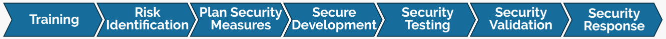
SAST, DAST, IAST and RASP

TIC 4302 - Information Security Practicum II

The material is adopted from “Secure Software Development on the Enterprise Level” presentation by Achim D. Brucker (University of Sheffield) and “Security Testing Comparison” from Software Testing Help Center

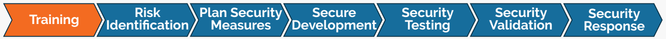
A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^{2}DL)



A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^2DL)

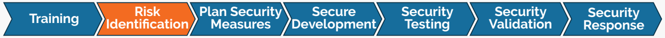


Training

- Security awareness
- Secure programming
- Threat modelling
- Security testing
- Data protection and privacy
- Security expert curriculum ("Masters")

A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^{2}DL)

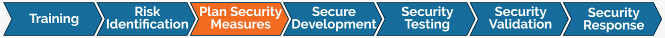


Risk Identification

- Risk identification ("high-level threat modelling")
- Threat modelling
- Data privacy impact assessment

A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^2DL)

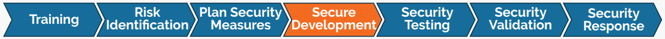


Plan Security Measures

- Plan product standard compliance
- Plan security features
- Plan security tests
- Plan security response

A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^{2}DL)

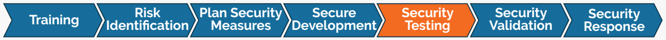


Secure Development

- Secure Programming
- Static code analysis (SAST)
- Code review

A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^{2}DL)

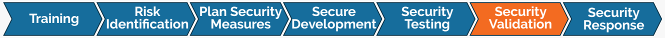


Security Testing

- Dynamic Testing (e.g., IAST, DAST)
- Manual testing
- External security assessment

A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S^2DL)



Security Validation ("First Customer")

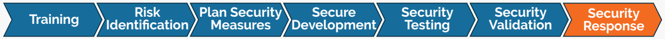
- Check for "flaws" in the implementation of the S^2DL
- Ideally, security validation finds:
- No issues that can be fixed/detected earlier
- Only issues that cannot be detect earlier
  (e.g., insecure default configurations, missing security documentation)

Penetration tests in productive environments are different:

- They test the actual configuration
- They test the productive environment (e.g., cloud/hosting)

A Path Towards (More) Secure Software

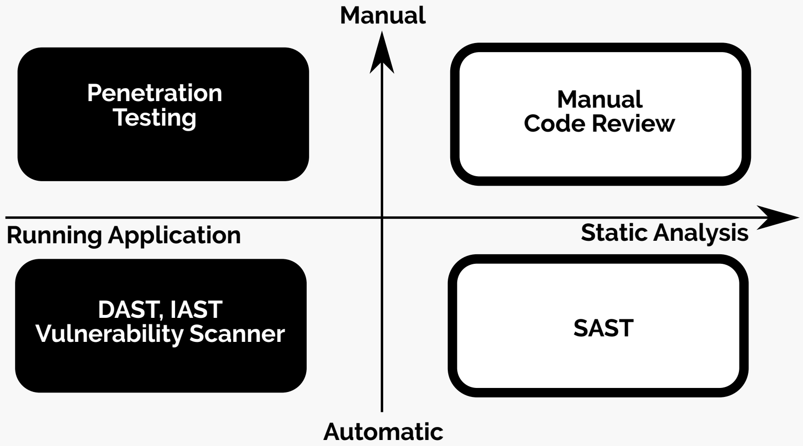
SAP's Secure Software Development Lifecycle (S^{2}DL)



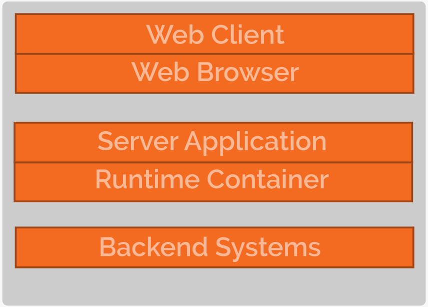
Security Response

- Execute the security response plan
- Security related external communication
- Incident handling
- Security patches
- Monitoring of third party components

Finding Security Vulnerabilities



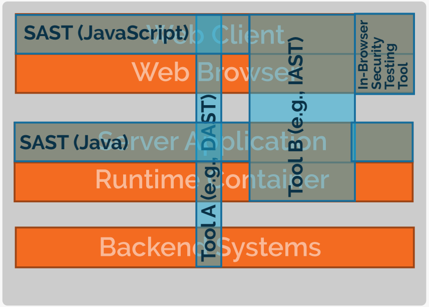
Combining Multiple Security Testing Methods and Tools



https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iast/

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack

Combining Multiple Security Testing Methods and Tools

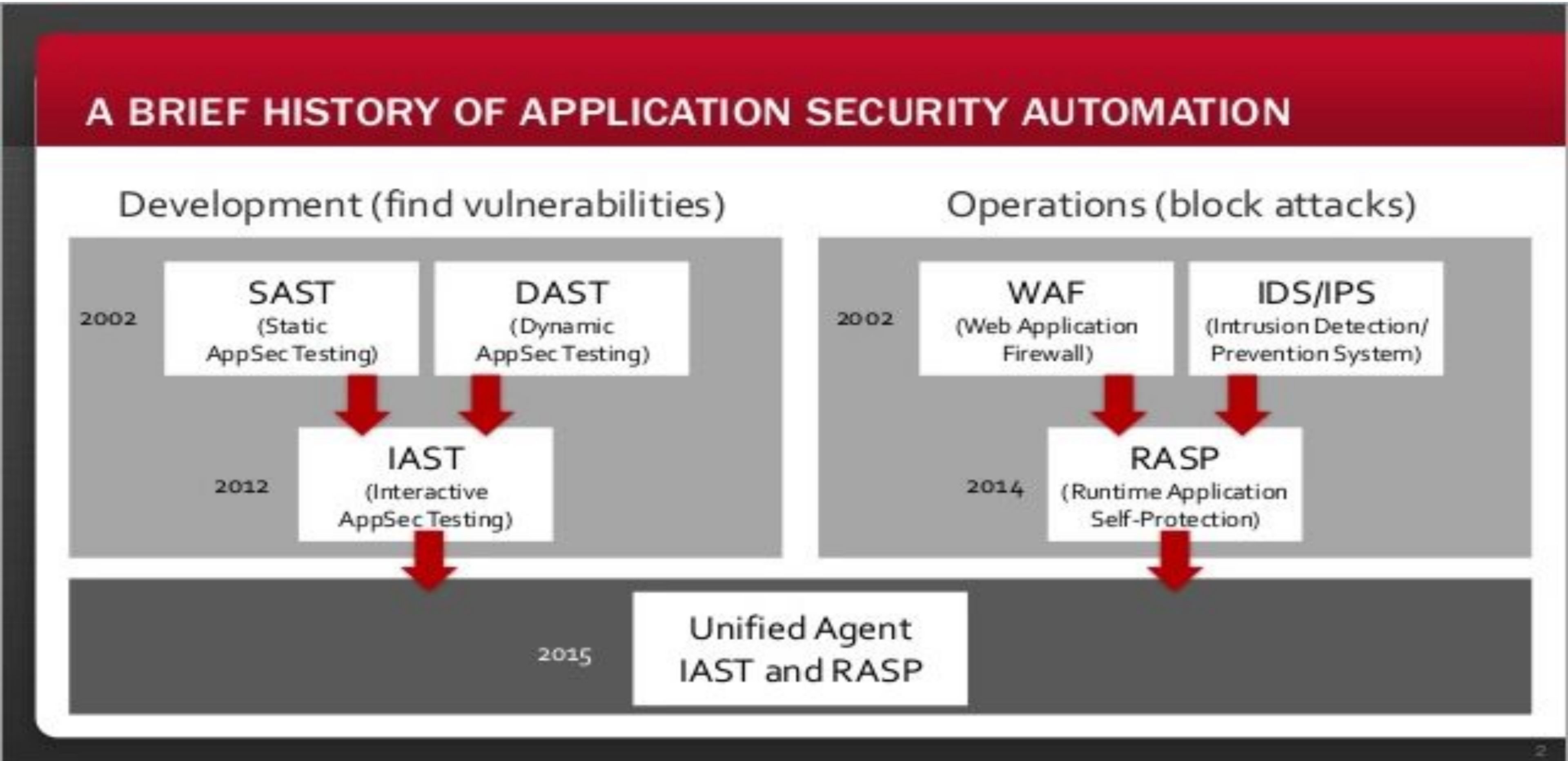


https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iast/

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack
- A comprehensive approach combines
  - Static approaches (i.e., SAST)
  - Dynamic approaches (i.e., IAST or DAST)

Document Classification and License Information

Differences Between SAST, DAST, IAST, And RASP


A BRIEF HISTORY OF APPLICATION SECURITY AUTOMATION
Development (find vulnerabilities)
Operations (block attacks)
2002
SAST
(Static AppSec Testing)
DAST
(Dynamic AppSec Testing)
2012
IAST
(Interactive AppSec Testing)
2002
WAF
(Web Application Firewall)
IDS/IPS
(Intrusion Detection/ Prevention System)
2014
RASP
(Runtime Application Self-Protection)
2015
Unified Agent IAST and RASP

SAST (Static Application Security Testing)

- Needs of an application that could automate or execute processes very fast and also improve performance and user experience without forgetting the negative impact an application that lacks security.
- Security testing is not about speed or performance rather it is about finding vulnerabilities.
- **Static** means that the test is done before an application is live and running, so **SAST** can help to detect vulnerabilities before the world finds them.
- **SAST** methodology analyzes a source code to detect any traces of vulnerabilities that could provide a backdoor for an attacker. **SAST** usually analyze and scan an application before the code is compiled.
- The process of **SAST** is also known as **White Box Testing**. Once a vulnerability is detected the next line of action is to check the code and patch the code before the code will be compiled and deployed to live.
- **White Box Testing** is an approach or method that testers uses to test the inner structure of software and see how it integrates with the external systems.

DAST (Dynamic Application Security Testing)

- This tool is used to scan any web applications to find security vulnerabilities.
- This tool is used to detect vulnerabilities inside a web application that has been deployed to production, so that **DAST tools** will always send alerts to the security team for immediate remediation.
- **DAST** can be integrated very early into the software development lifecycle and its focus is to help organizations to reduce and protect against the risk that application vulnerabilities.
- This tool is very different from **SAST** because **DAST** uses the **Black Box Testing Methodology**, it conducts its vulnerability assessment from outside as it does not have access to the application source code.
- DAST is used during the testing and QA phase of SDLC.

IAST (Interactive Application Security Testing)

- This tool was designed for both web and mobile applications to detect and report issues even while the application is running. **IAST** requires an understanding of **SAST** and **DAST**.
- It was developed to stop all the limitations that exist in both SAST and DAST, and it uses the **Grey Box Testing Methodology**.
- **IAST** occurs in real-time just like DAST while the application is running in the **staging** environment, but **IAST** can identify the line of code causing security issues and inform the developer for immediate remediation
- **IAST** also checks the source code just like **SAST** but this is at the post-build stage unlike the SAST that occur while the code is been built.
- **IAST** needs **agents** that usually deployed on the application servers, and when DAST scan and report a vulnerability then IAST agent that is deployed will return a line number of the issue from the source code.
- The IAST agents can be deployed on an application server and during functional testing performed by a QA tester, and it studies every pattern that a data transfer inside the application regardless it's dangerous or not.
- **For example**, if data is coming from a user and the user wants to perform an SQL Injection on the application by appending SQL query to a request, then the request will be flagged as dangerous.

RASP (Runtime Application Self Protection)

RASP is a runtime application that is integrated into an application to analyze inward and outward traffic and end-user behavioral pattern to prevent security attacks.

This tool is different from the other tools because it is used after product release which makes it a more security-focused tool when compared to the others that are known for testing.

RASP is deployed to a **web or application server** which makes it to sit next to the main application while it’s running to monitor and analyze both the inward and outward traffic behavior.

Immediately once an issue is found, **RASP** will send alerts to the security team and will immediately block access to the individual making request.

When you deploy **RASP**, it will secure the whole application against different attacks as it does **not just wait or try to rely only on specific signatures** of some known vulnerabilities.

RASP is a complete solution that observes every little detail of different attacks on your application and also knows **your application behavior**

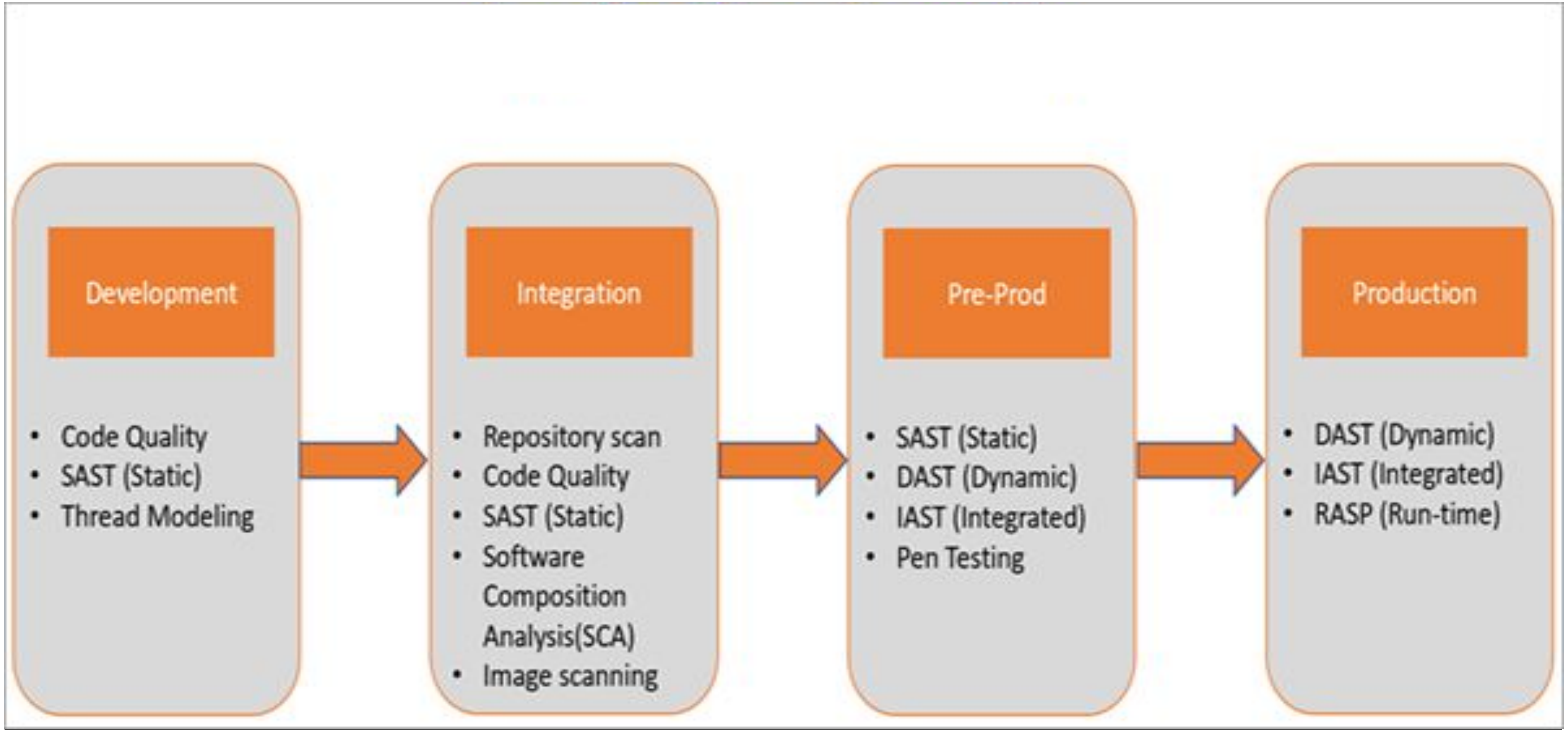
SAST vs DAST

White box testing / tested from the inside out / developer approach.	**Black box testing** / from outside in / the hacker approach.
It does **not need to be installed** rather needs the source code to act (change/modify)	It is **deployed on the application server** and does not need to have access to the source code before acting.
It is implemented immediately while the **code is being written.**	The **code should have been compiled** and the tool is used to scan the complete application
It is **not expensive tool** because the vulnerabilities are usually very early	It is **expensive tool** due to the fact that the vulnerabilities are usually discovered towards the end of the SDLC.
It is **difficult to discover** any run-time vulnerabilities.	The dynamic analysis **is used to to find** run-time vulnerabilities.
It is supports **any applications**.	It scans application like web app it **does not work with some other software**

IAST vs RASP

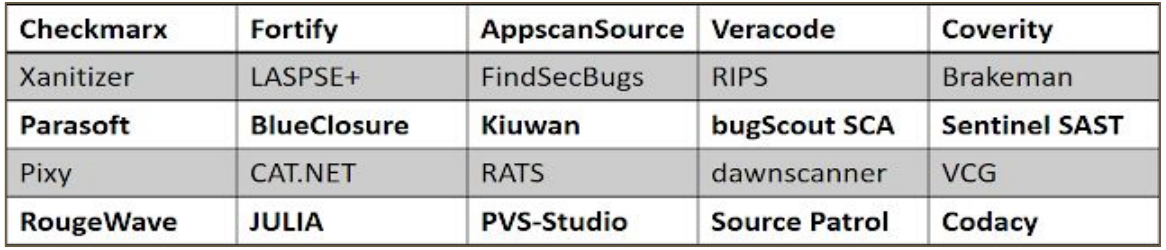
It is **mostly used** as a security testing tool.	It is **not just as a security testing tool** but used to protect the entire application by running alongside it.
It supports the accuracy of SAST through the use of the run-time analysis results from SAST.	It identifies and blocks threats in real-time and sometime it **does not even need any human intervention**
There is a **limited language support**.	It’s **not language or platform dependent**.
It is **easy to integrates** for the analysis of source code, runtime control and all the frameworks	It is **integrate seamlessly** with the application and it’s not reliant on any network-level protections like WAF.
It is the **best from the combination of SAST and DAST** functionality which equally helps it to discover vulnerabilities on a broader scale.	It **covers a broad range** of vulnerabilities
It is **gradually being accepted** even through it requires the deployment of an agent.	It is **not yet accepted** because it requires the deployment of an “intelligent” agent.

Integrating SAST/DAST/IAST/RASP into DevOps



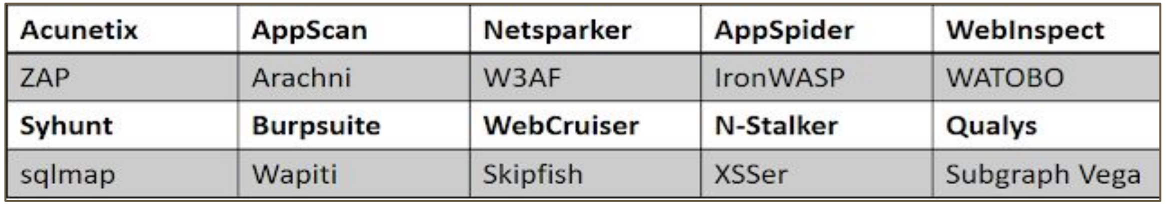
Security Testing Tools Lists

SAST



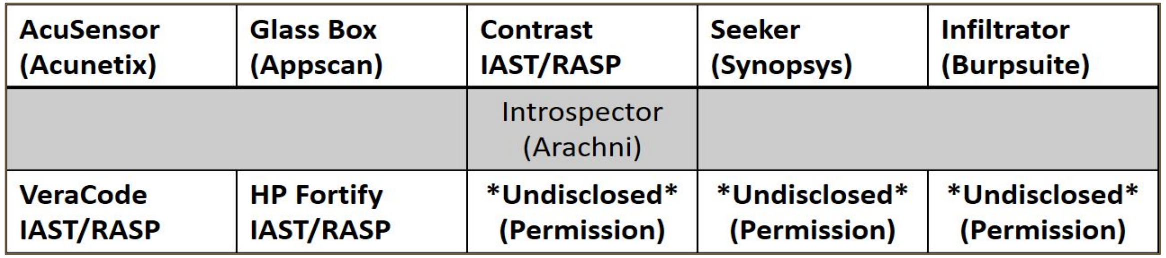
Checkmarx	**Fortify**	**AppscanSource**	**Veracode**	**Coverity**
Xanitizer	LASPSE+	FindSecBugs	RIPS	Brakeman
Parasoft	**BlueClosure**	**Kiuwan**	**bugScout SCA**	**Sentinel SAST**
Pixy	CAT.NET	RATS	dawnscanner	VCG
RougeWave	**JULIA**	**PVS-Studio**	**Source Patrol**	**Codacy**

DAST



Acunetix	**AppScan**	**Netsparker**	**AppSpider**	**WebInspect**
ZAP	Arachni	W3AF	IronWASP	WATOBO
Syhunt	**Burpsuite**	**WebCruiser**	**N-Stalker**	**Qualys**
sqlmap	Wapiti	Skipfish	XSSer	Subgraph Vega

IAST/RASP



AcuSensor (Acunetix)	**Glass Box (Appscan)**	**Contrast IAST/RASP**	**Seeker (Synopsys)**	**Infiltrator (Burpsuite)**
		Introspector (Arachni)		
VeraCode IAST/RASP	**HP Fortify IAST/RASP**	***Undisclosed* (Permission)**	***Undisclosed* (Permission)**	***Undisclosed* (Permission)**

http://sectooladdict.blogspot.com/2017/05/dast-vs-sast-vs-iast-modern-ssldc-best.html