`

line: `355`

```
next_url = "/product/" + str(product_id)
return redirect(next_url)
```

↓

```
return redirect(reverse('product', kwargs={'id': product_id}))
```

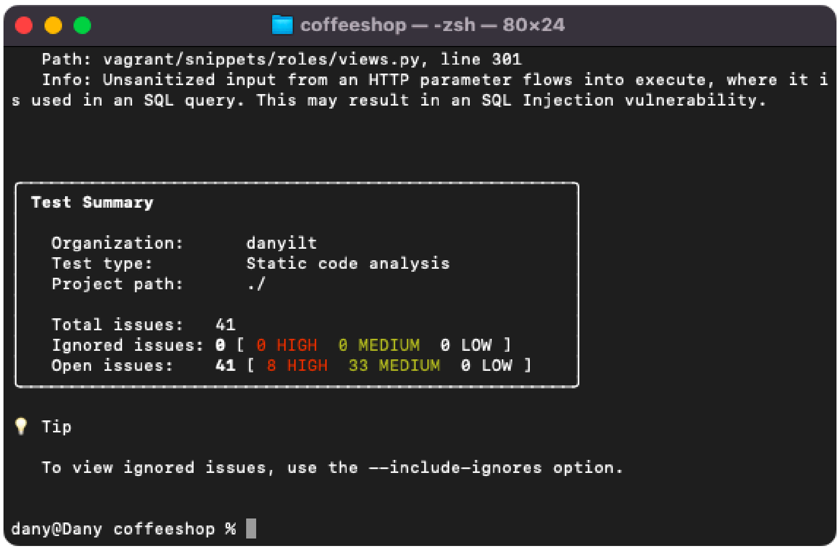
line: 380

```
next_url = "/product/" + str(product_id)
return redirect(next_url)
```

↓

```
return redirect(reverse('product', kwargs={'id': product_id}))
```

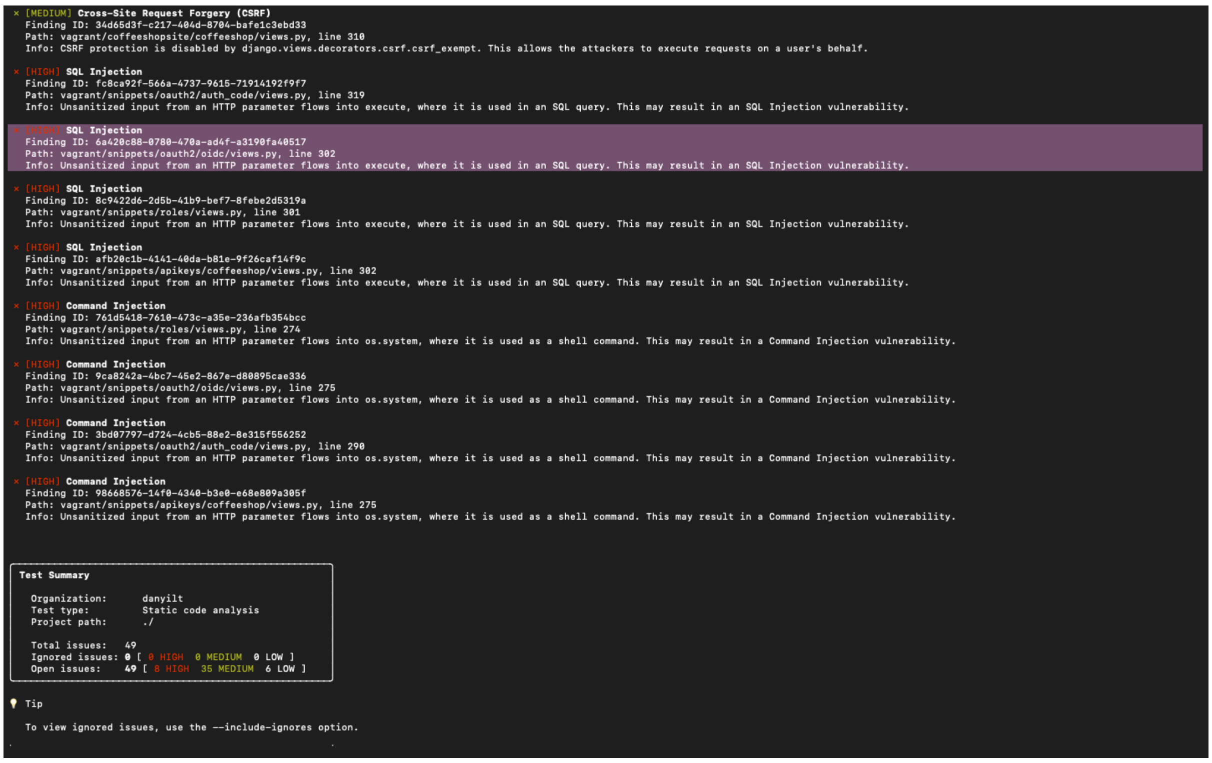
## After fixing these vulnerabilities, run Snyk check again



```
coffeeshop — -zsh — 80×24
   Path: vagrant/snippets/roles/views.py, line 301
   Info: Unsanitized input from an HTTP parameter flows into execute, where it i
s used in an SQL query. This may result in an SQL Injection vulnerability.



 ╭─────────────────────────────────────────────────────╮
 │ Test Summary                                        │
 │                                                     │
 │   Organization:      danyilt                        │
 │   Test type:         Static code analysis           │
 │   Project path:      ./                             │
 │                                                     │
 │   Total issues:   41                                │
 │   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]     │
 │   Open issues:    41 [ 8 HIGH  33 MEDIUM  0 LOW ]   │
 ╰─────────────────────────────────────────────────────╯

💡 Tip

   To view ignored issues, use the --include-ignores option.

dany@Dany coffeeshop % ▌
```

As we can see that there is now two fixed Medium vulnerabilities

## [HIGH] Level Issue — SQL Injection



```
 ✗ [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 34d65d3f-c217-404d-8704-bafe1c3ebd33
   Path: vagrant/coffeeshopsite/coffeeshop/views.py, line 310
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 ✗ [HIGH] SQL Injection
   Finding ID: fc8ca92f-566a-4737-9615-71914192f9f7
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 319
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 ✗ [HIGH] SQL Injection
   Finding ID: 6a420c88-0780-470a-ad4f-a3190fa40517
   Path: vagrant/snippets/oauth2/oidc/views.py, line 302
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 ✗ [HIGH] SQL Injection
   Finding ID: 8c9422d6-2d5b-41b9-bef7-8febe2d5319a
   Path: vagrant/snippets/roles/views.py, line 301
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 ✗ [HIGH] SQL Injection
   Finding ID: afb20c1b-4141-40da-b81e-9f26caf14f9c
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 302
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 ✗ [HIGH] Command Injection
   Finding ID: 761d5418-7610-473c-a35e-236afb354bcc
   Path: vagrant/snippets/roles/views.py, line 274
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.

 ✗ [HIGH] Command Injection
   Finding ID: 9ca8242a-4bc7-45e2-867e-d80895cae336
   Path: vagrant/snippets/oauth2/oidc/views.py, line 275
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.

 ✗ [HIGH] Command Injection
   Finding ID: 3bd07797-d724-4cb5-88e2-8e315f556252
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 290
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.

 ✗ [HIGH] Command Injection
   Finding ID: 98668576-14f0-4340-b3e0-e68e809a305f
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 275
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.



 ╭─────────────────────────────────────────────────────╮
 │ Test Summary                                        │
 │                                                     │
 │   Organization:      danyilt                        │
 │   Test type:         Static code analysis           │
 │   Project path:      ./                             │
 │                                                     │
 │   Total issues:   49                                │
 │   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]     │
 │   Open issues:    49 [ 8 HIGH  35 MEDIUM  6 LOW ]   │
 ╰─────────────────────────────────────────────────────╯

💡 Tip

   To view ignored issues, use the --include-ignores option.
```

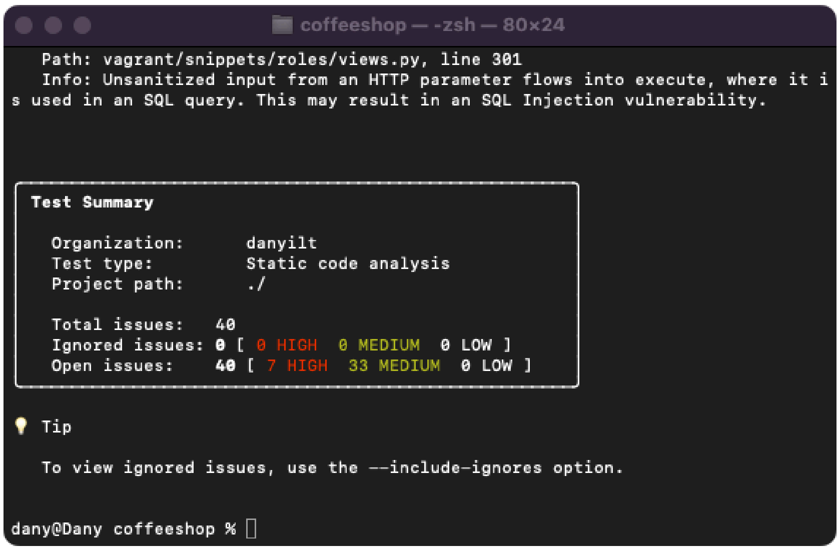
Fixing the same file: `coffeeshop/vagrant/snippets/oauth2/oidc/views.py`, line `302`

```
with connection.cursor() as cursor:
    sql = '''SELECT id, name, description, unit_price FROM coffeeshop_product WHERE
(LOWER(name) like '%{}%' or LOWER(description) like
'%{}%')'''.format(search_text.lower(), search_text.lower())
    print(sql)
    products = []
    try:
        cursor.execute(sql)
        for row in cursor.fetchall():
            (pk, name, description, unit_price) = row
            product = Product(id=pk, name=name, description=description,
unit_price=unit_price)
            products.append(product)
```

↓

```
with connection.cursor() as cursor:
    sql = '''SELECT id, name, description, unit_price FROM coffeeshop_product WHERE
(LOWER(name) LIKE %s OR LOWER(description) LIKE %s)'''
    products = []
    try:
        # Escape any % characters in user input to avoid unintended wildcards
        search_term = '%' + search_text.lower().replace('%', '%%') + '%'
        cursor.execute(sql, [search_term, search_term])
        for row in cursor.fetchall():
            (pk, name, description, unit_price) = row
            product = Product(id=pk, name=name, description=description,
unit_price=unit_price)
            products.append(product)
```

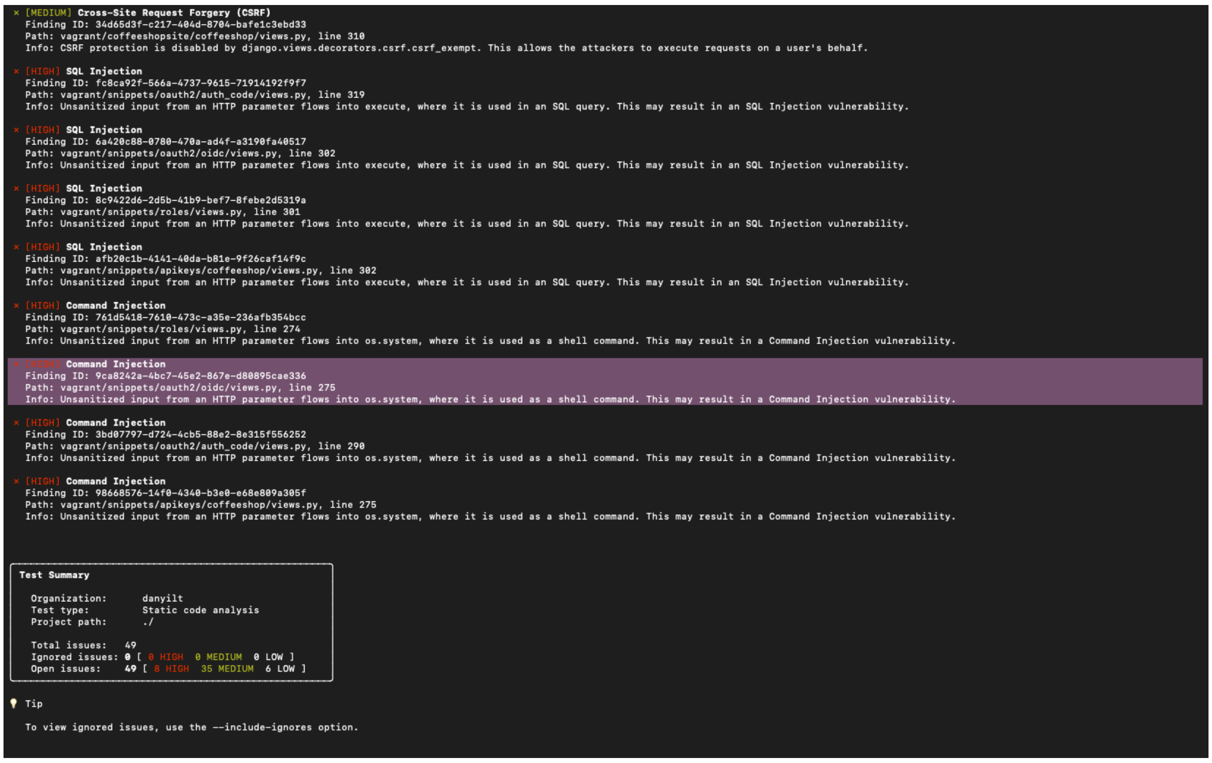
After fixing these vulnerabilities, run Snyk check again



```
   Path: vagrant/snippets/roles/views.py, line 301
   Info: Unsanitized input from an HTTP parameter flows into execute, where it i
s used in an SQL query. This may result in an SQL Injection vulnerability.


╭─────────────────────────────────────────────────────────╮
│ Test Summary                                            │
│                                                         │
│   Organization:      danyilt                            │
│   Test type:         Static code analysis               │
│   Project path:      ./                                 │
│                                                         │
│   Total issues:   40                                    │
│   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]         │
│   Open issues:    40 [ 7 HIGH  33 MEDIUM  0 LOW ]       │
╰─────────────────────────────────────────────────────────╯

💡 Tip

   To view ignored issues, use the --include-ignores option.

dany@Dany coffeeshop % 
```

We can see that there is now one fixed Hard vulnerability

## [HIGH] Level Issue — Command Injection



```
 × [MEDIUM] Cross-Site Request Forgery (CSRF)
   Finding ID: 34d65d3f-c217-404d-8704-bafe1c3ebd33
   Path: vagrant/coffeeshopsite/coffeeshop/views.py, line 310
   Info: CSRF protection is disabled by django.views.decorators.csrf.csrf_exempt. This allows the attackers to execute requests on a user's behalf.

 × [HIGH] SQL Injection
   Finding ID: fc8ca92f-566a-4737-9615-71914192f9f7
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 319
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 × [HIGH] SQL Injection
   Finding ID: 6a420c88-0780-470a-ad4f-a3190fa40517
   Path: vagrant/snippets/oauth2/oidc/views.py, line 302
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 × [HIGH] SQL Injection
   Finding ID: 8c9422d6-2d5b-41b9-bef7-8febe2d5319a
   Path: vagrant/snippets/roles/views.py, line 301
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 × [HIGH] SQL Injection
   Finding ID: afb20c1b-4141-40da-b81e-9f26caf14f9c
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 302
   Info: Unsanitized input from an HTTP parameter flows into execute, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 × [HIGH] Command Injection
   Finding ID: 761d5418-7610-473c-a35e-236afb354bcc
   Path: vagrant/snippets/roles/views.py, line 274
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.

 × [HIGH] Command Injection
   Finding ID: 9ca8242a-4bc7-45e2-867e-d80895cae336
   Path: vagrant/snippets/oauth2/oidc/views.py, line 275
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.

 × [HIGH] Command Injection
   Finding ID: 3bd07797-d724-4cb5-88e2-8e315f556252
   Path: vagrant/snippets/oauth2/auth_code/views.py, line 290
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.

 × [HIGH] Command Injection
   Finding ID: 98668576-14f0-4340-b3e0-e68e809a305f
   Path: vagrant/snippets/apikeys/coffeeshop/views.py, line 275
   Info: Unsanitized input from an HTTP parameter flows into os.system, where it is used as a shell command. This may result in a Command Injection vulnerability.


╭─────────────────────────────────────────────────────────╮
│ Test Summary                                            │
│                                                         │
│   Organization:      danyilt                            │
│   Test type:         Static code analysis               │
│   Project path:      ./                                 │
│                                                         │
│   Total issues:   49                                    │
│   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]         │
│   Open issues:    49 [ 8 HIGH  35 MEDIUM  6 LOW ]       │
╰─────────────────────────────────────────────────────────╯

💡 Tip

   To view ignored issues, use the --include-ignores option.
```

Fixing: `coffeeshop/vagrant/snippets/oauth2/oidc/views.py`, line `275`

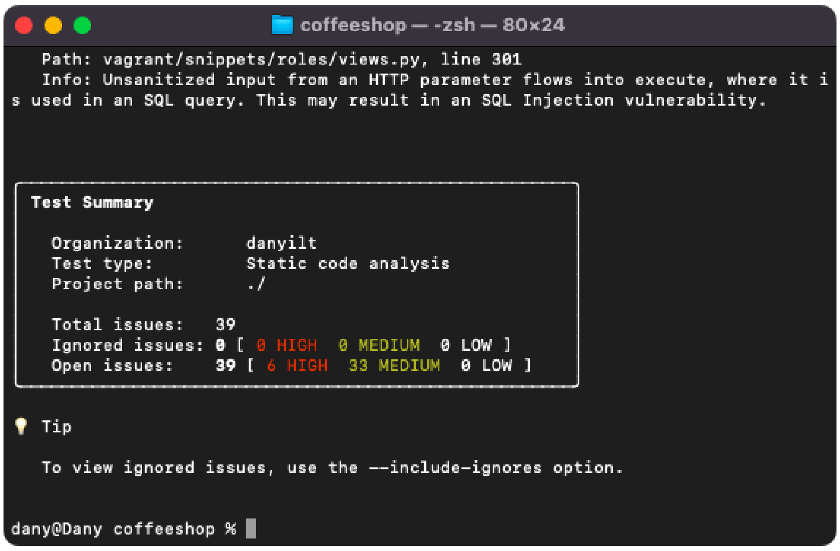
```
cmd = ' printf "From: ' + request.user.email + '\nSubject: CoffeeShop User
Contact\n\n' + body + '" | ssmtp contact@coffeeshop.com'
print(cmd)
os.system(cmd)
```

↓

```
import re
# Sanitize and safely send email without invoking a shell command
```

```
raw_body = request.POST['message']
# Allow common punctuation, limit length to prevent abuse
safe_body = re.sub(r'[^\w\s\.,!?@\-]', '', raw_body)[:2000]
    send_mail(
    'CoffeeShop User Contact',
    safe_body,
    request.user.email,
    ['contact@coffeeshop.com'],
    fail_silently=False,
)
```

After fixing these vulnerabilities, run Snyk check again


```
  Path: vagrant/snippets/roles/views.py, line 301
  Info: Unsanitized input from an HTTP parameter flows into execute, where it i
s used in an SQL query. This may result in an SQL Injection vulnerability.


 Test Summary

   Organization:      danyilt
   Test type:         Static code analysis
   Project path:      ./

   Total issues:   39
   Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]
   Open issues:    39 [ 6 HIGH  33 MEDIUM  0 LOW ]

 Tip

  To view ignored issues, use the --include-ignores option.

dany@Dany coffeeshop %
```

We can see that there is now one fixed Hard vulnerability

## Fixing the Vulnerability (link to the commit)

DanyilT/django-coffeeshop repo forked from stephen-oshaughnessy/django-coffeeshop

Snyk's vulnerabilities fix (settings.py, views.py):
DanyilT/django-coffeeshop/commit/0698114cad1c297d7b1683f7c568ca18b4d6199c