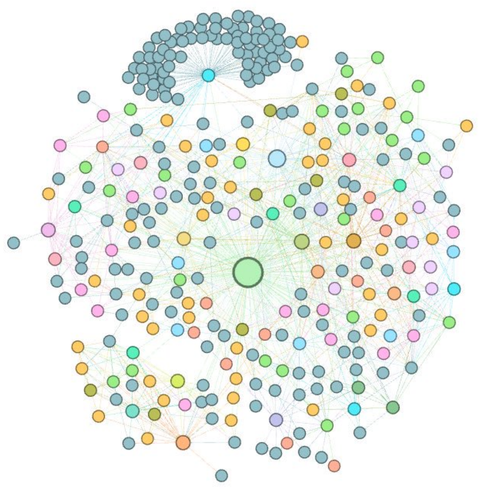
The Dynamics of Software Composition Analysis

Darius Foo, Jason Yeo, Xiao Hao, Asankhaya Sharma
Veracode

Motivation

- Software Composition Analysis (SCA): automated dependency management in CI/CD
- Key tasks: discovering dependencies, checking for reachability of exploitable code, remediation

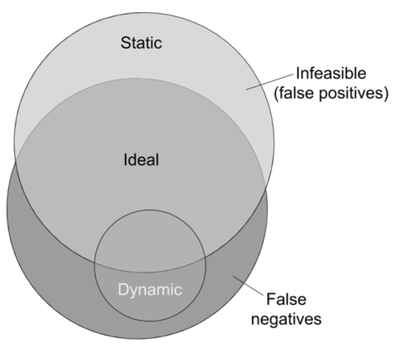


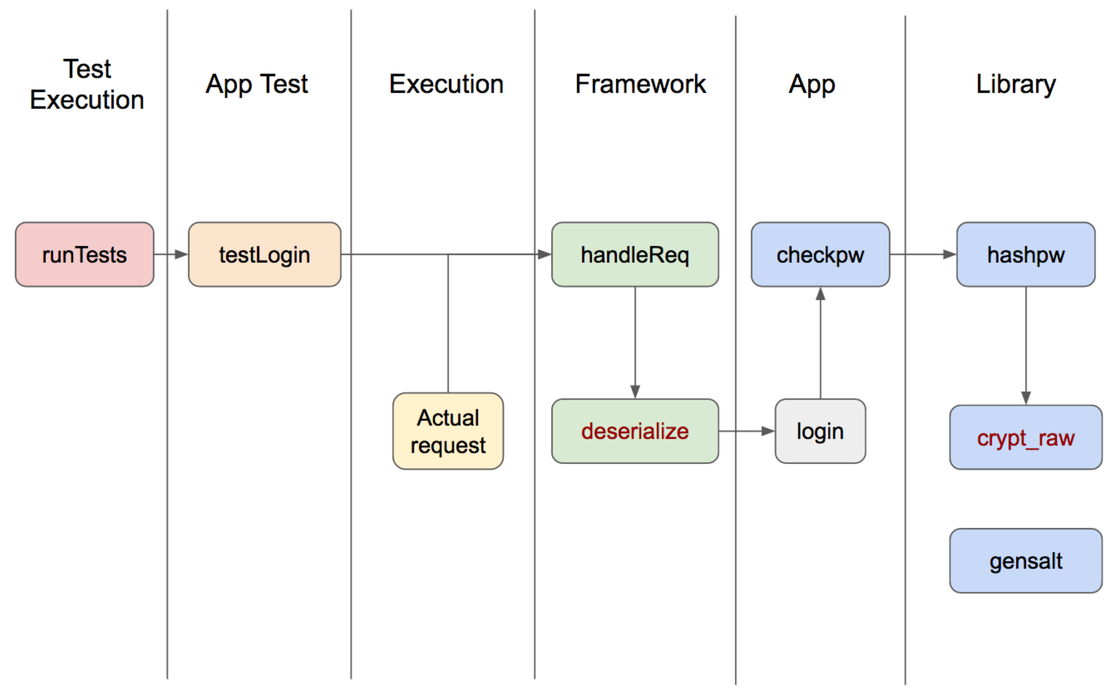
```
@PostMethod("/login")
void login(Credentials c) {
    BCrypt.checkpw(c.getPassword());
}


@Test
void testLogin() {
    Controller c = setupController();
    c.post("/login",
      "{\"password\": \"abc\"}");
}
```

```
class BCrypt {

    public void checkpw(String password) {
        hashpw(password);
    }

    void hashpw(String password) {
        crypt_raw(password);
    }

    void crypt_raw(String password) {
        // ...
    }

    public void gensalt() {
        // ...
    }
}
```

Dependency Analysis

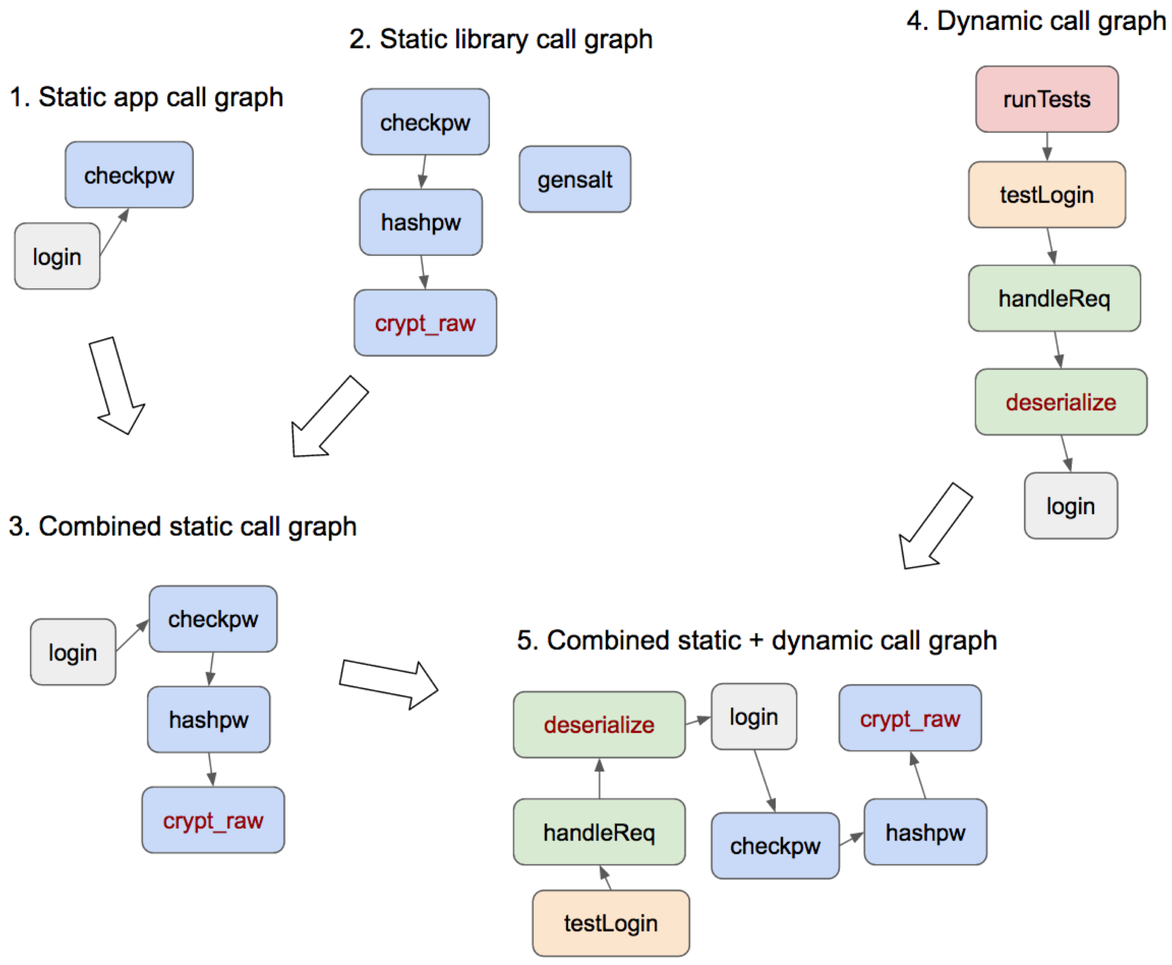
- Determining third-party dependencies used by a project
- Dependency resolution is unspecified, difficult
- Static analysis: lockfiles
- Dynamic analysis: integrating with and querying package manager; 204% more dependencies





Reachability of Exploitable Code

- Annotate vulnerability-specific sinks (“vulnerable methods”) by hand
- Check static call graph for reachability of sinks
- Problems: infeasible edges (false positives), soundy (false negatives)



Dynamic Analysis

- Instrument programs
- Execute test cases
- Check combined graph for reachable vulnerable methods

Combining Graphs

- Framework calls
- Static analysis false negatives (reflection, dynamic dispatch)
- Overhead, manual setup

Evaluation

Project	Static vertices	Static edges	Dynamic vertices	Dynamic edges	Static sinks	Dynamic sinks
helios	7930	26746	12287	39813	1	3616
immutables	30206	319934	398	874	5	5
java-apns	536	999	4240	8685	58	859
retrofit	1925	5269	7339	22565	6	6