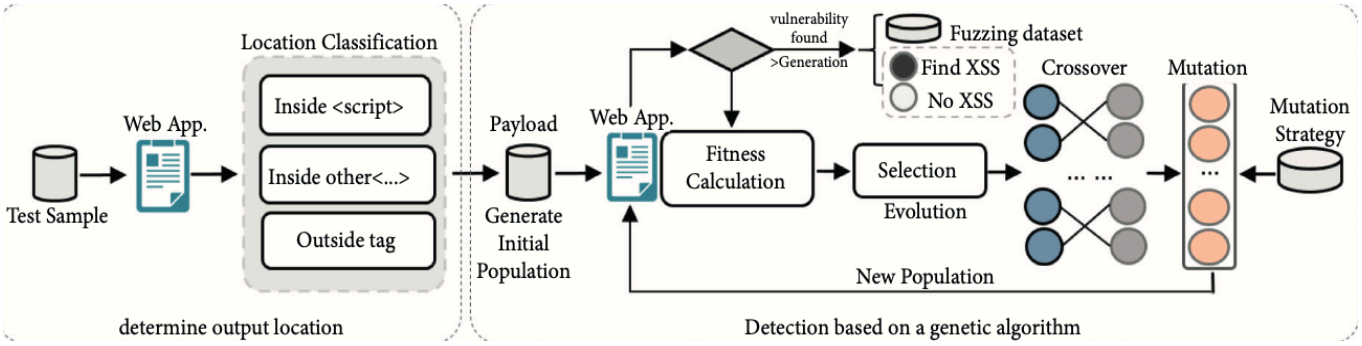
Roadmap For Genetic Algorithm Approach

1. **Define Objective and Approach**
    - Objective: Use genetic algorithms to automate vulnerability detection in web applications.
    - Approach: Encode payloads into DNA-like structures, evolve them using genetic operations, and evaluate fitness based on execution success, closure completeness, and input-output similarity.

2. **System Architecture Design**
    - Create a clear system diagram (Figure 2) outlining:
        - Data flow: From initial population generation through response analysis to vulnerability detection.
        - Components: Input submission, response analysis, and vulnerability detection criteria.



3. **Individual Encoding**
    - Design DNA structure:
        - Closing part (C), main part (B), mutation part (M).
        - Choose encoding type (e.g., binary) for XSS payloads.

4. **Fitness Function Design**
    - Define fitness components:
        - Execution success (Ex(I, O)): Evaluate successful XSS execution.
        - Closure completeness (CLOSED(I, O)): Measure HTML closure in response.
        - Input-output similarity (Dis(I, O)): Calculate similarity using Levenshtein distance.
        - Filter handling (Pu(I, O)): Penalise for filtered JavaScript or incomplete closures.

5. **Genetic Algorithm Operations**
    - Operations:
        - Selection: Probabilistic selection based on fitness.
        - Crossover: Combine DNA segments of selected individuals to create offspring.
        - Mutation: Introduce variations in offspring DNA to explore new vulnerabilities.
        - Termination: End after a defined number of generations or upon successful vulnerability detection.

6. **Implementation Steps**
    - Initialize: Set initial population (P0) with encoded XSS payloads.
    - Iterate: Perform selection, crossover, mutation, and fitness evaluation iteratively.
    - Terminate: Stop after reaching a set number of generations or achieving vulnerability detection goals.

7. **Testing and Validation**
    - Validate using test datasets and scenarios.
    - Evaluate effectiveness in detecting XSS vulnerabilities.
    - Analyse results to refine algorithm parameters and strategies.

8. **Conclusion and Future Work**
    - Summarise findings on algorithm effectiveness.
    - Discuss limitations and future improvements (e.g., handling complex DOM structures, enhancing mutation strategies).

9. **Documentation and Reporting**
    - Document process, architecture, algorithm details, and experimental results.
    - Prepare a comprehensive report for stakeholders, including researchers and cybersecurity experts.

10. **Integration and Deployment**
    - Integrate algorithms into existing vulnerability assessment tools or frameworks.
    - Plan deployment strategies for use in real-world web application security testing.