

**API** **ADDICTS**  
**DAYS**  **21**

EL MAYOR EVENTO  
DE EXPERTOS EN **APIS**

# Seguridad en el CI/CD de las APIs

**Roger Carhuatocto**

# Roger Carhuatocto

Más de 20 años en Seguridad y Sistemas Distribuidos; trabajé en diferentes proyectos en Latam y Europa. Actualmente ayudo a Organizaciones en UK a asegurar su Cloud Infrastructure, sus Aplicaciones y sus datos.

Especializado en:

- Security in Distributed Systems (SOA, Microservices, APIs & Service Mesh).
- Cloud Security and DevSecOps.
- Cryptography and PKI.

Twitter @Chilcano · <https://www.linkedin.com/in/chilcano> · <http://HolisticSecurity.io>

# ÍNDICE

- 1) Situación actual: API y Seguridad
- 2) SDLC para las APIs
- 3) Seguridad, cómo y dónde?
- 4) Controles de Seguridad
- 5) Buenas Prácticas
- 6) Planeando la implementación
- 7) Seguridad en tu Pipeline
- 8) Conclusiones





# 1. Situación actual: API+Seguridad

1. Transformación Digital -> API-first
2. Pandemia -> Remote Access
3. Sistemas Distribuidos -> Microservices
4. Abundancia de Recursos Tecnológicos



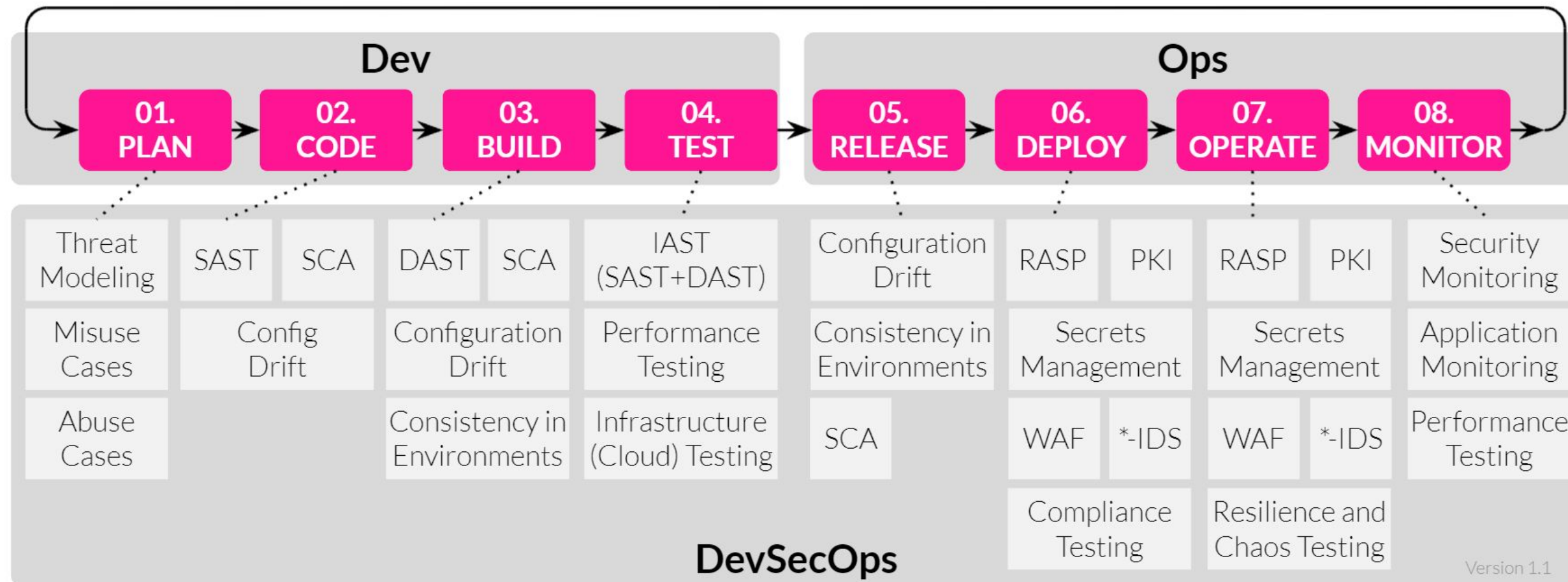
La implementación de un API, por defecto, extiende la “Superficie de Ataque”.

## 2. SDLC para las APIs



- Pregunta: El ciclo de desarrollo de APIs difiere al ciclo de desarrollo de software?, Existe un SDLC específico para APIs?
- Respuesta: No.

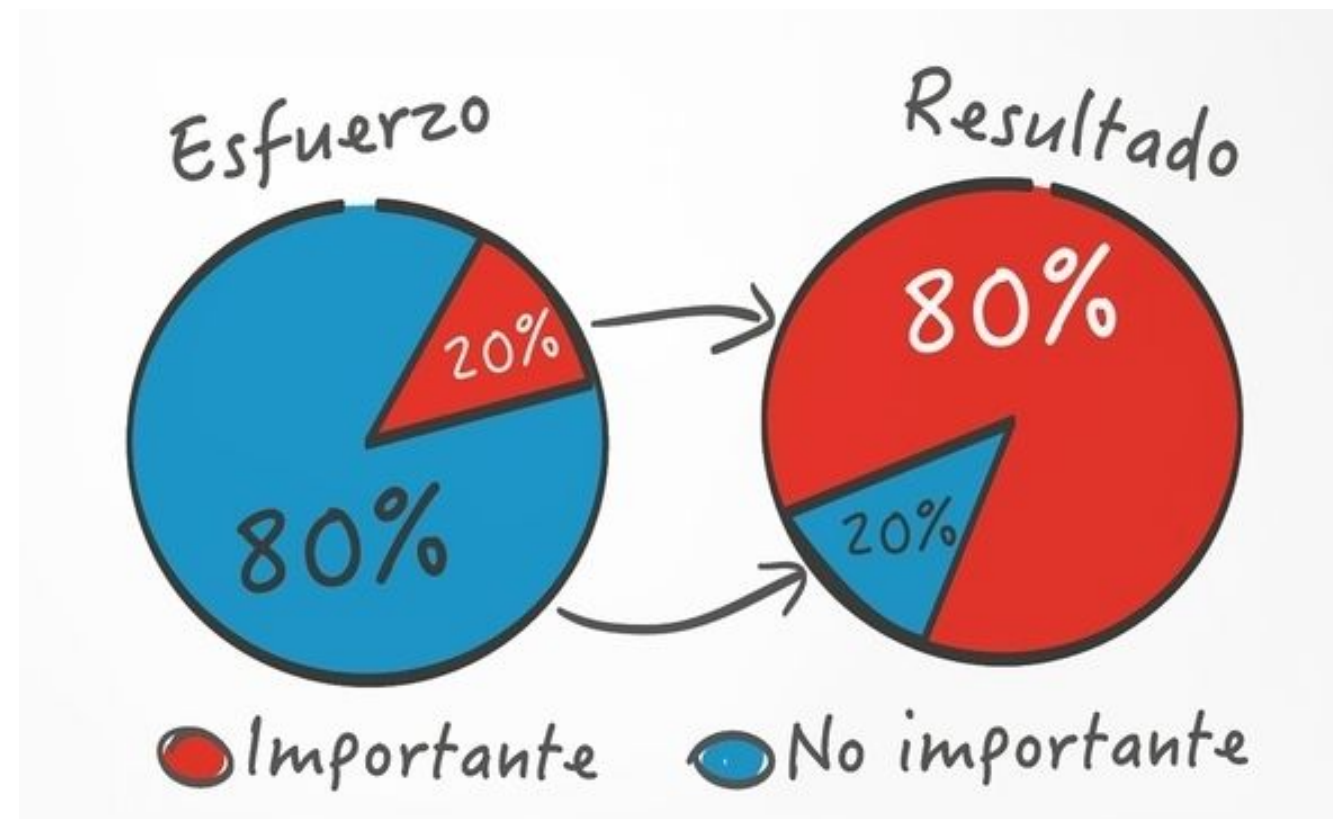
## 2. SDLC para las APIs



- Pregunta: Existen “controles de seguridad” específicos para APIs?.
- Respuesta: No, los controles son similares a los ya aplicados a tu SDLC, sin embargo, la implementación de los controles sí son diferentes.

# 3. Seguridad, cómo y dónde?

- Pregunta: Considerando todos los controles de seguridad existentes, cómo y dónde empiezo?
- Respuesta:
  - Pareto Principle



The Pareto principle states that for many outcomes, roughly 80% of consequences come from 20% of causes (the “vital few”). Other names for this principle are the 80/20 rule, the law of the vital few, or the principle of factor sparsity

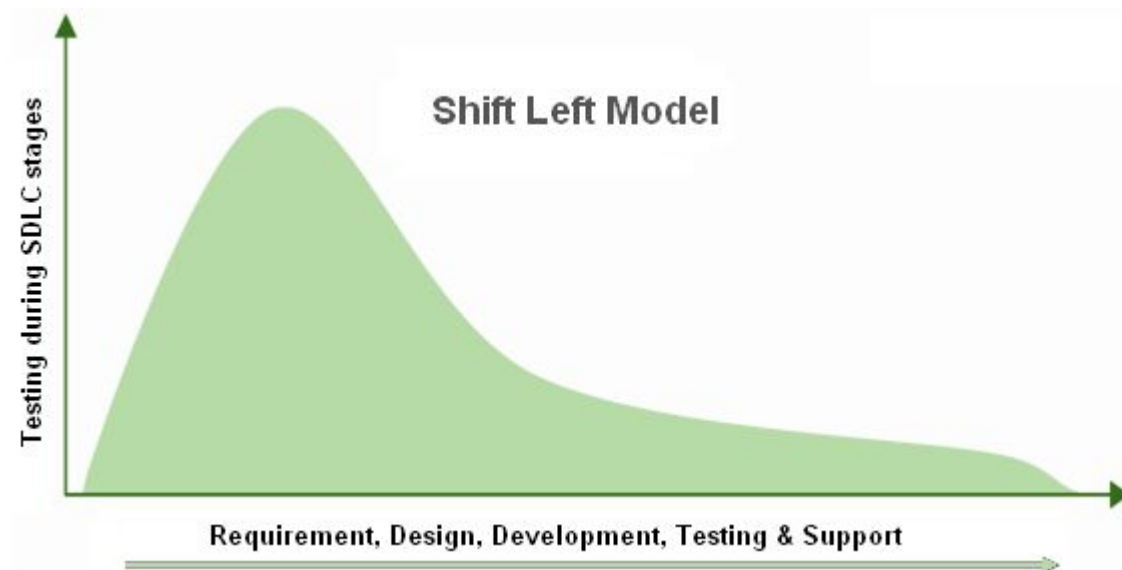
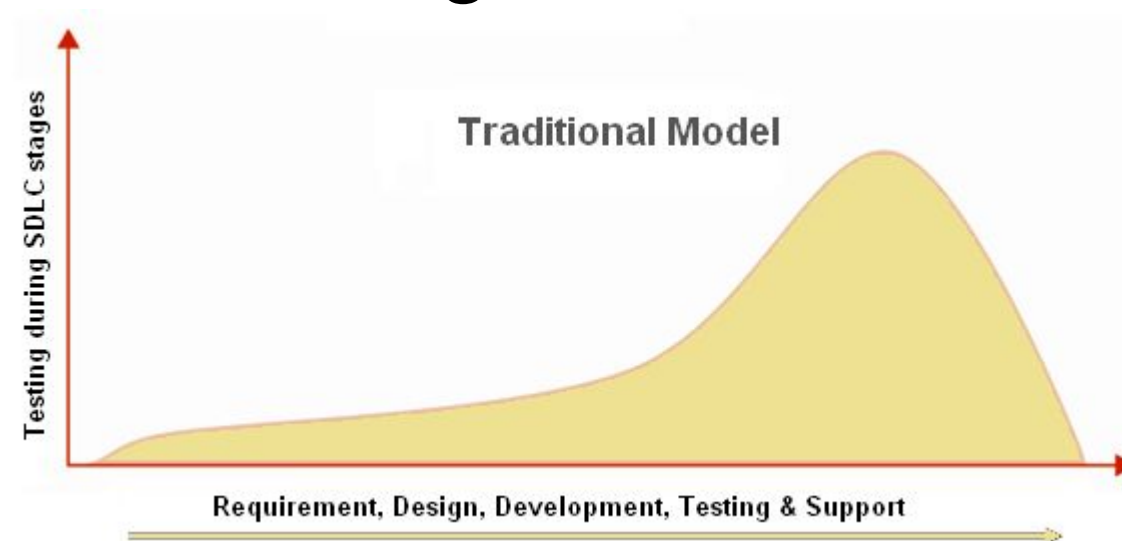


# 3. Seguridad, cómo y dónde?

- Pregunta: Considerando todos los controles de seguridad existentes, cómo y dónde empiezo?

- Respuesta:

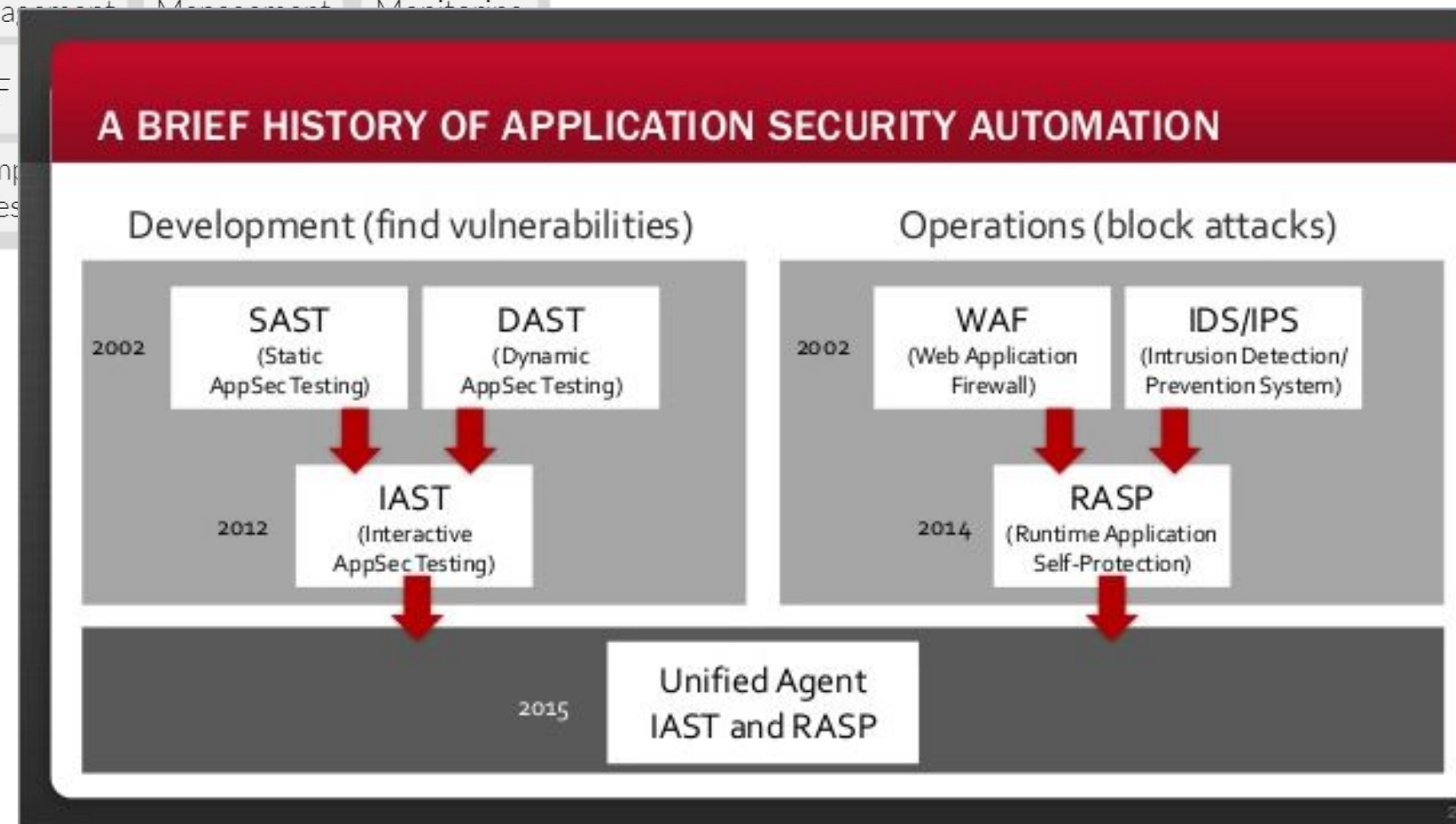
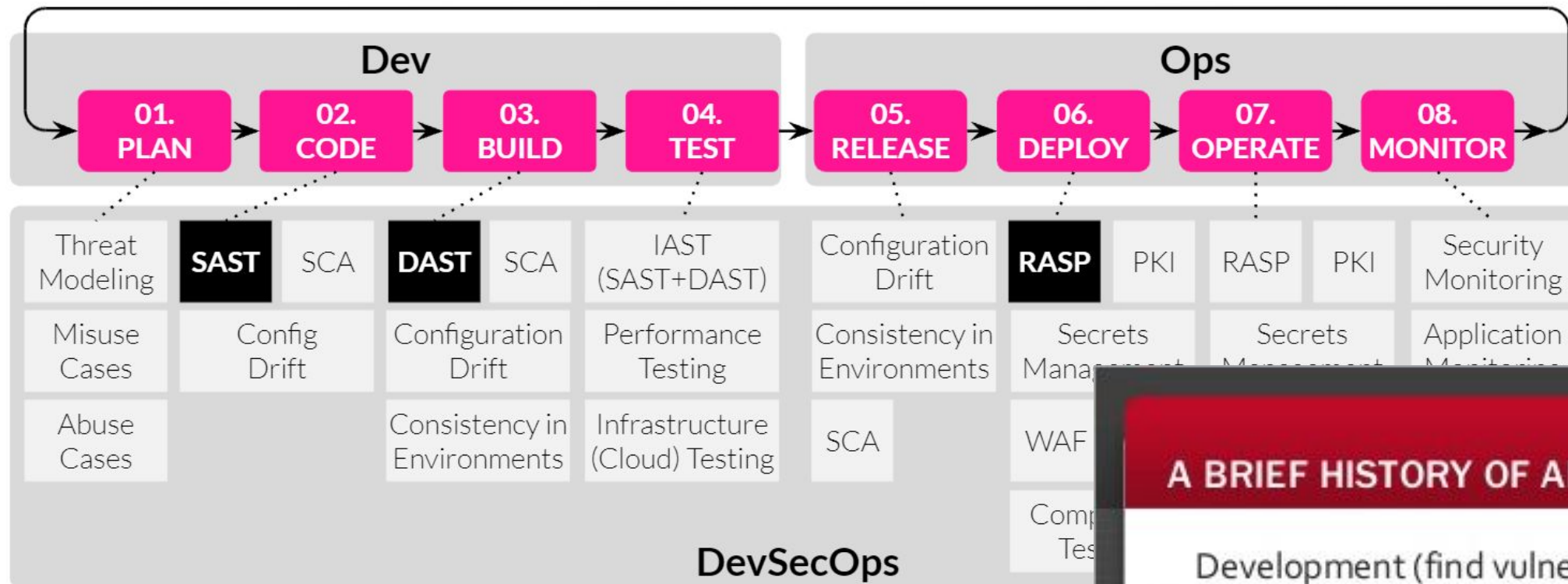
→ Shift Left Testing



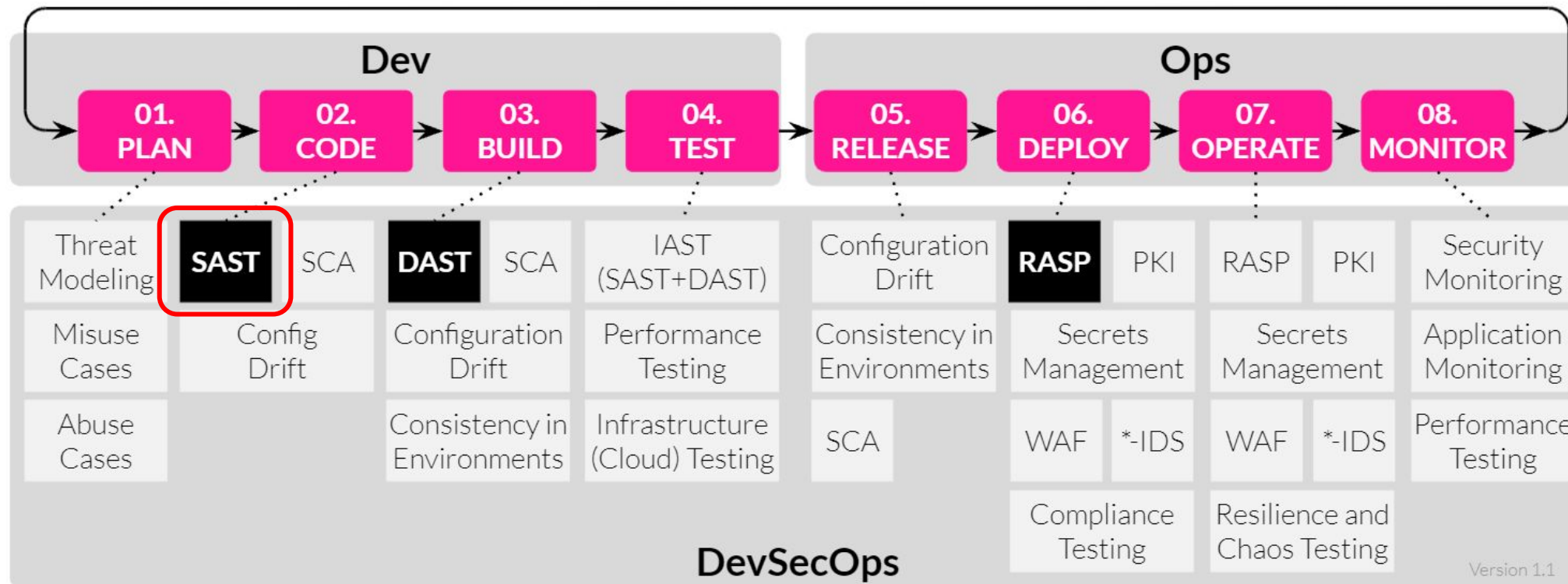
Research from the Ponemon Institute, in 2017, found that if vulnerabilities get detected in the early development process, they may cost around \$80 on an average. But the same vulnerabilities may cost around \$7,600 to fix if detected after they have moved into production.



# 4. Controles de Seguridad

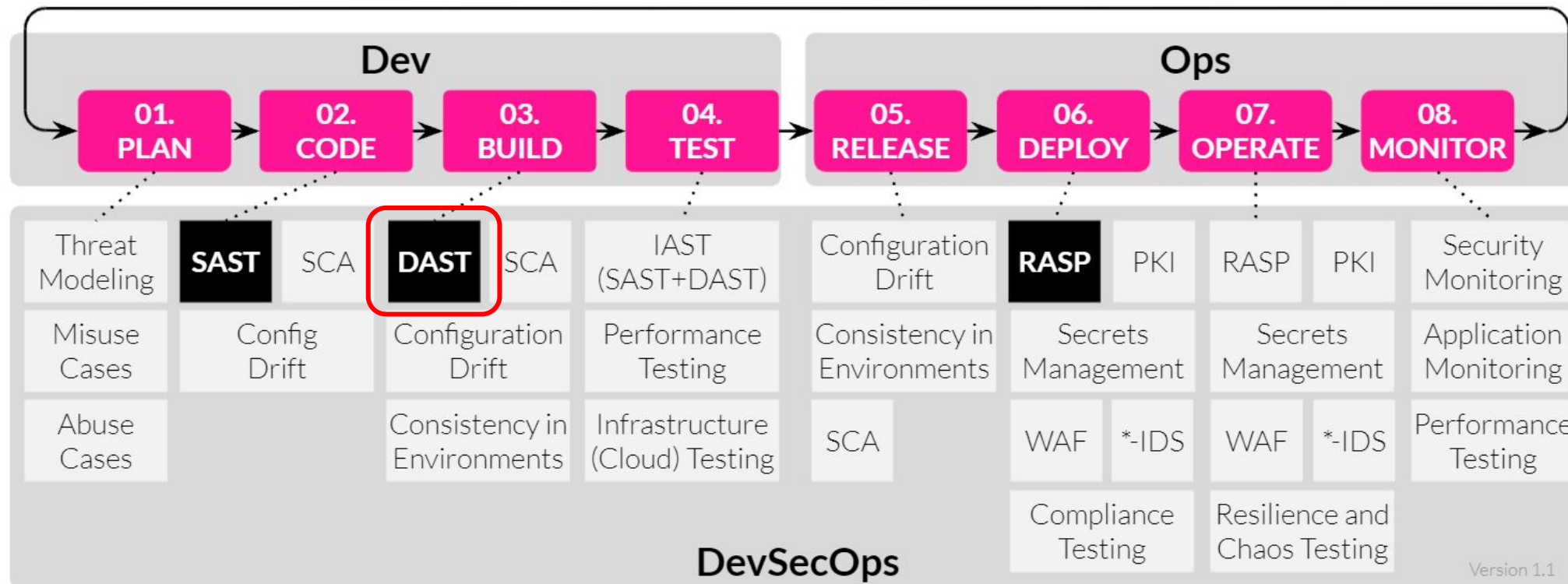


# 4. Controles de Seguridad



- SAST (Static Application Security Testing):
  - Analiza los archivos fuente de la aplicación para identificar vulnerabilidades potenciales de seguridad. Tal como lo hace un corrector ortográfico con un documento.
  - También llamado “White-box Testing” o “Glass-box Testing” porque analiza las estructuras internas o el funcionamiento de la aplicación.
  - Ejemplo: Command Injection.

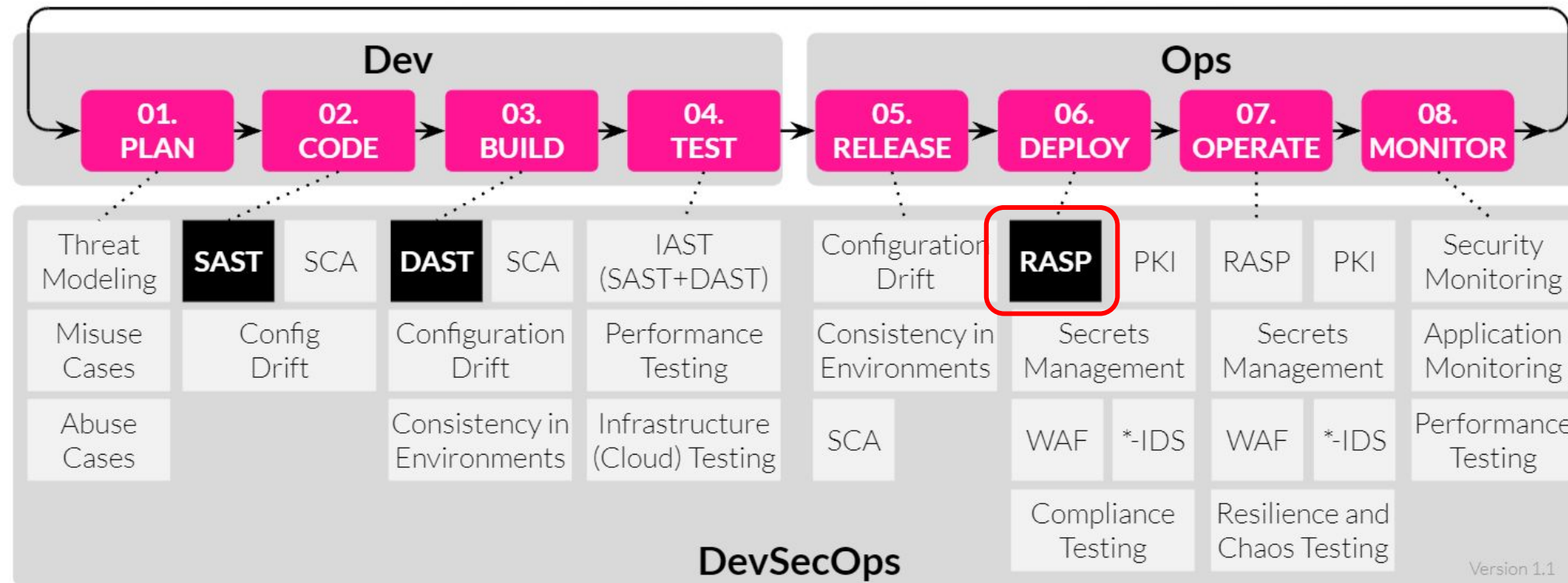
# 4. Controles de Seguridad



- DAST (Dynamic Application Security Testing):
  - Escanea la aplicación para identificar vulnerabilidades de seguridad. Se introduce fallas desde sus interfaces (UI, API Layer, etc.) simulando la ejecución de una funcionalidad.
  - También llamado “Black-box Testing” porque analiza la funcionalidad de la aplicación sin mirar sus estructuras internas o su funcionamiento (análisis desde el exterior).
  - Ejemplo: SQL Injection.



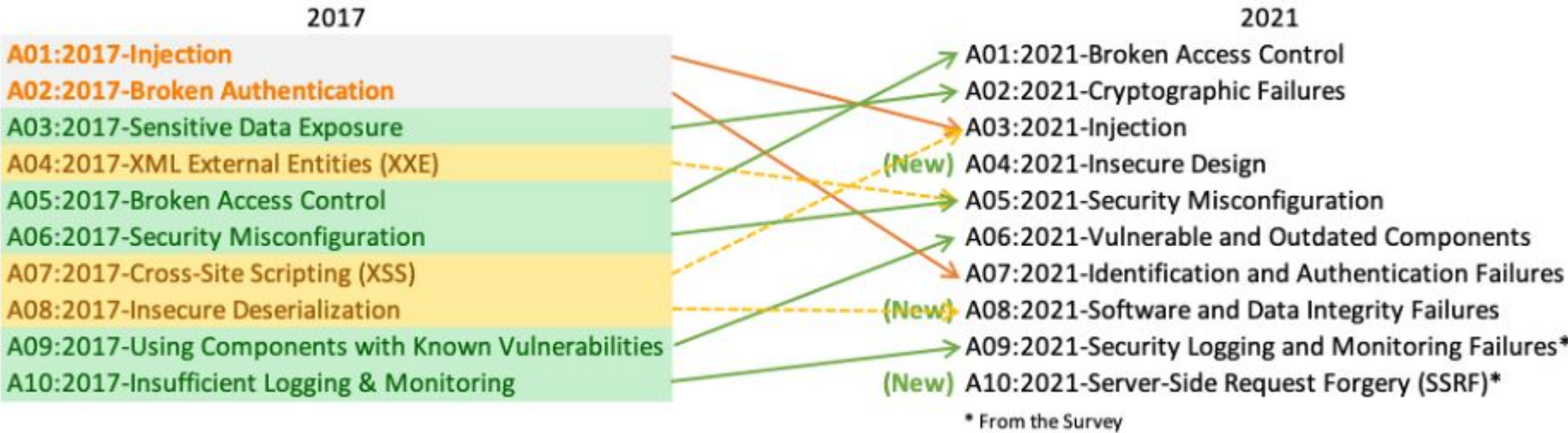
# 4. Controles de Seguridad



- RASP (Runtime Application Self Protection):
  - No hace “Testing”, RASP usa un agente en el lado de la aplicación para detectar y bloquear ataques tomando el control sobre el funcionamiento de la aplicación.
  - Ejemplo: Firewall, IDS, WAF, etc.

# 5. Buenas Prácticas

## OWASP Top 10



## OWASP API Security - Top 10 2019

API1:2019 - Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.
API2:2019 - Broken User Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising system's ability to identify the client/user, compromises API security overall.
API3:2019 - Excessive Data Exposure	Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.
API4:2019 - Lack of Resources & Rate Limiting	Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.
API5:2019 - Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.
API6:2019 - Mass Assignment	Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on a whitelist, usually lead to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.
API7:2019 - Security Misconfiguration	Security misconfiguration is commonly a result of insecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.
API8:2019 - Injection	Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
API9:2019 - Improper Assets Management	APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.
API10:2019 - Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems to tamper with, extract, or destroy data. Most breach studies demonstrate the time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

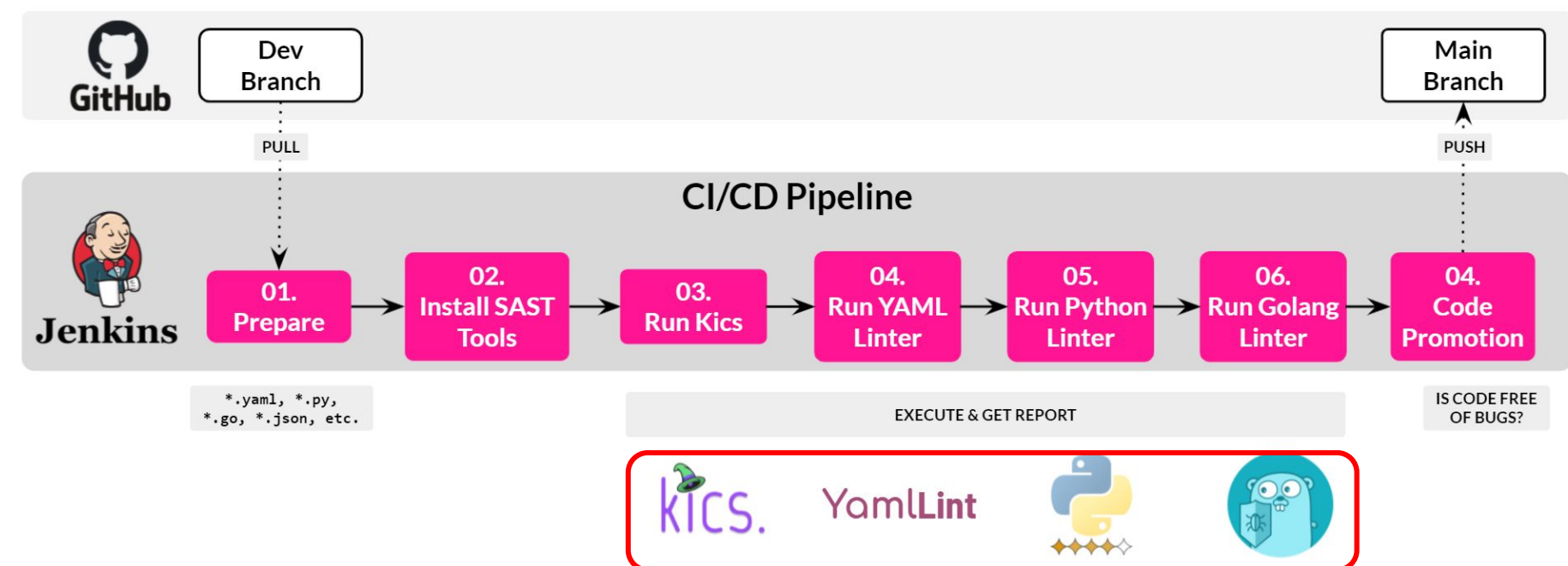


# 6. Planeando la implementación












1. Diseña/define tu workflow y stack.
2. Despliega tu CI/CD Tooling:
  - CI Server, Source Code System, Binary Artifact Server, Issue/Task System y QA Dashboard.
3. Selecciona tu DevSecOps Tooling:
  - SAST, DAST y RASP.
4. Inicia con una MVP:
  - Crea un Pipeline e integra SAST.
5. Extiende tu Pipeline a DAST:
  - Necesitarás definir o seleccionar un subconjunto de Casos de Pruebas de las Pruebas Funcionales que hayan sido definidas por el Equipo de QA.
  - Prepara la Aplicación: necesita estar en un entorno aislado y no estar siendo usada.
  - El subconjunto de Casos de Pruebas serán ejecutadas (automatizadas) desde el Pipeline.
  - Recoge los resultados de las pruebas.
6. Implementa RASP:
  - Lo más fácil es conectar un WAF a tu Gateway











# 7. Seguridad en tu Pipeline: SAST

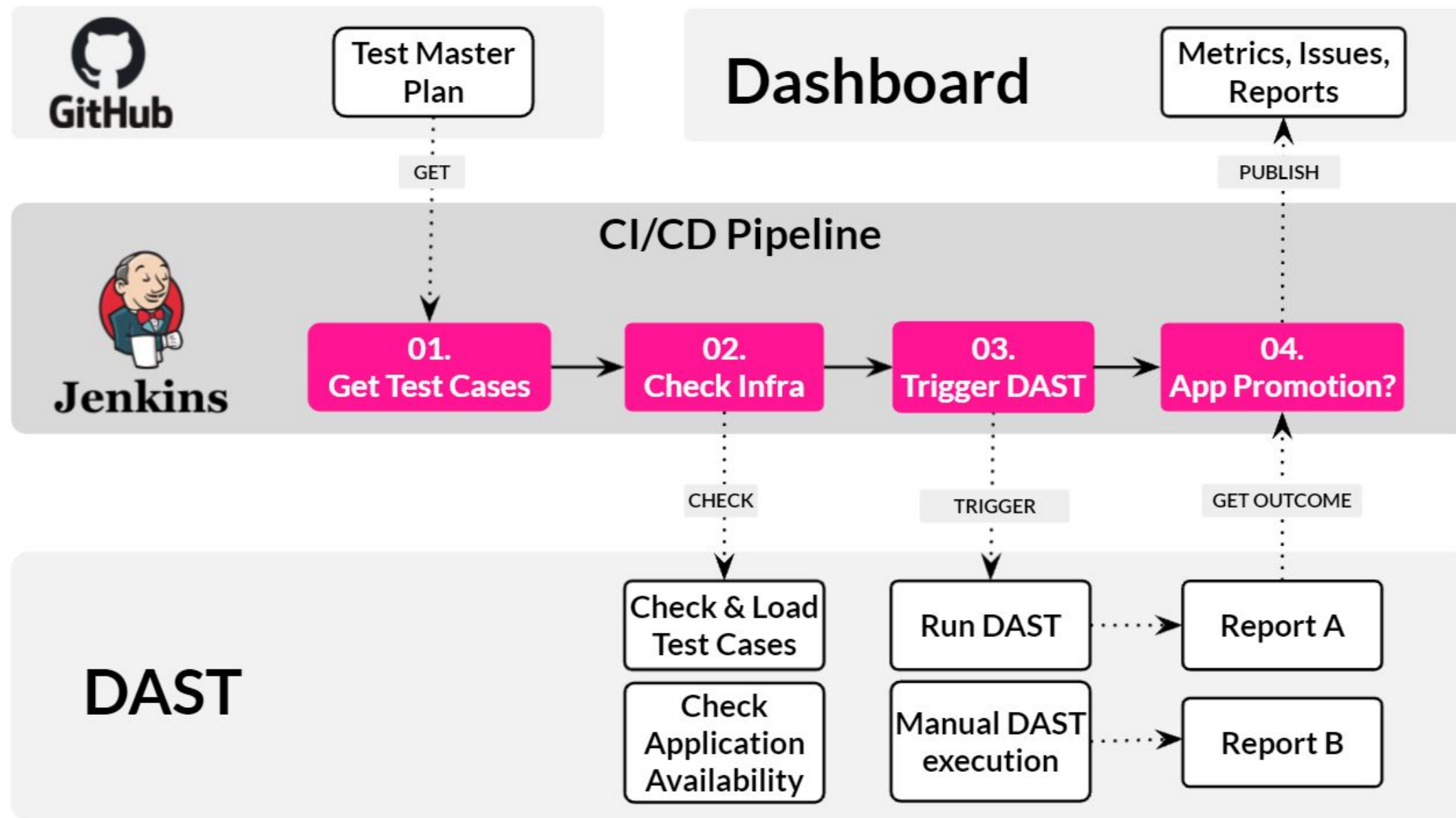










Lista completa de Productos de Seguridad Open Source:  
<https://holisticsecurity.io/2020/02/10/security-along-the-sdlc-for-cloud-native-apps/>

Name		Category
Semgrep		SAST
Anchore Gype		SAST
AquaSec Trivy		SAST
AWS CloudFormation Linter		SAST
Bandit (Python)		SAST
Checkmarx KICS		SAST
Checkov		SAST
Clair		SAST
Dockle		SAST
GolangCI-Lint		SAST
GoSec		SAST

Name		Category
huskyCI		SAST
PHPStan		SAST
Pylint (Python)		SAST
Skyscanner CFripper		SAST
SonarQube CE		SAST
SpotBugs (Java)		SAST
Stelligent cfn_nag		SAST
tfsec (Terraform)		SAST
AWS Serverless Rules		SAST
Dagda		SAST
Terraform Linter (tflint)		SAST

# 7. Seguridad en tu Pipeline: DAST

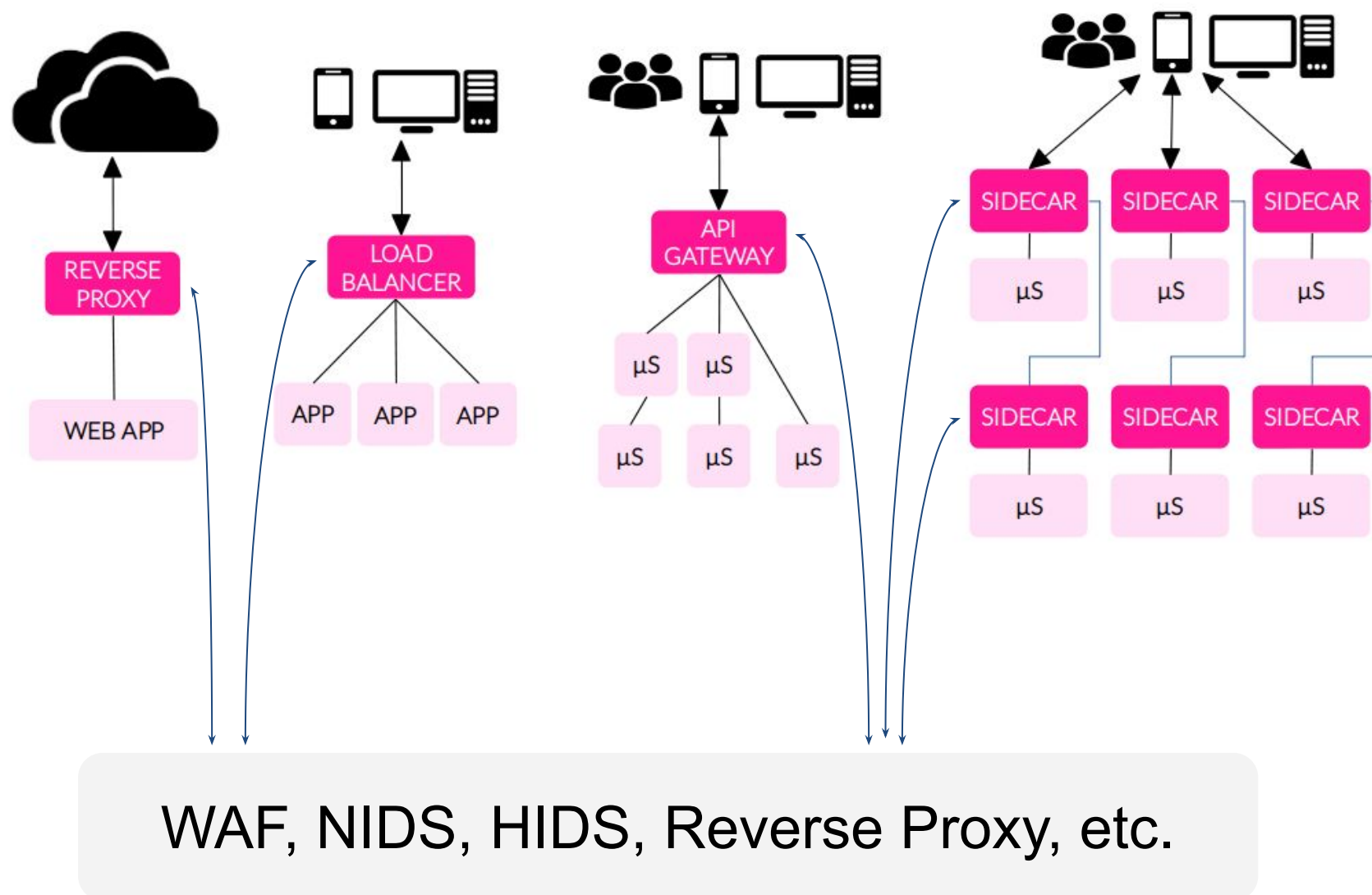


Name		Category
Nuclei	 nuclei	DAST
Burp Suite CE		DAST
AquaSec Kuber-Hunter	 kube-hunter	DAST
Arachni	 arachni	DAST
OpenSCAP	 OpenSCAP	DAST
OWASP Zed Attack Proxy (ZAP)	 ZAP	DAST
w3af	 w3af	DAST
Wapiti	 Wapiti	DAST
Grabber		DAST











Lista completa de Productos de Seguridad Open Source:  
<https://holisticsecurity.io/2020/02/10/security-along-the-sdlc-for-cloud-native-apps/>














# 7. Seguridad en tu Pipeline: RASP



Lista completa de Productos de Seguridad Open Source:  
<https://holisticsecurity.io/2020/02/10/security-along-the-sdlc-for-cloud-native-apps/>

Name		Category
ModSecurity		WAF
NAXSI		WAF
Heptio Ironclad		WAF
Ghostunnel Proxy		Proxy
Caddy Server		LB, Proxy, Ingress, Gateway
dapr		LB, Proxy, Ingress, Gateway
Datawire Ambassador (CE)		LB, Proxy, Ingress, Gateway
Envoy Proxy		LB, Proxy, Ingress, Gateway
Gloo Open Source		LB, Proxy, Ingress, Gateway
HAProxy Ingress Controller		LB, Proxy, Ingress, Gateway
Hashicorp Consul Connect		LB, Proxy, Ingress, Gateway

Name		Category
Istio		LB, Proxy, Ingress, Gateway
Linkerd		LB, Proxy, Ingress, Gateway
NGINX Ingress Controller		LB, Proxy, Ingress, Gateway
Rancher Load Balancer Controller		LB, Proxy, Ingress, Gateway
Traefik		LB, Proxy, Ingress, Gateway
Snort IDS		IDS
Capsule8 Sensor		HIDS, RASP
Sysdig Falco		HIDS, RASP
Sysdig Inspect		HIDS, RASP
OSSEC HIDS		HIDS
Wazuh HIDS		HIDS



# 8. Conclusiones

1. Adopta Shift-Left Testing.
2. Sigue el Principio de Pareto.
3. Automatiza tanto controles de seguridad como sea posible en tu SDLC.
4. Considera el Modelamiento de Amenazas. Aunque eso no se puede automatizar (por el momento), esa una “herramienta” que ayuda a explorar casos de uso y asociarlos a requerimientos y riesgos.

# Preguntas



# ¡GRACIAS POR ASISTIR!

## Contacto

+34 91 764 79 82

[contacta@apiaddicts.org](mailto:contacta@apiaddicts.org)

[www.apiaddicts.org](http://www.apiaddicts.org)



**Facebook**

[ApiAddicts](#)



**Linkedin**

[API Addicts](#)



**Twitter**

[@APIAddicts](#)



**Meet Up**

[API Addicts](#)



**Youtube**

[API Addicts](#)