CVE-2022-21662

ZeroDiglett Team
Marimo, '24.12.31
guswjddl4313/ZeroDiglett

1/0-Day Vulnerability Project
SMU Univ., Project began on December 23, 2024.

Vulnerability Type & Impact Version

Type : Stored XSS Vulnerability.

Impact Version : WordPress < 5.8.3, bbPress < 2.5.9

Vulnerability Description

해당 취약점(CVE-2022-21662)은 WordPress 5.8.2 이하의 버전에 영향을 미치는 Stored Cross-Site Scripting 취약점이다. 공격자가 JavaScript 페이로드를 삽입하면 해당 페이로드는 데이터베이스에 저장된다. 이후 관리자 대시보드와 같은 다양한 사용자 인터페이스를 감염 시켜 공격자가 관리자 세션을 탈취 할 수 있다.

Principle of Vulnerability Occurrence

WordPress 5.8.2 버전에서 발견된 Stored XSS 취약점(CVE-2022-21662)은 게시물 슬러그에 악성 자바스크립트 코드를 삽입함으로써 발생한다. 슬러그는 URL의 일부로 사용되며, 예를 들어 *www.example.com/blog/the-post-title*에서 *the-post-title* 부분이 슬러그이다.

wp-includes/post.php

```php
if ( empty( $post_name ) ) {
	if ( ! in_array( $post_status, array( 'draft', 'pending', 'auto-draft' ), true ) ) {
		$post_name = sanitize_title( $post_title );
	} else {
		$post_name = '';
	}
} else {
	// On updates, we need to check to see if it's using the old, fixed sanitization context.
	$check_name = sanitize_title( $post_name, '', 'old-save' );

	if ( $update && strtolower( urlencode( $post_name ) ) == $check_name && get_post_field( 'post_name', $post_ID ) == $check_name ) {
		$post_name = $check_name;
	} else { // new post, or slug has changed.
		$post_name = sanitize_title( $post_name );
	}
}
```

슬러그는 해당 코드대로 *sanitize_title()* 함수를 통해 게시물 제목을 URL 또는 HTML 속성에서 사용할 수 있는 슬러그로 변환되며 아래의 과정을 따른다.

*** 슬러그 생성 규칙**

1. 제목에서 특수 문자를 제거 (제거 시, 공백은 하이픈(-)으로 변환)
2. 악센트가 포함된 문자(예: á, ü, ç 등)를 ASCII 형태로 변환
3. 소문자 변환
4. 중복 하이픈 제거
5. 동일한 슬러그가 이미 존재하면 -2, -3 같은 숫자가 뒤에 추가
6. 사용자 수동 편집 가능
7. 슬러그에는 알파벳(a-z), 숫자(0-9), 언더바(_), 하이픈(-)만 포함

- **ex)** "Test Title! 2025" → test-title-2025

wp-includes/formatting.php

```php
function sanitize_title_with_dashes( $title, $raw_title = '', $context = 'display' ) {
	$title = strip_tags( $title );
	// Preserve escaped octets.
	$title = preg_replace( '|%([a-fA-F0-9][a-fA-F0-9])|', '---$1---', $title );
	// Remove percent signs that are not part of an octet.
	$title = str_replace( '%', '', $title );
	// Restore octets.
	$title = preg_replace( '|---([a-fA-F0-9][a-fA-F0-9])---|', '%$1', $title );
```

조금 더 자세한 과정을 코드로 보기 위해 *formatting.php*로 이동 후 *sanitize_title_with_dashes()* 함수를 확인해보면, *preg_replace()* 함수를 사용해 정규 표현식을 토대로 슬러그를 생성하는 것을 볼 수 있다. 해당 코드를 자세히 분석한 결과는 아래와 같다.

preg_replace: PHP에서 제공하는 함수로, 정규 표현식을 사용해 문자열에서 특정 패턴을 찾아 대체하는 데 사용된다.

```
// Preserve escaped octets.
$title = preg_replace( '|%([a-fA-F0-9][a-fA-F0-9])|', '---$1---', $title );
```

정규 표현식: |%([a-fA-F0-9][a-fA-F0-9])|

- |는 정규식 구분자.
- %는 퍼센트 기호 %를 찾음.
- [a-fA-F0-9]: 한 자리의 16진수 문자 (a-f, A-F, 또는 0-9)를 의미.
- [a-fA-F0-9][a-fA-F0-9]: 두 자리의 16진수 문자.
- ()는 캡처 그룹. % 뒤의 두 자리 16진수 부분을 캡처.

대체 문자열: '---$1---'

- $1은 캡처 그룹의 내용. % 뒤의 두 자리 16진수 값이 $1로 참조.
- ---$1---는 캡처된 값을 ---로 감싸는 형태로 변환.

예시: %20 → ---20---

```
// Remove percent signs that are not part of an octet.
$title = str_replace( '%', '', $title );
```

str_replace(): %를 ''으로 대체. 즉, $title에서 모든 % 문자가 제거.

```
// Restore octets.
$title = preg_replace( '|---([a-fA-F0-9][a-fA-F0-9])---|', '%$1', $title );
```

정규 표현식: |---([a-fA-F0-9][a-fA-F0-9])---|

- ---: 세 개의 하이픈 ---이 정확히 일치해야 함.
- ([a-fA-F0-9][a-fA-F0-9]): 두 자리 16진수 값을 캡처.
- 캡처 그룹 () 부분은 $1로 참조.

대체 문자열: '%$1'

- 캡처된 두 자리 16진수 값을 다시 %로 감싼 형태로 복원.

예시: ---20--- → %20

분석 결과 정규 표현식에서 볼 수 있듯이, 슬러그는 URL 인코딩된 문자를 포함할 수 있다. 일반적으로 URL 인코딩된 문자열은 다시 디코딩 되지 않는 한 크게 문제가 되지 않지만, 만약 이 문자열이 디코딩 되고 그대로

HTML이나 JavaScript로 렌더링 된다면, XSS 공격이 가능하다.

wp-includes/post.php

```
function _truncate_post_slug( $slug, $length = 200 ) {
        if ( strlen( $slug ) > $length ) {
                $decoded_slug = urldecode( $slug );
                if ( $decoded_slug === $slug ) {
                        $slug = substr( $slug, 0, $length );
                } else {
                        $slug = utf8_uri_encode( $decoded_slug, $length, true );
                }
        }

        return rtrim( $slug, '-' );
}
```

다시 *post.php*로 돌아와서 *_truncate_post_slug()* 함수를 찾아보면 *urldecode()* 함수를 사용해 슬러그를 URL 디코딩 하는 것을 볼 수 있다. *_truncate_post_slug()*는 슬러그의 길이를 제한하는 함수로 이 과정에서 URL 디코딩과 재인코딩이 발생한다.

$decode_slug === $slug: 디코딩된 결과가 원래 슬러그와 동일하면, 슬러그는 URL 인코딩 되지 않은 상태라고 판단 후, 슬러그를 *substr()* 함수를 사용해 단순히 *$length*를 토대로 잘라낸다.

$decode_slug !== $slug: 동일하지 않은 경우, 슬러그는 URL 인코딩된 상태이므로 디코딩된 결과를 *$length*에 맞게 자른 후 다시 인코딩 하여 저장한다.

디코딩된 슬러그를 재인코딩을 할 때 *utf8_uri_encode()* 함수를 사용한다. 이때 *utf8_uri_encode()*가 URL 인코딩의 완벽한 대응 함수가 아니라는 점에서 취약점이 발생한다. *utf8_uri_encode()* 함수는 Unicode 문자만 인코딩하고, 일반적인 HTML 태그나 ASCII 문자는 그대로 남기게 된다.

ex) 입력: <script>alert(66)</script> → 결과: <script>alert(66)</script>

이로써 URL 디코딩된 값이 완전히 재인코딩 되지 않고 잠재적으로 위험한 문자열이 남아있게 된다.

wp-includes/post.php

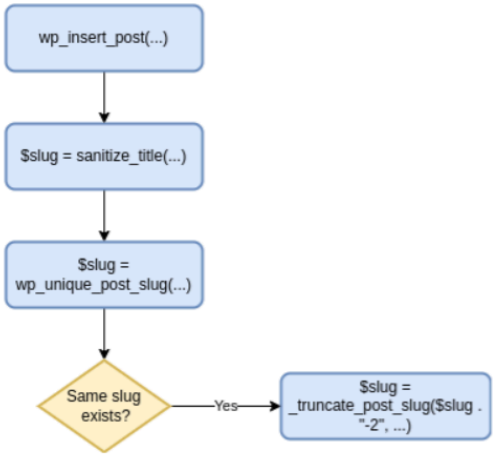
```php
function wp_unique_post_slug( $slug, $post_ID, $post_status, $post_type, $post_parent ) {
        if ( in_array( $post_status, array( 'draft', 'pending', 'auto-draft' ), true )
                || ( 'inherit' === $post_status && 'revision' === $post_type ) || 'user_request' === $post_type
        ) {
                return $slug;
        }
```

_truncate_post_slug() 함수는 위의 *wp_unique_post_slug()* 함수에서 주로 호출되며 *wp_unique_post_slug()* 함수는 게시물 저장 과정에서 슬러그가 중복 되지 않도록 처리한다.

예를 들어 앞서 슬러그 생성 규칙대로 이미 존재하는 슬러그의 경우, 새로운 게시물이 동일한 슬러그를 가지지 않도록 *-2, -3* 등의 숫자를 추가해 중복되지 않도록 처리한다.

만약 슬러그의 길이가 길다면*_truncate_post_slug()* 함수를 호출하여 길이를 제한하는데 이 과정에서 위에서 설명한 취약점이 발생한다.

Scenario



1. 게시물 A를 생성한다. 게시물 A의 슬러그는 **URL 인코딩된 JS 페이로드 + 길이 제한을 넘기기 위한 문자열**
2. 게시물 B를 생성한다. B의 슬러그를 A와 동일하게 설정한다.

두 번째 게시물에 슬러그를 설정할 때, 페이로드는 이미 같은 슬러그를 가진 게시물이 있으므로 데이터베이스에서 디코딩 되고, 대체 슬러그는 *wp_unique_post_slug()* → *_truncate_post_slug()*를 거쳐서 계산된다. 디코딩은 특정 길이(200자)를 초과해야 발생하기 때문에 슬러그에 문자열을 추가할 필요가 있다.

Test

해당 시나리오대로 취약점을 이용해 XSS 공격을 시도해도 슬러그는 URL에 안전한 문자인 알파벳, 숫자, 하이픈, 언더바만 포함되도록 설정되어 있기 때문에 escape 처리 없이 출력 되는 위치를 찾아야 했다. 그러한 위치 중 하나는 관리자 패널의 메인 게시물 목록이다.

관리자는 WordPress 관리 대시보드에 접속하여 게시물을 관리하고 게시물의 슬러그가 HTML 페이지에서 그대로 출력되기 때문에, JavaScript 페이로드가 HTML 응답 페이지에 삽입되고 관리자가 게시물 목록 페이지를 열면, 브라우저에서 실행된다.

이후 JavaScript 코드는 관리자 브라우저에서 실행되므로, 공격자는 해당 코드를 통해 관리자의 세션이나 인증 정보를 탈취할 수 있다.

bbPress

WordPress는 사용자마다 다른 역할(예: 관리자, 편집자, 작성자, 구독자)을 부여하는 권한 기반 시스템을 사용한다. 따라서 앞서 얘기한 취약점은 작성자 권한이 있는 공격자에 의해서 만 악용이 가능하다. 왜냐하면 게시물의 슬러그에 대한 제어가 필요하기 때문이다. 특별한 권한이 필요하지 않아도 취약점을 발생시키기 위해 조사한 결과 WordPress의 bbPress 플러그인이 설치되었을 때 가능했다.

bbPress 플러그인은 WordPress에서 포럼 기능을 제공하는 플러그인이다. 해당 플러그인은 사용자들이 포럼 주제(topic)을 작성할 수 있도록 허용한다. 여기서 포럼 주제(topic)은 WordPress 내부적으로 게시물로 관리되며 사용자가 입력한 제목을 기반으로 슬러그가 자동 생성된다.

ex) 제목: test-topic → **topic URL:** www.example.com/forum/topic/test-topic

bbPress는 모든 포럼 사용자가 제목을 설정할 수 있도록 허용하기 때문에 제목에 JavaScript 페이로드를 삽입해 앞서 얘기한 시나리오대로 공격하면 취약점을 악용할 수 있다.

🌐 WordPress 5.8.2 bbPress Stored XSS

Patch

🌐 WordPress 5.8.3 Security Release

- 해당 취약점은 WordPress 5.8.3에 패치 되었다. 보안 릴리스에 따르면 슬러그 처리 과정에서 입력 값을 철저히 검증하고, 특수 문자를 적절히 escape 하여 악의적인 코드가 실행되지 않도록 패치 하였다.

🌐 bbPress 2.6 – Better Great Than Never

- bbPress 플러그인도 새 버전인 bbPress 2.6.0이 출시되면서 취약점이 패치 되었다. 새 버전인 2.6.0에서는 topic 제목 최대 길이에 대한 서버 측 검증이 추가되어 악용이 불가능하다.

Others

- *utf8_uri_encode()* 대신 다른 encode 함수를 사용하여 일관된 디코딩/인코딩 함수 사용.

- **ex)** *rawurlencode()* 등.

Reference

Ehtisham Siddiqui (2022). WordPress: Stored XSS through authenticated users. wordpress-develop. Available at https://github.com/WordPress/wordpress-develop/security/advisories/GHSA-699q-3hj9-889w.

Karim El Ouerghemmi (2022). WordPress 5.8.2 Stored XSS Vulnerability. sonar. Available at https://www.sonarsource.com/blog/wordpress-stored-xss-vulnerability/.