



# 애플리케이션에 자가방어 능력 부여하기

RASP (Runtime Application Self Protection)



# RASP

**Runtime Application Self-Protection**  
런타임 애플리케이션 자가방어



# RASP – NIST SP 문서

(17) SOFTWARE, FIRMWARE, AND INFORMATION INTEGRITY | [RUNTIME APPLICATION SELF-PROTECTION](#)

**Implement [Assignment: organization-defined controls] for application self-protection at runtime.**

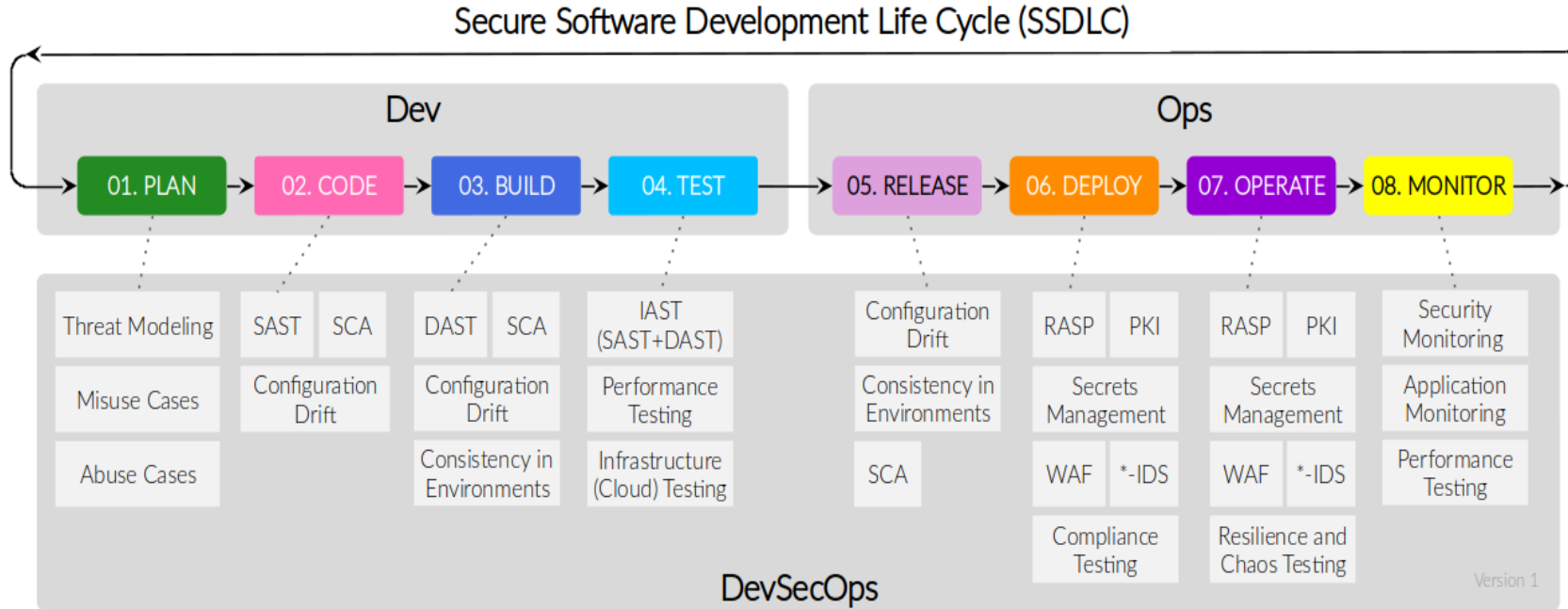
Discussion: Runtime application self-protection employs runtime instrumentation to detect and block the exploitation of software vulnerabilities by taking advantage of information from the software in execution. Runtime exploit prevention differs from traditional perimeter-based protections such as guards and firewalls which can only detect and block attacks by using network information without contextual awareness. Runtime application self-protection technology can reduce the susceptibility of software to attacks by monitoring its inputs and blocking those inputs that could allow attacks. It can also help protect the runtime environment from unwanted changes and tampering. When a threat is detected, runtime application self-protection technology can prevent exploitation and take other actions (e.g., sending a warning message to the user, terminating the user's session, terminating the application, or sending an alert to organizational personnel). Runtime application self-protection solutions can be deployed in either a monitor or protection mode.

Related Controls: [SI-16](#).

미 정부기관 및 관련 업체들이 정보보호 시스템 구축시 주요 참고문서로 활용하는 NIST(National Institute of Standards and Technology : 미표준기술연구소) SP(Special Publication) 800-53 문서에 RASP를 추가

(NIST SP 800-53 Rev.5 / 2020년 09월)

# 전문가 의견 – RASP



## SSDLC (Secure Software Development Life Cycle)

**Dev** 필수 솔루션으로 SAST(Static Application Security Testing) 및 DAST(Dynamic Application Security Testing)을  
**Ops** 필수 솔루션으로 RASP(Runtime Application Self-Protection)을

*(Holistic Security)*

# RASP

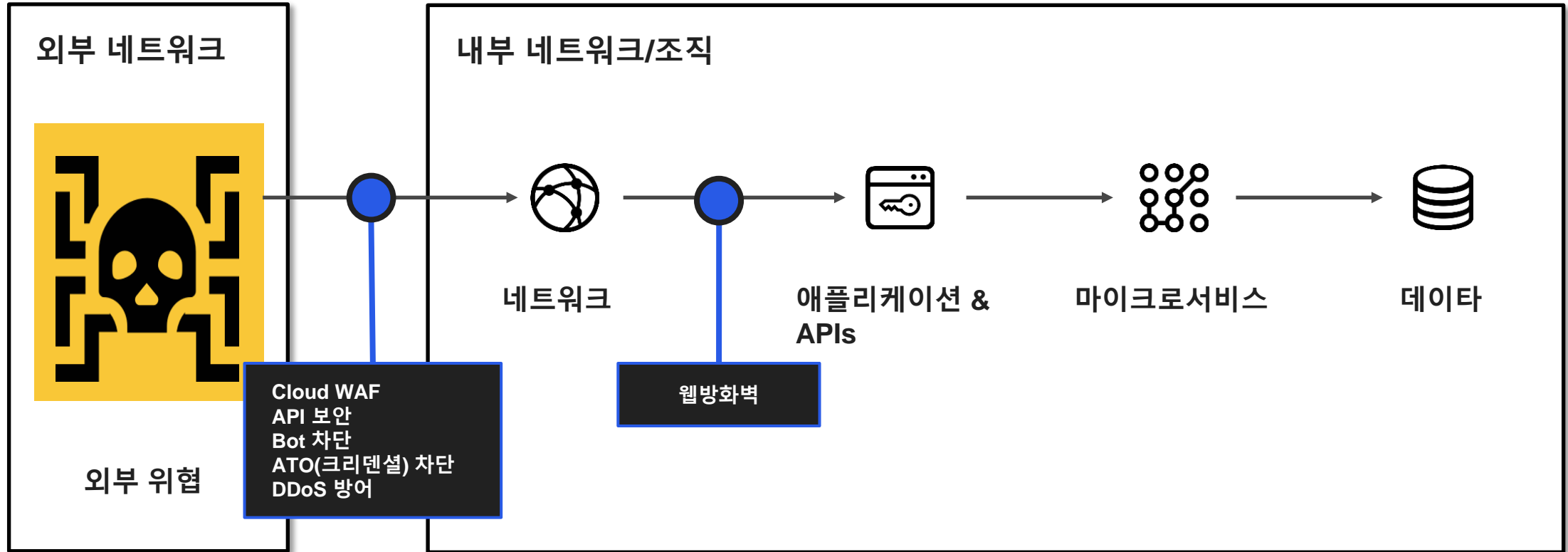
Runtime Application Self-Protection  
런타임 애플리케이션 자가방어

# Gartner®

**RASP는 애플리케이션 혹은 런타임 환경에 편성(Instrumentation)되어 애플리케이션이 스스로를 보호하는 기술**

**RASP는 애플리케이션 코드에 편성되어 애플리케이션의 행위를 모니터링 하고 애플리케이션의 실행을 통제하고 실시간으로 애플리케이션을 보호하는 기능을 제공**

# 기존 접근 방식 - 경계선을 지켜라



외부에서 내부로 향하는 침입이 대부분이었으며, 애플리케이션 들은 데이터 센터 내부에 존재

# 변화 – 그리고, 숙제



애플리케이션이 비즈니스의 주요 수단이 되다.

운영 환경의 변화 - Cloud

Open Source 및 3<sup>rd</sup> Party Source 사용의 증가

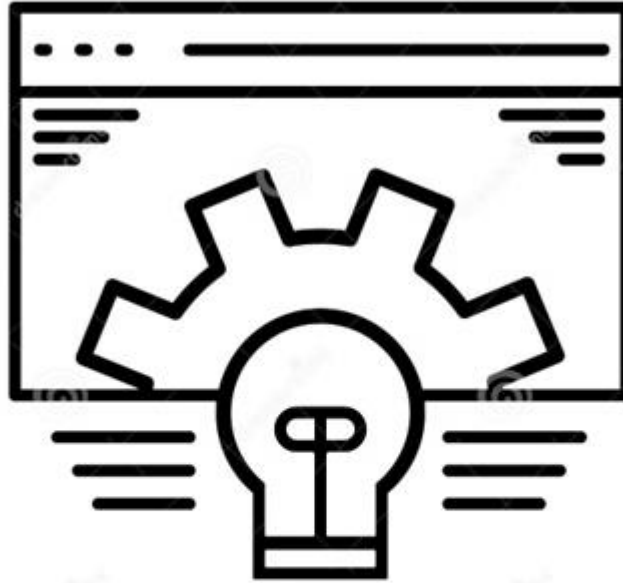
API 및 내/외부 애플리케이션간의 통신 증가

내부에서 시작되는 공격의 증가 – Supply Chain Attack

애플리케이션의 변경 주기의 단축 - Agile

자동화 요구의 증가 - DevOps

애플리케이션 취약점 공격 :  
전체 공격이 **80%** 이상



다크웹 해킹 의뢰 게시물중  
웹사이트 해킹 관련 **69%**


1일 평균 해킹 당하는  
웹사이트의 숫자 :  
**50,000**

# WEB APPLICATION





# 변화 – 애플리케이션 구성 요소

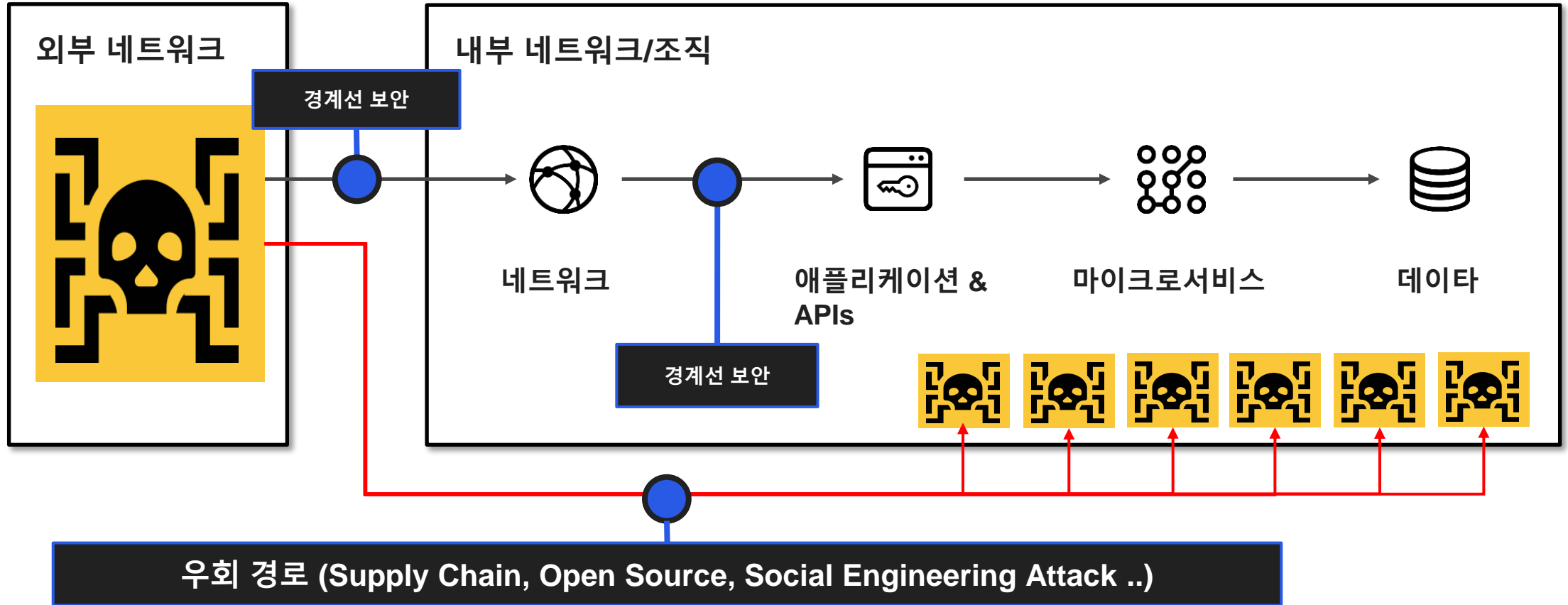
An iceberg floating in the ocean. The tip of the iceberg is above the water line, while the much larger, jagged base is submerged underwater. This visual metaphor represents the concept that most of an application's components are not owned by the user.

Operating Systems  
Containers  
Virtual Machines  
Application Runtimes  
Application Servers  
Databases  
Open Source Components

## Most of your software isn't yours

Application은 Open Source를 비롯한 다양한 요소들의 집합체이기 때문에 자신이 개발한 코드에 대하여 철저하게 Secure coding tool 등을 이용하여 검사를 한다고 하여도 한계가 존재 합니다.

# 변화 – 아무도 믿지 말라!



- 상기 그림에서와 같이 현대적인 공격들은 경계선 보안 제품이 설치된 (방어망이 구축된) 구간을 우회하여 내부로 침입하게 됩니다.
- 그리고 공격의 소스는 외부이지만 공격은 내부에서 시작되게 됩니다. (Cyber Kill Chain의 관점에서 보면 Delivery 이후에 공격이 시작 됨)
- 내부로 부터 시작되는 공격을 방어하기 위해서는 새로운 접근법을 기반으로 한 제품(Solution)이 요구 됩니다.

# RASP의 등장

Application이 스스로를 보호하다.

Gartner Research

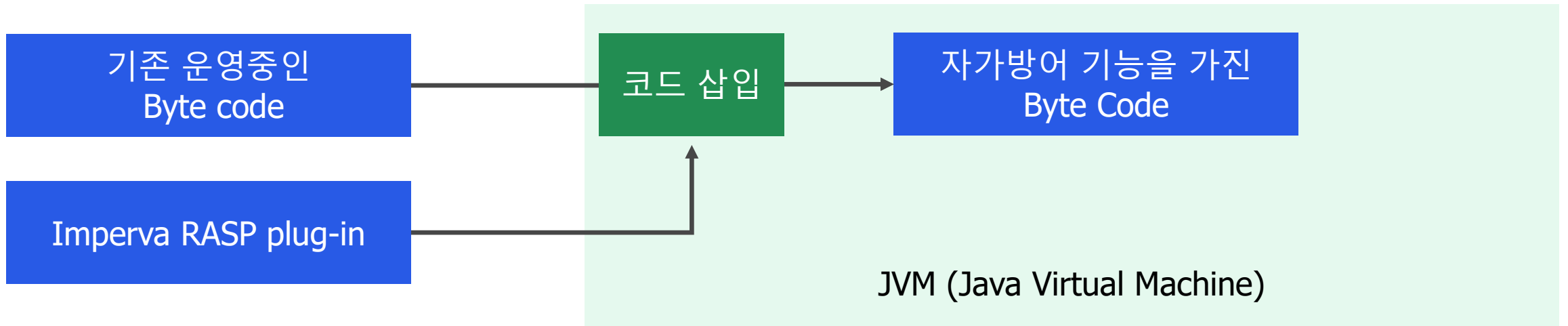
**Maverick\* Research: Stop Protecting Your Apps; It's Time for Apps to Protect Themselves**

애플리케이션을 외부에서 보호하는 방법에서  
애플리케이션이 스스로를 보호하게 하는 방법으로의 전환

# RASP (런타임 애플리케이션 자가 방어)



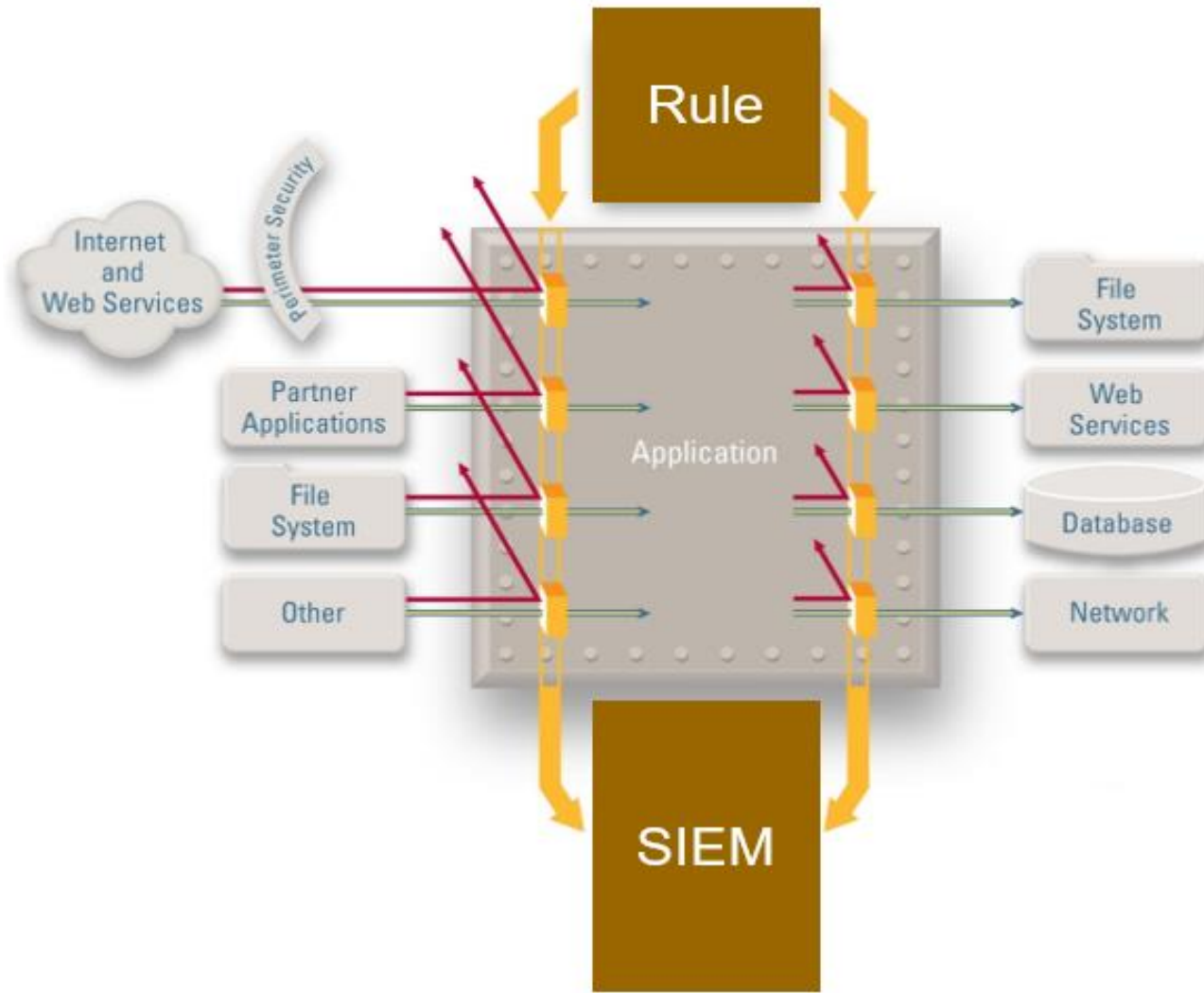
# RASP 작동원리 (BCI : Byte Code Instrumentation / 바이트코드 삽입)



- **Bytecode** (일종의 기계어/머신코드)
- 소스코드의 수정 없이 소스코드가 **Bytecode**로 변환되는 과정에서 **Bytecode**상에 원하는 기능을 삽입하는 기술
- **WAS** 모니터링 솔루션 툴(**Jennifer, Performizer** 등) 및 **RASP**에서 사용



# RASP 작동원리



소스코드 변경 없음

애플리케이션 내부에서 동작

전체 입력값 및 출력값에 대한 모니터링

East-West 탐지

Zero-day 공격 탐지

# RASP – NIST SP 문서

(17) SOFTWARE, FIRMWARE, AND INFORMATION INTEGRITY | [RUNTIME APPLICATION SELF-PROTECTION](#)

**Implement [Assignment: organization-defined controls] for application self-protection at runtime.**

Discussion: Runtime application self-protection employs runtime instrumentation to detect and block the exploitation of software vulnerabilities by taking advantage of information from the software in execution. Runtime exploit prevention differs from traditional perimeter-based protections such as guards and firewalls which can only detect and block attacks by using network information without contextual awareness. Runtime application self-protection technology can reduce the susceptibility of software to attacks by monitoring its inputs and blocking those inputs that could allow attacks. It can also help protect the runtime environment from unwanted changes and tampering. When a threat is detected, runtime application self-protection technology can prevent exploitation and take other actions (e.g., sending a warning message to the user, terminating the user's session, terminating the application, or sending an alert to organizational personnel). Runtime application self-protection solutions can be deployed in either a monitor or protection mode.

Related Controls: [SI-16](#).

**RASP는 Runtime Instrumentation 기술을 사용하여 실행중인 소프트웨어의 취약점을 이용한 공격을 탐지 차단한다.**

**RASP는 애플리케이션에 대한 이해 없이 네트워크정보 등을 이용하여 공격을 탐지하는 경계선보안과는 다르다.**

**RASP는 애플리케이션에 유해를 가할 수 있는 입력값을 모니터링하고 차단하여 애플리케이션 취약성을 경감 시킨다.**

**RASP는 악의적인 변경으로부터 런타임 환경을 보호하고 사용자 세션종료, 안내메시지 보내기 등의 기능을 수행한다.**



# RASP – 전문가들의 의견

protection alternatives. Organizations that have already deployed a WAF but find that attackers are **bypassing** it or experiencing too many **false positives** should also consider **RASP** solutions to augment their protection portfolios.

WAF를 도입하였지만 공격이 잘 탐지되지 않는다고 느끼거나 많은 오탐을 경험하고 있다면 RASP를 ...

(SANS Study : RASP vs WAF)

Year after year, attackers target application-level vulnerabilities. To address these vulnerabilities, application security teams have increasingly focused on shifting left - identifying and fixing vulnerabilities earlier in the software development life cycle. However, at the same time, development and operations teams have been accelerating the pace of software release, moving towards continuous delivery. As software is released more frequently, **gaps** remain in test coverage leading to the introduction of vulnerabilities in production. To prevent these vulnerabilities from being exploited, it is **necessary that applications become self-defending**. **RASP** is a means to quickly make both new and legacy applications self-defending. However, because most applications are

소프트웨어 릴리스 주기가 짧아 지면서 SAST로는 부족하며 Application 자가 방어 기능을 부여하여야 한다.

(SANS Study : Effectiveness of a RASP)

# RASP – OWASP 2017 RC1

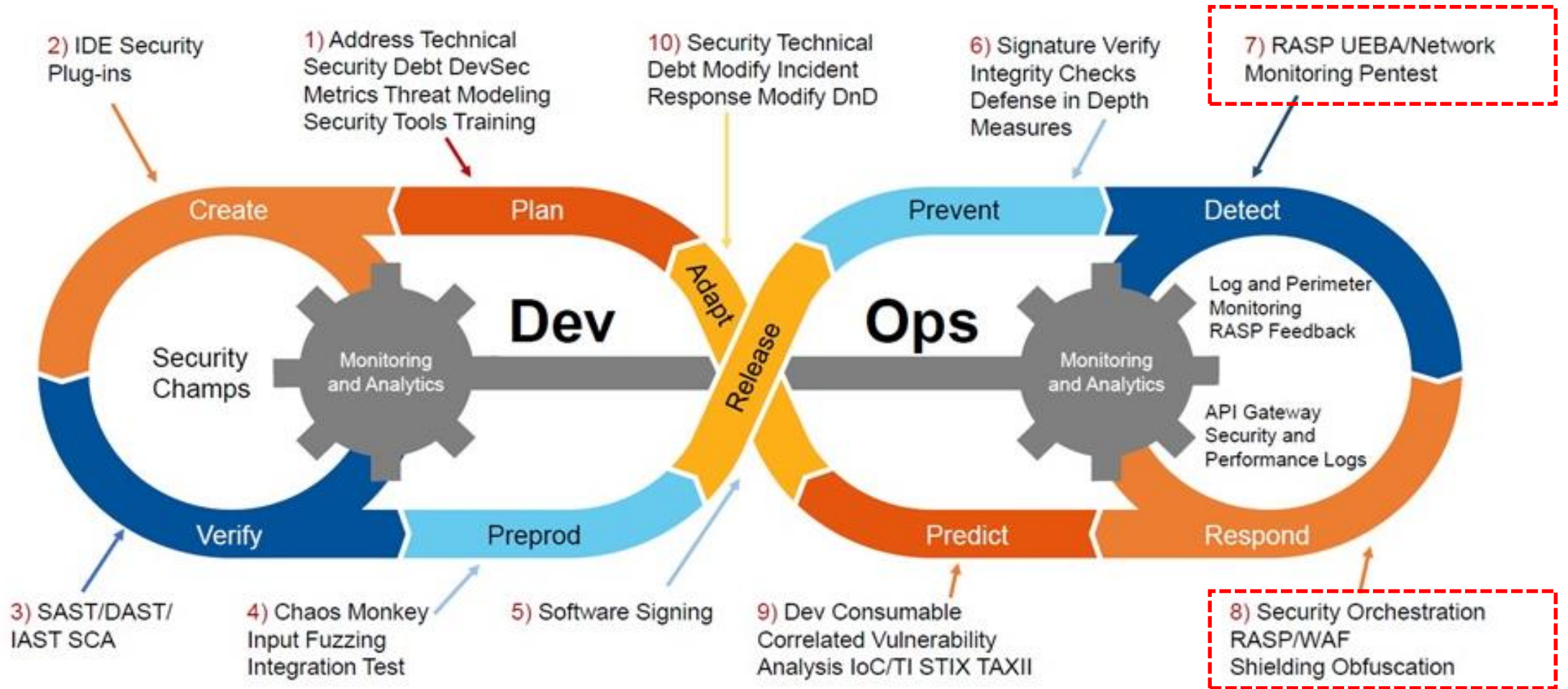


OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

“You can use technologies like WAFs, **RASP**, and OWASP AppSensor to detect or block attacks, and/or virtually patch vulnerabilities.”

비록 **RC2(Release Candidate 2)**가 **OWASP 2017** 정식 **Release**로 채택 되었지만, **OWASP**는 **RC1**에서 **Insufficient Attack Protection**을 이야기하면서 **RASP** 등의 사용을 권장하였습니다.

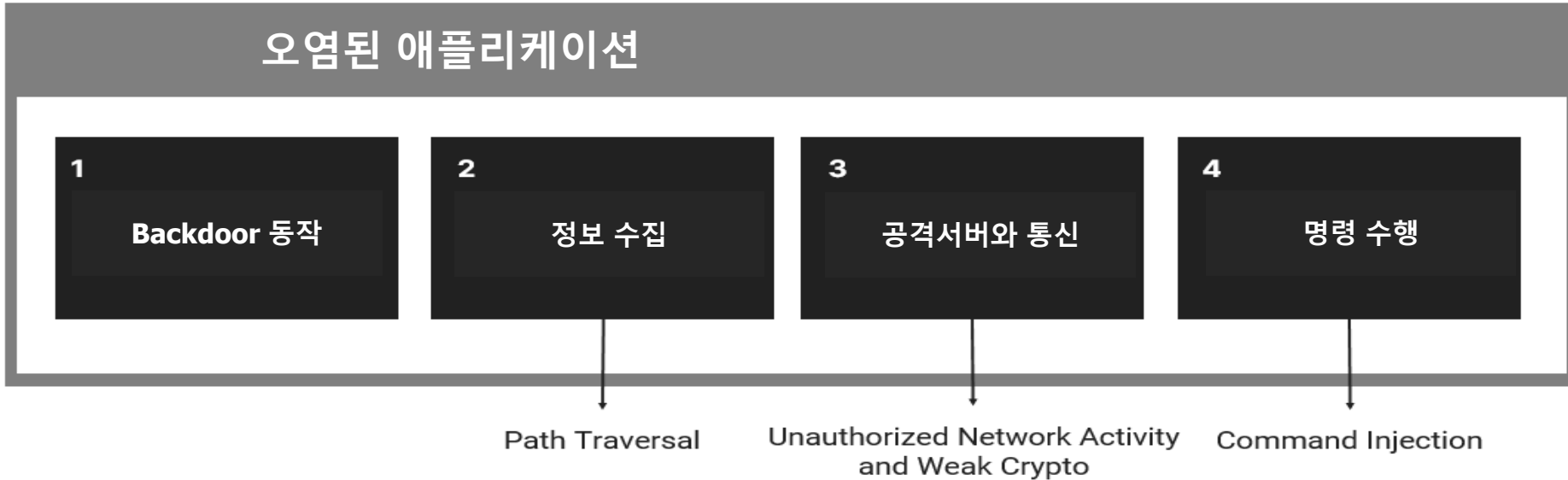
# RASP - DevSecOps



# RASP, WAF 비교

주요기능	WAF	RASP	Comment
Cloud를 포함한 다양한 환경 지원	제한적 지원	지원	
페일 오픈(Fail-Open)	과부하 시에 발생	없음	
North – South 위협 대응	지원	지원	
East – West 위협 대응(내부 애플리케이션 통신간 위협 대응)	미지원	지원	
DevOps 프로세스를 통한 자동화	미지원	지원	DevOps
공급망 공격(Supply Chain 공격 대응) malicious code like SolarWinds/Sunburst and 0-days like Struts 2 deserialization exploits	미지원	지원	
Serverless(Lambda) 보안위협 지원	제한적 지원	지원	
취약점 발생에 대한 포인팅(소스 코드, 데이터 흐름, 스택 추적)	미지원	지원	DevOps

# Backdoor & RASP



## 일반적인 Backdoor의 동작방식 및 RASP를 이용한 탐지

대부분의 백도어들은 유사한 동작방식을 가지고 있습니다. 일반적으로 Trigger를 작동시켜서 동작하게 되는데 SolarWinds의 경우에는 타이머를 트리거로 사용하였습니다. (SolarWinds의 경우 오염된 애플리케이션이 설치되고 약 2주후 트리거가 작동을 시작)

- 트리거 작동 후 백도어 코드는 파일 시스템을 검사하고 정보를 수집하게 되는데 보안적인 측면에서 Path Traversal(경로조작) 작업이 시행되게 됩니다. **(RASP -> Path Traversal 공격으로 탐지)**
- 이런 수집 과정을 통해 유용한 정보를 찾으면 외부서버와 통신을 연결하고 암호화된 메시지를 전송하게 됩니다. **(RASP -> Unauthorized Networks Activity and Weak Crypto 로 탐지)**
- 마지막으로 백도어는 공격자는 대상서버에 임의의 명령을 실행하게 됩니다. **(RASP -> Command Injection으로 탐지)**

# RASP 탐지 영역



## Attacks

- Clickjacking
- HTTP Response Splitting
- HTTP Method Tampering
- Large Requests
- Malformed Content-Types
- Path Traversal**
- Unvalidated Redirects

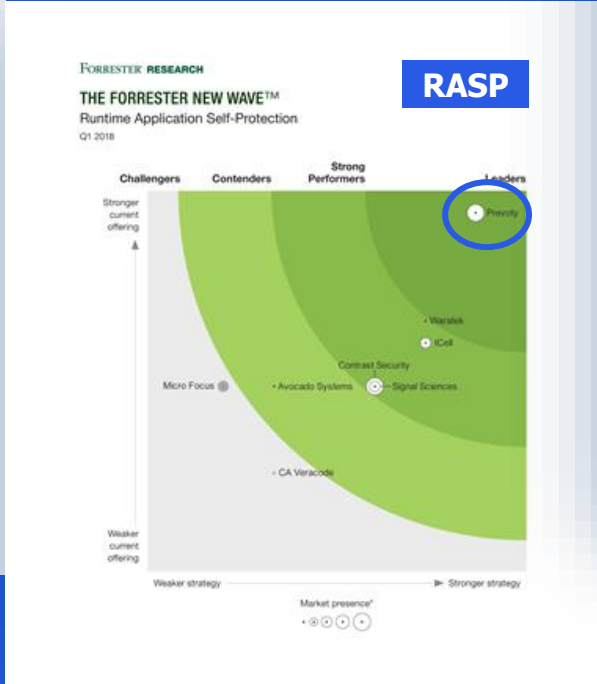
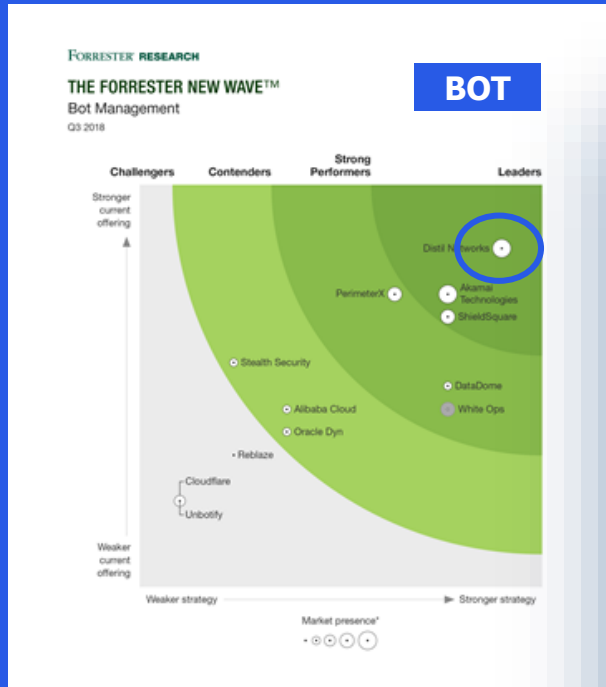
## Injections

- Command Injection**
- Cross-Site Scripting
- Cross-Site Request Forgery
- CSS & HTML Injection
- Database Access Violation
- JSON & XML Injection
- OGNL Injection
- SQL Injection

## Weaknesses

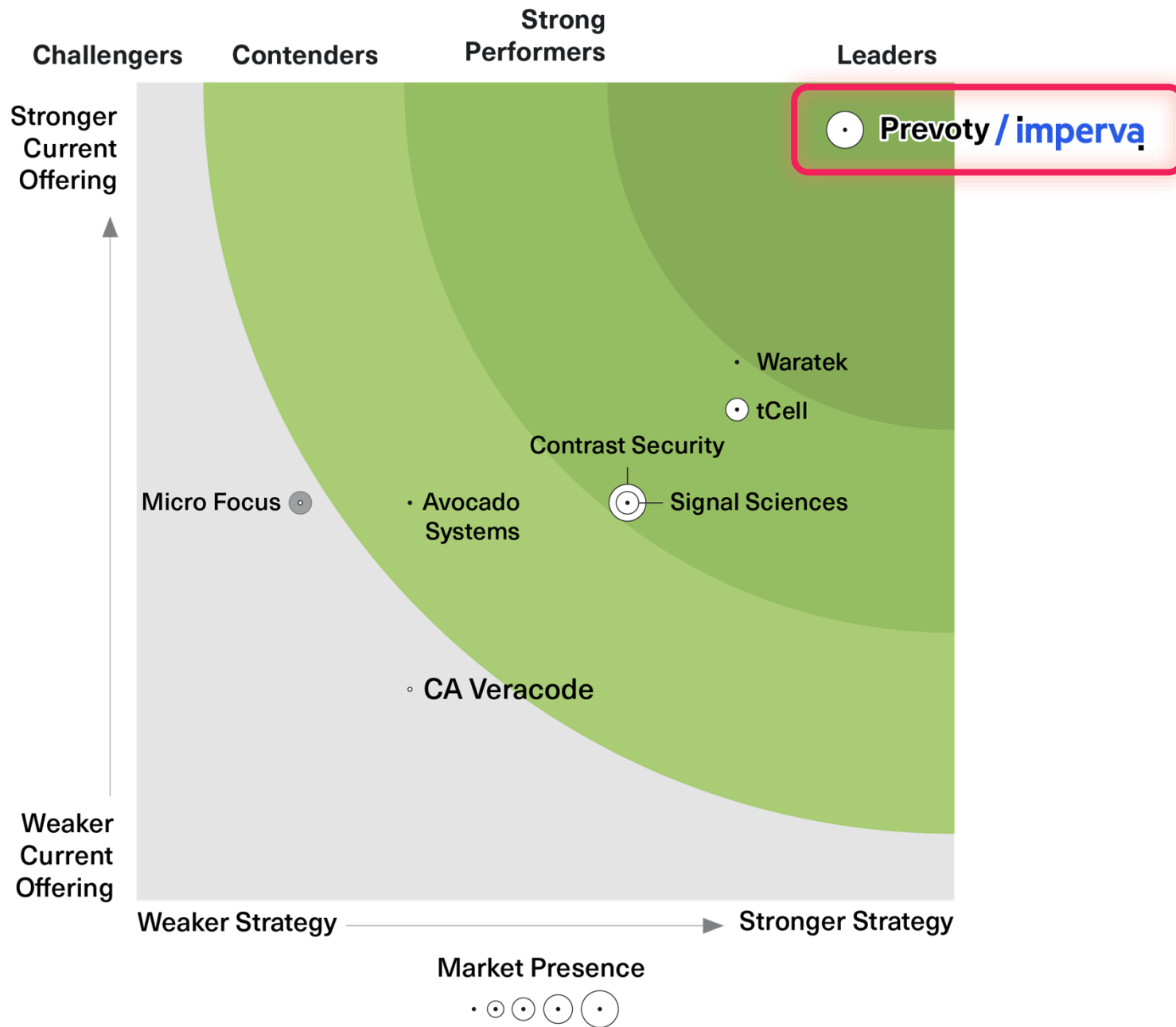
- Insecure Cookies
- Insecure Transport
- Logging Sensitive Information
- Unauthorized Network Activity**
- Uncaught Exceptions
- Vulnerable Dependencies
- Weak Authentication
- Weak Browser Caching
- Weak Cryptography**

# 애플리케이션 보안 부문 리더



임퍼바는 WAF/DDoS/BOT/RASP 등 주요 애플리케이션 보안 부문 리더 기업임  
RASP 솔루션은 빠르게 변화하는 개발 환경 및 운영 환경에 대응하고 기존 WAF의 부족한 부분을 보완하기 위한 핵심 솔루션





# Forrester Wave

Runtime Application Self-Protection

Q1 2018





Source: Gartner (September 2021)

# Gartner MQ

WAAP (Web Application & API Protection)

Sep 2021

# Case Study: 미국 금융사

## Problem

서비스 운영(production)환경에서 WebLogic 서버 600대의 심각한 취약성

## 고객의 요구 사항

서비스 영향없이 데이터를 보호할 수 있는 솔루션

알려진 취약성 및 알려지지 않은 취약성을 탐지하고 완화하는 비용 효율적인 솔루션

## Imperva RASP 도입 결과

애플리케이션에 대한 보호를 신속하게 구현하고, 향후 위협을 차단하며, WebLogic 서버를 완벽하게 보호

600+  
애플리케이션  
웹로직 서버  
보호

막대한 운영  
비용 절감

- 웹로직은 Oracle 에서 판매하는 애플리케이션 서버
- 대부분의 일반적인 소프트웨어 패키지와 마찬가지로 몇 가지 중요한 보안 취약성이 존재
- RASP는 WebLogic의 이러한 취약점 방어를 통하여 공격자의 행위 방어
- 해당 고객사의 경우 일련의 중요한 WebLogic 패치를 적용하지 않기 위해 RASP를 사용했으면 \$2M USD이상의 비용을 절감한 것으로 추산

# Case Study: Top 3 미국 통신사

## Problem

24개월 Struts 2 프레임워크에서 동작하는 애플리케이션을 보호하기 위하여 여러 번에 걸쳐 수천 대의 서버에 대한 긴급, 주기 외 패치 적용

운영 중단 및 상당한 비용 발생

## 고객의 요구 사항

Struts 2 프레임워크(및 다른 타사 코드)를 통한 제로데이 공격 위험을 완화  
긴급 패치 적용 및 관련된 시스템 중단으로 인해 발생하는 운영 비용 증가를 해소

## Imperva RASP 도입 결과

단일 릴리스 주기 동안 주요 매출 창출 웹 사이트에 Imperva RASP 구축

프로덕션 환경에서  
수천개의  
애플리케이션에  
RASP 적용

제로 데이  
위협으로부터  
보호,  
운영 비용  
절감

- 고객은 수백 개의 엔터프라이즈 웹 애플리케이션을 구축하는 데 사용한 공통 오픈 소스 패키지를 대상으로 제로데이 공격에 시달리고 있었음
- 이 문제가 매우 심각하여 CIO는 개발 프로세스 중에 모든 애플리케이션에 RASP를 적용하도록 의무화
- 서버에 대한 무단 액세스를 방지하고 관련 정리 비용을 절감하는 것이 가장 큰 이점

**Thank You**