Encouraging Secure Programming Practice in Academia

SCOTT KURODA
ADVISOR: DR. FRANZ KURFESS

Goals

- Develop a system that exposes students to:
  - Secure programming practice
  - Attack scenarios
  - Vulnerable code
- Develop system in a service oriented manner.
  - Accessible via Internet

Current Tools

- Static Code Analysis
- Sandbox Environments
- Courses

Static Code Analysis

- Lint
- PC-Lint
- JS-Lint
- Pylint
- Pychecker

Sandbox Environment

- Provide a controlled space for experiments.
  - Penetration test
- Allow “safe” environment for safe competitions
  - Defcon
  - International Capture the Flag Hacking Competition (UCSB)
    - Traditional CTF
    - “Treasure Hunt”
    - “Botnet” Scenario
    - Simulated attack against a rogue nation

Academics

- Courses
  - Theory and concepts of security
    - Encryption
    - Program Security
    - Network Security
  - Implementation of attacks
    - Buffer Overflow
    - Breaking encryption
    - Graceful failure
    - SQL Injection
- Clubs
  - White Hat

Current Research

- Teaching computer security
  - Course design
- Automated tools in academics
  - Checking for plagiarism
- In industry
  - Penetration testing
  - Automated software testing

Research in Academics

- Course design
  - Not practical to create an additional required course for many universities.
- Code analysis
  - Utilized by many institutions to reduce plagiarism.
    - Textual analysis
    - Structural analysis
    - Variable analysis

Research in Industry

- Threat Model Driven Approach for Security Testing
- Automated Software Testing as a Service

Threat Model Driven Approach for Security Testing

- Threats modeled as an UML
- Scenarios developed as sequence diagrams at design phase
- Determine security policy, then define model behavior that would violate said policy.

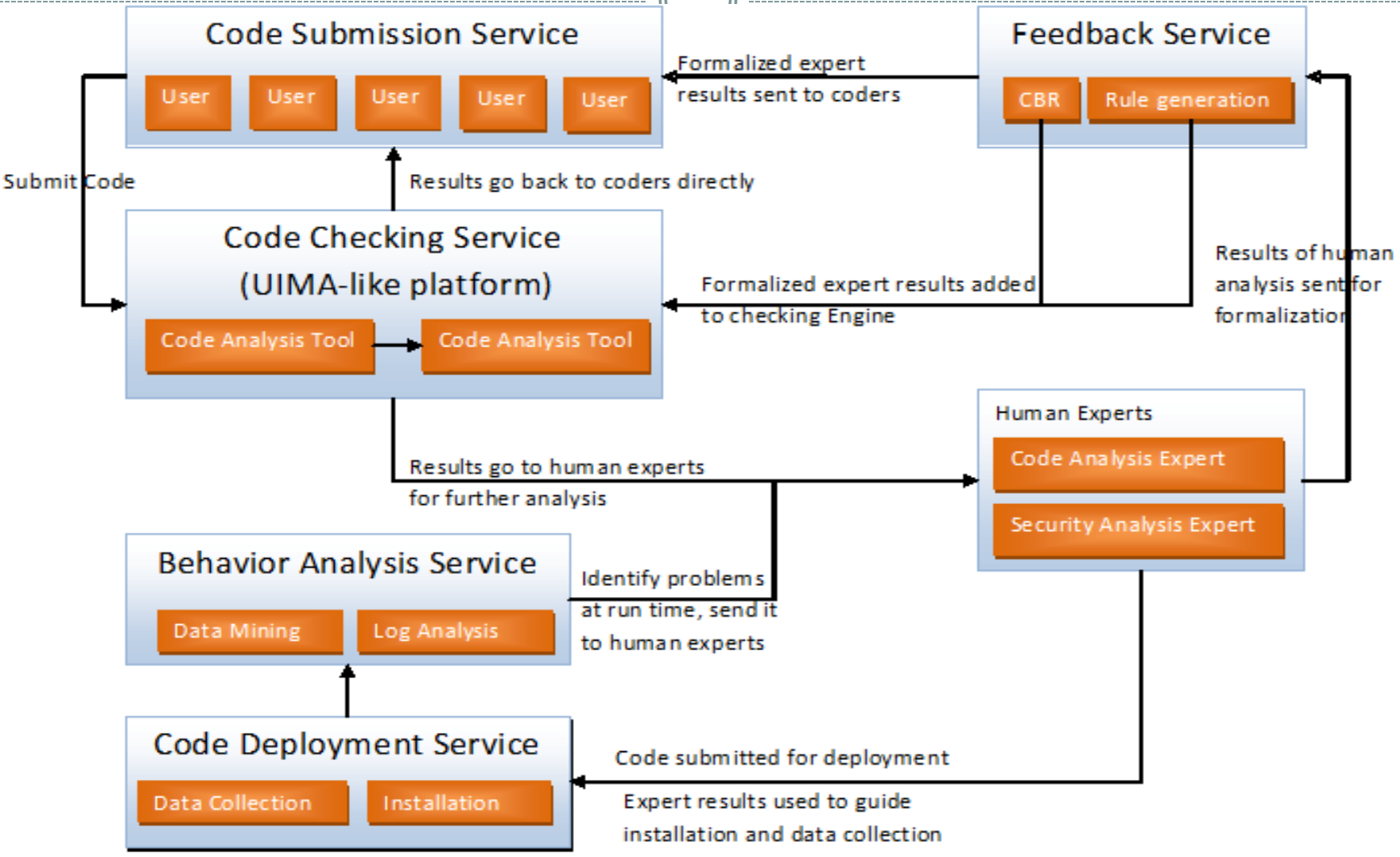
Automated Software Testing as a Service

- Leverage cloud services to test code.
- Reduce the load on a given system.
- Provide continuous testing of code to developers.
- Developers can define both high level specifications and lower level test predicates.
- Predicates broken into two categories, universal and application specific.

So what?

- Goals reiterated:
  - To expose students to computer security issues.
- Close the knowledge gap for student developers
  - Students will be exposed to security issues, at a minimum, through submitting and receiving feedback on their code.
  - Students may choose to extend their knowledge by becoming “experts” in the system.

Proposed Architecture



From proposal by Dr. Seng, Dr. Kurfess, Dr. Nico, and Dr. Assal

Code Checking

- Perform static code analysis
- Generate annotated report for both user and experts
- Intermediate agent to potentially combine reports
  - Shorten final report
  - Reduce redundancy of a given error
  - Several challenges
    - Reports from each tool may appear differently.
    - Text parsing and language processing to accurately create final report

Human Expert

- Second level of analysis.
- Use levels to define how much of an “expert” in the field of computer/network security.
  - E.g. Students providing feedback vs. Industry Expert

Code Deployment

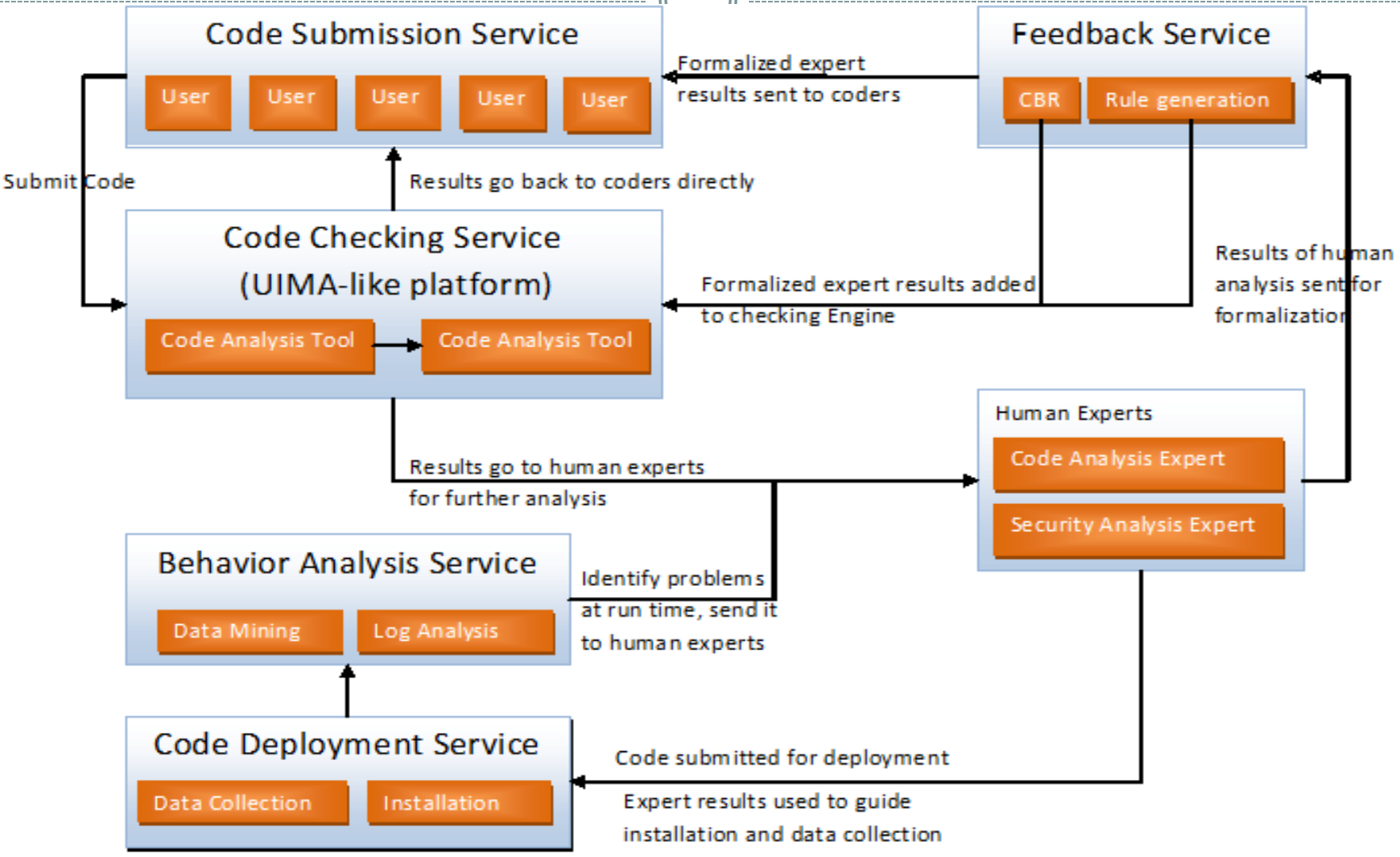
- Actual running of submitted code.
- Collect various metrics about deployed code.
- Potentially utilize non-static code analysis methods
- Requires building a safe closed environment to run code.
  - Must be isolated from external influences.
  - Must be restricted if malicious code is submitted.

Behavior Analysis

- Analysis of code behavior
- Various analysis methods performed on data generated from code deployment.

Questions?

Proposed Architecture



From proposal by Dr. Seng, Dr. Kurfess, Dr. Nico, and Dr. Assal