Vul AI
Vulnerability Detection Report
File Name: views.py
Date: 2024-09-10 22:47:24.446962

Our System has analyzed the file views.py and found total of 16 functions in the file
Each function is analyzed by our fined tuned model to detect for possible vulnerabilities
A Final report has been generated covering all functions

GENERATED REPORT

TOTAL FUNCTIONS IN FILE 16

No.	Function Name	Original Code	Vulnerability Status	CWE IDS
1	def get(self,request)	def get(self,request): if request.user.is_authenticated: log_type = 'User' cuser = request.user.email if UserModel.objects.filter(uemail=cuser): return redirect('developer') elif RecruiterModel.objects.filter(remail=cuser): return redirect('recruiter') elif request.user.is_superuser: return redirect('Administrator') else: return render(request,'portal.html')	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
2	def get(self,request)	def get(self,request): logout(request) return redirect('/')	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
3	def post(self,request)	def post(self,request): if request.method == 'POST': username = request.POST['aname'] password = request.POST['psswd'] user = authenticate(username=username, password=password) print("AAYA1") try: if user.is_staff: login(request, user) return redirect('Administrator') # Redirect to a success page. else: # message = messages.error(request, 'Sorry you are not an admin') message= "Sorry you are not authoirized to login as admin Stay In your Limits !!" return render(request, 'admin_login.html',{'message':message})	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
4	def get(self,request)	def get(self,request): return render(request,'admin_login.html')	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
5	def get(self,request)	def get(self,request): if request.user.is_authenticated: if request.user.is_superuser: w = request.GET.get('who') developers = UserModel.objects.all() recruiters = RecruiterModel.objects.all() if 'users' == w: return render(request,'administrator.html',{'developers':developers,'who':'user'}) elif 'recruiter' == w: return render(request,'administrator.html',{'recruiters':recruiters,'who':'recruiter'}) elif w == 'pending': recruiters = RecruiterModel.objects.filter(status='pending') return render(request,'administrator.html',{'recruiters':recruiters,'who':'recruiter'})	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']

6	def get(self,request,pid,type)	def get(self,request,pid,type): print(pid,'pid==========',type) if request.user.is_authenticated: if type == 'users': # user = UserModel.objects.get(id=pid) obj = User.objects.get(id=pid) elif type == 'recruiter': print('recruiter==========') # user = RecruiterModel.objects.get(id=pid) obj = User.objects.get(id=pid) print(obj,'obj==========') print('USERID+++++++',obj) # user.delete() obj.delete()	The code is vulnerable to SQL Injection . This occurs when user input is not properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
7	def get(self,request,pid)	def get(self,request,pid): if request.user.is_authenticated: user = RecruiterModel.objects.get(id=pid) if user.status == ('pending' or 'Pending'): user.status = 'active' else: user.status = 'pending' user.save() return redirect(reverse('Administrator')+'?who=recruiter') return render(request,'administrator.html')	The code is vulnerable to SQL Injection . This occurs when user input is not properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
8	def post(self,request)	def post(self,request): if request.method == 'POST': inorup = request.POST['inorup'] # FOR LOGIN if inorup == 'login': username = request.POST['uname'] password = request.POST['psswd'] user = authenticate(username=username, password=password) if user: try: # userm = UserModel.objects.get(user=user) if UserModel.objects.filter(user=user): # print(userm.type) # if userm.user_type == 'Applicant': login(request, user) print('SUCCSESS') message = "Login Successfull" return redirect('developer') else: message = "Sorry you are not an applicant" return render(request,'user_login.html',{'message':message}) except Exception as e: print(e) messages.error(request, 'Invalid Username or Password') return render(request, 'user_login.html',e) else: message = 'Invalid Username or Password' return render(request, 'user_login.html',{'message':message}) return redirect('Portal')	The code is vulnerable to SQL Injection . This occurs when user input is not properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
9	def get(self,request)	def get(self,request): if request.user.is_authenticated: log_type = 'User' cuser = request.user.email return redirect('developer') else: log_type = 'User'	The code is vulnerable to SQL Injection . This occurs when user input is not properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']

10	def post(self,request)	def post(self,request): if request.method == 'POST': inorup = request.POST['inorup'] # FOR LOGIN if inorup == 'login': username = request.POST['rname'] password = request.POST['psswd'] user = authenticate(username=username, password=password) if user: try: # userm = RecruiterModel.objects.get(user=user) if RecruiterModel.objects.filter(user=user): # if userm.user_type == 'Recruiter': login(request, user) print('SUCCSESS') message = "Login Successfull" return redirect('recruiter') else: message = "Sorry you are not an recruiter" return render(request,'recruiter_login.html',{'message':message}) except Exception as e: print(e) messages.error(request, 'Invalid Username or Password') return render(request, 'recruiter_login.html',e) else: message = 'Invalid Username or Password' return render(request, 'recruiter_login.html',{'message':message}) return redirect('Portal')	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
11	def get(self,request)	def get(self,request): if request.user.is_authenticated: log_type = 'Recruiter' return redirect('recruiter') else: log_type = 'Recruiter' return render(request,'recruiter_login.html',{'log_type':log_type})	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
12	def post(self,request)	def post(self,request): if request.method == 'POST': try: uemail = request.POST.get('uemail') uname = request.POST.get('uname') uphno = request.POST.get('uphno') domain = request.POST.get('domain') user_type = request.POST.get('user_type') age = request.POST.get('age') experience = request.POST.get('experience') resume = request.FILES.get('resume') image = request.FILES.get('image') city = request.POST.get('city') aboutu = request.POST.get('aboutu') freelance = request.POST.get('freelance') if freelance == 'on': freelance = 1 else: freelance = 0	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
13	def get(self,request)	def get(self,request): return render(request,'signup.html')	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
14	def get(self,request)	def get(self,request): if not request.user.is_authenticated: return redirect('Portal') else: return render(request,'change_password.html')	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']

15	def post(self,request)	def post(self,request): if request.method == 'POST': old_password = request.POST.get('old_password') new_password = request.POST.get('new_password') confirm_password = request.POST.get('confirm_password') who = request.POST.get('who') if new_password == confirm_password: user = authenticate(username=request.user.email, password=old_password) if who == 'recruiter': userm = RecruiterModel.objects.get(user=user) elif who == 'developer': userm = UserModel.objects.get(user=user) if user: user.set_password(new_password) user.save() userm.password = new_password userm.save() # return redirect('Portal') return render(request,'change_password.html',{'message':'Password Changed'})	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
16	def get(self,request)	def get(self,request): if not request.user.is_authenticated: return redirect('job/user-login') else: user = request.user cuser = RecruiterModel.objects.get(remail=user.username) developers = UserModel.objects.all() recruiters = RecruiterModel.objects.all() return render(request,'recruiter.html',{'cuser':cuser,'recruiters':recruiters,'developers':developers })	The code is vulnerable to SQL Injection . This occurs when user input is not CWE-53 properly sanitized CWE-23 CWE 089 before being used in a SQL query. In this case, the 'username' variable is directly inserted into the SQL query, which allows an attacker to manipulate the query and potentially delete any user from the database	['CWE-53', 'CWE-23']
Total Vulneratble Functions				2

Vulnerability Description And Effects on Robotic Systems

CWE ID	How it can effect robotic systems
CWE-23	"Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')", allows an attacker to inject malicious SQL code into a robotic system's database, compromising its: Data integrity and confidentiality System operations and availability Authentication and authorization mechanisms This can lead to unauthorized data access, system downtime, and manipulation of the robotic system's actions or movements, posing a risk to human safety and system reliability.
CWE-53	"Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')", allows an attacker to inject malicious SQL code into a robotic system's database, compromising its: Data integrity and confidentiality System operations and availability Authentication and authorization mechanisms This can lead to unauthorized data access, system downtime, and manipulation of the robotic system's actions or movements, posing a risk to human safety and system reliability.
Total Vulneratble Functions	2