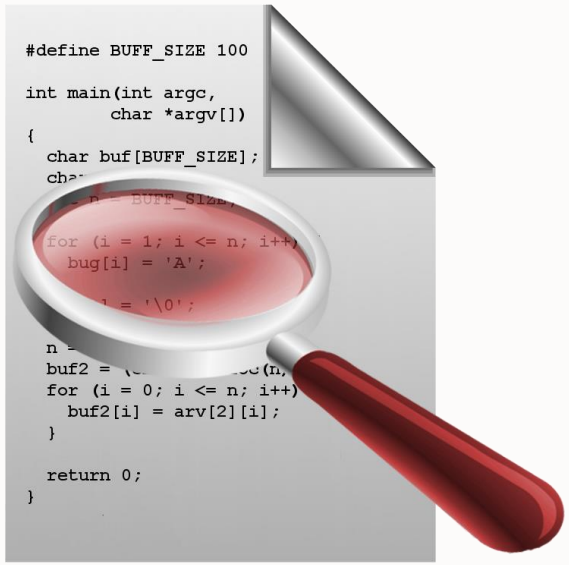
ORACLE

Security Research at Oracle Labs, Australia

Program Analysis Meets Security

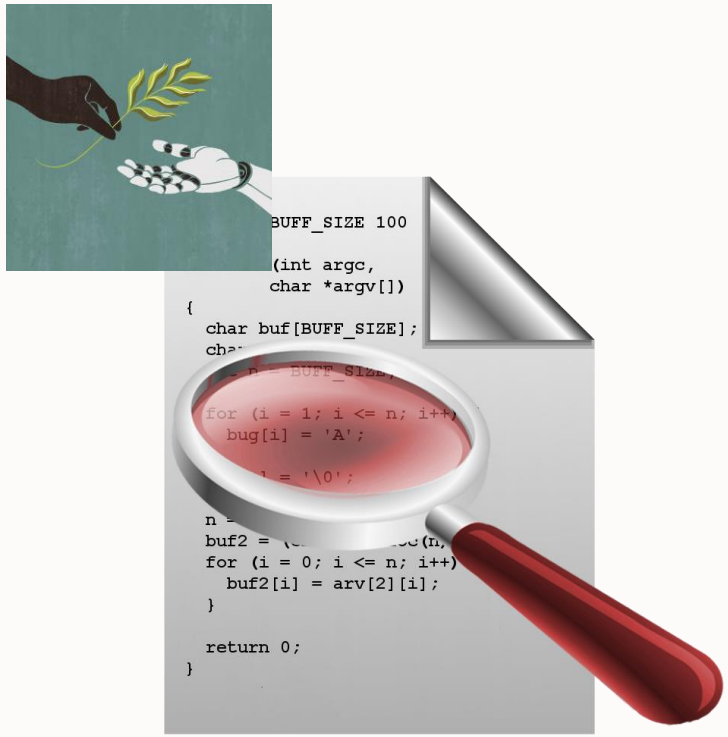
Paddy Krishnan
Research Director
Oracle Labs, Brisbane, Australia
2021



We focus on automatically detecting security vulnerabilities and preventing security attacks by analysing programs.

Oracle Labs Australia

Established 2010



“To make *intelligent application security*, at scale, a reality”

Oracle Labs Australia vision

General Principles

Information and Software Security

- Security breaches aim to steal data: Other's data is valuable
- Cloud provider's responsibility to protect customer's data
  - Oracle has numerous cloud offerings

- Information needs protection
- Software security mechanisms protect information
- Assurance shows how can one rely on software for security

Exploit should not be the first security feedback

Sample Attacks

Remote code execution through .Net deserialization attack, June 2020

Sample Attacks

Australian governmen
routinely audited for c
Most fail

By Michael Slezak and Ariel Bogle

Posted Sat 20 Jun 2020 at 5:08am, updated Mon 22 Jun 2

Trust factor: Professor Dali Kaafar (pictured) says people expect links to be safe and information well-protected on government websites.

The audit found that more than 70 per cent of state/territory governments' webpages and 57 per cent of federal government webpages had at least one JavaScript library with publicly

10% Australian government websites use outdated code that's vulnerable to XSS, October 2020

Remote code exec
deserialization attack

Sample Attacks

Data breach through SQL injection attack, February 2021

Sample Attacks

Data breach through SQL inj
February 2021

Supply chain attack introduces malware,
March 2021

Oracle was not affected by this attack
to XSS

Sample Attacks

High-Level View

- Insecure information flow

- Assurance
- Compliance
- Security Standards

- Stop insecure flow
  - Network: Firewalls
  - Infrastructure: Scanners
  - ✓ Applications: Check have correct behaviour

Agenda

- Information vs software security
- Software development life cycle and vulnerability detection
- Program analysis techniques
- Strengths and open challenges

Information Security

- Information received and released have the right properties

- **Integrity**: Information received not malicious
  - Attacker may send malicious data but it is sanitised before use
- **Confidentiality**: Receiver authorised to view data
  - Data leaks violate confidentiality, needs declassification
- **Non-repudiation**: Neither sender nor receiver can deny their actions

- Other properties like **Availability** not considered here
- Need **policies** to specify permitted and forbidden information flows

Software Security

- Check that software does not have security vulnerabilities
  - Link software behaviour to information security
  - Detect potential security violations: E.g., OWASP Top 10
  - Information flow policies converted to program properties

- Example: Can confidential information leak via SQL injection?
  - Need to consider threat environment
  - What is exposed to the attacker and how can it be used

- Ideal: Mitigate all risks - **Defence in depth**

Examples: Linking Information and Software Security

- Access control: Have right credentials
  - Authentication
  - Authorisation
- Boundary protection mechanisms
  - Firewalls
- Labelling information and protection domains
- Flow control
  - Encrypted tunnels

Policy and Configurations: Integrity

- Any user-controllable data is potentially malicious and hence tainted
  - Entry points; **taint sources** (can lead to integrity violations)

- Tainted data flows through the program via data flows in the application
  - **Sanitisation**/validation routines make tainted value untainted

- Security-sensitive statements are considered **taint sinks**
  - If a tainted value reaches a taint sink, a vulnerability is exposed

Policy and Configurations: Confidentiality

- Data that is security-sensitive should not reach an exit point unless that data is made available to the same security level
  - **Sources**: points where security sensitive values generated: identify credentials required for access
  - **Sinks**: points where information leaks (can lead to confidentiality violations)
  - **Declassifiers**: confidential information redacted

General Approach

Software Development Process, Operating Environment

Attack Surface

The attack surface of a software environment is the sum of the different points (the "**attack vectors**") where an unauthorized user (the "**attacker**") can try to *enter data* to or *extract data* from an environment.

OWASP

Attack Surface: Inputs and Resources

- User interface (UI) forms and fields
- HTTP headers and cookies
- APIs
- Files
- Databases and other local storage
- Email or other kinds of messages
- Run-time arguments provided by user

Attack Surface: Mapped to Code

- Login/authentication entry points
- Admin interfaces
- Inquiries and search functions
- Data entry (CRUD) forms
- Business workflows
- Transactional interfaces/APIs
- Operational command and monitoring interfaces/APIs
- Interfaces with other applications/systems

Application Security Testing (AST) Definitions: Gartner

SAST

- Static testing tools analyse the code (source code, bytecode, or binary code) for security vulnerabilities
- May or may not compile the code
- White-box testing

IAST

- Interactive testing tools combine elements of SAST and DAST simultaneously
- Typically implemented as an agent within the test runtime environment
- Uses test cases
- Can test whether known vulnerabilities are actually exploitable in the running app

DAST

- Dynamic testing tools run at runtime, while the application is running
- It simulates attacks against an application, analyses the app's reactions, and determines whether it's vulnerable
- Black-box testing

Application Security Testing (AST) Definitions: Gartner

SCA

- Software composition analysis
- Examines software to determine origin of components and libraries
- Aim is to identify vulnerabilities in commonly used components (not in in-house developed components)

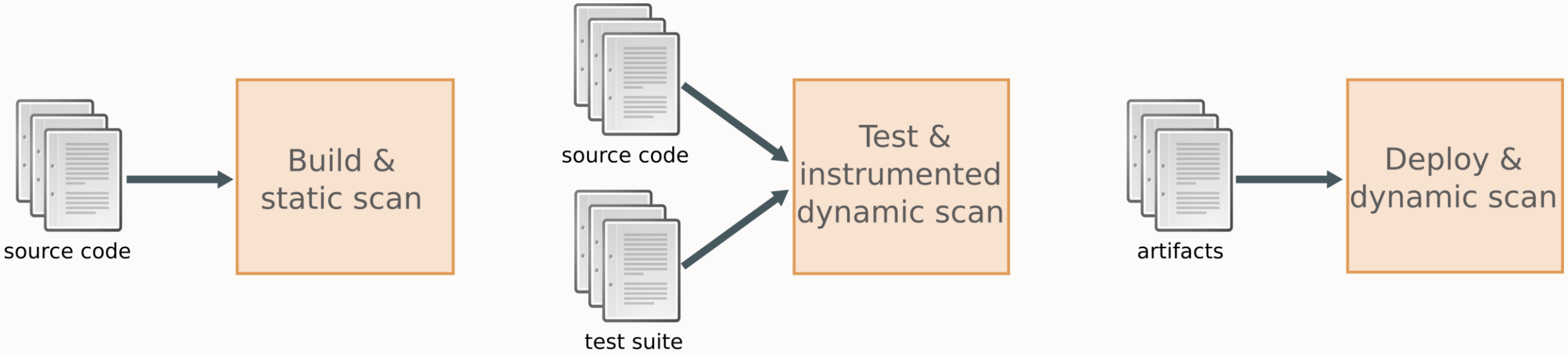
RASP

- Runtime application self-protection
- Blocks computer attacks through runtime instrumentation
- Monitors inputs and reports (monitor mode) or blocks (protection mode) attacks

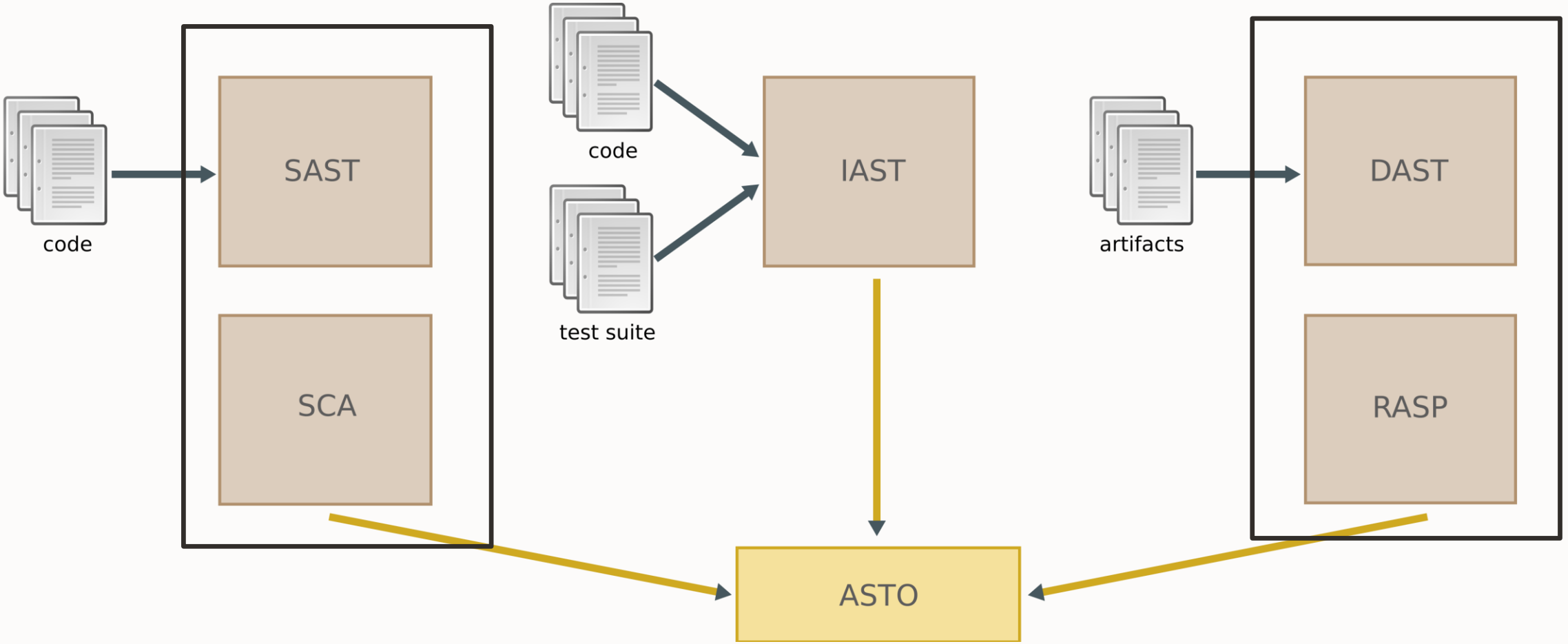
ASTO

- Application Security Testing Orchestration
- Central management and reporting of all the different AST tools running in an ecosystem

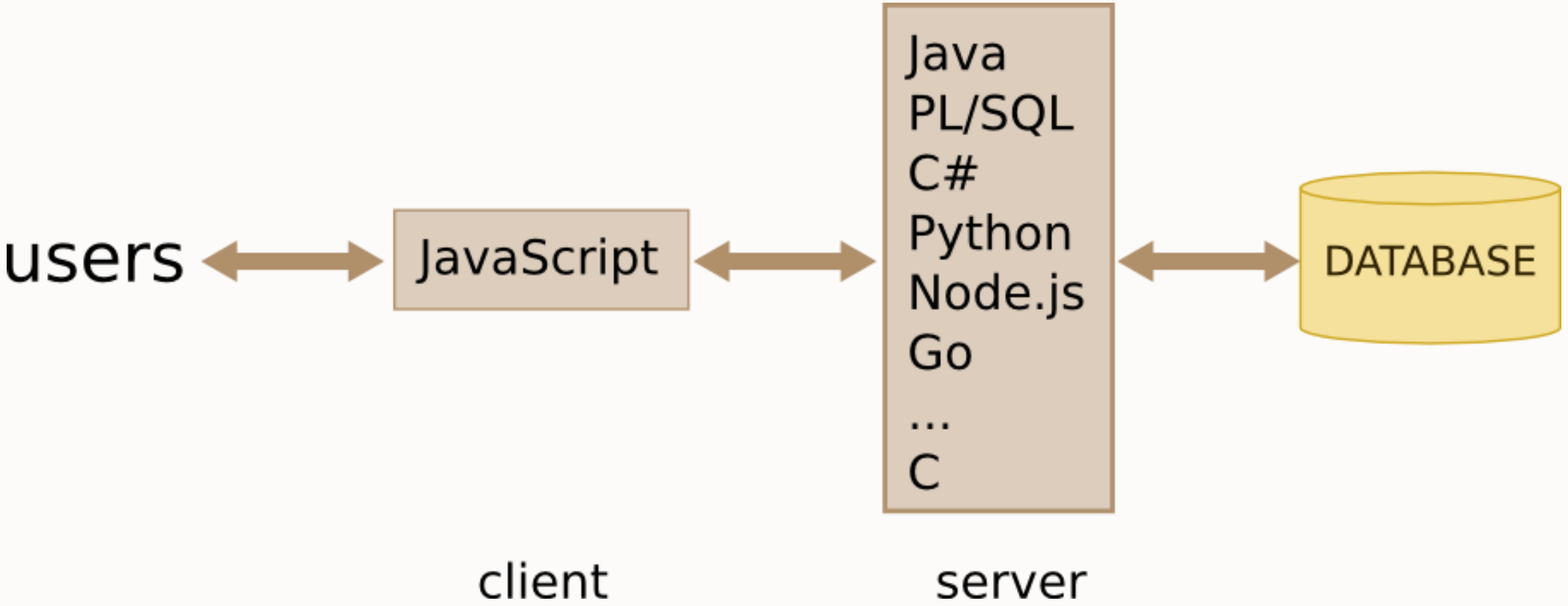
Building Security Checking Into the Lifecycle



The Application Security Testing Suite



Detecting Vulnerabilities in Web Applications



Strategies

- Development
  - Unit testing, CI/CD, *Static Analysis*
- Complete System
  - End-to-end testing, *Dynamic Analysis*
- Trial deployment
  - Test with different configurations, *Dynamic Analysis*

Specific Projects

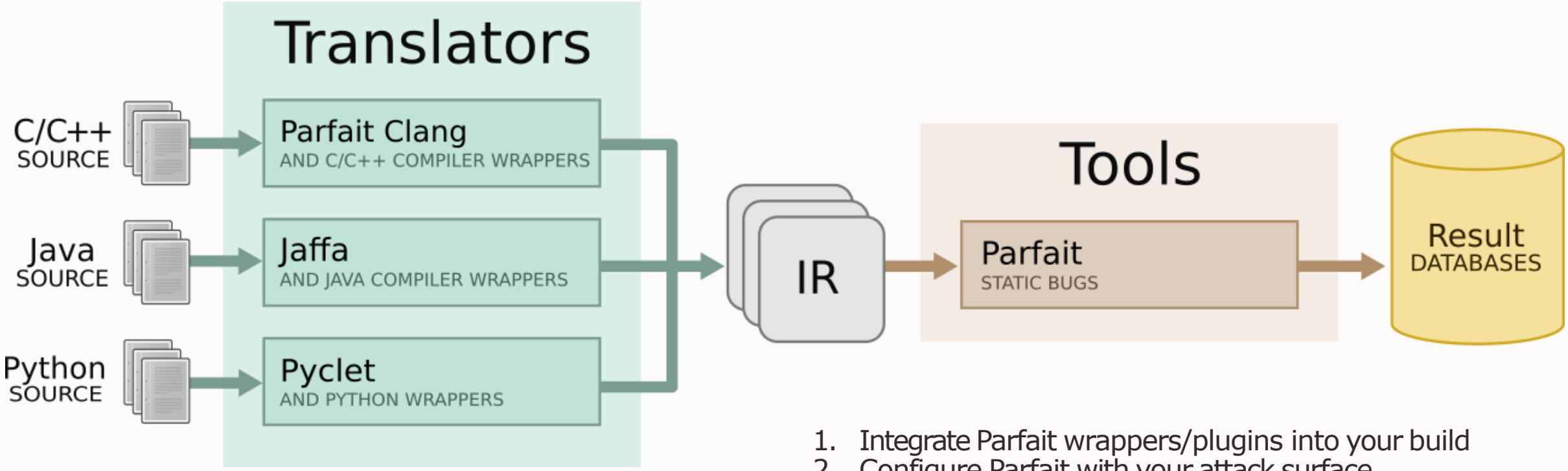
Projects

- Static Analysis
  - Source code
- Dynamic Analysis
  - Client-side
  - Server-side
- Other Approaches
  - Malware Detection including Machine Learning
  - Synthesis for RASP
  - SCA

Parfait: Static Analysis

- Key Features
  - High precision: 90% true positives
  - Scalable: 10 mins per MLOC on a 2.6 GHz Intel Xeon E5-2690
- Vulnerabilities that matter
  - SQL injection, cross-site scripting (XSS), LDAP injection, OS injection
  - XXE/XEE
  - Using weak crypto
  - Insecure deserialization
  - Buffer overflows
  - Dereference of untrusted pointer
  - Trusted boundary vulnerabilities

Parfait Architecture



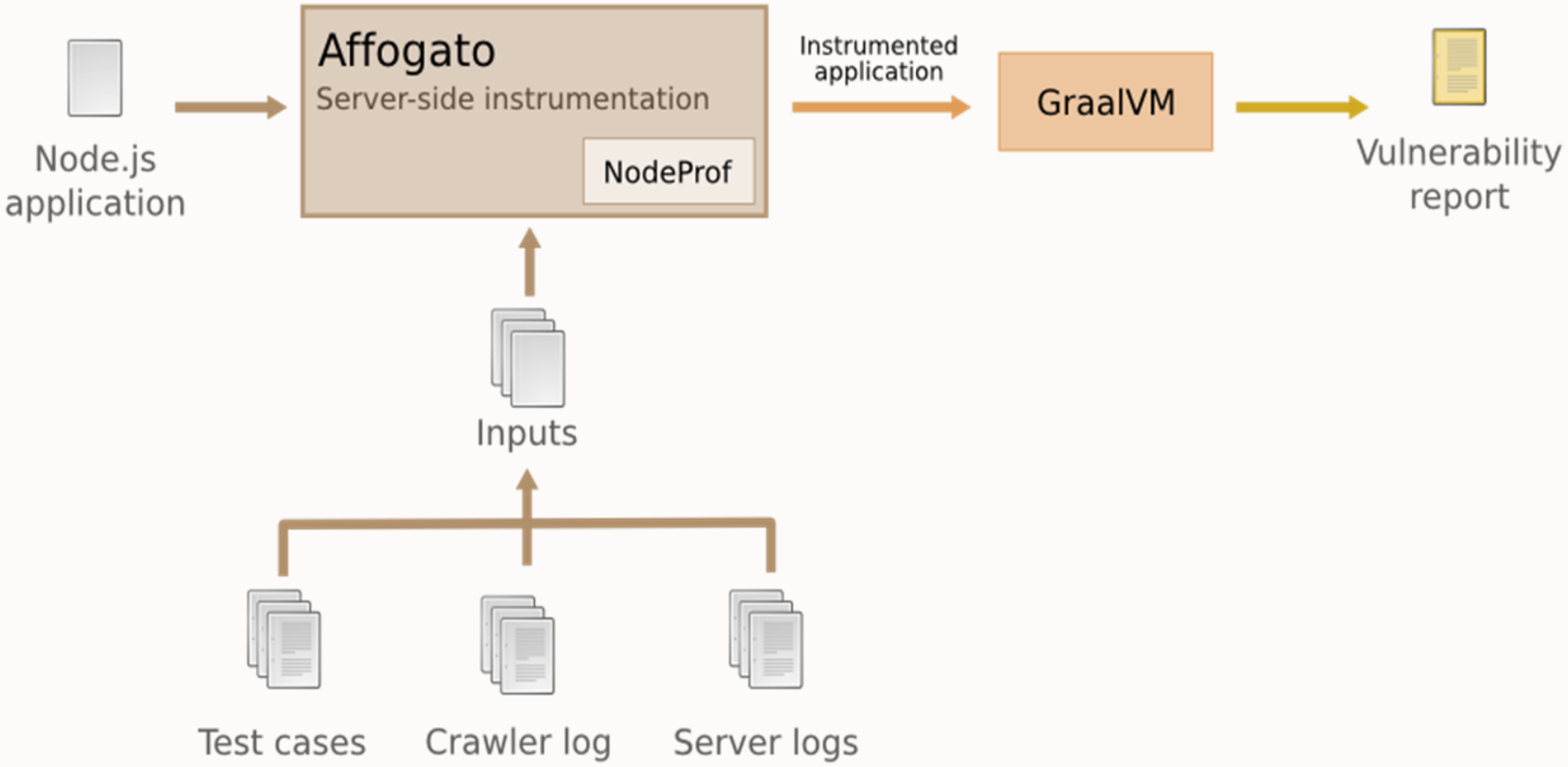
1. Integrate Parfait wrappers/plugins into your build
2. Configure Parfait with your attack surface
3. Run Parfait proper over the generated .bc files

Affogato: Analysis at Testing Time

- Instrumentation-based dynamic analysis tool for Node.js
- Tunable precision through configuration, with optional post-processing
- Overhead amortises over time for long runs, high for short-running tests
- Vulnerabilities that can be detected
  - SQL injection
  - Cross-site scripting
  - Path traversal
  - Unvalidated redirects and forwards
  - Command injection
  - Information leak

AFFOGATO: runtime detection of injection attacks for Node.js (Workshop at ISSTA/ECOOP 2018)

Affogato Architecture

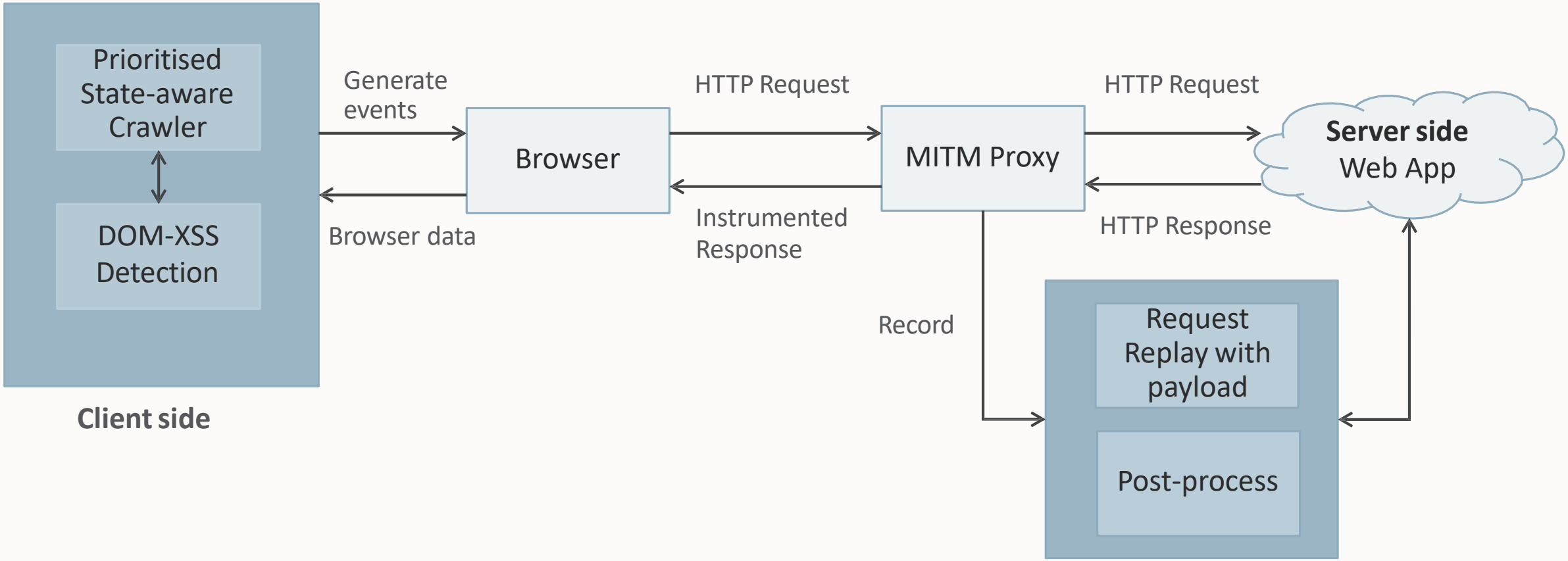


Gelato: Analysis at Deployment Time

- Dynamic analysis tool for web applications
  - Treats server as black-box
  - Analyses client-side JavaScript code
  - Detects end-points on the server with high precision
- Vulnerabilities that can be detected
  - DOM-XSS on the client
  - Reflected XSS on the server
  - Information leak: Session tokens in URL parameters

Gelato: Feedback-driven and Guided Security Analysis of Client-side Web Applications

Gelato Architecture



Strengths

- Parfait
  - ✓ Has very low FPs: Identifies actionable items
  - ✓ Can handle incomplete code
- Affogato
  - ✓ Works on running instance with test cases
  - ✓ Acceptable overhead of instrumentation acceptable
- Gelato
  - ✓ Works on running instance
  - ✓ No test cases required

Challenges

- Parfait
  - ? Support for frameworks: Code not invoked directly
  - ? Use of reflection makes static analysis hard
- Affogato
  - ? Instrument diverse systems without affecting semantics
  - ? What does semantics preservation mean in general?
- Gelato
  - ? Is Gelato making progress or just stuck?
  - ? How to reach a state to start a new exploration?

Other Projects

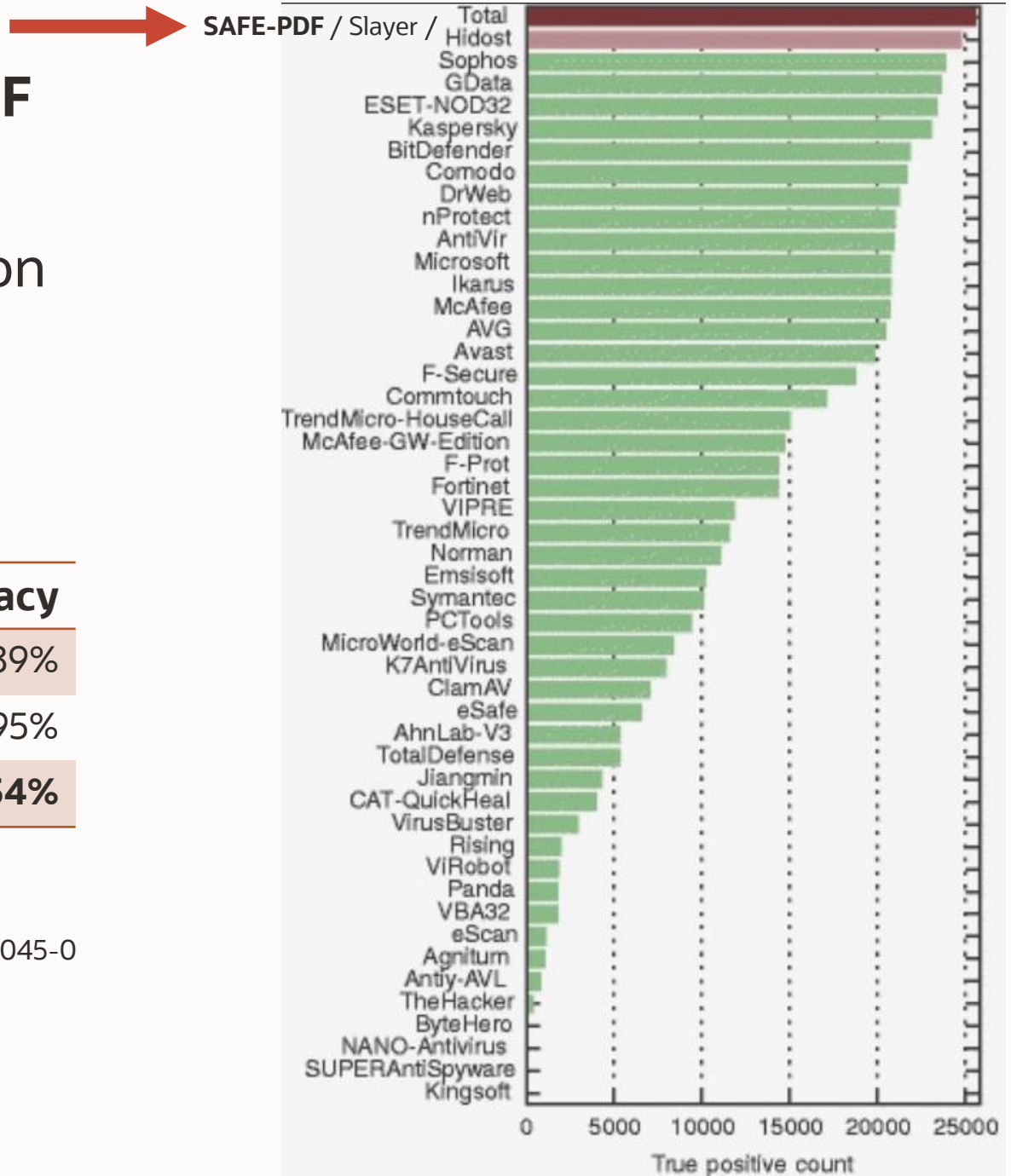
- Malware detection: Go beyond syntactic patterns
  - Abstract Interpretation based analysis of JavaScript in PDF
  - ML techniques for Office documents
- Synthesis
  - Generate program-point specific allowlists
  - Blocklists can be evaded
- Software Composition Analysis (SCA)
  - Nascent stage: Uses and enhances our existing techniques

Detecting Malicious PDFs with SAFE-PDF

- Based on the SAFE abstract interpretation framework (SAFE-PDF).
- 9,410 malicious, 14,306 benign PDFs.
- On average, < 4 sec. per document.

	Tool	FP Ratio	Recall	Accuracy
ML	Slayer	2.99%	99.23%	97.89%
ML	Hidost	1.53%	99.67%	98.95%
Abs. Int.	**SAFE-PDF**	**2.70%**	**99.93%**	**98.34%**

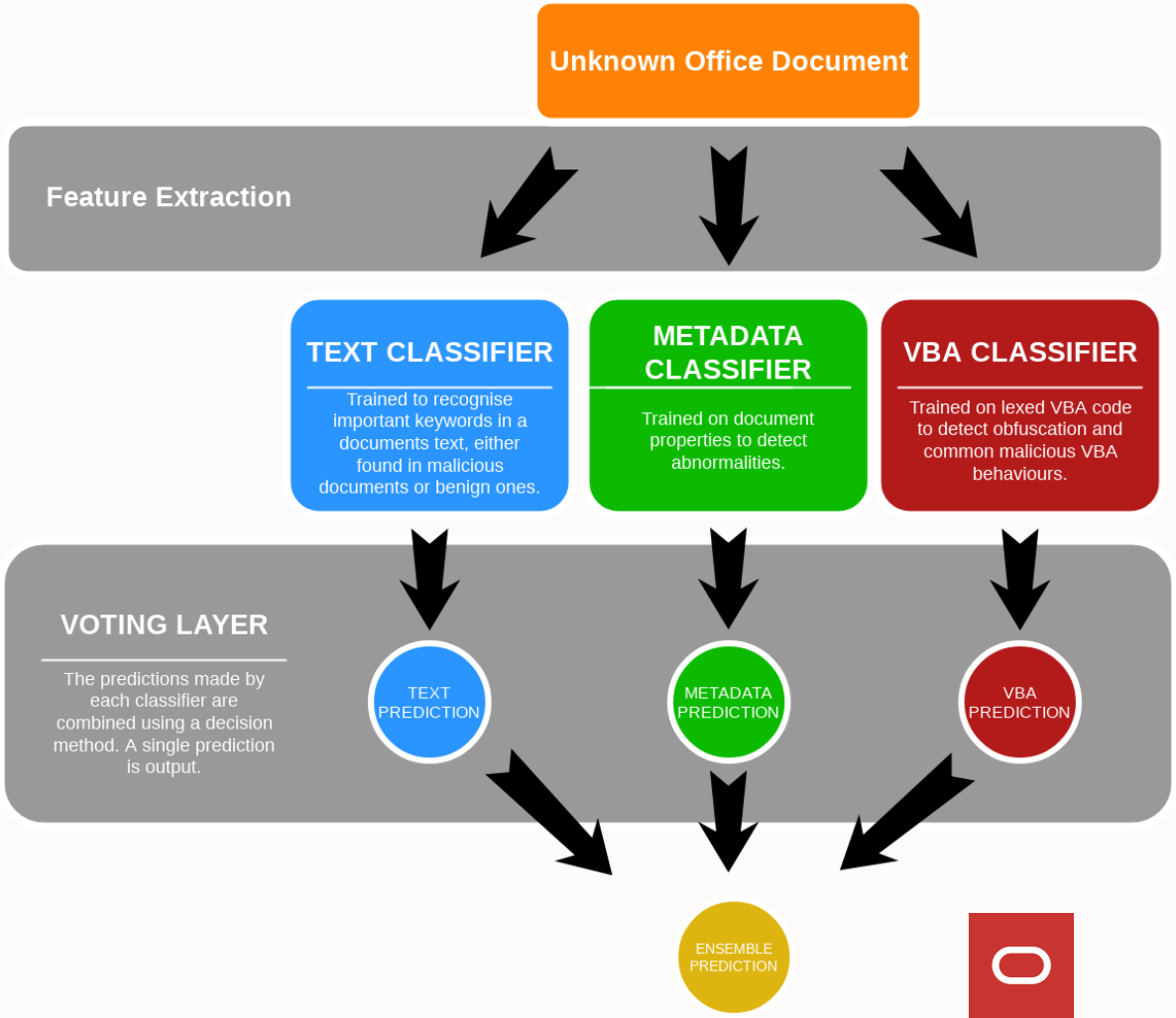
Sources:
https://github.com/sukyoung/safe
https://jis-eurasipjournals.springeropen.com/articles/10.1186/s13635-016-0045-0



Learning-Based Detection of Malicious Office Documents

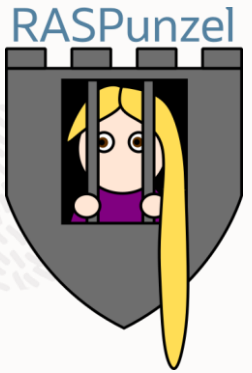
- 11,551 malicious email attachments
- 32,450 benign documents
- On average < 1 sec. per document

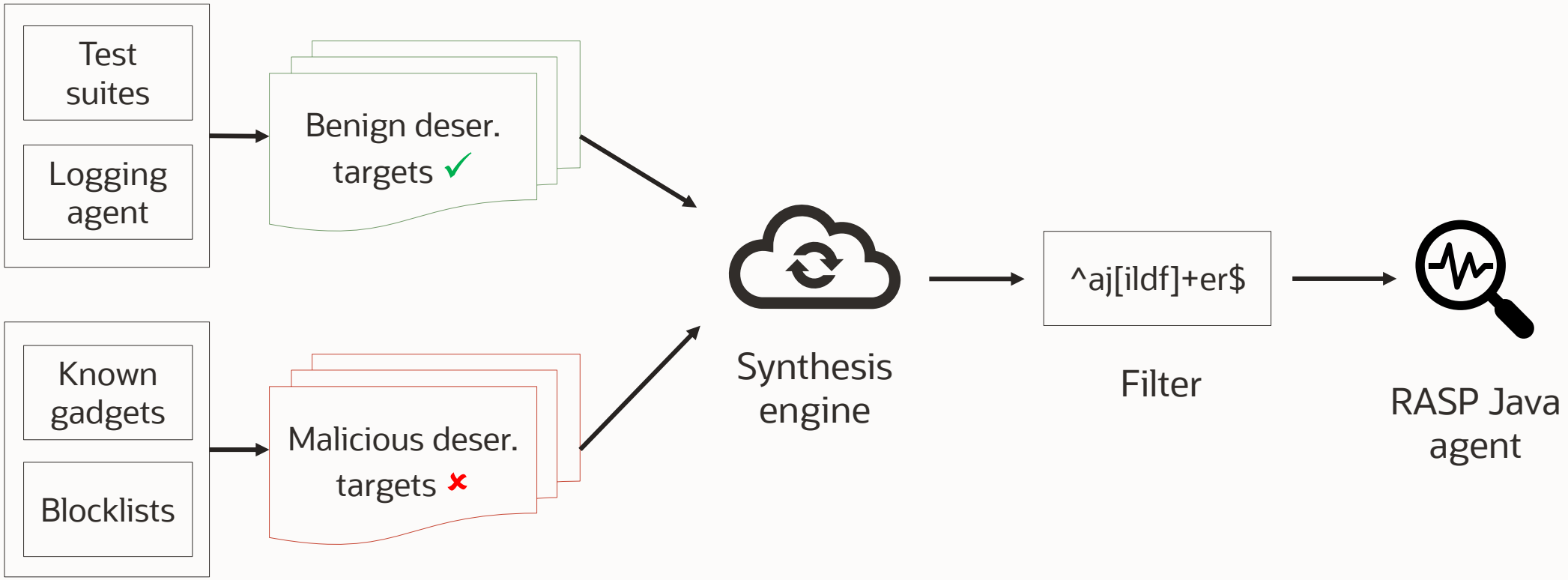
Model	Precision	Recall	Accuracy
Ensemble	96.94%	98.00%	97.45%



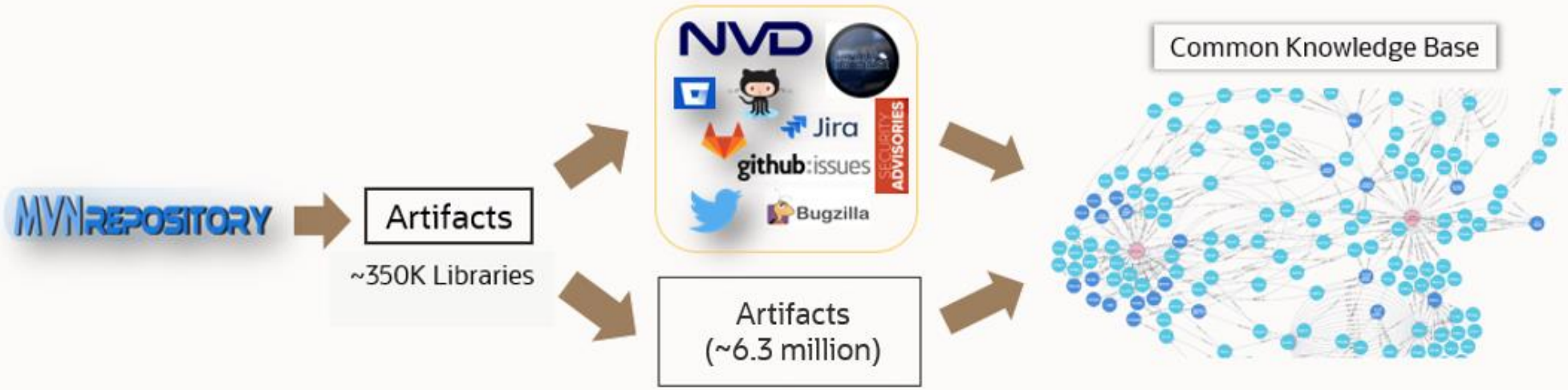
Data-Driven Runtime Application Self-Protection (RASP)

Automatically synthesizing and enforcing protections





SCA Analysis: Overview



Maven Repository: Centralized location that stores all the open source Java libraries

NVD: National Vulnerability Database

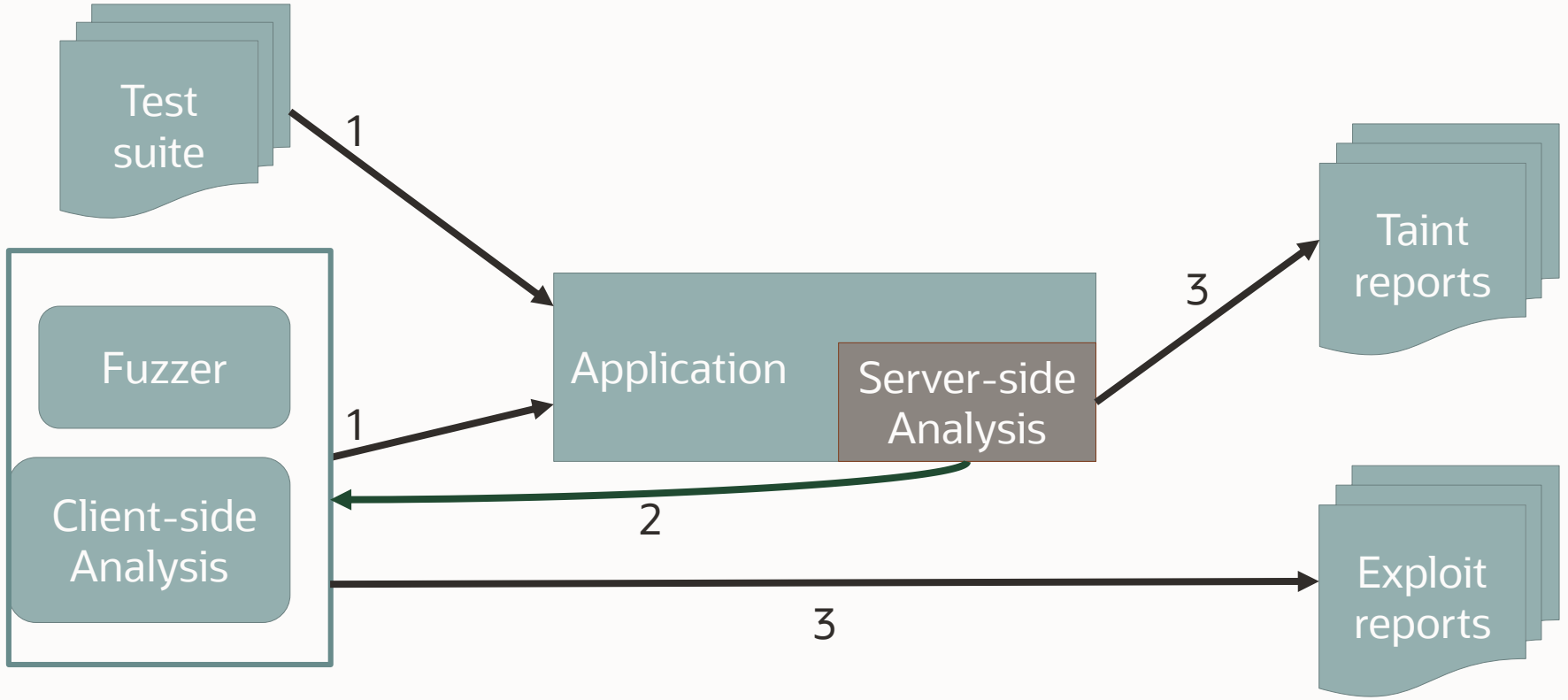
CVE: Common Vulnerabilities and Exposures. It's a list of publicly disclosed vulnerabilities and exposures

Intelligent Application Security At Scale

- Characteristics of real systems
  - Large: Multi-million lines of code, Uses many 3rd party libraries
  - Use different technologies
- Security game
  - Attacker has to find one exploit
  - Defenders have to protect against all possible attacks
- Different tools generate different signals
  - Automated handling of these signals
  - Examples: Integrate in CI/CD, Process logs from monitoring

Example of Combined Dynamic Analysis

1. Send or **generate** inputs
2. **Gather and use taint feedback**
3. Generate taint and **exploit** reports



Intelligent Application Security At Scale

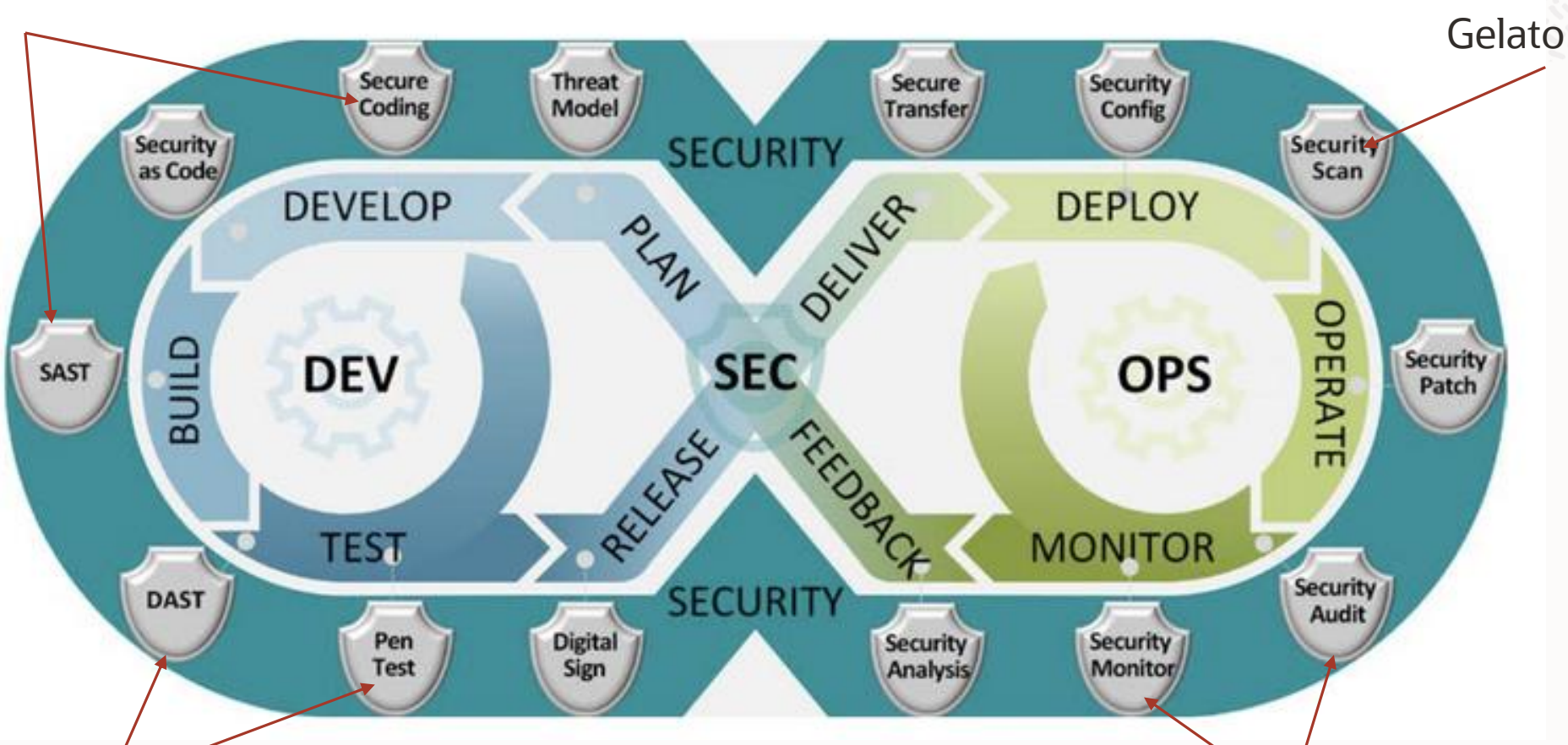
- Need to integrate security signals from different sources
  - Parfait and Affogato: Static and dynamic traces
  - Parfait and Gelato: Gelato finds taint sources to configure Parfait
  - SCA and Affogato: Vulnerable component used at runtime

- Relies of feedback from different phases of the life-cycle
  - DevSecOps: Code, build, test, deploy, operate, monitor
  - Security underpins all these activities

DevSecOps

Parfait, SCA

Gelato



Affogato/Gelato

Malware/Synthesis

Intelligent Application Security At Scale

- Intelligent Coding: Support for developers
  - Prevent security vulnerabilities from being introduced
- Intelligent Security Testing: Support for testers
  - Check for potential violations of policies
- Intelligent Security Monitoring: Deployment, Operations
  - Prevent attacks at runtime

General Research Questions

- What are the security issues that matter?
- Can we prevent 0-day attacks?
- How can one leverage different analysis concepts to detect the security issues?
- What are the tradeoffs when handling industrial scale systems?
- How to handle different technologies used in a single complex application?

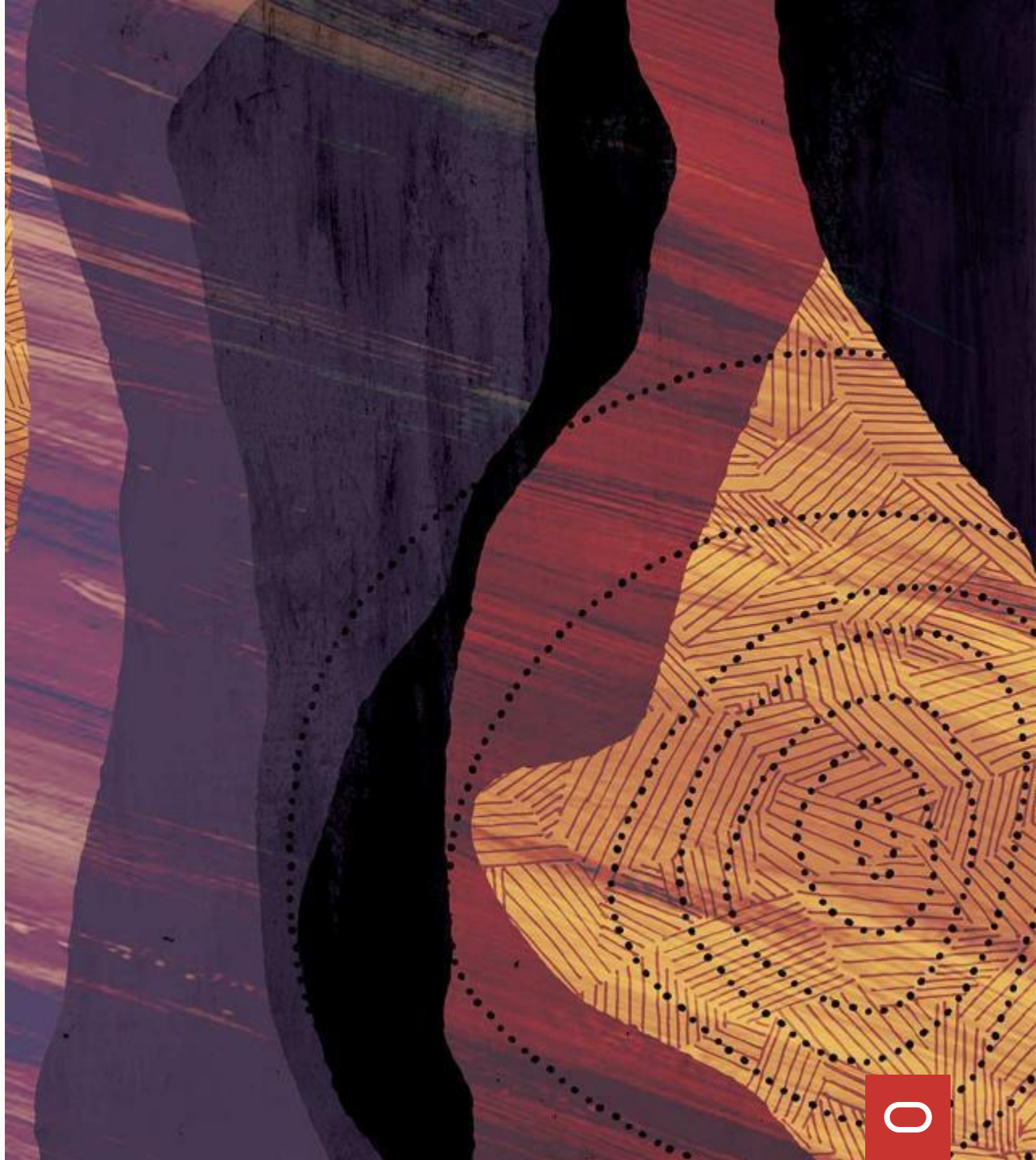
Conclusion

- Application Security vast field
- Different techniques needed to defend against potential attackers
- Parfait, Gelato, Affogato and other tools examples of different techniques
  - Useful at different phases in the DevSecOps cycle
- These tools need to work together
- Numerous research and engineering challenges remain
- Looking for collaborators
  - Have various types of internships
    - Mix of research and engineering tasks
    - Duration: E.g., 3-6 months full time, part-time

Thank You

Questions?

Paddy Krishnan
paddy.Krishnan@oracle.com



Our mission is to help people see data in new ways, discover insights, unlock endless possibilities.