분석보고서

2024.12.23
이정현

취약점

php filter chain

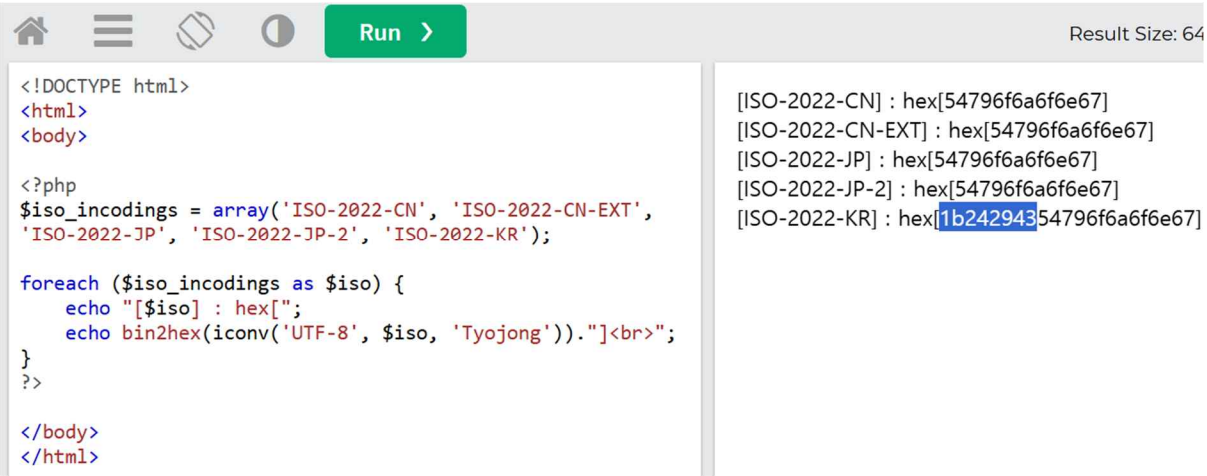
해당 취약점은 php에 존재하는 Wrapper와 Conversion filter을 사용해서 임의의 코드조각 또는 string을 생성하여 원격 코드 실행이 가능하도록 하는 취약점이다.

원리

여러 인코딩 방식이 있는데 php에서 iconv 명령이 활성화 된 경우(default) php://convert.iconv.*.* Wrappers를 사용해서 글자의 인코딩 형식을 변경하는 것이 가능하다. (wrapper란 실제 데이터의 앞에서 어떤 틀을 잡아 주는 데이터 또는 다른 프로그램이 성공적으로 실행되도록 설정하는 프로그램, 개발을 도와줌)

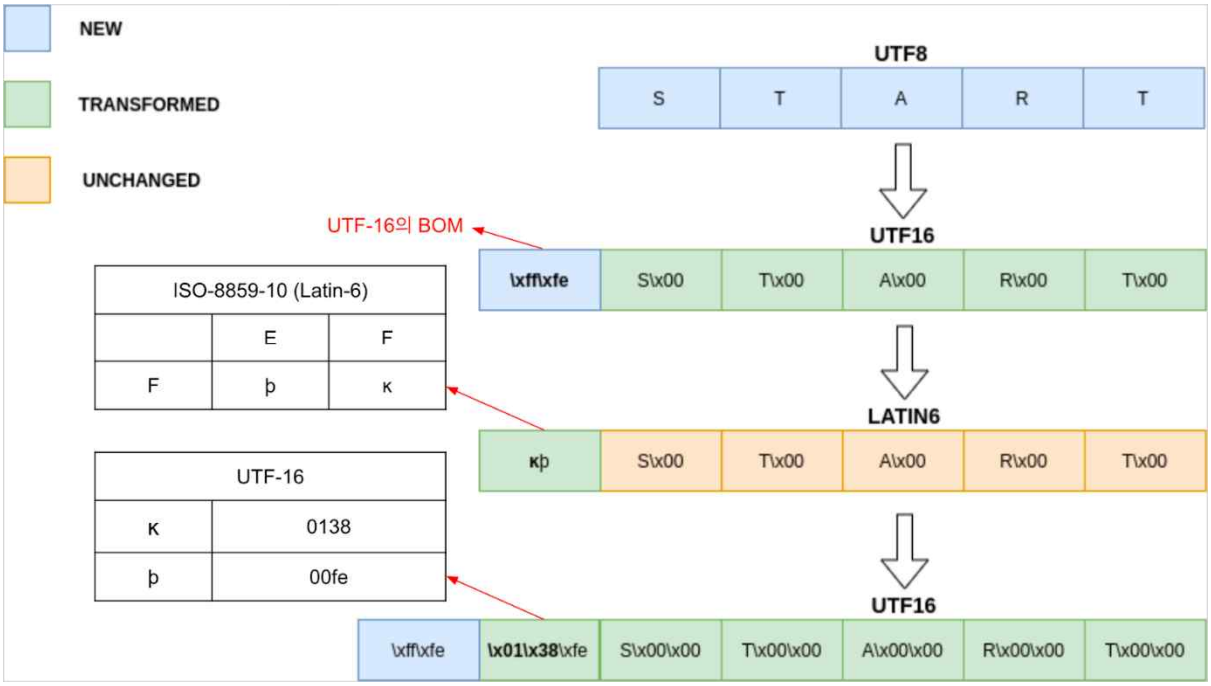
위 사진처럼 base64-decode함수는 =문자를 제대로 처리하지 못한다. 하지만 인코딩 형식을 UTF-7로 변경 후 디코딩하면 =문자를 처리하는 것을 확인 할 수 있다.

Unicode에서는 인코딩 방식을 알아내기 위해 BOM(Byte Order Mark)이라는 것을 사용한다. (눈에 보이지 않는 특정 바이트를 삽입해서 바이트 값을 보고 인코딩 형식을 알아낸다) 이것은 디코딩 시 무시되어 문자열만 출력되는데 @_>문자열은 BOM으로 인식되어서 원본 문자열에 영향을 주지 않는다.



다른 인코딩 방식들을 보면 모두 동일한 결과가 나오지만 유일하게 iconv함수에서 사용 가능한 인코딩인 ISO-2022-KR을 보면 앞에 0x1b, 0x24, 0x29, 0x43인 BOM값이 붙어서 출력되는 것을 알 수 있다.

그렇다면 우리가 원하는 문자열을 어떻게 만들어서 rce를 발생시킬 수 있을까?



예를 들어서 start라는 UTF-8형식의 문자열을 UTF-16으로 변경하면 앞에 0xff, 0xfe라는 UTF-16의 BOM값이 붙는다

UTF-16으로 인코딩 된 값을 다시 LATIN6으로 인코딩 시키면 앞에 0xff, 0xfe라는 BOM값도 인코딩 되어 ĸþ라는 값으로 바뀐다.

이 값을 다시 UTF-16으로 인코딩 시키면 맨 앞에 UTF-16의 BOM값이 붙고 ĸþ는 인코딩 되어 0x01, 0x38, 0x00, 0xfe값으로 바뀐다. 이때 0x38이 UTF-8에서 8로 정의 되어있기 때문에 없던 8이 추가로 붙어서 출력된다.

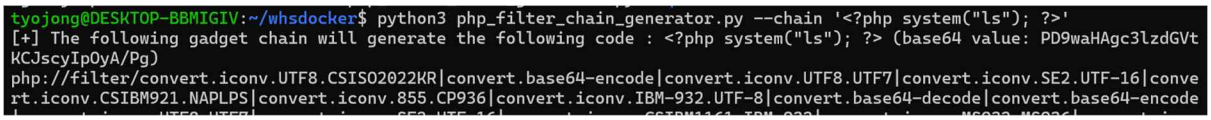
Run

```php
<!DOCTYPE html>
<html>
<body>

<?php
$str = "start";
echo "[UTF-8] <br>";
echo "[STR] : $str <br>";
echo "[hex] : ".bin2hex($str);
echo "<br>==========<br>";
echo "[UTF-8 -> UTF-16]";
$step1 = iconv('UTF-8', 'UTF-16', $str);
echo "[str] : $step1 <br>";
echo "[hex] : ".bin2hex($step1);
echo "<br>==========<br>";
echo "[LATIN6 -> UTF-16]";
$step2 = iconv('LATIN6', 'UTF-16', $step1);
echo "[str] : $step2 <br>";
echo "[hex] : ".bin2hex($step2);
?>

</body>
</html>
```

```
[UTF-8]
[STR] : start
[hex] : 7374617274
==========
[UTF-8 -> UTF-16][str] : ��start
[hex] : fffe7300740061007200740000
==========
[LATIN6 -> UTF-16][str] : ��8┌start
[hex] : fffe3801fe00730000007400000061000000720000007400000
```

이 로직을 이용해서 php쉘 코드를 만들어 php include가 사용된 위치에 입력값으로 넣

어준다면 include는 php 코드로 해석되는 부분이 있어 실행시킨다.(require도 가능) 하지만 file_get_contents함수를 사용하게 되면 문자열로 처리를 하기 때문에 해당 취약점을 대응할 수 있다.(하지만 wrapper를 사용한 LFI는 발생할 수 있음)



php filter chain에서 생성된 코드를 보면 php://처럼 wrapper를 사용한 것을 볼 수 있는데 php filter chain의 경우 궁극적으로 php의 코드를 생성하여 그 코드가 실행되도록 만드는 것이기 때문에 일반적인 wrapper를 이용한 lfi취약점과는 다르다. (filter chain으로 생성된 php코드가 실행이 안되는 것이지 wrapper자체는 작동한다.)

이러한 인코딩을 이용해 BOM값 대신 우리가 원하는 문자열을 넣어서 base64 문자열의 무결성을 손상시키지 않고 문자를 추가해서 php filter chain에 이용할 수 있다.

추가적으로

Filter chain 맨뒤에 붙는 /resource=php://temp는 인코딩 과정 시 변하게 되는 문자열 원본을 유지하기 위해 붙인다.

참고자료

https://www.synacktiv.com/publications/php-filters-chain-what-is-it-and-how-to-use-it
https://book.hacktricks.xyz/pentesting-web/file-inclusion/lfi2rce-via-php-filters#improvements
https://mokpo.tistory.com/542#toc-%EB%B6%84%EC%84%9D
https://y0un.tistory.com/62
https://www.rfc-editor.org/rfc/rfc2781#section-3.2
https://learn.dreamhack.io/15#39