Short description of findings	Related location	File location	Start line	Start column	End line	End column	Severity	Justification	Remediation
1)Clear-text logging of sensitive information	[["Sensitive data (password)"|"relative:///bad/brute.py:9:13:18:1"]] is logged here. [["Sensitive data (password)"|"relative:///bad/brute.py:20:17:20:25"]] is logged here.	/bad/brute.py	23	15	23	73	Critical	Hardcoding passwords is an unacceptable programming practice. An adversary can fish for the passswords and tamper with the application in unintended ways.	We should use secrets and .env files instead to work with passwords. We can also use encrypted passwords and decrypt before use as an alternative but it is better to separate the password from the code to reduce likeliness of password guessing.
2)SQL query built from user-controlled sources	This SQL query depends on [["a user-provided value"|"relative:///bad/mod_api.py:32:12:32:18"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_api.py:34:12:34:18"]]. This SQL query depends on [["a user-provided value"|"relative:///bad/mod_user.py:14:8:14:14"]]. This SQL query depends on [["a user-provided value"|"relative:///bad/mod_user.py:16:20:16:26"]]. This SQL query depends on [["a user-provided value"|"relative:///bad/mod_user.py:17:20:17:26"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:15:8:15:14"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:17:20:17:26"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:18:20:18:26"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:44:8:44:14"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:46:20:46:26"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:47:20:47:26"]]. This SQL query depends on [["a user-provided value"|"relative:///good/mod_user.py:69:24:69:30"]].	/bad/libuser.py	12	22	12	111	High	A skilled adversary that can work with database queries can manipulate the database with the inputs for username and password that are passed onto libusers.py. They could potentially manipulate the database to let themselves login as the first user or reveal a list of all usernames and password. In the worst case depending on the access controls of the database an adversary could gain administrative access by finding the administrator credentials through a manipulated query.	Using the following stucture to process passwords and usernames is a best practice for sqlite3. c.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password)) This way the values are derived from tuples and matched based on their value instead of being appended directly to the query and potentially causing a manipulation of special characters (* ") or commands.
3)SQL query built from user-controlled sources	This SQL query depends on [["a user-provided value"|"relative:///bad/mod_user.py:43:8:43:14"]]. This SQL query depends on [["a user-provided value"|"relative:///bad/mod_user.py:45:20:45:26"]]. This SQL query depends on [["a user-provided value"|"relative:///bad/mod_user.py:46:20:46:26"]].	/bad/libuser.py	25	15	25	159	High	This vulnerability is similar to the one above as a skilled adversary can add input to a query and it will concatenate to the data base. Someone could potentially add " '); DELETE FROM users; -- "as the mfa_secret if they know enough about the structure to the database or see the code and with this they delete all users from the database.	The following is the recommended way to call parametized inserts for sqlite3 c.execute ("INSERT INTO users (username = ?, password = ?, failures = ?, mfa_enabled = ?, mfa_secret = ?) VALUES (?, ?, ?, ?, ?)", (username, password, failures, mfa_enabled, mfa_secret)
4)Reflected server-side cross-site scripting	Cross-site scripting vulnerability due to [["a user-provided value"|"relative:///bad/libsession.py:5:12:5:19"]]. Cross-site scripting vulnerability due to [["a user-provided value"|"relative:///bad/libsession.py:7:5:7:12"]]. Cross-site scripting vulnerability due to [["a user-provided value"|"relative:///bad/libsession.py:8:12:8:19"]]. Cross-site scripting vulnerability due to [["a user-provided value"|"relative:///bad/mod_user.py:14:8:14:14"]]. Cross-site scripting vulnerability due to [["a user-provided value"|"relative:///bad/mod_user.py:16:20:16:26"]]. Cross-site scripting vulnerability due to [["a user-provided value"|"relative:///bad/libsession.py:5:12:5:19"]].	/bad/mod_user.py	33	16	33	23	Medium	This vulnerability can allow a person to cause code to be stored on server-side to execute limited instructions that can cause potential future issues	It is recommended to sanitize the 'username' variable using the command 'escape' available in the flask library
5)Server side request forgery	SSRF vulnerability because it uses untrusted input (HTTP request parameter) directly to construct a URL	/bad/api_list.py	10	9	10	75	High	This vulnerability will allow someone to potentially list out files, including the API key file	It is recommended to sanitize the 'username' variable using the command 'escape' available in the flask library
6)Flask app is run in debug mode	A Flask app appears to be run in debug mode. This may allow an attacker to run arbitrary code through the debugger.	/bad/vulpy.py	55	1	55	71	Critical	This vulnerability will allow for Remote Code Execution and therefore, the attacker can do anything they would like on the system	It is recommended to turn off debug mode by setting the 'debug' variable to 'False'
7)Empty except	except' clause does nothing but pass and there is no explanatory comment.	/bad/libsession.py	21	5	21	21	Medium	This can allow unexpected and unpredictable behavior	Except block should be used for error handling and should handle different errors in different ways
8)Unused import	Import of 'render_template' is not used. Import of 'redirect' is not used. Import of 'g' is not used. Import of 'session' is not used. Import of 'make_response' is not used. Import of 'flash' is not used.	/bad/mod_api.py	1	1	1	106	Low	While this causes readability issues it has no serious security implications	Remove the unused imports
9)Unreachable code	This statement is unreachable.	/bad/mod_mfa.py	54	5	54	45	Low	Unreachable code doesn't cause any unexpected behavior it simply clutters the code up	Simply remove the unreachable code or if the code should be reachable restructure the code so there is a path there ie change the else to an elif
10)Code injection	This code execution depends on [["a user-provided value"|"relative:///badguys/vulnerable/views.py:56:20:56:26"]].	/badguys/vulnerable/views.py	72	18	72	64	High	This line of code is excecuting a string provided by the other without santizing the input. This presents a high vulnerability as it is open to attacks like sql injection.	Sanitize the user input or remove the exec() line in total, I'm not sure what the point of that could be anyway
11)Code injection	This code execution depends on [["a user-provided value"|"relative:///badguys/vulnerable/views.py:56:20:56:26"]].	/badguys/vulnerable/views.py	76	22	76	52	High	This line of code is excecuting a string provided by the other without santizing the input. This presents a high vulnerability as it is open to attacks like sql injection.	sanitize the user input or remove the exec() line in total, I'm not sure what the point of that could be anyway
12)URL redirection from remote source	Untrusted URL redirection depends on [["a user-provided value"|"relative:///badguys/vulnerable/views.py:176:26:176:32"]].	/badguys/vulnerable/views.py	178	21	178	23	High	This is an incredibly dangerous line of code as the page is being redirected to a URL given by the user without validation. This can lead to phishing attacks.	User inputs should be sanitized before being put into a url used for redirection.
13)CSRF protection weakened or disabled	NA	/badguys/settings.py	102	5	102	49	High	This is another incredibly dangerous line of code as the CSRF token is absent which can lead to cross site request forgery.	CSRF tokens must be used to authenticate the user or website. This would eliminate the possibility of an unverified source to run this code unless they can somehow forge a CSRF token which is unlikely.