


Vulnerability List

Vuln. Num.	Summary	Location	Consequences/ Impact	CVSS Score	Proof of concept
ex)	A brief description of the vulnerability.	Information needed to locate the vulnerability in the system, such as components, file(s), and lines of code.	What are the consequences of exploitation? For example, "Remote, unauthenticated code execution" and conduct "denial-of-service".	Use CVSS version 3.1 to compute the score.	Steps to trigger the vulnerability, either written in prose or as code to demonstrate exploitation.
1	Allow GET method in login	<a href="#">Server: usermanagement.py</a>  <pre>@app.route("/login", methods=['GET', 'POST']) def login():     if request.method == 'GET':         form_user = request.args.get("username")         form_pass = request.args.get("password")         elif request.method == 'POST':             form_user = request.form.get("username")             form_pass = request.form.get("password")             print(form_user, form_pass)             if form_user == app.config["USER"] and form_pass == app.config["PASS"]:                 session["username"] = form_user                 #does the user have 2FA enabled? Let's assume so...                 return redirect(url_for("OTP_auth"))             else:                 flash("Invalid credentials. Please try again.")                 return redirect(url_for("index"))</pre>	<p>GET method in login can disclose user credential in URL parameters.</p> <p><a href="https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#sensitive-information-in-http-requests">https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#sensitive-information-in-http-requests</a></p> <p><b>INFORMATION DISCLOSURE</b></p>	High 8.3 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:L/A:L	<p>curl -X POST https://myflask-ltkud.run.goorm.io/login -H "Content-Type: application/x-www-form-urlencoded" -d "username=username&amp;password=password" ==&gt; Allowed and login OK</p> <p>curl https://myflask-ltkud.run.goorm.io/login?username=username&amp;password=password ==&gt; Allowed and login OK too</p>
2	The contents of JWT token is base64 encoded but no ciphering	<a href="#">Server: usermanagement.py</a>  <pre>token = jwt.encode({'public_id': 'brian', 'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=45)}, _JWT_SECRET_KEY, "HS256")</pre>	<p>ASSUMPTION: an attacker has access to a JWT token.</p> <p>This attacker can extract information stored in it. Information can be for example the security roles, login format, etc.</p> <p><a href="https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html#token-information-disclosure">https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html#token-information-disclosure</a></p> <p><b>INFORMATION DISCLOSURE</b></p>	High 7.7 CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:C/C:H/I:L/A:H	<p>token = jwt.encode({'public_id': 'brian', 'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=45)}, _JWT_SECRET_KEY, "HS256")</p> <p>eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiJicmlhbiIsImV4cCI6MTY1NjQxNDMwMX0.XoSBNNuEyYms0fOhPMCa4Qz0uZ3FruHcc5wjaM0YFrg</p> <p>Simply decode base64 ==&gt;</p> <pre>{ "typ": "JWT", "alg": "HS256" } { "public_id": "brian", "exp": 1656414301 }   MG▼:!!  KQkw-▯3F </pre>
3	JWT tokens signed using weak keys can be found by bruteforce attack	<a href="#">Server: usermanagement.py</a>  <pre>_JWT_SECRET_KEY = 'life is good' token = jwt.encode({'public_id': 'brian', 'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=45)}, _JWT_SECRET_KEY, "HS256")</pre>	<p>ASSUMPTION: an attacker has access to a JWT token.</p> <p>If JWT tokens signed using weak keys, an attacker can use Hashcat to bruteforce the correct secret key.</p> <p><b>SPOOFING, TAMPERING</b></p>	High 7.4 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:C/C:H/I:L/A:H	<pre>\$ echo 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiJicmlhbiIsImV4cCI6MTY1NjkyNDE2MX0.jt6dl35XijwVzii9C-A6u2LsC1UKb39YbGDO3uEa4n8' &gt; hash.txt</pre> <pre>\$ hashcat -a 0 -m 16500 hash.txt /usr/share/wordlists/rockyou.txt --force</pre> <pre>... eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiJicmlhbiIsImV4cCI6MTY1NjkyNDE2MX0.jt6dl35XijwVzii9C-A6u2LsC1UKb39YbGDO3uEa4n8:life is good ... \$</pre>
4	No built-in token revocation by the client	The server does not provide token revocation explicitly.	<p>ASSUMPTION:</p> <p>1) An attacker has access to a JWT token.</p> <p>2) The token expires after one day.</p> <p>This problem is inherent to JWT because a token only becomes invalid when it expires. The user has no built-in feature to explicitly revoke the validity of a token. This means that if it is stolen, a user cannot revoke the token itself thereby blocking the attacker.</p> <p><a href="https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html#no-built-in-token-revocation-by-the-user">https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html#no-built-in-token-revocation-by-the-user</a></p> <p><b>SPOOFING, REPUDIATION</b></p>	Medium 6.4 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:C/C:L/I:L/A:H	<p>1) The client sign-ins the server.</p> <p>2) The client receives a JWT token with one day expiration.</p> <p>3) The attacker has access the token.</p> <p>4) The client sign-outs the server.</p> <p>5) The attacker can still use the token before expiration, because the server does not provide token revocation explicitly upon client sign-out.</p>
5	No input file validation by ALPR.exe	Client: ALPR.exe	<p>ASSUMPTION: an attacker has access to video files.</p> <p>The attacker delivers a file for malicious intent.</p> <p><a href="https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html#file-upload-threats">https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html#file-upload-threats</a></p> <p><b>TAMPERING</b></p>	Medium 6.4 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	<pre>C:\Users\user\source\repos\OpenALPR\x64\Release&gt;copy "C:\Users\user\Downloads\myflask2.zip" my.avi 1개 파일이 복사되었습니다.</pre> <pre>C:\Users\user\source\repos\OpenALPR\x64\Release&gt;alpr my.avi [ERROR:0@1.702] global C:\build\master_winpack-build-win64-vc15\opencv\modules\videoio\src\cap.cpp (166) cv::VideoCapture::open VIDEOIO (CV_IMAGES): raised OpenCV exception:  OpenCV(4.5.5) C:\build\master_winpack-build-win64-vc15\opencv\modules\videoio\src\cap_images.cpp:253: error: (-5:Bad argument) CAP_IMAGES: can't find starting number (in the name of file): my.avi in function 'cv::icvExtractPattern'</pre> <p>C:\Users\user\source\repos\OpenALPR\x64\Release&gt;</p>
6	TOCTTOU attack to ALPR.exe input file	Client: ALPR.exe	<p>ASSUMPTION: an attacker has access to video files.</p> <p>The attacker changes the file selected by the user before clicking 'Start view'.</p> <p><b>TAMPERING</b></p>	Medium 6.4 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	<p>1) The user selects a video file named 'my.avi' and don't click 'Start view'.</p> <p>2) The attacker changes 'my.avi' with the same name but corrupted video file.</p> <p>3) The user clicks 'Start view' without knowing that the file has been changed.</p>

Vulnerability List

Vuln. Num.	Summary	Location	Consequences/ Impact	CVSS Score	Proof of concept
7	A route to find user's OTP QR code is still open	Server: <a href="#">usermanagement.py</a>  @login_view.route("/login/auth/<user>", methods=['GET']) def OTP_auth(valid_user, user): print("OTP_auth") data = valid_user      # ID 조회Query 실행  if data is not None: #ToDo : QR코드 제공 시 e-mail 정보를 삭제 가능한지 확인 필요 return render_template('auth.html', secret_key=data.otp, prov_uri=pyotp.TOTP(data.otp).provisioning_uri (issuer_name='3team Studio Project')) else: flash("Invalid user. Please try again.") return redirect(url_for("login.index"))	ASSUMPTION: an attacker knows one of user ID (email).  An attacker can discover development items left behind that are just hooked into the server. The OTP QR code is one of user credentials to access the system. If the attacker has access to it, spoofing is completely possible.  SPOOFING	Medium 6.7 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:C/L:I/L:A:H	1) curl -X GET https://myflask-ltkud.run.goorm.io/login/auth/testid 2) OTP QR code and registration code will be returned as response
8	Easily change whether you are admin in user information	Server: <a href="#">database.py</a>  class User(db.Model): tablename = "user"  id = db.Column(db.Integer, primary_key=True) username = db.Column(db.String(80), unique=True) password = db.Column(EncryptedType(db.Unicode, key, AesEngine, "pkcs5")) otp = db.Column(EncryptedType(db.Unicode, key, AesEngine, "pkcs5")) passwordfent = db.Column(db.Integer, default=0) otpfent = db.Column(db.Integer, default=0) admin = db.Column(db.Boolean)	There is a path for regular users to become administrators without creating a separate administrator account. Elevation of privilege allows malicious behavior.  SPOOFING, INFORMATION DISCLOSURE, EOP	High 7.4 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:C/L:I/H:A:L	
9	Turn off debug PIN of Flask app	Server: The Werkzeug Debugger of Flask app  export WERKZEUG_DEBUG_PIN=off	A publicly exposed debugger will subject the machine to remote code execution. Once attackers can execute arbitrary Python code on the server, he can directly leak all the sensitive data stored on the server.  <a href="https://medium.com/swlh/hacking-flask-applications-939eae4bffed">https://medium.com/swlh/hacking-flask-applications-939eae4bffed</a>  INFORMATION DISCLOSURE, EOP	High 7.4 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:C/L:I/H:A:L	1) The first step is to induce an internal server error in the application. 2) The debugger gets triggered when there is an internal server error. 3) Notice that there is an interactive Python console at the bottom of the traceback page. 4) Inside the console, you can execute any kind of Python code on the local machine.
10	The client source code is visible from the Sources tab of the dev tools	Client: The Sources tab of the browser dev tools	The client source code shows the server request URL and format details. It also tells how/where the client stores a token. An attacker can refer to these information in DDoS attacks.  INFORMATION DISCLOSURE, DENIAL-OF-SERVICE	Medium 6.1 CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:C/L:I/N:A:H	1) Open the browser dev tools and go to the Sources tab 2) You can see "https://test-jskak.run.goorm.io/plate?plate_number=", "https://test-jskak.run.goorm.io/signin?username=" etc. in server.js 3) You can see "const jwt = sessionStorage.getItem("key");" in index.js

**ASSUMPTION: an attacker has access to video files.**

The client can use a network drive to load or save video files. The reason for using a network drive is to provide convenience when submitting captured videos to the police station or viewing videos sent by other police officers. An attacker has found the network drive location and managed to get access to it.

**ASSUMPTION: an attacker has access to a JWT token.**

The client runs in a browser and keeps the token in a browser session storage. When a police officer creates a new browser tab and visits malicious site, an attacker can have access to the storage using XSS attacks.

CVSS v3.1 Base Score Calculator

ATTACK VECTOR (공격 벡터)	ATTACK COMPLEXITY (공격의 복잡성)	PRIVILEGES REQUIRED (필요한 권한)	USER INTERACTION (사용자 참여 정도)
Network	Low	None	None
Adjacent	High	Low	Required
Local		High	
Physical			

SCOPE (공격 범위)	CONFIDENTIALITY (기밀성)	INTEGRITY (무결성)	AVAILABILITY (가용성)
Changed	High	High	High
Unchanged	Low	Low	Low
	None	None	None

SEVERITY SCORE VECTOR

CVSS:3.1/AV:./AC:./PR:./UI:./S:./C:./I:./A:./