# Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey

Seyed Mohammad Ghaffarian and Hamid Reza Shahriari

# Software Vulnerability

- A software vulnerability is an instance of a flaw, caused by a mistake in the design, development, or configuration of software such that it can be exploited to violate some explicit or implicit security policy.
- error, fault, failure, mistake

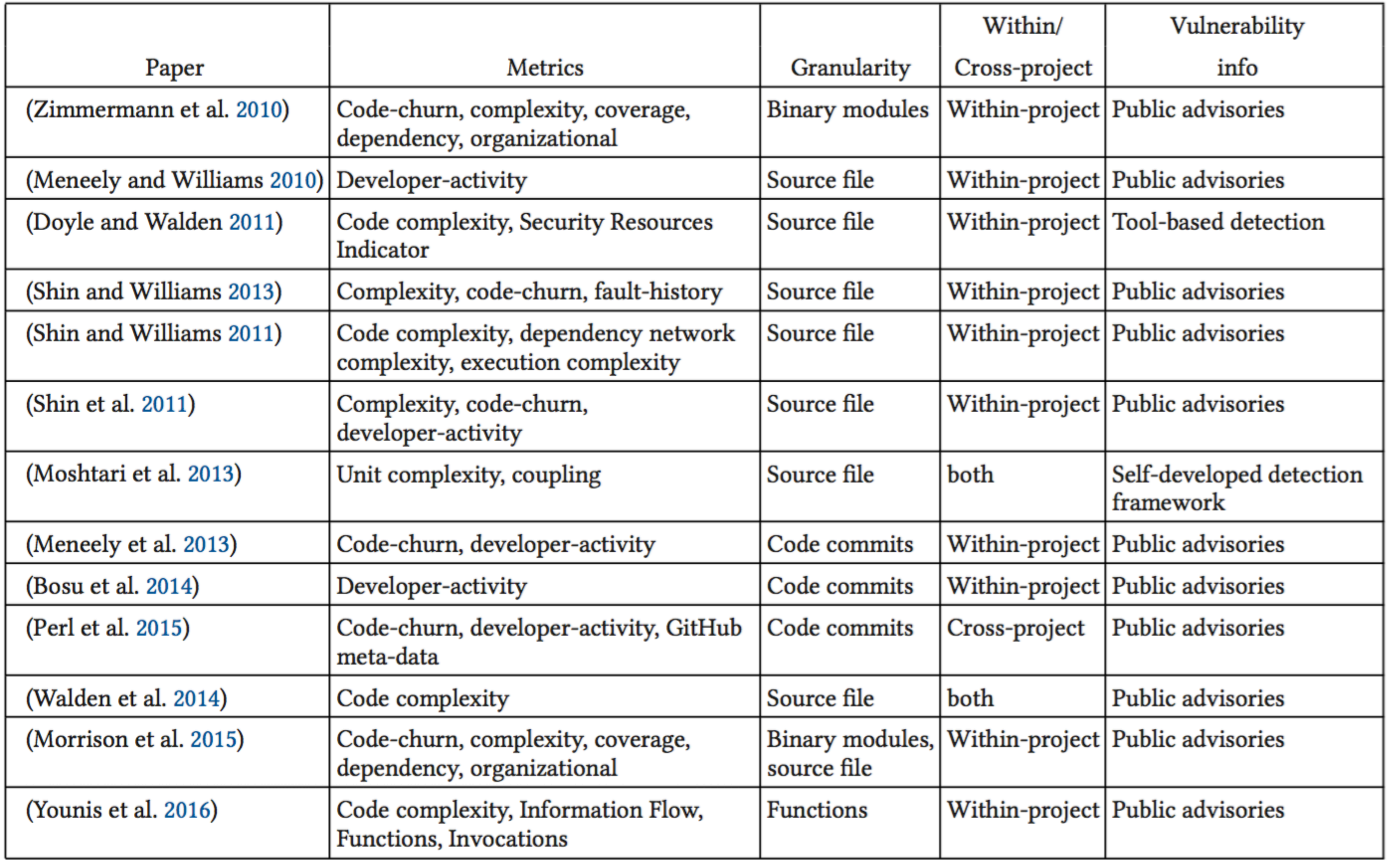
# Conventional Approaches

- Static Analysis
- Dynamic Analysis
- Hybrid Analysis
- Software Penetration Testing
- Fuzz-Testing
- Static Data-Flow Analysis

# Machine-learning and Data-mining techniques

- Vulnerability Prediction Models based on Software Metrics
- Anomaly Detection Approaches
- Vulnerable Code Pattern Recognition
- Miscellaneous Approaches

# Vulnerability Prediction Models based on Software Metrics

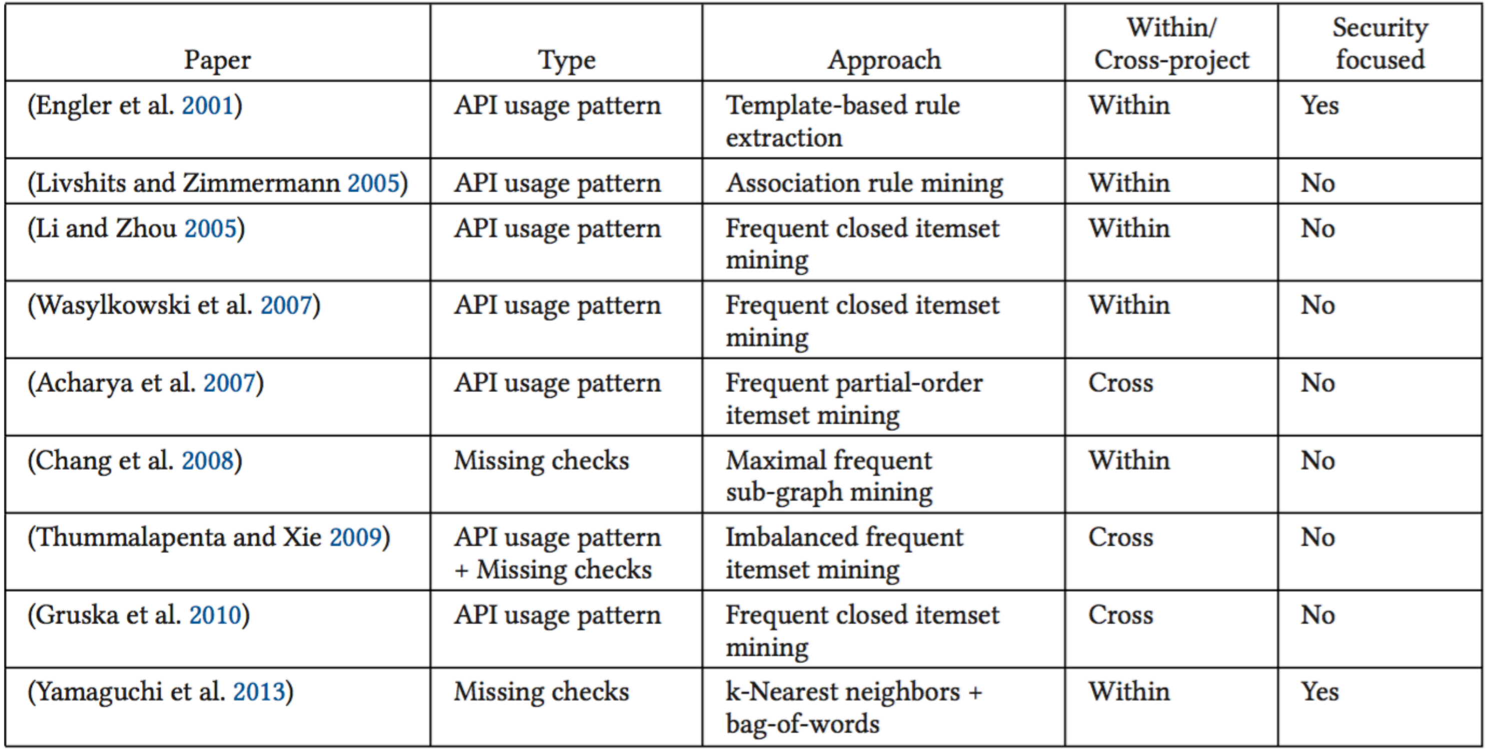


| Paper | Metrics | Granularity | Within/<br>Cross-project | Vulnerability<br>info |
|---|---|---|---|---|
| (Zimmermann et al. 2010) | Code-churn, complexity, coverage, dependency, organizational | Binary modules | Within-project | Public advisories |
| (Meneely and Williams 2010) | Developer-activity | Source file | Within-project | Public advisories |
| (Doyle and Walden 2011) | Code complexity, Security Resources Indicator | Source file | Within-project | Tool-based detection |
| (Shin and Williams 2013) | Complexity, code-churn, fault-history | Source file | Within-project | Public advisories |
| (Shin and Williams 2011) | Code complexity, dependency network complexity, execution complexity | Source file | Within-project | Public advisories |
| (Shin et al. 2011) | Complexity, code-churn, developer-activity | Source file | Within-project | Public advisories |
| (Moshtari et al. 2013) | Unit complexity, coupling | Source file | both | Self-developed detection framework |
| (Meneely et al. 2013) | Code-churn, developer-activity | Code commits | Within-project | Public advisories |
| (Bosu et al. 2014) | Developer-activity | Code commits | Within-project | Public advisories |
| (Perl et al. 2015) | Code-churn, developer-activity, GitHub meta-data | Code commits | Cross-project | Public advisories |
| (Walden et al. 2014) | Code complexity | Source file | both | Public advisories |
| (Morrison et al. 2015) | Code-churn, complexity, coverage, dependency, organizational | Binary modules, source file | Within-project | Public advisories |
| (Younis et al. 2016) | Code complexity, Information Flow, Functions, Invocations | Functions | Within-project | Public advisories |

# Discussion

- Vulnerabilities are few and sparse in the datasets(imbalance class data)
- Cross-project studies are few(transfer learning)
- Poor results(defining security-specific metrics)
- Deep learning

# Anomaly Detection Approaches

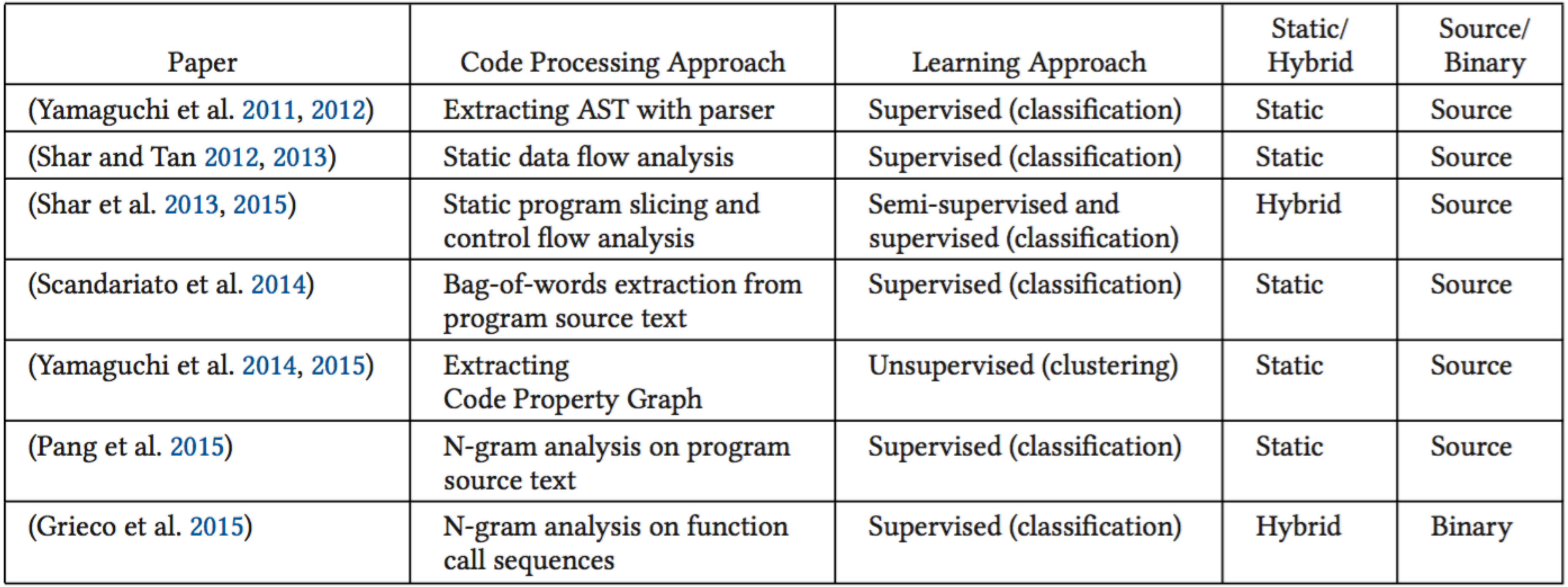


| Paper | Type | Approach | Within/ Cross-project | Security focused |
|---|---|---|---|---|
| (Engler et al. 2001) | API usage pattern | Template-based rule extraction | Within | Yes |
| (Livshits and Zimmermann 2005) | API usage pattern | Association rule mining | Within | No |
| (Li and Zhou 2005) | API usage pattern | Frequent closed itemset mining | Within | No |
| (Wasylkowski et al. 2007) | API usage pattern | Frequent closed itemset mining | Within | No |
| (Acharya et al. 2007) | API usage pattern | Frequent partial-order itemset mining | Cross | No |
| (Chang et al. 2008) | Missing checks | Maximal frequent sub-graph mining | Within | No |
| (Thummalapenta and Xie 2009) | API usage pattern + Missing checks | Imbalanced frequent itemset mining | Cross | No |
| (Gruska et al. 2010) | API usage pattern | Frequent closed itemset mining | Cross | No |
| (Yamaguchi et al. 2013) | Missing checks | k-Nearest neighbors + bag-of-words | Within | Yes |

# Discussion

- Anomaly detection approaches are only effectively applicable for mature software systems
- Anomaly detection approaches are unable to specify the type of defect or vulnerability
- False positive rate is high
- Graph based detection

# Vulnerable code pattern recognition

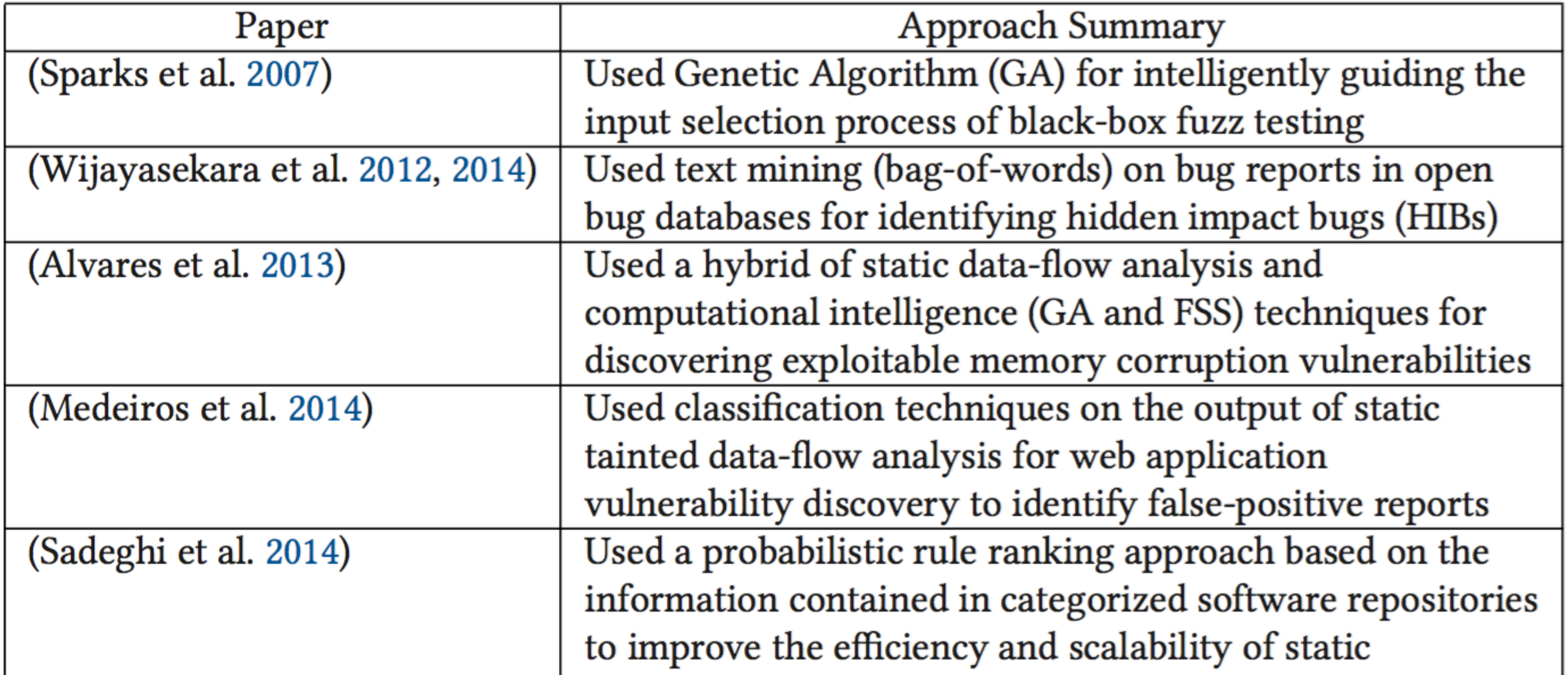


| Paper | Code Processing Approach | Learning Approach | Static/ Hybrid | Source/ Binary |
|---|---|---|---|---|
| (Yamaguchi et al. 2011, 2012) | Extracting AST with parser | Supervised (classification) | Static | Source |
| (Shar and Tan 2012, 2013) | Static data flow analysis | Supervised (classification) | Static | Source |
| (Shar et al. 2013, 2015) | Static program slicing and control flow analysis | Semi-supervised and supervised (classification) | Hybrid | Source |
| (Scandariato et al. 2014) | Bag-of-words extraction from program source text | Supervised (classification) | Static | Source |
| (Yamaguchi et al. 2014, 2015) | Extracting<br>Code Property Graph | Unsupervised (clustering) | Static | Source |
| (Pang et al. 2015) | N-gram analysis on program source text | Supervised (classification) | Static | Source |
| (Grieco et al. 2015) | N-gram analysis on function call sequences | Supervised (classification) | Hybrid | Binary |

# Discussion

- Specific type of vulnerability is not determined
- Define rich feature extraction and description techniques to achieve higher precision and recall
- Use deep learning methods to build powerful cross-project vulnerable code pattern recognition systems

# Miscellaneous Approaches



| Paper | Approach Summary |
|---|---|
| (Sparks et al. 2007) | Used Genetic Algorithm (GA) for intelligently guiding the input selection process of black-box fuzz testing |
| (Wijayasekara et al. 2012, 2014) | Used text mining (bag-of-words) on bug reports in open bug databases for identifying hidden impact bugs (HIBs) |
| (Alvares et al. 2013) | Used a hybrid of static data-flow analysis and computational intelligence (GA and FSS) techniques for discovering exploitable memory corruption vulnerabilities |
| (Medeiros et al. 2014) | Used classification techniques on the output of static tainted data-flow analysis for web application vulnerability discovery to identify false-positive reports |
| (Sadeghi et al. 2014) | Used a probabilistic rule ranking approach based on the information contained in categorized software repositories to improve the efficiency and scalability of static |

# Fruitful research areas

- Engineering rich features with high discriminative and expressive power for various types of software vulnerabilities
- Designing new machine-learning and data-mining algorithms, tailored to the characteristics of the problem of software vulnerability analysis and discovery