목차

Apache HTTPD 보안취약점 점검

— *이강우 2025/07/09 01:09*

Method 제한

전역설정으로 제한 걸기 `/etc/httpd/conf.d/method.conf`

```
<Location "/*">
    <LimitExcept GET POST>
        Order deny,allow
    Deny from all
    </LimitExcept>
</Location>
```

Apache httpd logfile permission(umask)

아파치 웹서버 로그 파일은 기본적으로 644로 생성된다. 이것은 실행하는 루트의 기본 umask가 0022로 설정되어있기 때문이다. 따라서 웹서버를 기동하는 계정의 기본 umask값만 변경하면 자동으로 설정된다.

하지만 계정의 기본값은 그대로 두고 아파치 웹서버의 설정변경만 하고 싶은경우 아래와 같이 설정하도록 한다.

rpm 으로 설치한경우

rpm 패키지로 설치한경우에는 `/etc/sysconfig/httpd` 파일에 아래 내용을 추가하면 된다.

```
umask 0027
```

envvars 사용법

rpm이 아닌 소스컴파일 설치한경우에 이 방법을 사용한다. apache 기동 스크립트인 `apachectl`내에 보면 envvars 파일을 로딩하는 부분이 존재한다. (소스 설치인경우)

```
# pick up any necessary environment variables
if test -f /APP/httpd-2.4.39/bin/envvars; then
  . /APP/httpd-2.4.39/bin/envvars
fi
```

위와같이 envvars 파일을 로딩하는 부분이 존재한다면 envvars 파일 내에 umask 설정을 추가해주면 된다.

```
umask 0027     #<- 반드시 4자리 설정을 하여야 한다.
```

apachectl 자체를 수정하는 방법

만약 apachectl스크립트 내에 envvars를 로딩하는 부분이 없다면 (Redhat JBCS(Jboss core services)의 경우 존재하지 않음) apachectl 내에 직접 수동으로 umask 설정을 넣어주면 된다.

```
umask 0027  #<- 이경우에도 반드시 4자리로 설정한다.
```

위의 모든 방법을 써도 안되는경우 그냥 실행하는 계정의 umask값을 수정하는 수밖에 없다.

Host Header Poisoning / Injection

HTTP Host Header 변조 공격을 방지하기 위해서는 아래와 같이 설정하여야 한다. www.example.com 과 example.com 도메인만 허용할 경우

- **Apache HTTPD** 의 경우

```
RewriteEngine On
RewriteCond %{HTTP_HOST} !^(www.example.com|example.com)$ [NC]
RewriteRule .* - [F]
```

또는 VirtualHost 설정에서 기본 호스트를 최상단에 두어 걸러지도록 설정한다.

```
<VirtualHost _default_:*>
DocumentRoot /www/default
</VirtualHost>
```

- **JBoss EAP 7** 의 경우 Undertow 하위 시스템의 expression-filter 를 사용하십시요

```
/subsystem=undertow/configuration=filter/expression-filter=host-
checker:add(expression="not(equals(%{i,Host}, www.example.com) or
equals(%{i,Host}, example.com)) -> response-code(403)")
/subsystem=undertow/server=default-server/host=default-host/filter-ref=host-
checker:add
```

결과 xml 파일은 아래와 같습니다.

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
    <buffer-cache name="default"/>
    <server name="default-server">
        ...
        <host name="default-host" >
            ...
            <filter-ref name="host-checker"/>
        </host>
    </server>
    <filters>
        ...
        <expression-filter name="host-checker"
expression="not(equals(%{i,HOST}, www.example.com) or equals(%{i,HOST},
example.com)) -> response-code(403)"/>
    </filters>
</subsystem>
```

- **JBoss EAP 6**의 경우 rewrite 하위 시스템을 사용하십시요

```
<subsystem xmlns="urn:jboss:domain:web:2.1" default-virtual-server="default-
host" native="false">
    <connector name="http" protocol="HTTP/1.1" scheme="http" socket-
binding="http"/>
    <connector name="ajp" protocol="AJP/1.3" scheme="http" socket-
binding="ajp"/>
    <virtual-server name="default-host" enable-welcome-root="true">
        <alias name="localhost"/>
        <alias name="example.com"/>
        <!-- added -->
        <rewrite pattern="^/(.*)$" substitution="-" flags="F">
            <condition test="%{HTTP:HOST}"
pattern="!(www.example.com|example.com)" flags="NC"/>
        </rewrite>
    </virtual-server>
</subsystem>
```

참조링크

- https://access.redhat.com/solutions/3166341
- https://access.redhat.com/solutions/3057511

Content-Security-Policy (CSP)

누락된 Content-Security-Policiy (CSP) 헤더는 HTTP 응답 메시지에 CSP 헤더가 없는 취약점입니다. CSP는 Same-Origin-Policy(SOP) 에 근간을 둔 콘텐츠 보안 정책입니다. CSP는 신뢰할 수 있는 콘텐츠 소스의 허용 목록을 생성하고, 브라우저가 이런 소스에서 받은 리소스만을 실행하거나 렌더링

하도록 합니다. 공격자는 이 취약점을 이용하여 사용자의 브라우저에서 악의적인 클라이언트 스크립트 실행을 시도할 수 있습니다. 이를 통해 공격자는 사용자의 중요 정보 또는 권한을 탈취할 수 있으며, 사용자가 의도치 않은 행동을 하도록 유도할 수 있습니다. 이를 해결하기 위해서는 Content-Security-Policy 헤더를 추가하여 신뢰할 수 없는 소스에서 리소스를 받지 못하게 막아야 합니다.

적용방법

```
<IfModule mod_headers.c>

# 누락된 Content-Security-Policy 헤더 취약점을 해결하기 위해서는 웹 애플리케이션 서버에서 Content-Security-Policy 헤더를 추가해야 합니다. 모든 웹 사이트 내의 콘텐츠를 서브 도메인을 제외한 같은 도메인에서만 제공 받으려면 다음과 같이 사용합니다.
Header set Content-Security-Policy "default-src 'self'"

# 서브 도메인을 포함시키려면 다음과 같이 사용합니다.
Header set Content-Security-Policy "default-src 'self' *.mydomain.com"

# 이외에도 신뢰할 수 있는 도메인의 리소스를 특정 웹 페이지 요소에서만 사용하도록 설정할 수도 있습니다.또한, 신뢰할 수 없는 도메인의 리소스가 사용되지는 것을 막지 않고 사용되었다는 정보만을 제공받으려면 다음과 같이 사용합니다.
Header set Content-Security-Policy-Report-Only "policy"

# 특정 사이트의 특정 리소스를 포함하려면 아래처럼 적용합니다.
Header set Content-Security-Policy "default-src 'self'; img-src
*.cloudflare.com; script-src 'self' www.google-analytics.com
*.cloudflare.com"

</IfModule>
```

참조링크

- https://content-security-policy.com/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

X-Content-Type-Options

누락된 X 콘텐츠 타입 옵션 취약점은 HTTP 응답 메시지 내에 nosniff 플래그가 설정된 X-Content-Type-Options 헤더가 없는 취약점입니다. nosniff 플래그는 MIME 타입 스니핑을 막습니다. MIME 타입 스니핑이란 MIME 타입이 없거나 잘못 설정됐다 판단되었을 때, 브라우저가 MIME 타입을 추측하여 판단하는 기능을 의미합니다. 공격자는 악의적인 코드가 포함되어 있는 비실행 MIME 타입을 MIME 타입 스니핑을 통해 실행 MIME 타입인 것처럼 속여서 웹 애플리케이션에서 악의적인 코드를 실행하게 할 수 있습니다. 이를 통해 공격자는 악의적인 코드를 웹 애플리케이션에 업로드하여 해당 코드를 웹 애플리케이션이 반복적으로 실행하도록 할 수 있습니다. 이를 해결하기 위해서는 HTTP 응답 메시지에 값이 nosniff인 X-Content-Type-Options 헤더를 포함시켜야 합니다.

적용방법

```
<IfModule mod_headers.c>
    Header set X-Content-Type-Options nosniff
</IfModule>
```

참조링크

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

X-Frame-Options

누락된 X 프레임 옵션 취약점은 HTTP 응답 메시지 내에 X-Frame-Options 헤더가 없거나 올바른 값이 아닌 취약점입니다. X-Frame-Options 헤더는 frame, iframe, object 태그 안의 페이지를 브라우저가 렌더링할지 여부를 설정합니다. 공격자는 악의적인 사이트를 frame, iframe 또는 object 태그 안에 렌더링하여 클릭잭킹을 시도할 수 있습니다. 클릭잭킹이란 사용자가 자신이 클릭하고 있다고 인지하는 것과 다른 것을 클릭하게 속이는 공격 방식입니다. 이를 통해 공격자는 사용자의 정보를 유출하거나 사용자의 컴퓨터를 제어할 수 있습니다. 이를 해결하기 위해서는 HTTP 응답 메시지에 올바른 값을 가진 X-Frame-Options 헤더를 포함시켜야 합니다.

적용방법

```
<IfModule mod_headers.c>
    Header set X-Frame-Options DENY
</IfModule>
```

참조링크

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

X-XSS-Protection

이 헤더는 이제 **사용 중단(Deprecated)**되었고 모든 최신 브라우저에서 제거되었습니다. 오히려 이 기능 자체가 또 다른 취약점을 만들 수 있어 사용하지 않는 것이 표준이 되었습니다. 강력한 CSP 정책이 이 헤더의 역할을 완벽하게 대체하고도 남습니다. 따라서 이 헤더는 적용하지 않아야 합니다.

누락된 XSS 보호 헤더 취약점은 HTTP 응답 메시지의 X-XSS-Protection 헤더가 없거나 제대로 설정되어 있지 않은 취약점입니다. X-XSS-Protection 헤더는 비 지속적 크로스 사이트 스크립팅 감지 시, 페이지 로딩을 중단합니다. 공격자는 이 취약점을 확인하여 비 지속적 크로스 사이트 스크립팅 공격을 시도할 수 있습니다. 그 결과 공격자는 사용자의 정보를 탈취하거나, 서버로 하여금 의도하지 않은 동작을 수행하도록 강제할 수 있습니다. 이를 해결하기 위해서는 웹 애플리케이션의 설정을 변경하여 X-

XSS-Protection 헤더를 추가하고 그 값을 0으로 설정해야 합니다.

적용방법

```
<IfModule mod_headers.c>
Header set X-XSS-Protection "1; mode=block"
</IfModule>
```

참조링크

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

CSRF (Cross-Site Request Forgery)

웹 쿠키는 악의적인 사용자의 주요 공격 벡터인 경우가 많으므로 애플리케이션은 쿠키를 보호하기 위한 보안 속성을 사용해야 합니다. SameSite 속성으로 쿠키의 자사 또는 동일 사이트로 컨텍스트 제한 여부를 선택할 수 있습니다. 이는 사이트 간 요청 위조(CSRF)를 방지하기 위한 방법으로도 사용됩니다. 사이트 간 요청 위조(CSRF)를 막기 위해 SameSite는 Lax나 Strict로 사용하는 것이 좋습니다. 구글 크롬도 80버전으로 업데이트 되면서 SameSite 속성의 기본 값을 None에서 Lax로 상향 조절했습니다. 하지만 None으로 사용해야 한다면 반드시 Secure 속성과 함께 사용해야 합니다.

적용방법

```
<ifmodule mod_headers.c>
# none으로 사용할 경우
Header always edit Set-Cookie (.*) "$1; Secure; SameSite=None;"

# strict로 사용할 경우
Header always edit Set-Cookie (.*) "$1; SameSite=strict"
</ifmodule>
```

참조링크

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie#samesitesamesite-value

설정 예제

위의 설정을 이용하여 권장하는 설정방법 예제이다. 사이트 도메인이 atl.kr이고 추가적으로 Google Analytics를 이용한다고 할때를 가정해서 설정하면 아래와 같다.

```
```

```
<IfModule mod_headers.c>
    # ----------------------------------------------------------------------
----
    # 1. CSRF 공격 보조 방어 (SameSite Cookie Attribute)
    # ----------------------------------------------------------------------
----
  # 애플리케이션에서 발행하는 모든 쿠키에 SameSite=Lax 속성을 추가합니다.
    # 'Lax'는 대부분의 CSRF 공격을 막아주면서, GET 링크를 통한 사이트 간 이동은 허용하여
사용자 편의성을 유지합니다.
    # 'Strict'는 보안이 더 강력하지만, 다른 사이트에서 내 사이트로 링크를 타고 들어올 때 로그
인이 풀리는 등 불편을 줄 수 있습니다.
    Header edit Set-Cookie ^(.*)$ "$1; SameSite=Lax"

    # ----------------------------------------------------------------------
----
  # 2. 콘텐츠 보안 정책 (CSP)
    # ----------------------------------------------------------------------
----
    # atl.kr 도메인과 Google Analytics(GA4) 작동을 기준으로 정책을 구성합니다.
    Header set Content-Security-Policy "default-src 'self'; \
        script-src 'self' https://www.googletagmanager.com
https://www.google-analytics.com; \
        style-src 'self' 'unsafe-inline'; \
        img-src 'self' data: https://www.google-analytics.com
https://stats.g.doubleclick.net; \
        connect-src 'self' https://*.google-analytics.com; \
        font-src 'self'; \
        object-src 'none'; \
        frame-ancestors 'self'; \
        form-action 'self'; \
        base-uri 'self'; \
        upgrade-insecure-requests;"

    # ----------------------------------------------------------------------
----
  # 3. 기타 필수 보안 헤더
    # ----------------------------------------------------------------------
----
  # 클릭재킹(Clickjacking) 방어 (구형 브라우저 호환용)
    Header set X-Frame-Options SAMEORIGIN

    # MIME 타입 스니핑(Sniffing) 방어
    Header set X-Content-Type-Options nosniff

    # HSTS (Strict-Transport-Security): 모든 접속을 HTTPS로 강제
    Header set Strict-Transport-Security "max-age=31536000;
includeSubDomains; preload"

    # Referrer-Policy: 사용자 프라이버시 보호
    Header set Referrer-Policy "strict-origin-when-cross-origin"
```

```
    # Permissions-Policy: 브라우저 기능 접근 제어
    Header set Permissions-Policy "geolocation=(), midi=(), camera=(),
microphone=()"
</IfModule>
```