AWS WAF 를 활용한
OWASP Top 10 웹애플리케이션
취약점 보완 방법

July 2017

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

이 문서는 정보제공을 목적으로 제공됩니다. 문서에 설명된 제품들과 방법들은 본 문서가 공개된 날짜를 기준으로 유효한 내용이며 추후 사전 공지 없이 변경될 수 있습니다. 이 문서에 나와있는 서비스나 설명 및 방법들에 대한 최종 판단은 고객 여러분의 개별적인 판단에 달려있으며, 이를 어떤 종류로든 보장하지는 않습니다. 이 문서에 대한 내용에 대해서 AWS 에서는 어떠한 보장도 제공하지 않습니다. AWS 와 고객 여러분 간의 법적인 책임은 AWS 와의 합의(Agreement)에 따르며 이 문서는 AWS 와 고객간의 맺는 합의(Agreement)에 해당하지 않으며 어떠한 영향도 주지 않습니다.

본 문서는 아래 문서의 번역본으로 약간의 오류 또는 잘못된 표현이 있을 수 있습니다. 보다 정확한 정보에 대해서는 아래 영어로 출판 된 원문을 참조하십시오.

원본 문서 : Use AWS WAF to Mitigate OWASP's Top 10 Web Application Vulnerabilities

한글번역: 임기성 보안 솔루션즈 아키텍트 - AWS Security Solutions Architect

목차

개요

AWS WAF 는 HTTP 프로토콜 수준에서 발생하는 다양한 공격 행위로부터 웹 사이트 나 웹 애플리케이션을 보호하는 웹 애플리케이션 방화벽입니다. 이 백서에서는 이 서비스를 사용하여, 가장 흔한 유형의 애플리케이션 보안 결함들을 다루는 Open Web Application Security Project(OWASP) Top 10 리스트에 기술되어 있는 애플리케이션의 취약점들을 완화하는 방법을 설명합니다 . 이 문서는 웹 사이트나 애플리케이션을 보호하고 보안 환경 및 그것의 가용성을 유지할 의무가 있는 사람을 대상으로 하고 있습니다.

서론

Open Web Application Security Project(OWASP)는 웹 애플리케이션 보안 분야에서 누구나 자유롭게 이용할 수 있는 기사, 방법론, 문서, 도구, 기술들을 발굴하는 온라인 커뮤니티입니다.[1] 그들은 OWASP Top 10[2] 으로 알려져 있는 가장 중요한 10 가지의 웹 애플리케이션 보안 결함 순위를 공개하고 있습니다. 현재 버전은 2013 년에 출판되었지만, 새로운 2017 릴리스의 후보 버전이 현재 공개되어 있습니다.

OWASP 상위 10 가지 요소는 각계에서 인정받고 있는 가장 중요한 웹 애플리케이션의 보안 결함을 담고 있습니다. 이것은, 웹 애플리케이션의 보안 수준을 평가하여 웹 사이트나 웹 기반의 애플리케이션에 대한 보완 전략을 구축하려는 목적으로 널리 통용되는 방법이기도 합니다. 또한 웹 애플리케이션이 공격에 취약할 수 있는 상위 10 개 영역을 선정하고, 그러한 영역들에서 발견되는 공통 취약점에 대해 설명하고 있습니다.

웹 사이트나 웹 기반 애플리케이션의 보안 수준을 향상시키고자 하는 어떤 프로젝트라도, OWASP Top 10 을 이해하고 그것을 자신의 워크로드에 어떻게 적용할 수 있는 지를 이해하는 것은 매우 좋은 시도입니다. 이를 통해 효과적인 완화 전략을 적용하는 데 많은 도움을 받을 수 있게 됩니다.

AWS WAF 는 애플리케이션 가용성에 영향을 주거나 보안을 침해하고 과도한 리소스를 소비하게 만드는 일반적인 형태의 웹 공격으로부터 웹 애플리케이션을 보호하기위한 웹 애플리케이션 방화벽 (WAF)입니다.[3] AWS WAF 를 사용하면 사용자가 직접 웹 보안 규칙을 정의하여 웹

애플리케이션으로의 요청에 대해 허용하거나 거부할 수 있습니다. 또한 AWS WAF 를 사용하여 일반적인 공격 패턴이나 여러분의 애플리케이션을 타겟으로 한 특정 공격 패턴을 차단하는 규칙을 만들 수 있습니다.

AWS WAF 는 AWS 의 글로벌 Content Delivery Network (CDN) 서비스인 Amazon CloudFron[4] 및 Elastic Load Balancing[5] 의 옵션인 Application Load Balancer 에서 작동합니다. 이들을 함께 이용하여, 유입되는 HTTP 요청을 분석하고 일련의 규칙을 적용하거나, 이 규칙에서 탐지된 대로 대응을 할 수 있습니다.

AWS WAF 은 HTTP 요청에서 흔하게 감지될 수 있는 패턴을 사용하는 OWASP Top 10 이나 혹은 기타 다른 형태의 웹 애플리케이션 보안 취약점을 완화하는 데 도움이 됩니다

여러분들은 탐지 할 수 있는 패턴을 검사하는 규칙을 만들고 이에 해당하는 요청들이 여러분들의 워크로드에 도달하지 않도록 할 수 있습니다.

그러나 웹 애플리케이션 방화벽을 사용하더라도 웹 애플리케이션의 근저에 있는 결함은 수정할 수 없다는 것을 이해하는 것이 중요합니다. 이를 위해 추가적인 보호 계층을 제공하여 악용 되는 위험을 줄일 필요가 있습니다. 이것은 소프트웨어 개발 싸이클이 점점 더 빨라지고 있는 현대적인 개발 환경에서 특히 중요합니다.

웹 애플리케이션 취약점 완화

2017 년 4 월 OWASP 는 10 가지의 새로운 범주를 공개했습니다. 새롭게 제안된 Top10 에 나열된 범주는 2013 년 Top10 이나 그 이전 버전에서 제기한 애플리케이션의 결함 카테고리들을 많이 포함하고 있습니다:

A1	인젝션
A2	훼손된 인증 및 세션 관리
A3	크로스 사이트 스크립팅(XSS)
A4	훼손된 접근제어(신규)
A5	잘못된 보안 설정
A6	민감한 데이터 노출
A7	불충분한 공격 방어(신규)
A8	크로스 사이트 요청 위조(CSRF)
A9	알려진 취약점을 갖는 구성 요소 사용
A10	API 에 대한 미흡한 보호(신규)

새로운 A4 카테고리는 2013 년 Top10 의 *'안전하지 않은 직접 객체 참조(Insecure Direct Object References)*와 *'기능 수준 액세스 제어의 미적용(Missing Function Level Access Controls)'*를 통합한 것입니다. 이전 A10 카테고리의 *'리다이렉션과 포워드에 대한 미검증(Unvalidated Redirects and Forwards)'* 는 API 보안에 초점을 맞춘 새로운 카테고리로 옮겨졌습니다. 본 백서에서는 이전 카테고리와 새로운 카테고리 모두에 대해 논의하겠습니다.

AWS WAF 를 도입하여 위의 카테고리 중 많은 대표적인 공격 벡터들을 효과적으로 완화할 수 있습니다. 물론 다른 카테고리들에서도 유효합니다. 그러나 그 효과는 보호되는 개별 워크로드 별로 다를 수 있고, HTTP 요청의 패턴을 얼마나 정확히 인식하는 지에 따라 좌우됩니다. 공격 기법과 악용하는

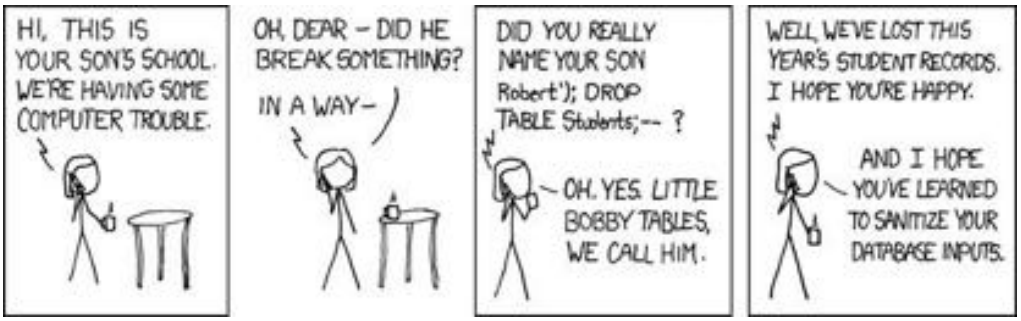
사례가 지속적으로 발전하고 있음을 감안할 때, 어떤 단일 웹 애플리케이션 방화벽도 이러한 범주에 포함되는 취약점들을 이용한 모든 가능한 시나리오들을 완벽하게 완화할 수 있는 것은 없습니다.

이 문서에서는 애플리케이션의 취약점을 완화하기 위해 쉽게 구현할 수 각 카테고리의 권장 사항에 대해 설명합니다. 이 백서의 마지막 부분에서는, 여기에서 설명한 일반적인 완화 기법의 일부를 구현하고 있는 AWS CloudFormation 템플릿 샘플을 다운로드 할 수 있습니다. 그러나 이러한 규칙을 특정 웹 애플리케이션에 곧바로 적용할 수 있는지에 대한 부분에는 다양한 경우가 존재한다고 말씀드릴 수 있습니다.

A1 – 인젝션

인젝션 취약점은 애플리케이션이 신뢰할 수 없는 데이터를 인터프리터에 보낼 때 발생합니다.[6] 인터프리터는 대부분 자신의 도메인 특정 언어를 가지고 있습니다. 그 언어를 사용하여 불순한 데이터를 인터프리터의 요청에 삽입함으로써, 공격자는 요청의 의도를 변경하고 예기치 않은 동작을 일으킬 수 있습니다.

아마도 가장 잘 알려진 광범위한 인젝션 결함은 **SQL 인젝션 결함**입니다. 이들은 입력값이 적절하게 필터링되지 않은 채, SQL 문에 직접 삽입 된 경우에 발생합니다. 입력 값 자체에 SQL 구문이 포함되어 있는 경우 데이터베이스 쿼리 엔진은 그것을 그대로 실행합니다. 이렇게 하면 원래 의도했던 것과는 다른 동작이 발생하고 그 결과 위험한 상태가 될 수 있습니다.



Credit: XKCD: Exploits of a Mom, published by permission.

AWS WAF 를 이용하여 완화하는 방법

SQL 인젝션 공격은 일반적인 시나리오에서 비교적 쉽게 탐지할 수 있습니다. 이 공격은 보통의 경우, HTTP 요청의 각 구성 요소들에서 잠재적으로 유효한 SQL 쿼리를 표시하기 위한 SQL 키워드를 파악하는 방식으로 감지됩니다. 그러나 더 복잡하고 위험한 변종은 백엔드에서 애플리케이션이 어떻게 구성되어 있는 지에 대한 사전 정보를 바탕으로, 여러 개의 입력 매개 변수 또는 HTTP 요청의 여러 구성 요소들에 악성 쿼리 (및 관련 키워드)를 삽입시킬 수 있습니다. 이들은 WAF 만 가지고는 대응하는 것이 어려울 수 있으며, 애플리케이션 레벨에서 다룰 필요가 있을 수도 있습니다.

AWS WAF 에는 SQL 인젝션 공격에 대응하고 완화하는 기능이 내장되어 있습니다. SQL 인젝션 탐지 조건을 사용하여 이러한 공격을 완화하기위한 규칙을 배포 할 수 있습니다.[7] 다음 표는 이를 위한 전형적인 조건 설정을 보여줍니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	설명
QUERY_STRING	URL_DECODE, HTML_ENTITY_DECODE	탐지하기 위한 **가장 일반적인** 구성 요소입니다. 쿼리 문자열 매개 변수는 데이터베이스 조회에서 자주 사용됩니다.
URI	URL_DECODE, HTML_ENTITY_DECODE	애플리케이션이 인코딩 변환(Dirified) URL 을 사용하는 경우 매개 변수는 URL 경로 세그먼트의 일부로 표시됩니다 - 쿼리 문자열이 아님(나중에 서버 상에서 재작성 됨.) 예를 들면: https://example.com/products/<product_id>/reviews/
BODY	URL_DECODE, HTML_ENTITY_DECODE	애플리케이션이 Form 입력을 이용한다면, **전형적인** 탐지 구성 요소 입니다. 단, AWS WAF 는 BODY 내용의 첫 부분 8KB 만 검사합니다.
HEADER: Cookie	URL_DECODE, HTML_ENTITY_DECODE	**덜 일반적인** 탐지 요소입니다. 그러나 애플리케이션이 데이터베이스 조회에서 Cookie 기반의 매개 변수를 사용하는 경우,이 구성 요소의 점검도 고려하십시오.
HEADER: Authorization	URL_DECODE, HTML_ENTITY_DECODE	**덜 일반적인** 탐지 요소입니다. 그러나 애플리케이션에서 데이터베이스의 검증을 위해 헤더 값을 사용하는 경우,이 구성 요소의 점검도 고려하십시오.

또한 애플리케이션이 데이터베이스 Look Up 을 위해 매개 변수로 사용하는 커스텀 요청 헤더의 다른 구성 요소도 고려하십시오. SQL 인젝션의 탐지 조건에서 이러한 구성 요소들을 점검할 수 있습니다.

다른 고려 사항들

이 탐지 패턴은 만약 여러분의 워크로드에서 사용자가 SQL 쿼리를 포함하여 요청을 전송하는 것을 허용하도록 설정되어 있는 경우라면, 그다지 효과적이지 않습니다. 이를 위해선 이런 형태의 입력이 유효하도록 특정 URL 패턴에서는 SQL 인젝션 탐지 규칙을 무시하게 하는 예외 조건의 범위를 엄격하게 좁히는 것을 고려하실 필요가 있습니다. 위의 표에서 설명한 바와 같이 SQL 인젝션 탐지 조건을 사용하고, 이와 동시에 문자열 탐지 조건을 사용하여 검사에서 제외 시킬 URL 패턴을 목록화 함으로써 구현 할 수 있습니다: [8]

Rule - action:**BLOCK**

when request matches **SQL Injection MatchCondition**

and request does not match **String Match Condition** for excluded Uniform Resource Identifiers (URI)

문자열 탐지 조건을 사용하게 되면, 지정된 키워드와 일치하는 지를 따져서 정당한 입력값이 아님을 탐지하는 방식으로, 다른 도메인의 특정 언어에 대한 또 다른 형태의 Injection 공격을 완화 할 수 있습니다.

A2 – 훼손된 인증 및 세션 관리

웹 애플리케이션의 인증 및 세션 관리 메커니즘을 구현할 때 생긴 결함은 의도하지 않은 데이터 노출, 자격증명이나 세션의 도용, 정당한 사용자로 위장하는 형태로 이어질 수 있습니다.[9] 그리고 이러한 결함을 웹 방화벽을 사용하여 해결하는 것은 어렵습니다.

대체로 공격자는 클라이언트와 서버 사이의 통신이 구현되는 과정에서

포함된 취약점을 이용합니다. 또는 이러한 자격 증명을 얻기 위해 애플리케이션에서 세션 토큰 이나 인증 토큰을 생성, 저장, 전송, 재사용, 시간 초과 또는 비활성화 시키는 상황을 공략하기도 합니다. 자격 증명을 취득한 후, 공격자는 그 토큰을 사용하여 합법적인 사용자로 가장한 채, 웹 애플리케이션에 요청을 보냅니다.

예를 들어, 공격자가 웹 클라이언트와 RESTful API 사이의 통신을 허용하는 JWT 토큰을 취득한 경우, 부정하게 취득 된 인증 토큰을 사용하여 HTTP 요청을 시작하면 토큰이 만료 될 때까지 정상 사용자로 위장 할 수 있습니다.[10]

AWS WAF 를 이용하여 완화하는 방법

이미 승인 되었던 자격 증명 세션 또는 토큰을 훔쳐서 행해진 악의적인 요청은 정당한 요청과 구별하는 것이 어렵기 때문에, AWS WAF 는 다음과 같이 제한된 형태로 역할을 수행합니다.

여러분들이 애플리케이션 보안 기능에서 토큰이 도난당한 것을 감지하게 되면, 그 토큰을 블랙리스트 AWS WAF 규칙에 추가 할 수 있습니다. 이 규칙은 영구적으로 또는 토큰이 만료될 때까지, 지정된 패턴을 이용하여 도난 당한 토큰을 이용한 추가 요청을 차단하게 됩니다. 여러분들은 이런 식의 대응을 자동화하여 조치 시간을 단축시킬 수 있습니다. AWS WAF 는 서비스와 상호 작용하기위한 API 를 제공합니다.[11] 이를 이용하여, 여러분들은 현재 운영하고 있는 인프라 또는 애플리케이션 전용의 모니터링/로깅 도구와 연동하여 이런 불법적인 패턴을 찾고 바로 대응할 수 있습니다. AWS WAF 규칙의 자동화에 대해서는 "A7 - 불충분한 공격 방어"에 자세히 설명되어 있습니다.

차단해야 될 블랙리스트를 작성하려면 **문자열 탐지 조건**을 사용하게

됩니다. 다음 표는 이런 패턴의 사례를 보여줍니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	연관된 조건	검색할 값
QUERY_STRING	브라우저의 주소 표시 줄 또는 서버 로그에 표시되어 쉽게 얻을 수 있기에, 세션 토큰은 URI 또는 QUERY_STRING 에 포함하지 않도록 합니다.		
URI			
HEADER: Cookie	URL_DECODE, HTML_ENTITY_DECODE	CONTAINS	Session ID 또는 access tokens.
HEADER: Authorization	URL_DECODE, HTML_ENTITY_DECODE	CONTAINS	JWT token 또는 다른 전달자의 authorization tokens.

여러분은 다양한 메커니즘을 사용하여 유출된 또는 부정 사용된 세션 토큰이나 인증 토큰을 탐지할 수 있습니다. 그 중 한가지는 사용자가 평상시, 애플리케이션에 액세스하는 위치와 사용하는 클라이언트 장치를 추적하는 것입니다. 이를 통해 동일한 토큰을 가지고 완전히 다른 위치나 또는 다른 클라이언트 장치로부터 요청이 들어 왔는지 여부를 신속하게 감지하고 안전하게 그들을 블랙리스트에 등록 할 수 있습니다.

AWS WAF 는 요청율에 기반한 규칙도 지원하고 있습니다. 요청율 기반 규칙은 IP 주소 별 요청율이 여러분이 정의한 임계값 (5 분 동안의 요청수 총합)을 초과하면 더 이상 요청을 받지 않고 차단합니다. 이러한 규칙은 AWS WAF 에서 사용할 수 있는 다른 규칙(조건)과 결합 할 수 있습니다. 요청율 기반의 조건을 적용하여 여러분 애플리케이션의 인증 또는 인가 URL 과 앤드포인트를 Brute-Force 공격으로부터 보호할 수 있습니다. 문자열 탐지 조건을 사용하여 애플리케이션의 인증 URI 경로를 검사할 수도 있습니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	연관된 조건	검색할 값
URI	URL_DECODE, HTML_ENTITY_DECODE	STARTS_WITH	/login *(or relevant application- specific URLs)*

이 조건은 IP 주소 별로 전송한 요청들을 대상으로 요청율 기반의 임계값이 설정된 규칙을 통해 체크할 때 사용됩니다:

Rule - action: **BLOCK;** rate limit: **10**; rate key: **IP**

위 사례는 주어진 문자열 탐지 조건에 일치하는 요청만 집계됩니다. 그 횟수가 5 분 간격으로 10 건을 초과하면 원본 IP 주소가 차단됩니다.

A3 – 크로스 사이트 스크립팅(XSS)

크로스 사이트 스크립팅 (XSS) 취약점은 웹 애플리케이션이, 브라우저로 전송된 웹 페이지에서 사용자가 제출한 데이터를 포함하여 다시 회신될 때, 적절한 방역 조치없이 서버로 전달되도록 제작된 경우에 발생합니다.[12] 제출된 데이터가 제대로 검증되지 않거나 제거되지 않는 경우, 공격자는 이러한 통로를 이용하여 인라인 프레임 또는 의도된 페이지로 유도(리플렉션)하는 객체를 삽입할 수 있습니다. 이를 악용하여 예를 들면, 시스템 악성 코드를 설치하기 위해 키로거를 사용하여 사용자의 자격 증명을 탈취하는 등 다양한 악의적인 목적으로 사용할 수 있습니다. 조작된 사용자 데이터가 서버 측의 데이터 저장소에 저장된 후, 다수의 다른 사용자에게 전달되면 공격의 영향이 확대될 수 있습니다.

사용자의 의견을 받는 인기있는 일반적인 블로그의 예를 생각해 봅시다. 사용자의 의견이 제대로 방역 조치 되지 않는 경우, 악의적인 사용자는 댓글에 악성 스크립트를 삽입 할 수 있습니다. 예를 들어:

```
<script src="https://malicious-site.com/exploit.js"
type="text/javascript" />
```

정당한 사용자가 블로그 게시물을 로드하면 위의 코드가 실행됩니다.

AWS WAF 를 이용하여 완화하는 방법

XSS 공격은 HTTP 요청에서 특정 주요 HTML 태그가 필요 하기 때문에, 일반적으로는 공격을 완화하는 것이 비교적 간단합니다.

AWS WAF 는 XSS 공격을 탐지하고 완화하는 기능이 포함되어 있습니다. 크로스 사이트 스크립팅의 탐지 조건을 사용하여 이러한 공격을 완화하기위한 규칙을 배포 할 수 있습니다.[13] 다음 표를 통해 전형적인 조건 설정 케이스들을 확인할 수 있습니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	설명
BODY	URL_DECODE, HTML_ENTITY_DECODE	애플리케이션이 폼 입력을 받아 들일 경우 **매우 일반적인 구성 요소**입니다. AWS WAF 는 본문의 첫 번째 8 KB 만을 체크합니다.

QUERY_STRING	URL_DECODE, HTML_ENTITY_DECODE	쿼리 문자열 매개 변수가 웹 페이지에 반영되는 경우에 **권장됩니다**. 일례로 페이지 나누기 목록에 있는 현재 페이지 번호를 들 수 있습니다.
HEADER: Cookie	URL_DECODE, HTML_ENTITY_DECODE	애플리케이션이 웹 페이지에 반영된 Cookie 기반의 매개 변수를 사용하는 경우에 **권장됩니다**. 예를 들어 현재 로그인 한 사용자의 이름은 Cookie 에 저장된 페이지 헤더에 포함됩니다.
URI	URL_DECODE, HTML_ENTITY_DECODE	**일반적이지는 않습니다**. 그러나 애플리케이션이 인코딩 변환(Dirified) URL 을 사용하는 경우 매개 변수는 쿼리 문자열이 아니라 URL 경로 세그먼트의 일부로 표시됩니다.(이 부분은 나중에 서버 측에 다시 쓰여집니다). 쿼리 문자열과 유사한 우려가 있습니다.

다른 고려 사항들

이 탐지 패턴은 콘텐츠 관리 시스템(CMS)[14] 의 편집기 등과 같이, Rich HTML 형태로 작성하여 제출하는 것을 허용하는 워크로드에는 덜 효과적입니다. 이 경우에는 유효하다고 인정한 특정 URL 패턴을 XSS 규칙에서 예외처리하는 범위를 좁혀서 엄격하게 적용할 것을 권고 드립니다. 더불어 예외처리되는 URL 패턴들에 대해서도 대체 통제 수단을 적용하여야 합니다. 또한 특정 이미지 혹은 사용자 정의 데이터 포맷과 그것들에 대한 탐지 규칙의 설정으로 인해, 오탐(False-Positive) 레벨이 상승할 수도 있습니다. HTML 구문에서 XSS 공격으로 보이는 패턴들이 특정 이미지 혹은 기타 데이터 포맷에서는 유효할 수도 있습니다. 예를 들어, SVG 그래픽 포맷[15] 은 <script> 태그를 허용하고 있습니다. HTML 요청에 다른 데이터 포맷이 포함되어있는 경우, XSS 규칙을 요청 내용의 유형에 맞게 조정할 필요가 있습니다.

A4 – 훼손된 접근 제어

이 유형의 애플리케이션 결함은 이번에 제안된 2017 년 Top 10 에서 새롭게 등장했으며, 인증 된 사용자에게 어떤 것을 허가할 지에 대한 통제가 부족하거나 부적절하게 통제되는 경우를 다루고 있습니다. 그것은 2013 년 Top 10 의 다음 2 개의 범주를 통합합니다: A4 – 안전하지 않은 직접 객체 참조(Insecure Direct Object References)와 A7 - 함수 수준의 접근 제어 미적용(Missing Function Level Access Controls).

이 유형의 애플리케이션 결함을 이용하면 요청자의 권한이 제대로 검증되지 않은 채, 내부 웹 애플리케이션의 개체를 조작 할 수 있습니다.[16] 더 나아가, 특정 워크로드에 따라서는 권한 없는 데이터를 공개하거나, 내부 웹 애플리케이션 상태를 조작하고, 작업 경로 탐색이나 파일을 포함시킬 수도 있습니다.

여러분의 애플리케이션에서는 자체적으로 사용되는 인증 및 인가 방식을 통해, 개별 모듈, 구성 요소 또는 기능에 대한 액세스를 적절하게 확인하고 제한해야 합니다. 기능 수준의 액세스 제어의 결함은 액세스 제어가 초기 시스템 개발 단계에서 구현되지 않고, 나중에 추가 된 경우에 가장 흔하게 발생합니다.[17]

이러한 결함은 경계 보안 접근법(Perimeter Security)을 접근제어 방법으로 적용하고 있는 애플리케이션에서도 발생합니다. 이런 경우 접근 가능 레벨의 확인은 요청의 초기 단계에서 한 번만 확인하게 됩니다. 따라서 그 요청에서 파생한 다양한 서브 루틴들이 호출될 때 마다 확인하는 과정은 수행되지 않습니다. 이 경우, 호출된 서브루틴 코드가 인가된 사용자 대신 다른 모듈, 구성 요소 또는 함수를 호출 할 수 있게 되는 암묵적인 신뢰관계가 만들어 지게

되는데, 이런 상황이 모든 경우에 안전한 것은 아닙니다.

여러분의 웹 애플리케이션이 액세스 수준 또는 구독 수준에 따라 사용자 별로 서로 다른 구성 요소를 게시하는 경우에는, 해당 기능이 호출 될 때마다 권한 체크를 수행해야합니다.

이해를 돕기 위해, 이와 같은 결함이 있는 다음의 사례들을 살펴 봅시다:

1. 인증된 사용자가 자신의 프로파일을 편집 할 수 있게 만들어 진 웹 애플리케이션에서 인증에 성공하면, 본인 프로파일을 편집할 수 있는 페이지에 대한 링크가 다음과 같이 생성된다고 합시다:

   https://example.com/edit/profile?user_id=**3324**

   그러나 그 프로파일 편집 페이지에서는 전달된 매개 변수가 현재 사용자와 일치하는지 여부는 별도로 확인하지 않는다고 합시다. 이렇게 되면, 로그인하는 모든 사용자는 단순히 파라메터로 전달되는 사용자 ID 를 임의로 변경하고 반복하여 다른 사용자에 대한 정보를 찾을 수 있게 됩니다. 이는 허용되지 않은 정보가 공개되는 결과가 됩니다:

   https://example.com/edit/profile?user_id=**3325**

2. 또 다른 예로 문서 공유 사이트에서 파일 다운로드를 표시하거나 허용하는 일종의 도우미 기능의 서버 사이드 스크립트를 살펴보겠습니다. 이것은 다음과 같이 파일 이름을 쿼리 문자열 매개

변수로 받아들입니다:

https://example.com/download.php?file=**mydocument.pdf**

스크립트의 어느 단계에서, 전달된 파라메터를 다음과 유사하게 파일을 읽어 들이는 함수로 전달할 것 입니다:

```
$content = file_get_contents("/documents/path/{$_GET[file]}");
```

서버 구성이 취약하고 별도의 검증이나 필터링 단계가 없다고 한다면, 전달된 파일 파라메터로 인해 다음과 같이 서버 상의 파일을 읽고 반영할 수 있게 됩니다:

https://example.com/download.php?file=
..%2F..%2Fetc%2Fpasswd

이것은 디렉토리 탐색 공격(directory traversal)[18]과 로컬 파일 포함(local file inclusion) 공격[19]의 두가지 예입니다.

3. 모델-뷰-컨트롤러 (MVC) 프레임 워크를 사용하는 애플리케이션이나 확장된 형태의 콘텐츠 관리 시스템에서는 아주 전형적인 패턴인 모듈형 웹 애플리케이션을 생각해 봅시다. 일반적인 경우, 애플리케이션에 대한 진입 지점은 공통 루틴 (인증 / 인가 등)을 처리한 후, 요청 파라메터에 따라 적절한 컨트롤러를 호출하는 라우터가 됩니다:

https://example.com/?module=**myprofile**&view=**display**

정당하게 인증된 사용자는 위의 URL 을 호출하여 본인의 프로파일을 볼 수 있을 것입니다. 당연히 악의적인 사용자도 자신을 인증한 뒤, 본인의 프로파일을 볼 수 있습니다. 그러나 그들은 아래와 같이 요청 URL 을 조작하여 관리 모듈을 호출하려고 할 수도 있습니다:

https://example.com/?module=**usermanagement**&view=**display**

이 경우, 호출되는 관리 모듈이 관리자에게 요구되는 권한을 확인하는 추가 검사를 수행하지 않는다면, 공격자는 시스템의 허가되지 않은 부분에 액세스 할 수 있습니다.

AWS WAF 를 이용하여 완화하는 방법

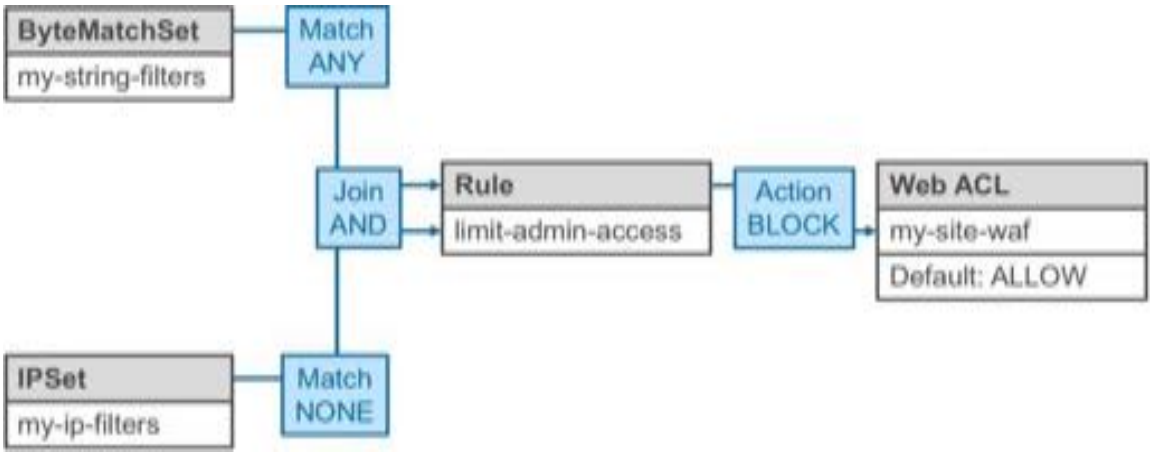
AWS WAF 를 사용하여 이 유형의 취약점을 이용한 특정 공격 방법을 완화 시킬 수 있습니다. 일반적으로 WAF 를 사용하여 권한 검증으로 인한 취약점을 완화하는 것은 어렵습니다. 이것은 잘못된 요청과 유효한 요청을 구별하는 근거가 사용자(요청자)의 세션과 권한의 컨텍스트에서 찾아 지거나, 드물게는 HTTP 요청 자체의 표현에서 찾을 수 밖에 없기 때문입니다. 그러나 만약 악의적인 HTTP 요청 안에, 정당한 요청에서는 포함되지 않을, 탐지 가능한 패턴이 포함되어 있다면, 그것을 탐지하는 WAF 규칙을 만들 수 있습니다.

또한 AWS WAF 를 사용하여 경로 탐색 시도나 원격 및 로컬 파일 포함 (RFI / LFI)을 나타내는 위험한 HTTP 요청 패턴을 필터링 할 수 있습니다.
아래의 표는 이러한 전형적인 조건의 일부를 보여줍니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	연관된 조건	검색할 값
QUERY_STRING	URL_DECODE, HTML_ENTITY_DECODE	CONTAINS	../, ://
URI	URL_DECODE, HTML_ENTITY_DECODE	CONTAINS	../, ://

또한 여러분의 애플리케이션이 파일 시스템 경로를 조합하거나 참조하는 데 활용하는 HTTP 요청의 또 다른 구성 요소들도 고려해야 합니다. 위의 카테고리에서 제안된 패턴과 마찬가지로, 애플리케이션이 URL 들이나 복잡한 파일 시스템 경로를 유효하다고 인정할 경우에는 별로 효과가 없을 수 있습니다.

관리를 위한 모듈, 컴포넌트, 플러그인 또는 그 기능에 대한 접근이 지정된 특권 사용자 집합으로 한정되어 있는 경우, 아래 그림처럼, 정해진 소스 IP 위치와 화이트리스팅 된 패턴을 가질 경우에만 접근이 허용되도록 하는 방식을 이용하여 중요 기능들에 대한 엑세스를 제한 시킬 수 있습니다:



다른 고려 사항들

만약 인가 정보가 HTTP 요청의 일부로 클라이언트에서 전송되고, JWT 토큰(또는 이와 유사한 것)을 사용하여 캡슐화되어 있는 경우, 여러분들은 요청된 모듈, 플러그인, 구성 요소 또는 기능과 이 정보를 비교하고 평가할 수 있습니다. AWS Lambda @ Edge 함수를 사용하여, HTTP 요청을 미리 확인하고, 관련 요청 파라메터가 전달된 인가 토큰의 유효성 및 허용된 권한 설정내용과 일치하는지 체크할 것을 권장드립니다.[20] 여러분들은 이런

방식으로 Lambda @ Edge 를 사용하여 부적합한 요청이 백엔드 서버에 도달하기 전에 차단할 수 있습니다.

A5 – 잘못된 보안 설정

서버의 매개 변수, 특히 보안에 영향을 미치는 매개 변수의 잘못된 설정은 애플리케이션 스택의 모든 레벨에서 발생할 수 있습니다.[21] 이것은 애플리케이션의 운영 체제, 미들웨어 플랫폼 서비스, 웹 서버 소프트웨어, 애플리케이션 코드 또는 데이터베이스 레이어에서 발견될 수 있습니다. 이들 구성 요소가 가지고 있는 디폴트 설정은 보안 권장 사항을 항상 준수한다고 볼 수 없으며, 모든 워크로드에 적합하지도 않습니다.

보안 설정이 잘못된 예를 몇 가지 들어 보겠습니다:

1. 실 운영 시스템에서 Apache 웹 서버 *ServerTokens Full*(기본값) 구성을 그대로 둡니다. 이렇게 하면 서버가 생성한 응답을 통해 웹 서버 및 관련 모듈의 정확한 버전이 노출됩니다. 공격자는 이 정보를 사용하여 서버 소프트웨어의 알려진 취약점을 식별할 수 있습니다.
2. 실 운영 웹 서버의 디폴트 디렉토리 조회 기능을 활성화된 상태로 둡니다. 이를 통해 악의적인 사용자가 웹 서버에서 호스팅 되는 파일들을 볼 수 있습니다.
3. 사용자에게 리턴되는 오류 메세지에 stack trace 를 포함하도록 구성된 애플리케이션 구성. 공격자는 적용된 소프트웨어 구성 요소를 잠재적으로 발견 할 수 있습니다. 그들은 여러분의 코드를 리버스 엔지니어링하고 결함을 발견 할 수도 있습니다.
4. PHP 의 이전 기능. 몇 년 전, PHP 의 기본 설정은 요청 파라메터(쿼리

문자열, Cookie 기반, POST 기반)를 전역 변수로 등록 할 수 있었습니다. 이후 이 기능은 폐지되어 완전히 제거되었습니다. 다음과 같이 PHP 의 취약한 버전을 이용하여, HTTP 요청을 통해 내부 서버 변수를 덮어 쓸 수 있었습니다:

http://example.com/**?_SERVER[DOCUMENT_ROOT]=http://bad.com/bad.htm**

이런 방식으로 사용자가 방문하는 사이트에 악성 사이트 주소를 포함시킬 수 있습니다.

AWS WAF 를 이용하여 완화하는 방법

악용하려고 하는 HTTP 요청 패턴이 탐지 가능하다면, AWS WAF 를 사용하여 다양한 방법으로, 서버의 설정 오류를 악용하려는 시도를 완화할 수 있습니다. 그러나 이러한 패턴은 애플리케이션 스택 고유의 것도 있습니다. 그들은 운영 체제, 웹 서버 프레임 워크 또는 코드로 활용하는 언어에 따라 달라집니다. 개별 스택에 국한되지 않는 범용적인 규칙을 적용하여 WAF 에서 먼저 차단함으로써, 백 엔드 서버에서 이런 부분들을 직접 처리할 필요가 없도록 만들기 때문에 웹서버의 부담을 더는데 도움이 됩니다.

여러분들이 사용할 수 있는 몇 가지 전략이 있습니다:

- 기본적으로 설치 또는 활성화 되어 있는 관리 콘솔의 구성 및 상태 페이지 경로에 대한 액세스를 차단해야 합니다. 또한 신뢰할 수 있는 소스 IP 주소를 접근제어에 적용 중인 경우, 그 밖의 액세스를 제한해야 합니다. 특정 기능을 사용했는지, 삭제했는 지와 무관하게 이렇게 해야합니다 (이후 필요하다면, 관련 기능은 다시 활성화하거나 재

설치할 수 있습니다).

- 여러분의 플랫폼을 알려진 공격 패턴으로 부터 보호합니다. 특히 플랫폼의 오래된 기능에 의존하는 애플리케이션이 있는 경우, 그러한 잘 알려진 공격 패턴으로 부터 보호해야 합니다. 예를 들어, PHP 를 사용하는 경우 "*_SERVER [*"가 포함 된 쿼리 문자열 요청을 차단할 수 있어야 합니다.

화이트리스트 규칙의 패턴은 바로 이전에 살펴본 훼손된 접근 제어(Broken Access Control) 범주에서 설명한 유형과 마찬가지로, WordPress 사이트의 관리 콘솔과 같은 특정 하위 기능을 화이트리스트에 등록하는 데 도움이 됩니다.

다른 고려 사항들

Amazon Inspector 를 도입하여 소프트웨어 설정을 확인하는 것도 검토하십시오.[22] 이것은 AWS 에 구축되는 애플리케이션의 보안 및 규정 준수를 향상시키는 자동화된 보안 평가 서비스입니다. Amazon Inspector 는 모범 사례를 벗어난 부분이나 취약점에 대해서 애플리케이션을 자동으로 평가합니다.

신속하게 시작할 수 있도록, Amazon Inspector 에는 일반적인 보안 모범 사례 및 취약성 정의(CVE)에 매핑된 수백 개의 규칙을 가진 지식 기반이 포함되어 있습니다. 빌트인 규칙의 사례로, 원격 root 로그인을 사용하거나 취약한 소프트웨어 버전을 설치했는지 검사하는 것 등을 들 수 있습니다. 이러한 규칙들은 AWS 의 보안 연구자에 의해 정기적으로 업데이트됩니다.

이와 같은 탐지 통제 외에도, 안전한 설정을 구현하고 유지함으로써 이 유형의 공격에 대한 가장 좋은 방어를 제공할 수 있습니다. CIS 벤치 마크 [23] 와 같은

설정 지침은 보안 설정을 배포하는 데 도움이 됩니다. 여러분들은 AWS Config[24] 와 Amazon EC2 Systems Manager[25] 와 같은 서비스를 사용하여 시간 순서에 따라 구성 변경을 추적하고 관리하는데 도움을 받을 수 있습니다.

A6 – 민감한 데이터의 노출

'민감한 데이터 노출' 애플리케이션 결함은 일반적으로는 웹 애플리케이션 방화벽을 사용하여 경감하기 어렵습니다.[26] 이러한 결함은 통상적으로 불충분하게 구현된 암호화 프로세스에 의해 발생합니다. 이에 대한 몇가지 사례는, 악의적 사용자가 데이터를 가로 채고 디코딩 할 수 있을 정도로 취약한 암호화 키를 사용하거나,[27] 민감한 데이터를 부족한 수준의 암호화를 사용하여 전송하거나 저장하는 경우를 생각할 수 있습니다.

일반적이지는 않지만, 애플리케이션 및 프로토콜의 구현체와 클라이언트 브라우저에 취약점들이 존재할 가능성이 있고, 이로 인해 중요한 데이터가 노출될 수 있습니다. 궁극적으로 민감한 데이터의 유출을 야기하는 공격은 여러 OWASP 카테고리에 걸쳐 있을 수 있습니다. 취약한 암호화 알고리즘의 사용을 유발하는 부적절한 보안 설정으로 인해 암호화 수준이 저하되고, 결국 공격자는 데이터 스트림을 캡처해서 기밀 데이터를 해독할 수 있게 됩니다.

AWS WAF 를 이용하여 완화하는 방법

들어오는 데이터 스트림이 복호화 된 후 AWS WAF 의해 HTTP 요청이 평가되기 때문에, 통신 채널 상에 충분히 안전한 암호화 방식을 강제하는 것과 WAF 규칙을 적용하는 것은 상호 간에 무관합니다.

흔한 경우는 아니지만, 민감한 데이터의 유출로 이어질 HTTP 요청을 감지할 수 있는 검색 패턴을 가지고 있다면, 그 패턴을 대상으로 하는 **문자열 탐지 조건**을

사용하여 그 가능성을 줄일 수 있습니다. 그러나 이러한 패턴은 애플리케이션 마다 고유한 것으로, 결국 해당 애플리케이션의 자세한 정보가 필요합니다.

예를 들어, 애플리케이션이 SHA-1 해시 알고리즘을 아주 많이 사용하는 경우,[28] 복수의 공격자들이 특수하게 조작 된 PDF 문서의 쌍을 사용하여, 해시 충돌(Hash Collision)을 시도할 가능성이 있습니다.[29] 만약 여러분의 애플리케이션에서 업로드를 허용한다면, 해당 PDF 파일의 base64 로 인코딩 된 값의 일부가 요청의 Body 에 포함하는 지 검사해서 차단하는 WAF 규칙을 설정하는 것이 좋습니다.

AWS WAF 를 사용하여 업로드 된 파일을 서명값 기준으로 차단하려고 하면, WAF 서비스가 이런 종류의 규칙에게 부과하는 몇가지 제한을 고려해야 합니다. 업로드 된 데이터는 base64 로 인코딩되어 있어야 하기 때문에, 이 경우 문자열 검색 조건의 값은 base64 형식이어야 합니다.

AWS WAF 는 일반적으로 HTTP 요청 본문의 첫 8 KB 를 검색하게 되는데, 만약 요청의 멀티 파트 인코딩에 다른 필드 파라메터들이 파일 데이터 자체보다 앞에 존재한다면, 검색하는 부분이 더 적어지게 됩니다. 검색하는 패턴은 최대 50 바이트의 크기가 될 수 있습니다. 대부분의 표준화 된 파일 포맷들은 각각 동일한 시작 부분을 가지기 때문에, 파일의 처음 몇 바이트는 그 형식의 모든 파일이 동일합니다. 때문에, 여러분들은 파일의 나머지 부분에서 검색할 만한 패턴을 적절하게 찾아야 합니다.

다른 고려 사항들

여러분들은 AWS 에코 시스템 내의 다른 서비스를 사용하여 통신 채널 레벨에서 사용되는 암호화 프로토콜과 암호 데이터를 제어 할 수 있습니다:

- Elastic Load Balancing 의 클래식 로드 밸런서의 경우,[30] 사전 정의 또는 사용자 정의된 보안 정책을 선택할 수 있습니다.[31] 이러한 정책은 로드 밸런서가 클라이언트와의 보호된 채널 연결을 협상하는 데 사용할 수 있는 프로토콜과 암호를 지정합니다.
- Elastic Load Balancing 애플리케이션 로드 밸런서는 [32] 사전 정의 된 일련의 보안 정책에서 선택할 수 있습니다.[33] 클래식 로드 밸런서와 마찬가지로, 이러한 정책은 허가된 프로토콜과 암호를 지정합니다.
- Amazon CloudFront[34] 의 콘텐츠 전송 네트워크 (CDN) 서비스에서는, 지원해야 되는 최하위의 SSL 프로토콜 버전 [35] 요건과, 사용자 오리진에 연결할 때 CloudFront 에서 사용해야 하는 SSL 프로토콜을 설정할 수 있습니다.

A7 – 불충분한 공격 방어

이 카테고리는 2017 년 톱 10 에 신규로 제시되었으며, 공격 패턴이 빠르게 진화하는 현실을 반영하고 있습니다. 악의적인 주체가 그들의 도구들을 신속하게 발전시켜 새로운 취약점을 악용하여 거대한 규모로 자동 공격을 시도하고 취약한 시스템을 검색 할 수 있습니다. 이 카테고리는 새로운 공격 경로나 비정상적인 요청 패턴 또는 애플리케이션의 발견된 결함에 대해 적시에 신속하게 대응하는 능력에 중점을 두고 있습니다.

다양한 종류의 공격 벡터가 이 범주에 들어가며, 다른 카테고리들과 많은 부분에서 중복됩니다. 이 부분을 더 잘 이해하기 위해 다음 질문을 고려해 보시기 바랍니다:

- 요청 레벨에서 일정 수준의 통제 관리를 적용할 수 있습니까? 여러분의

애플리케이션이 존재할 것으로 예상했던, 또는 없으면 동작하지 못하는 HTTP 요청의 구성 요소가 있는가요?

- 여러분의 애플리케이션이 비정상적인 요청 패턴이나 대량의 요청들이 유입될 때를 감지하고 인식할 수 있습니까? 그런 발견을 자동화 할 수 있는 시스템이 있습니까? 있다면, 그 시스템은 이러한 불필요한 트래픽을 감지하고 차단할 수 있습니까?
- 악의적인 주체가 당신의 애플리케이션의 결함을 이용하여 특정 표적 공격을 시작했을 때를 감지할 수 있습니까? 이 기능이 거의 실시간으로 대응할 수 있도록 자동화되어 있습니까?
- 얼마나 빨리, 발견된 애플리케이션 혹은 애플리케이션 스택의 취약점을 패치하고 그로 인한 공격을 완화할 수 있습니까? 패치 배포 후 적용 효과를 검증하는 메커니즘이 있나요?

AWS WAF 를 이용하여 완화하는 방법

AWS WAF 를 사용하여 인바운드 HTTP 요청의 검사 수준을 제어할 수 있습니다. 크기 제약 조건 [36] 은 HTTP 요청의 구성 요소가 특정 범위에 맞는지를 검사하는 규칙을 만드는 데 사용됩니다. 이들을 사용하면 비정상적인 요청이 처리되는 경우를 피할 수 있습니다. 일례로 URI 또는 쿼리 문자열의 크기를 애플리케이션에 맞는 값으로 제한하는 것입니다. 또한 그들을 사용하여 RESTful API 의 API 키 등 특정 헤더의 존재를 검사할 수도 있습니다.

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	비교 연산자	크기

URI	NONE	GT (greater than)	*바이트 단위의 URI 경로 최대값*
QUERY_STRING	NONE	GT	*바이트 단위의 쿼리크기 최대값*
BODY	NONE	GT	*바이트 단위의 요청 Body 의 최대값*
HEADER:x-api-key	NONE	LT (less than)	1 *(혹은 API 키의 실제 크기)*
HEADER:cookie	NONE	GT	*바이트 단위의 쿠키 크기 최대값*

이 절에서 설명하는 조건 사례를 블랙리스트 규칙과 함께 사용하여 조건에 맞지 않는 요청을 거부 할 수 있습니다.

비 정상적인 요청 패턴을 감지하기 위해, 단일 IP 주소에서 유입되는 요청의 속도가 지정된 임계값을 초과한 경우(5 분 간격 기준)에 트리거되는 AWS WAF 의 요청 속도 기반의 규칙을 사용할 수 있습니다. 또한 이 규칙을 AWS WAF 가 제공하는 다른 탐색 조건들과 결합 할 수 있습니다.

예를 들어, 속도 기반의 규칙과 문자열 탐지 규칙을 결합하여 특정 user agent 의 요청 만을(말하자면 user-agent = "abc" 식으로) 계산할 수 있습니다. 이 규칙의 조합으로 user-agent = "abc"에 맞는 요청들만 개별 IP 주소를 기준으로 하는 요청 속도 위반 판정에 집계됩니다.

AWS WAF 의 주요 장점은 개발의 유연성입니다. 여러분들은 언제든지 프로그램 API 를 사용하여 AWS WAF 웹 액세스 제어 목록 (ACL), 규칙과 조건을 설정하거나 변경할 수 있습니다. 이런 변경 내역은 일반적으로 Amazon CloudFront 와 통합된 글로벌 서비스에 1 분 이내에 반영됩니다.

API 를 사용하면 애플리케이션 자체의 비정상적인 상태에 대응하고, 의심스러운 트래픽 소스를 차단하는 작업을 수행하거나, 운영자에게 상세한 조사를 통지하는 자동화 된 프로세스를 구축할 수 있습니다. 이러한 자동화는 트랩 또는 허니팟 URL 경로를 통해 실시간으로 작동 가능합니다. 또한 애플리케이션 로그 파일과 요청 패턴들 간의 분석 및 상관 관계에 따라 대응할 수도 있습니다.

앞에서 이야기한 바와 같이, AWS 는 AWS WAF 보안 자동화라는 일련의 기능을 제공합니다. 이 기능은 앞에서 살펴본 주요 패턴들에 대응하도록 동작합니다. 몇가지 유형을 보자면, 이벤트 기반 컴퓨팅의 AWS Lambda 를 비롯하여 일부 AWS 서비스를 사용하여 다음과 같은 시나리오를 쉽게 구성할 수 있습니다:[38]

- **스캐닝과 불법 탐지의 경감.** 악성 소스는 취약점을 찾기 위해 인터넷에 접한 웹 애플리케이션을 검색하여 확인합니다. 그들은 HTTP 4xx 오류 코드를 생성하는 일련의 요청들을 전송하게 됩니다. 여러분들은 이 패턴을 사용하여 악의적인 소스의 IP 주소를 식별하고 차단할 수 있습니다. 이 솔루션은 액세스 로그를 자동으로 분석하여 소스 IP 주소 기준으로 잘못된 요청의 수를 집계하고, 더이상 스캐닝을 못하도록 AWS WAF 규칙에 반영하는 AWS 람다 함수를 작성 합니다.
- **알려진 공격자 IP 차단**. 많은 조직이 스패머, 멀웨어 배포자, 봇넷 등의 알려진 공격에 활용되는 IP 주소들을 담고 있는 평판 목록을 이용하고 있습니다. 이 솔루션은 이러한 평판 목록의 정보를 활용하여 악성 IP 주소로부터 유입되는 요청을 차단하는데 도움이 됩니다.

- **봇과 스크래퍼의 완화.** 공개된 웹 애플리케이션의 운영자는 자신의 콘텐츠에 액세스하는 클라이언트가 누구인지 정확하게 파악하고 목적에 맞게 서비스를 사용하고 있는지를 확인하게 됩니다. 그러나 컨텐츠 수집기(Content Scraper)나 악성 봇 등의 자동화된 클라이언트는 그런 통제를 회피하기 위해 자신을 숨기게 됩니다.
  이 솔루션은 악성 봇과 스크래퍼를 식별하고 차단하는 데 도움이 되는 허니팟을 구현하고 있습니다. 이 솔루션에서, robots.txt 파일의 'disallow' 부분에 허니팟 URL 들이 목록화 되어 있습니다.[39] 따라서 이 URL 에 액세스하는 모든 IP 는 악의가 있거나 또는 안전하지 않은 것으로 간주하여 블랙리스트에 실리게 됩니다.

또한 AWS WAF 를 사용하여 새로 발견된 애플리케이션의 결함이나 스택의 취약점을 완화하는 방법도 있습니다. 자세한 내용은 뒤에 다루도록 하겠습니다. (A9 - 알려진 취약점을 갖고 있는 구성 요소의 사용 부분을 참조).

A8 – 크로스 사이트 요청 위조(CSRF)

크로스 사이트 요청 위조 공격(CSRF)은 주로 웹 애플리케이션의 상태 변경 기능을 노리게 됩니다.[40] 상태 변화를 일으키는 원인 URL 경로와 HTTP 요청 (예를 들면, Form 양식 제출 요구 등)을 살펴봅시다. 여러분들은 사용자가 진짜 그 행위를 실행할 의도가 있었는지를 확인할 방법이 있나요? 이러한 메커니즘이 없다면, 그 요청이 악의적인 공격자가 위조하지 않았고 정당한 요청이라는 것을 판단할 수 있는 효과적인 방법은 없습니다. 세션 토큰이나 소스 IP 주소 등, 클라이언트 측의 속성에만 의존해서는, 악의를 가진 주체가 이러한 값을 조작하여 복제할 수도 있기 때문에, 효과적으로 대응할 수 없습니다.

CSRF 공격은 특정 작업의 모든 세부 사항들이 예측 가능하다는 사실 (Form 양식 필드, 쿼리 문자열 매개 변수)을 잘 활용합니다. 공격은 크로스 사이트 스크립팅이나 파일 첨부 등 다른 취약점을 이용하는 방법으로 이루어지기 때문에, 사용자 입장에서는 본인의 자격 증명과 활성화 된 세션을 사용하여 악의적인 작업이 실행된 것을 알아 차리기가 어렵습니다.

AWS WAF 를 이용하여 완화하는 방법

여러분들은 아래와 같은 방식으로 CSRF 공격을 완화시킬 수 있습니다:

- 행위를 유발하는 HTTP 요청 내부에 예측할 수 없는 토큰값 삽입.
- 행위 요청을 보내는 사용자에 대해 인증을 유도.
- 행위 요청을 보내는 데 대해 CAPTCHA 방식을 통해 재확인.[41]

첫번째 옵션은 사용자 입장에서는 안보이며, 히든 Form 필드, 커스텀 헤더, 혹은 덜 권장하지만 쿼리 스트링 파라메터 같은 곳에도 토큰을 포함시키게 됩니다. 나머지 두 가지 옵션은 최종 사용자에게 불필요한 마찰을 초래할 수 있으며, 일반적으로 기밀 작업 요청에 대해서만 구현됩니다. 또한 CAPTCHA 는 명확한 의도를 가진 주체가 값의 조합을 반복하는 방식으로 우회할 수 있습니다.[42] 이런 방식에서는 이들 옵션들도 그다지 바람직한 CSRF 완화책이 될 수 없습니다.

AWS WAF 를 사용하면 이러한 고유의 토큰이 존재하는지 여부를 확인할 수 있습니다. 예를 들어, CSRF 토큰으로 랜덤 universally unique identifier(UUIDv4)[43] 을 활용하기로 하고, *x-csrf-token* 이라는 커스텀 HTTP 헤더 값을 기대하는 경우에는 **크기 제약 조건**을 적용할 수 있습니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	비교 연산자	크기
HEADER:x-csrf-token	NONE	EQ (equal to)	36 *(bytes/ASCII characters, canonical format)*

여러분은 요청이 위 조건과 **일치하지 않을 때 차단**하는(부정) 규칙을 만듭니다. 예를 들어, POST HTTP 요청 만을 확인하는 방식으로, 규칙의 범위를 더욱 좁힐 수 있습니다. 위의 부정 조건 및 아래와 같은 추가적인 **문자열 탐지 조건**을 사용하여 규칙을 만들기 바랍니다:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	상대적인 위치 조건	탐지 값
METHOD	LOWERCASE	EXACTLY	post

다른 고려 사항들

이러한 규칙은 토큰을 우회하는 형태의 CSRF 공격을 제거하는 데 효과적입니다. 그러나 HTTP 요청 내부에 무효화된, 이상한, 해지된, 도난당한 토큰들이 포함되어 있는지 여부를 확인하는 데는 유효하지 않습니다. 그 이유는 HTTP 요청 단위의 조사에서는 애플리케이션 컨텍스트를 감안할 수 없기 때문입니다. 따라서 예상되는 토큰을 추적하고 그것이 한번만 사용되었는지 확인하려면 여러분의 애플리케이션에 서버 측 메커니즘이 필요합니다.

예를 들어, 서버가 고유한 토큰을 숨김 필드에 포함해서 간단한 Form 형태로 클라이언트 브라우저에 보낸다고 합시다. 동시에 이 값은 사용자가 양식을

제출할 때 브라우저가 제출한 토큰 값들을 저장하게 되는 서버 측 세션 저장소에 유지됩니다. 사용자가 양식을 제출한 뒤, 고유의 숨겨진 토큰을 포함한 POST 요청이 서버로 전송됩니다. 이를 이용하게 되면, 서버는 해당 세션에서 기대되는 값이 포함되지 않는 다른 모든 POST 요청들을 안전하게 무시할 수 있게 됩니다. 한번 사용되고 나면, 세션 저장소에서 이 값을 삭제해야 합니다. 이렇게 하면 그 값이 향후 재사용되지 않는다는 것도 보장할 수 있게 됩니다.

A9 – 알려진 취약점을 갖고 있는 구성 요소의 사용

현재 대부분의 웹 애플리케이션들은 고도로 복잡하게 구성되어 있습니다. 그들은 다양한 소스, 상용 또는 오픈 소스 프레임 워크와 라이브러리를 사용합니다. 그리고 이러한 구성 요소들의 버전을 항상 최신 상태로 유지하는 것이 주요 과제가 됩니다. 만약 기본 라이브러리나 프레임 워크에서 서로 다른 구성 요소들을 사용하는 경우 상황이 더 어려워 질 수 있습니다.

알려진 취약점이 있는 구성 요소를 공격의 통로로 사용하는 것은 가장 일반적인 공격 방법 중 하나입니다.[44] 알려진 취약점과 본 문서에서 설명된 다른 형태의 공격 방법들 중 일부를 이용하여, 여러분의 웹 애플리케이션이 공격받을 수 있는 가능성이 확대될 수 있습니다. 그럼에도 불구하고, 이러한 취약한 구성 요소를 사용하게 되는 의사결정은 기존 코드와의 호환성을 유지하기 위한 사유 등을 적극적으로 따져서 고민한 뒤 이루어 져야 합니다. 또한 취약한 하위 구성 요소에 의존하는 컴포넌트를 사용하게 되는 경우, 결과적으로 시스템 전체적으로 취약해 질 수 있다는 점을 분명히 알아야 합니다.

이러한 구성 요소의 취약점을 완화하는 작업은, 모든 취약점들이 CVE 와 같은 중앙 관리 체계에 의해 빠짐없이 보고되고 추적될 수 없기 때문에 매우 어려운 작업이 될 수 있습니다.[45] 따라서 애플리케이션 개발자는 각각의 벤더, 작성자 또는 공급자와 개별 구성 요소의 상태를 일일히 추적해야 할 책임을 갖게 됩니다. 종종 취약점은 기존 버전을 수정하는 것이 아니라, 새로운 기능을 포함한 새로운 버전의 구성 요소를 적용함으로써 해결되기도 합니다. 따라서 이러한 구성 요소의 신규 버전을 구현, 테스트 및 배포하기 위해 개발자가 수행해야 하는 작업량이 늘어날 수 있습니다.

AWS WAF 를 이용하여 완화하는 방법

구성 요소의 알려진 취약점을 완화하기 위한 주요한 방법은 이러한 구성 요소의 라이프 사이클에 대응하는 포괄적인 프로세스를 갖추는 것입니다. 즉, 애플리케이션의 종속성과 그것의 기본 하위 구성 요소의 종속성을 식별하고 추적할 수 있는 방법이 필요합니다. 또한 이러한 구성 요소의 보안성을 추적하는 모니터링 과정이 필요합니다.

기본 구성 요소들의 패치 혹은 배포 주기와 그것들이 허용하고 있는 라이센스 모델을 고려하여, 소프트웨어 개발 프로세스와 정책을 수립하시기 바랍니다. 이렇게 하면 구성 요소 공급자가 본인들 코드의 취약점을 공개했을 때, 여러분들이 이를 해결하기 위해 신속하게 대응할 수 있게 됩니다.

또한 AWS WAF 를 사용하여 애플리케이션에서 사용하지 않는 구성 요소의 기능에 대한 HTTP 요청을 필터링 및 차단 될 수 있습니다. 이렇게 하면 사용하지 않는 기능에서 취약점이 발견되더라도, 이러한 구성 요소를 통로로 한 공격 가능성이 제거 될 수 있습니다.

애플리케이션 서버 사이드에 포함된 구성 요소를 사용하고 있습니까? 이것들은 일반적으로 런타임에 로드되고, HTTP 응답을 직접 또는 간접적으로 조합한 코드가 포함된 파일 형태입니다. 예를 들면, PHP 의 include [47] 또는 require [48] 구문을 통해 로딩된 코드나, Apache Server-Side Include 같은 것들이 있습니다. 다른 언어와 프레임 워크도 비슷한 구조를 가지고 있습니다.

이러한 구성 요소들을 웹 서버 상의 퍼블릭 웹 경로에 배포하지 않게 관리할 것을 권장 드립니다. 그러나 때로는 다양한 이유로 이 권고를 무시할 경우도 있습니다. 만약 이러한 구성 요소가 퍼블릭 웹 경로에 존재하더라도, 이 파일은 직접 액세스 할 수 있도록 설계되어 있지 않습니다. 그럼에도 불구하고, 그들에 접근을 시도하는 과정에서, 내부 애플리케이션 정보가 공개되거나 공격 벡터가 제공 될 수 있습니다.

이 같은 URL 접두사에 대한 액세스를 차단하려면 **문자열 탐지 조건**의 사용을 고려하십시오:

HTTP 요청의 구성요소 중 점검 할 영역	**적용해야 될 입력값 처리 작업**	**상대적인 위치 조건**	**탐지 값**
URI	URL_DECODE	STARTS_WITH	/includes/ *(혹은 애플리케이션의 연관된 접두어)*

마찬가지로 여러분의 애플리케이션이 3rd party 구성 요소들 중 일부만을 사용하는 경우에도, 동일한 AWS WAF 조건을 적용하여, 사용하지 않는 구성

요소의 기능에 접근할 수 있는 public URL 경로를 차단하는 것이 좋습니다.

다른 고려 사항들

침투 테스트 또한 취약점을 발견하기위한 효과적인 방법이기도 합니다.[49] 배포 및 테스트 프로세스에 통합하여 잠재적인 취약점을 탐지할 수 있을 뿐만 아니라 배포된 패치로 인해 대상 애플리케이션의 결함이 적절하게 완화되었는지 확인할 수도 있습니다.

AWS Marketplace[50] 에서는 쉽고 빠르게 시작할 수 있도록 설계되어 있는 파트너 업체의 다양한 취약점 점검 솔루션을 제공하고 있습니다. 그리고 AWS 에서 호스팅되는 리소스에 대하여, 이러한 테스트를 실시하기 전에 AWS 로부터 승인을 얻을 필요가 있다는 것을 반드시 기억하시기 바랍니다. 또한 AWS Marketplace 에는 이러한 사전 승인 절차 없이 사용할 수 있는 사전 인증된 몇가지 솔루션도 있습니다. 해당 솔루션 제목에 그 부분이 표시되어 있습니다.

A10 – API 에 대한 미흡한 보호

2017 년 Top 10 에 신규로 제안된 Underprotected APIs 카테고리는 악용될 가능성이 있는 특정 애플리케이션의 결함 패턴이 아니라, 잠재적인 공격 대상에 대해 초점을 맞추고 있습니다. 이 카테고리는 API 의 보급률과 향후의 성장률에 주목하고 있습니다. 현재 사용자를 위한 UI 를 가지고 있지 않는 애플리케이션들 전체가 공개되어 있습니다. 대신 이들은 다른 애플리케이션 벤더가 느슨하게 연결된(loosely coupled) 애플리케이션을 구축하는 데 활용할 수 있는 API 형태로 사용될 수 있습니다. 이외의 많은 애플리케이션들은 사용자를 위한 UI 와 API 를 모두 가지고 있고, 이 API 는 또 다른 곳에서 이용될 목적을 가지고 있지는 않습니다.

이 공격 형태는 A1 에서 A9 카테고리에서 설명한 것과 동일한 것으로, 많은 최종 사용자를 서비스하는 기존의 웹 애플리케이션 환경에서 흔한 유형입니다.

하지만 API 는 프로그래밍 방식의 액세스를 위해 설계되어 있기 때문에, 보안 검사에 대한 몇 가지 추가적인 이슈를 더 고려해야 합니다. 사용자 UI 는 상대적으로 간단한 데이터 구조와 사용자와의 상호 작용으로 인한 단절되고 높은 지연 단계를 가지고 있기 때문에, 보안 테스트 케이스를 개발하는 것이 좀더 쉽습니다. 이와 반대로, API 는 종종 더 복잡한 데이터 구조로 동작하고 더 높은 요청 빈도와 더 넓은 범위의 입력 값을 사용하도록 설계됩니다. 이것은 RESTful API[52] 와 SOAP[53] 과 같이 표준화되고 잘 알려진 프로토콜을 사용하는 경우에도 동일합니다.

AWS WAF 를 이용하여 완화하는 방법

API 에 대한 공격 방법은 기존의 전통적인 웹 애플리케이션에 대한 공격 방법과 동일할 경우가 많기 때문에, 이 문서 전체에 설명되어 있는 완화 메커니즘과 동일한 방식들이 API 공격유형에도 적용될 수 있습니다. 다양한 방법으로 AWS WAF 를 사용하여 이러한 다양한 공격 방법을 완화 할 수 있습니다.

보안 강화가 필요한 주요 구성 요소는 API 프로토콜 파서 자체입니다. 표준화된 프로토콜에서는 사용하게 되는 파서를 추정하는 것이 비교적 간단합니다. SOAP 은 XML[54] 을 사용하게 되고, RESTful API 는 일반적으로는 JSON[55] 을 사용할 수 있지만 XML, YAML[56] 등의 형식도 사용할 수 있습니다. 따라서 파서 구성 요소의 설정을 효과적으로 보호하고 이에 대한 취약점을 완화하는 것은 이 부분에 대한 대단히 중요한 성공 요인이 될 수 있습니다.

파서의 결함을 악용하려는 특정 입력 패턴이 발견되면 AWS WAF **문자열 탐지 조건** 또는 **크기 제한**을 사용하여 그와 같은 요청 패턴을 차단할 수 있습니다.

이전 Top10 2013 의 A10 – 리다이렉트와 포워드에 대한 미 검증

대부분의 웹 사이트나 웹 애플리케이션은 사용자를 다른 페이지(내부 사이트 및 파트너 사이트)로 리디렉션 또는 전송하는 메커니즘이 포함되어 있습니다. 이러한 메커니즘이 리디렉션 또는 포워드 요청을 별도로 확인하지 않는 경우, 악의적인 주체가 여러분의 합법적인 도메인을 사용하여 사용자를 의도치 않은 목적지로 유도할 수 있습니다. 이러한 링크는 여러분이 소유한 정당한 평판을 가지고 있는 도메인을 사용하여 사용자를 속이게 됩니다.

아래와 같은 사례를 살펴보도록 합시다:

> *비디오 공유 사이트를 구성하고, URL 단축 메커니즘을 이용하여 사용자가 모바일 디바이스 상에서 문자 메시지로 비디오를 공유할 수 있도록 합니다. 스크립트를 사용하여 아래와 같이 URL 을 만듭니다:*
>
> https://example.com/link?target=
> **https%3A%2F%2Fexample.com%2Fvideo%2Fe439853%3Fpos%3D200%2 6mode%3Dfullscreen**
>
> *사용자는 다음과 같은 URL 을 받게 되고, 그들은 이를 통해 해당 컨텐츠 페이지로 이동하게 됩니다:*

https://example.com/to?**vrejkR6T**

링크 생성기 스크립트가 대상 페이지의 허용 입력 도메인을 확인하지 않으면, 아래의 경우처럼, 악의적인 사용자가 원치 않는 사이트로의 링크를 생성할 수 있습니다:

https://example.com/link?target=
https%3A%2F%2Fbadsite.com%2Fmalware

그들은 이 내용을 포장해서 여러분의 사이트에서 유도되도록 사용자에게 보낼 수 있습니다:

https://example.com/to?**br09FtZ1**

AWS WAF 를 이용하여 완화하는 방법

이런 형태를 완화시키기 위한 첫 단계는 애플리케이션에서 리디렉션 및 전달이 어디에서 발생하는지를 이해하는 것입니다. 어떤 URL 요청 패턴이 직접 또는 간접적으로 어떤 조건에서 리디렉션을 발생시키는지를 발견하여, 잠재적으로 취약한 영역의 목록을 작성할 수 있게 도와 줍니다. 만약 여러분의 애플리케이션이 리디렉션 기능을 포함하는 경우라면, 애플리케이션에서 사용하는 외부 컴포넌트들 모두에 대해 동일한 분석을 수행해야합니다.

위의 예와 같이 최종 사용자의 HTTP 요청에 따라 리디렉션 및 전송이 생성되는 경우, AWS WAF 를 사용하여 요청을 필터링하고, 리디렉션 / 전송의 타겟으로 신뢰할 수 있는 도메인의 목록을 유지할 수 있습니다. 매개 변수에 포함된 도메인이 화이트리스트에 있는지 확인하는 방법으로 HTTP 요청의 구성 요소를 대상으로 **문자열 탐지 조건**을 사용할 수 있습니다. 위의 사례를 탐지하기 위한

일련의 조건은 다음과 같습니다:

1. 리다이렉트가 허용되는 도메인 화이트 리스트(여기에 없는 곳으로의 리다이렉트는 전부 차단):

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	상대적인 위치 조건	탐지 값
QUERY_STRING	URL_DECODE	CONTAINS	target=https://example.com
QUERY_STRING	URL_DECODE	CONTAINS	target=https://partnersite.com

2. (redirector 혹은 라우팅 스크립트로의) 특정 HTTP 요청들만 탐지:

HTTP 요청의 구성요소 중 점검 할 영역	적용해야 될 입력값 처리 작업	상대적인 위치 조건	탐지 값
URI	URL_DECODE	STARTS_WITH	/link

두가지 조건들에 전부 해당하는 지를 점검하기 위해, 이들 두가지 조건을 단일 AWS WAF 규칙에 포함시켜야 합니다.

준비된 CloudFormation 템플릿 활용하기

본 문서에서 추천드렸던 웹 ACL 및 조건 유형과 규칙을 포함하는 AWS CloudFormation 템플릿 [58] 을 준비했습니다. 여러분들은 이 템플릿을 사용하여 몇 번의 클릭으로 이러한 환경을 설정할 수 있습니다(이에 대한 전체 API 지원도 사용할 수 있습니다). 이 템플릿은 실 운영환경에 바로 적용할 수 있는

포괄적인 규칙 집합이 아니라, 그런 수준으로 발전시킬 수 있는 출발점으로 활용되도록 설계되어 있습니다. CloudFormation 템플릿을 사용한 작업에 대한 자세한 내용은 "템플릿 기초를 배우기"를 참조하십시오.[59]

템플릿은 아래 링크에서 받으실 수 있습니다:

https://s3.us-east-2.amazonaws.com/awswaf-owasp/owasp_10_base.yml

다음 사례는 이 템플릿에 포함되어 있는 규칙들입니다:

- **악성 트래픽 소스**. 악성 소스로 확인된 곳으로 부터 유입된 요청들을 차단시키는 범용 IP 차단 목록 규칙.
- **훼손된 접근제어:**
    - 의심스러운 요청을 차단하기 위해, 공통 파일 시스템 경로 탐색과 로컬 및 원격 파일 인젝션 (LFI / RFI) 패턴을 감지하는 경로 탐색 및 파일 삽입에 대한 규칙.
    - 관리 모듈의 액세스를 알려진 소스 IP 로 제한하는 특권 모듈 액세스 제한 규칙. 이 템플릿에서는 하나의 경로 접두사와 소스 IP 주소를 설정할 수 있게 되어 있습니다. 나중에 조건을 직접 변경하여 패턴을 추가 할 수 있습니다. 자세한 내용은 웹 Access Control List 작성 및 구성을 참조하십시오.[60]
- **훼손된 인증 및 세션 관리**. JSON 웹 토큰과 세션 ID 등 도난 또는 유출된 자격 증명을 사용하는 잘못된 요청을 차단할 수 있는 차단 목록.
- **크로스 사이트 요청 변조(CSRF)**. CSRF 완화 토큰의 존재를 확인하기 위한 규칙.

- **크로스 사이트 스크립트(XSS)**. HTTP 요청 컴포넌트에 포함된 XSS 공격을 완화하기 위한 규칙.
- **인젝션**. HTTP 요청 컴포넌트에 포함된 SQL 인젝션 공격을 완화하기 위한 규칙.
- **불충분한 공격 방어**. 템플릿 매개 변수를 사용하여 다양한 HTTP 요청 컴포넌트들의 최대 사이즈를 구성하고, 설정된 최대 크기를 초과하는 비정상적인 요청을 차단해 주는 필터링 규칙.
- **잘못된 보안 설정**. PHP 서버 상의 설정 실수를 악용하는 부분을 탐지하는 규칙. PHP 기반의 애플리케이션을 실행하지 않는 경우, 이 규칙은 그다지 효과적이지 않지만, PHP 취약점을 파악하는 의도치 않은 자동화된 HTTP 요청을 차단시켜 주는 효과가 있습니다.
- **알려진 취약점을 가진 컴포넌트의 사용**. 여러분의 애플리케이션에서 사용되지 않는, 서버 사이드에 포함된 구성 요소와 그 기능에 대해 URL 기반으로 직접적인 접근을 하는 것을 제한하는 규칙입니다.

몇 가지 간단한 클릭으로 전체 규칙 집합을 프로비져닝하는 과정을 쉽고 반복적으로 할 수 있게 하기 위해, 샘플 AWS WAF 규칙 세트를 CloudFormation 템플릿으로 패키징 하였습니다. AWS CloudFormation 문서에서는, 스택을 만드는 방법[61] 에 대한 간단한 연습을 제공하는데, 스택이란 여러분들이 단일 단위로 관리할 수 있도록 해주는 자원의 집합을 말합니다.

그 절차대로 **템플릿 선택** 페이지에서 해당 템플릿을 제출하시기 바랍니다. **Amazon S3 에 템플릿을 업로드**하는 옵션을 선택하고 로컬 컴퓨터에서 지금 다운로드한 템플릿을 제출하면 됩니다. 다른 방법으로는 템플릿 URL

(https://s3.us-east-2.amazonaws.com/awswaf-owasp/owasp_10_base.yml)을 **Amazon S3 템플릿 URL 지정 텍스트 필드에 입력**하시기 바랍니다.

여러분들은 **고급 지정 페이지**에서 템플릿 매개 변수를 설정할 수 있습니다. 주목해야 할 중요한 매개 변수는:

- **WAF 적용.** 이 매개 변수를 사용하면 템플릿을 사용하여 Amazon CloudFront 웹 배포지점이나, 현재 리전의 Application Load Balancer(ALB)를 보호하기 위한 규칙 세트를 선택적으로 적용할 수 있습니다. AWS WAF 의 웹 ACL 은 애플리케이션 구축에 사용되는 서비스 별로, CloudFront 웹 배포 지점 혹은 ALB 리소스를 보호할 수 있습니다. 동시에 같은 스택을 적용할 수는 없지만, 멀티 스택을 적용할 수는 있습니다. 또한 스택을 업데이트하여 이 매개 변수의 값을 나중에 변경할 수 있습니다.
- **Rule effect.** 이 매개 변수는 규칙 세트의 효과를 결정합니다. 우선은 어떤 규칙이라도 바로 차단/허용을 하지 않고, 탐지되는 요청 건을 단순히 **집계**만 하는 규칙 세트를 가지고 시작하는 것이 좋습니다. 이를 통해 실제 트래픽에 영향을 주지 않고 규칙의 효과를 측정할 수 있습니다. 규칙의 효과가 정확하게 작동한다는 것이 확실해 지면, 탐지된 요청을 **차단**하도록 스택을 변경할 수 있습니다.

AWS CloudFormation 의 연습 단계를 따라서 스택을 적용해 보시기 바랍니다. 스택을 적용한 후, 규칙 세트가 활용되기 위해서는 스택에 의해 작성된 웹 ACL[62] 을 보호 대상인 로드 밸런서 또는 웹 배포 리소스에 연결해야 합니다.

결론

AWS WAF 를 사용하면 HTTP 프로토콜 수준에서 발생하는 다양한 공격 방식으로부터 웹 사이트나 웹 애플리케이션을 보호할 수 있습니다. 이 문서에서 살펴본 대로, AWS WAF 는 OWASP 의 보안 결함에 관련하여, HTTP 요청에서 이러한 공격 패턴을 탐지하고 연관된 취약점을 완화하는 데 매우 효과적입니다.

또한 다른 AWS 서비스와 AWS WAF 기능을 연계하여 포괄적인 보안 자동화를 구축할 수 있습니다. 이러한 도구 세트는 당사의 웹사이트 중 AWS WAF Security Automations[63] 에서 구할 수 있습니다.

이러한 도구를 사용하면 애플리케이션이 직면할 가능성이 있는 다양한 변종 공격들에 대응할 수 있는 일련의 보호 기능들을 만들 수 있습니다. 이 솔루션은 요청율 기반의 IP 블랙리스트, 평판 기반 IP 블랙리스트, 스캐닝과 불법탐색 완화, 봇과 스크래퍼의 감지 및 차단 기능을 구현할 수 있는 CloudFormation 템플릿 형태로서, 쉽게 적용 가능한 자동화 수단을 제공합니다.

공헌자들

본 문서를 위해 도움을 준 분들입니다:

- Vlad Vlasceanu, Sr. Solutions Architect, Amazon Web Services
- Sundar Jayashekar, Sr. Product Manager, Amazon Web Services
- William Reid, Sr. Manager, Amazon Web Services
- Stephen Quigg, Solutions Architect, Amazon Web Services
- Matt Nowina, Solutions Architect, Amazon Web Services
- Matt Bretan, Sr. Consultant, Amazon Web Services
- Enrico Massi, Security Solutions Architect, Amazon Web Services
- Michael St.Onge, Cloud Security Architect, Amazon Web Services
- Leandro Bennaton, Security Solutions Architect, Amazon Web Services
- 임 기성, Security Solutions Architect, Amazon Web Services

더 읽어볼 거리들

좀 더 자세한 내용은 다음을 참고하시기 바랍니다.:

- AWS WAF Security Automations: https://aws.amazon.com/answers/security/aws-waf-security-automations/
- OWASP Top 10 – 2017 rc1:

https://github.com/OWASP/Top10/raw/master/2017/OWASP%20Top%2010%20-%202017%20RC1-English.pdf

- OWASP Top 10 – 2013: https://www.owasp.org/index.php/Top_10_2013

문서 개정이력

Date	Description
July 2017	영문 초판 배포
Aug 2017	한글 번역본 배포

비고

1 https://www.owasp.org/

2 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

3 https://aws.amazon.com/waf/

4 https://aws.amazon.com/cloudfront/

5 https://aws.amazon.com/elasticloadbalancing/applicationloadbalancer/

6 https://www.owasp.org/index.php/Top_10_2013-A1-Injection

7 http://docs.aws.amazon.com/waf/latest/developerguide/web-acl-sql- conditions.html

8 http://docs.aws.amazon.com/waf/latest/developerguide/web-acl-string- conditions.html

9 https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management

10 https://jwt.io/

11 http://docs.aws.amazon.com/waf/latest/APIReference/Welcome.html

12 https://www.owasp.org/index.php/Top_10_2013-A3-Cross- Site_Scripting_(XSS)

13 http://docs.aws.amazon.com/waf/latest/developerguide/web-acl-xss- conditions.html

14 https://en.wikipedia.org/wiki/Content_management_system

15 https://developer.mozilla.org/en-US/docs/Web/SVG

16 https://www.owasp.org/index.php/Top_10_2013-A4- Insecure_Direct_Object_References

17 https://www.owasp.org/index.php/Top_10_2013-A7- Missing_Function_Level_Access_Control

18 https://en.wikipedia.org/wiki/Directory_traversal_attack

19 https://en.wikipedia.org/wiki/File_inclusion_vulnerability

20 http://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html

21 https://www.owasp.org/index.php/Top_10_2013-A5- Security_Misconfiguration

22 https://aws.amazon.com/inspector/

23 https://www.cisecurity.org/cis-benchmarks/

24 https://aws.amazon.com/config/

25 https://aws.amazon.com/ec2/systems-manager/

26 https://www.owasp.org/index.php/Top_10_2013-A6- Sensitive_Data_Exposure

27 https://en.wikipedia.org/wiki/Cipher

28 https://en.wikipedia.org/wiki/SHA-1

29 https://shattered.io/

30 http://docs.aws.amazon.com/elasticloadbalancing/latest/classic/introduction.html

31 http://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-ssl- security-policy.html

32 http://docs.aws.amazon.com/elasticloadbalancing/latest/application/introdu ction.html

33 http://docs.aws.amazon.com/elasticloadbalancing/latest/application/create- https-listener.html

34 http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Int roduction.html

35 http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-values-specify.html#DownloadDistValuesMinimumSSLProtocolVersion

36 http://docs.aws.amazon.com/waf/latest/developerguide/web-acl-size- conditions.html

37 https://aws.amazon.com/answers/security/aws-waf-security-automations/

38 https://aws.amazon.com/lambda/

39 https://en.wikipedia.org/wiki/Robots_exclusion_standard

40 https://www.owasp.org/index.php/Top_10_2013-A8-Cross- Site_Request_Forgery_(CSRF)

41 https://en.wikipedia.org/wiki/CAPTCHA

42 https://en.wikipedia.org/wiki/CAPTCHA#Circumvention

43 https://en.wikipedia.org/wiki/Universally_unique_identifier

44 https://www.owasp.org/index.php/Top_10_2013-A9- Using_Components_with_Known_Vulnerabilities

45 http://cve.mitre.org/

46 https://httpd.apache.org/docs/current/howto/ssi.html

47 http://php.net/manual/en/function.include.php

48 http://php.net/manual/en/function.require.php

49 https://en.wikipedia.org/wiki/Penetration_test

50 https://aws.amazon.com/marketplace/search/results?x=0&y=0&searchTerms=vulnerability+scanner&page=1&ref_=nav_search_box

51 https://aws.amazon.com/security/penetration-testing/

52 https://en.wikipedia.org/wiki/Representational_state_transfer

53 https://en.wikipedia.org/wiki/SOAP

54 https://www.w3.org/XML/

55 http://www.json.org/

56 http://yaml.org/

57 https://www.owasp.org/index.php/Top_10_2013-A10- Unvalidated_Redirects_and_Forwards

58 https://aws.amazon.com/cloudformation/

59 http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/gettingstarted.templatebasics.html

60 http://docs.aws.amazon.com/waf/latest/developerguide/web-acl.html

61 http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn- console-create-stack.html

62 http://docs.aws.amazon.com/waf/latest/developerguide/web-acl-working- with.html#web-acl-associating-cloudfront-distribution

63 https://aws.amazon.com/answers/security/aws-waf-security-automations/