1a)

i) can use nmap to scan for TCP services:

nmap -sS 10.39.26.32/26 (as we only need to scan between 32-63 so 31 hosts, which are covered by the last 6 bits).

to cover hosts between this range, with -O to identify OS information.

2a)

i)

Tampering: Not sanitising inputs on comments could lead to e.g. SQL injections to affect the db to modify credentials or other information. (Mitigation: sanitise inputs).

Information disclosure: If a user can query for a status update, an attacker could provide a malicious parameter with a search that displays information about the system. (Mitigation: sanitise parameters)

Spoofing: An attacker may either provide a malicious JS library that could extract credentials of users when a user logs in and send them to the attacker so the attacker could log in as that user. Alternatively, an existing third party library could be exploited to extract or show credentials.(Mitigation: use https/apply same origin policy).

Spoofing: discovering another user’s password and logging in as that user: forcing harder passwords to mitigate it

ii) Stored XSS: malicious input is saved to db and when retrieved causes information to be displayed. Deployed through user input e.g. comments where the string is sanitised to not initially cause harm, but may cause information to be extracted later.

Resident XSS: user inputs malicious code to exploit browser storage to get or set items in storage. Deployed by attacker inputting malicious JS that is not sanitised.

Self XSS: user inputs malicious code into browser that executes from their browser and reveals info to the attacker. Deployed by advertising some code which claims to benefit the user, leading to them executing the code (following instructions given by advertisement).

3a) i) Passive intelligence gathering: gather information without directly using target (db). Build DFD of connected services, network model, etc.

Active intelligence gathering: query db through altering URLs or using presented UI. Test with various parameters to identify any vulnerabilities. Query any services linked to the db to see if they can be exploited.

Exploiting: using any exploits revealed to query DB, perform SQL injections etc. to the extent specified in the specification (so e.g. not dropping tables if not permitted to).

ii) Online: query with many different dictionary words for random users until a correct username/password combination is identified. Countermeasure: limit num attempts per user/IP before blocking, or enforce more complex passwords so they are harder to guess, or use CAPTCHA to avoid automating attempts, or 2FA/MFA?

Offline: attempt to guess hash of password (if hash is known) or decrypt password. Check against various dictionary words until a password is identified; can also warp words as in John the Ripper to test variations of dictionary words. Countermeasure: as before, make passwords more complex so they are harder to guess, and if not introduced already add salt to the hashes so a rainbow table cannot be constructed from the hashes provided to work out passwords common to all users (so each password would need to be checked with each word separately even if the salt was known) + use OAuth?

4a) i) Phishing: website pretending to be another website (either by copying HTML or mimicking URL) to try to convince user to submit credentials on it that an attacker can then use. If a user logs in the attacker will get the credentials, and the legitimate website would receive the request which it either replies to with an error message or acknowledges.

Click-jacking: website that hides a transparent frame of another website so when the user performs an action e.g. clicking on a button, they interact with the other website. Attacker could use this so the user tampers with their own account e.g. deletes it, and user would be confused by the attacker’s overlay that hides the other website.

Drive by download: Code that exploits some vulnerability to download and install malware (e.g. from a plugin). Through the malware the attacker has access to the user’s device + information. Could utilise device as part of a botnet or run scripts from the user’s device/browser. User may be confused by not seeing any scripts run or any noticeable effect from the website.

ii) A malware author either wants to stop the service running or ensure that its own URLs are not correctly identified as hosting malware.

If inputs are not sanitised, could use SQL injection to affect results by updating rows for their URLs to register as not holding malicious content to fool users that query the website. (Mitigation: sanitise inputs)

Could also prevent the service from functioning through using a botnet to perform a Denial of Service attack by making many requests from the website. (Mitigation: allocate more resources to website or limit requests per user, + some queuing framework to handle subsequent requests)

Could perform SQL injection to gain access to credentials (or similarly intercept requests made if using only HTTP instead of HTTPS, which would mitigate this) to be able to log in and remove any data about their URLs or change them as before. (Mitigation: sanitise inputs).

Remote file inclusion: including a link to a malicious script within the URL?