How To Build Advanced Threat Intelligence Queries Utilising Regex and TLS Certificates - (BianLian)

By Matthew

Published: 2023-11-27 · Archived: 2026-04-05 17:40:11 UTC

In this post we will investigate a Bianlian C2 address and use TLS certificates to obtain another ~50 servers.

Our primary focus will be building a regex query that matches the unique TLS structure used by recent BianLian.

Regex and TLS Certs present a great opportunity to build queries without introducing geographic limitations like ASN numbers and hosting providers. By using TLS Certificates, we can also avoid limiting searches to specific port numbers or port counts. Effectively catching actors that avoid patterns demonstrated in previous posts. (Eg here and here)

Our final query will look something like this.

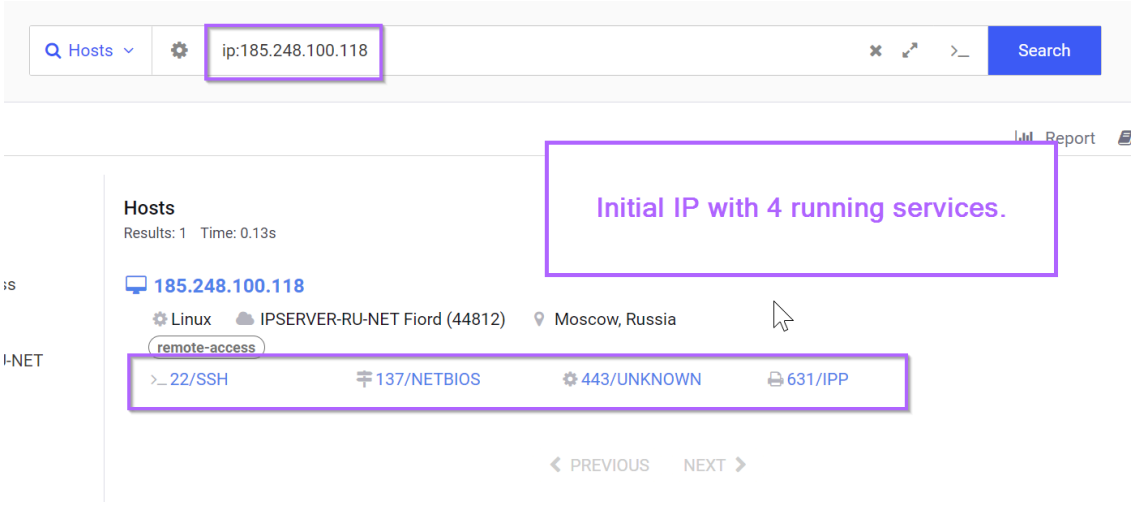
```
services.banner_hashes="sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855" and services.tls.certificates.leaf_data.subject_dn:/C=.{16}, O=.{16}, OU=.{16}/
```

> **Note that regular expression searches are a paid feature of Censys. But for the purposes of demonstrating interesting concepts I thought this blog would be useful**
>
> At the end of this blog, there is a query that will catch the same certificates without utilising Regex. This can be used if you don't have a paid Censys account.

Performing an Initial Search

We can begin the search by looking up the initial IP of `185.248.100[.]118` which was obtained on ThreatFox.

> Note that we can prepend `ip:` if we want an overview before jumping directly to the IP page.

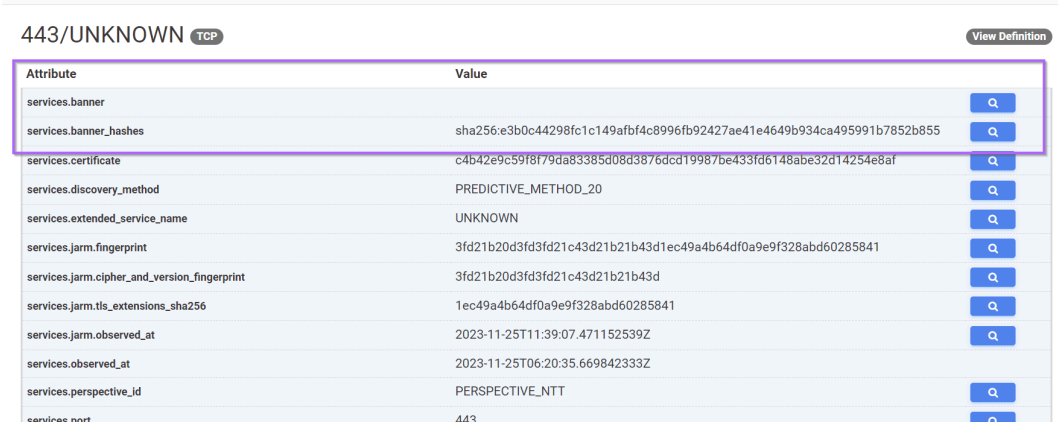


Clicking on the IP Address takes us to the full page, where we can see an interesting TLS certificate on port 443.

If we click on the "view all data" tab in the top right of the above screenshot, we can view the full information for port 443.

> This enables us to see all available information for a given port, which is typically significantly more than what is available within the default summary view.

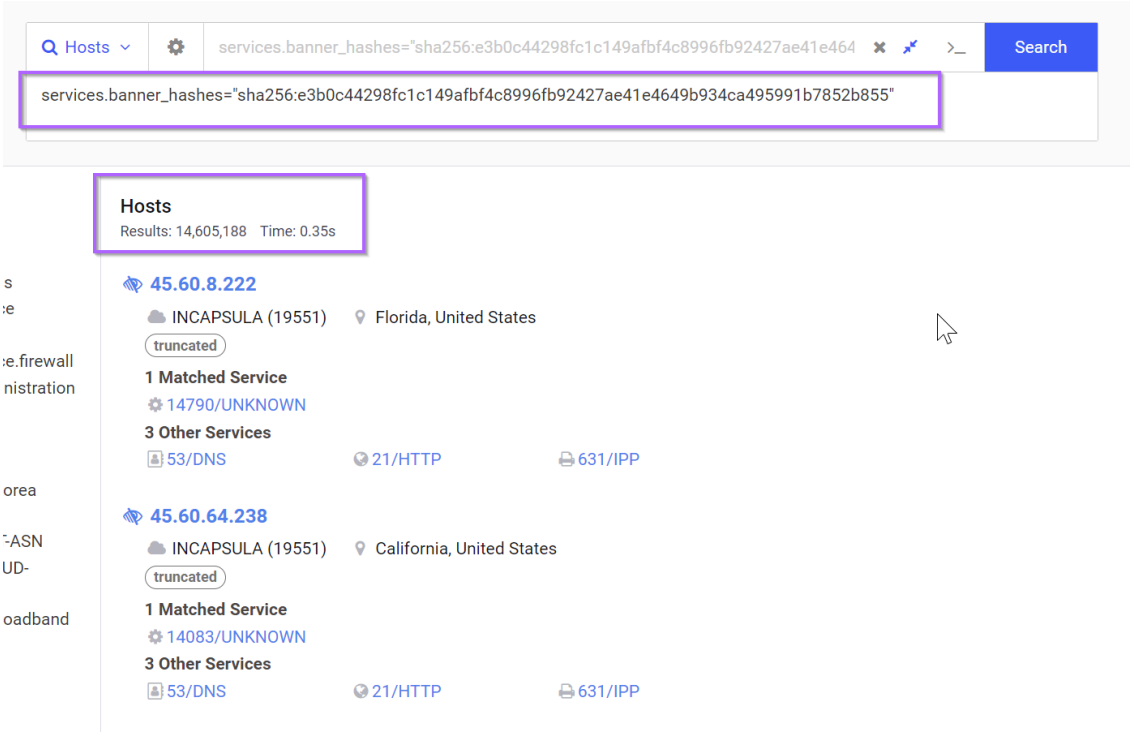
The first thing that stands out is the empty `services.banner` field. A lack of service banner (although boring at initial glance) could be a great pivot point.



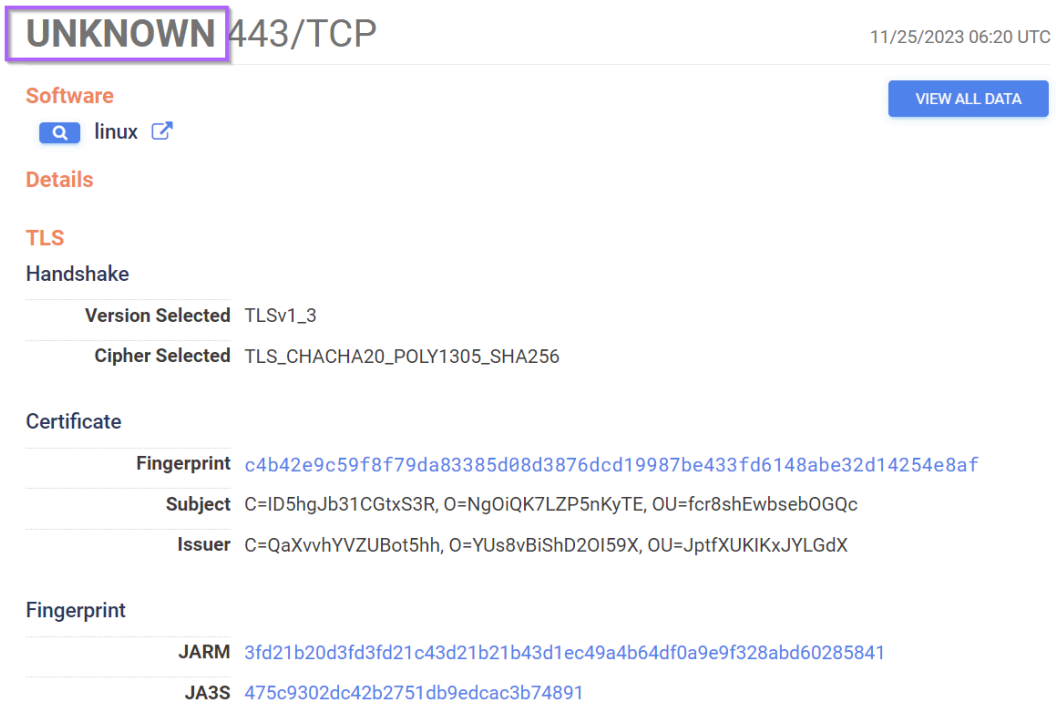
We can click on the blue search box next to the `services.banner` field to perform a pivot. This will pre-build the query and save us time finding the exact field to search.

> It's often better to use the banner hash when we want an exact match, this will avoid any issues where a value might be incorrectly typed when searching. This also ensures an exact match is queried.

Performing a basic pivot from the banner hash, we have an initial result of `14,605,188` servers. An admittedly huge number, But we'll improve this later.

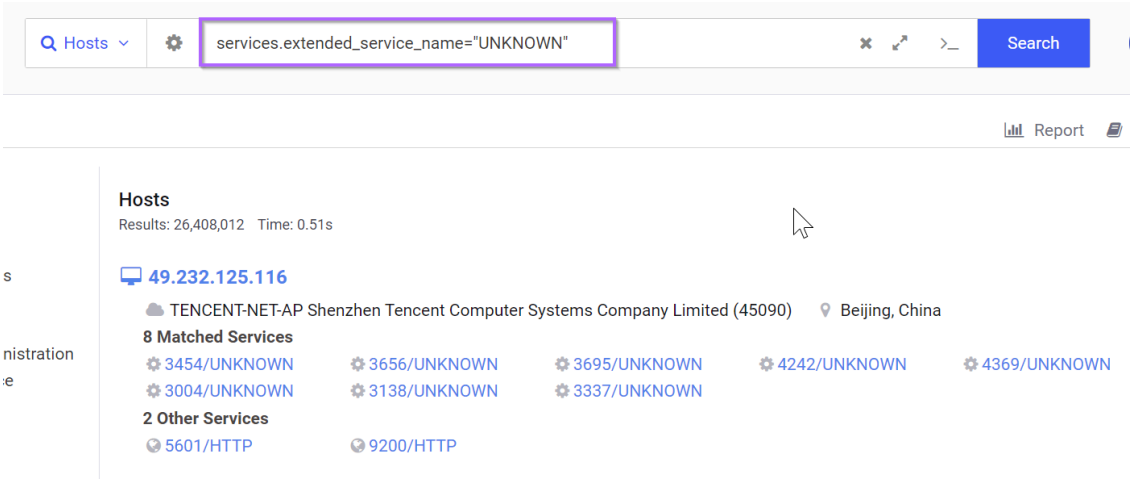


If we go back to the initial summary view, we can see that the service was identified as "Unknown".



We can again use the "view all data" to find the "Unknown" field and perform a pivot with the search box.

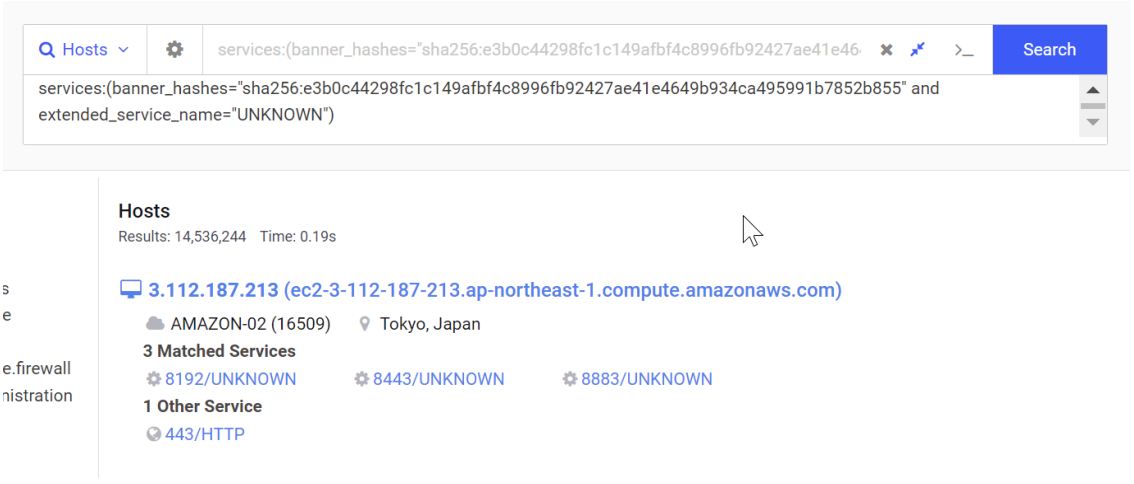
This returns a lot of results, so we can take the resulting query and add it to our initial empty banner search.



Sadly this doesn't actually reduce the hits to any useful degree (in this case), but it's a useful technique to know so I wanted to include it in the post.

Since the addition of `UNKNOWN1 only reduced the hits by a very very small degree, it doesn't matter whether we include it or not for the remainder of this search.

For the remaining queries we can leave it out.

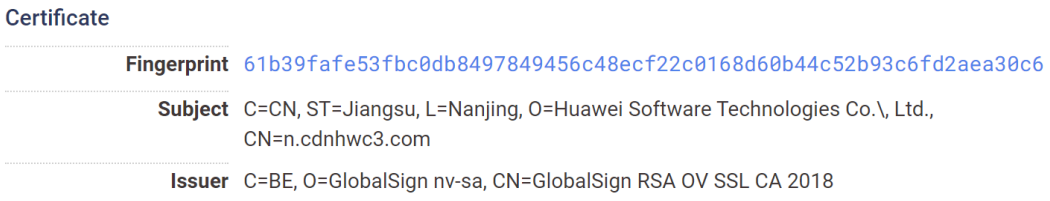


Moving on, we can go back to the field details page and observe the certificate information in more detail.

Of particular interest here is that only the `C=,O= and OU=` fields are present in the subject and issuer fields.

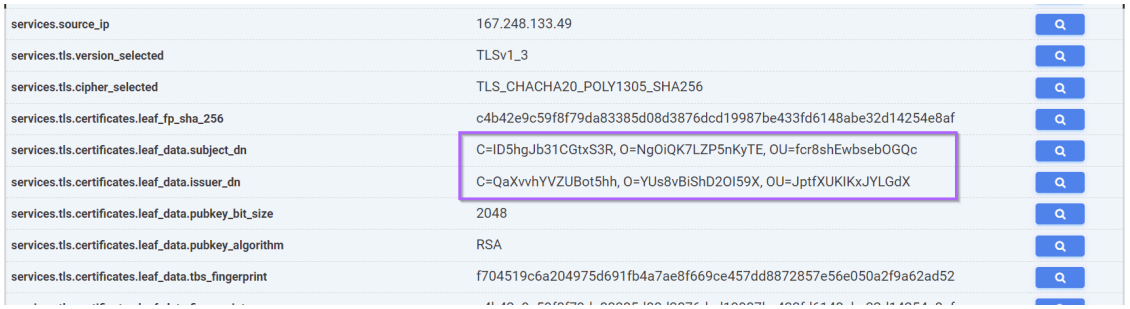
> Typically there are additional values like Location `L` , State `ST` and others. The absence of these values is an additional indicator that can be used.

For example, a "normal" certificate should look more like this.

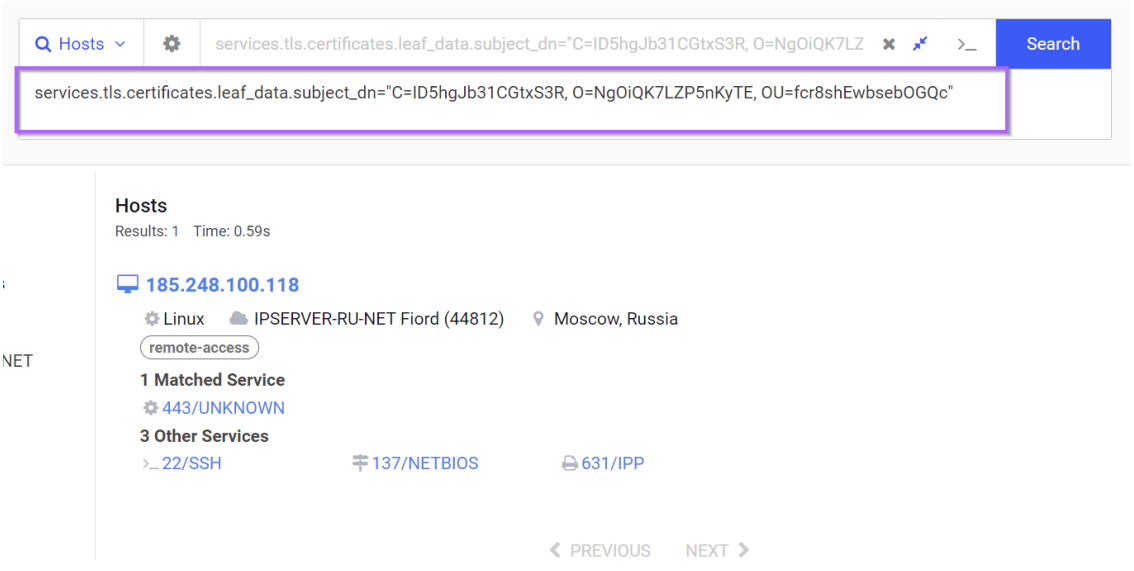


We can also note that all of the `C=,O=,OU=` values are exactly 16 characters in length.

The lack of "regular" fields and presence of exactly 16 characters is itself a pattern worth signaturing.



We can use the blue search box again (right side of the detailed fields view) to automatically build an exact query.



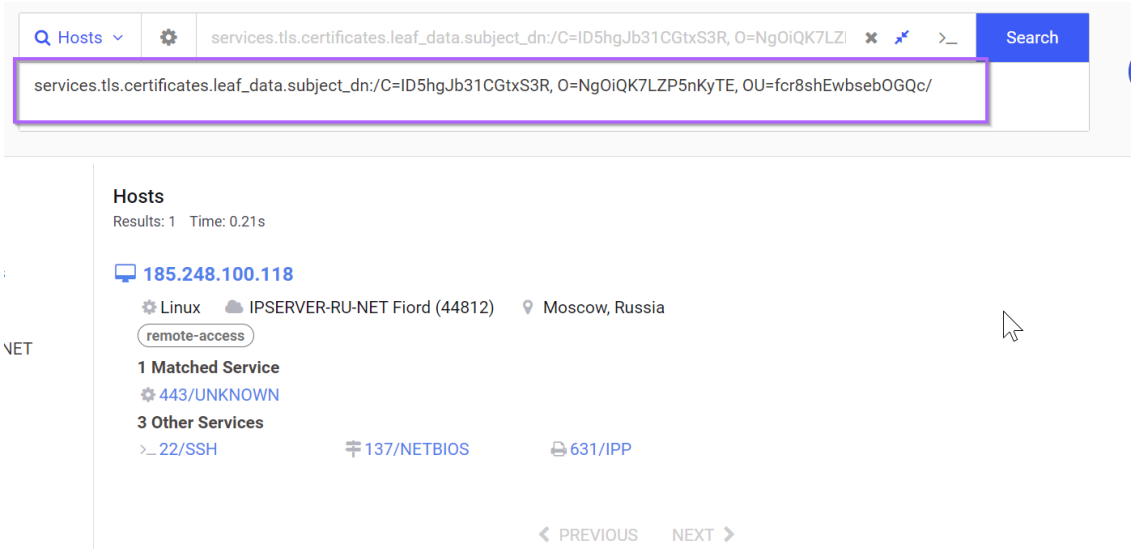
Converting a Field to Regex

The initial pre-built query is an exact string search. (Indicated by the presence of `=` and the entire thing encased in quotes)

`services.tls.certificates.leaf_data.subject_dn="C=ID5hgJb31CGtxS3R, O=NgOiQK7LZP5nKyTE, OU=fcr8shEwbsebOGQc"`

We can turn this into a regex by adjusting the equals `=` to a colon `:` and changing the double quotes `"` to forward slashes `/`

We can then validate the syntax by modifying the query and ensuring the same result is returned.

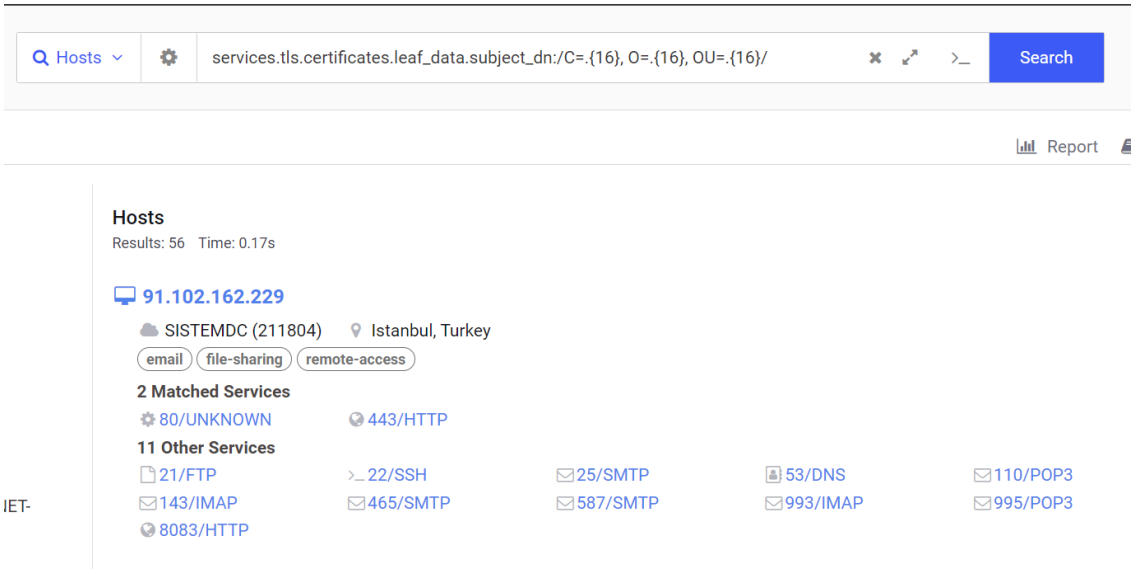


After validating the syntax, we can remove the main values and replace them with `.{16}`.

This ensures that exactly 16 characters are present following any of the `C=, O= or OU=` values.

> In a real example you would probably want to avoid the `.` wildcard and use a more specific query like `[a-zA-Z0-9]{16}`, `[^,]{16}` or `\w{16}`. But for this example we will keep it simple with a `.`

Adjusting the query returns `56` results for matching certificates.



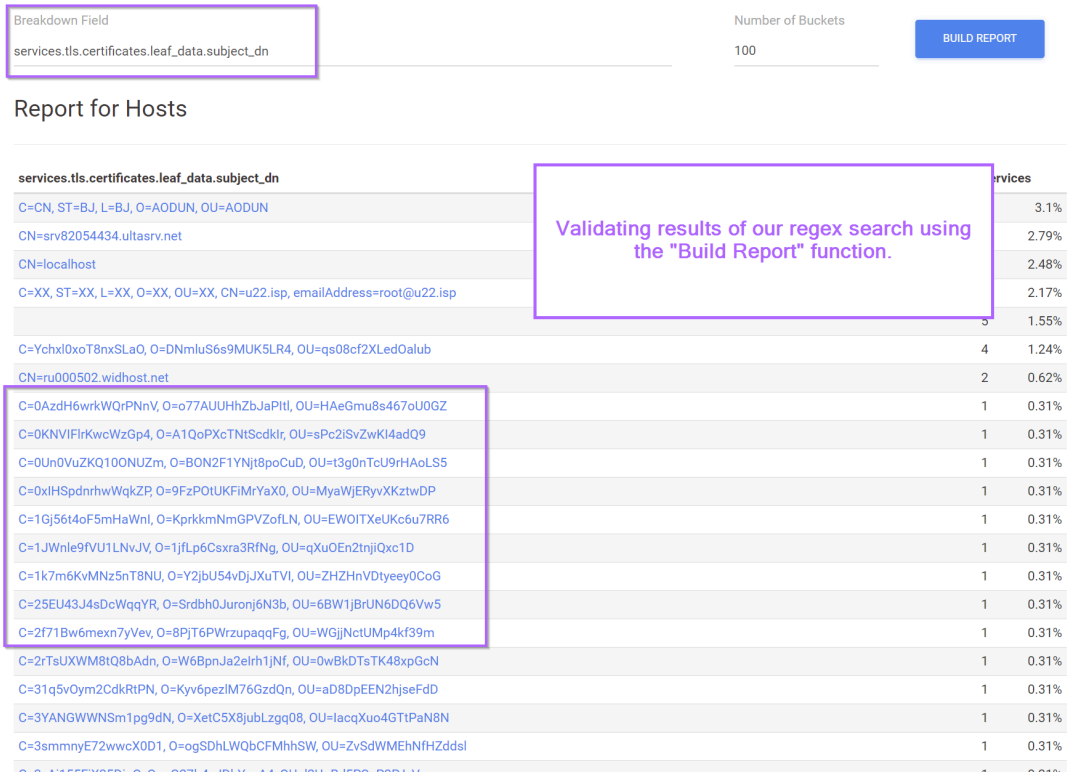
Validating a Regex Query with the "Build Report" Feature

By using the "report" button in the top-right corner, we can view a specific field for all of our returned results.

This can be extremely useful for validating that results are matching as intended.

> Note in the screenshot below, that the "build report" function will show results for other services running on returned servers.

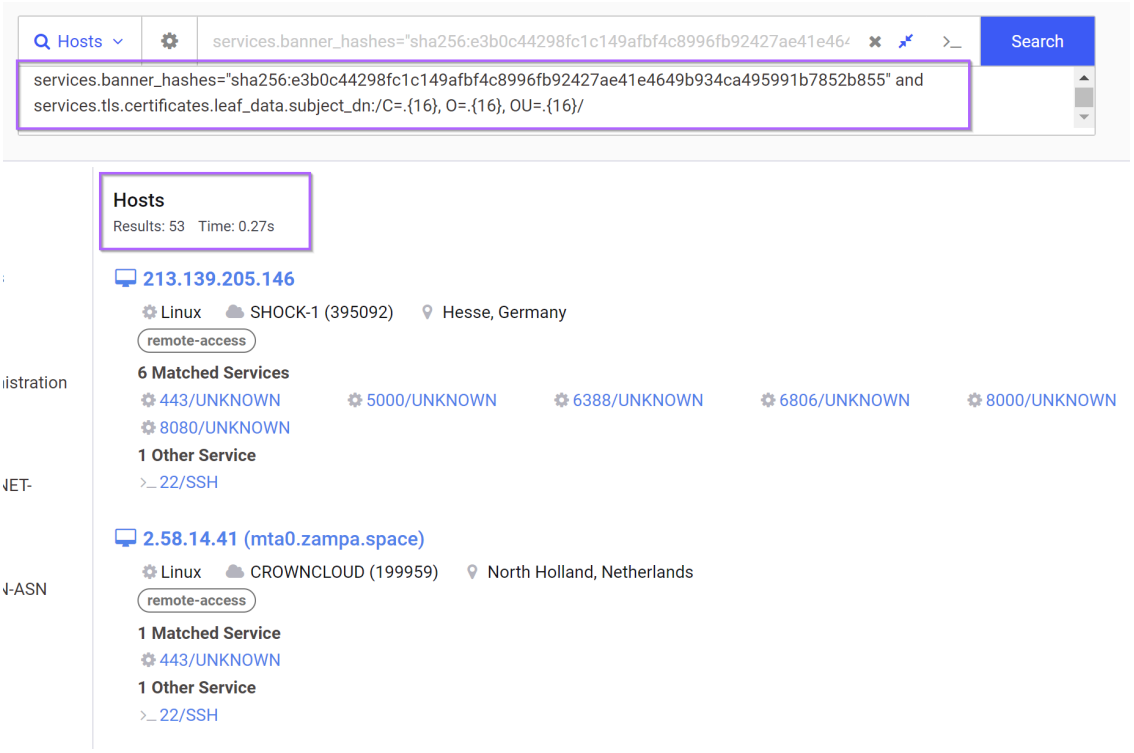
> For example, a server with a matching hit on port 443, may also have a non-matching hit on port 8080. If both services contain a relevant field, they both will be displayed. Hence the additional results that don't match our query.
>
> Either way this function is still useful for validating results.



With the query now validated further, we can add it to our initial empty banner search.

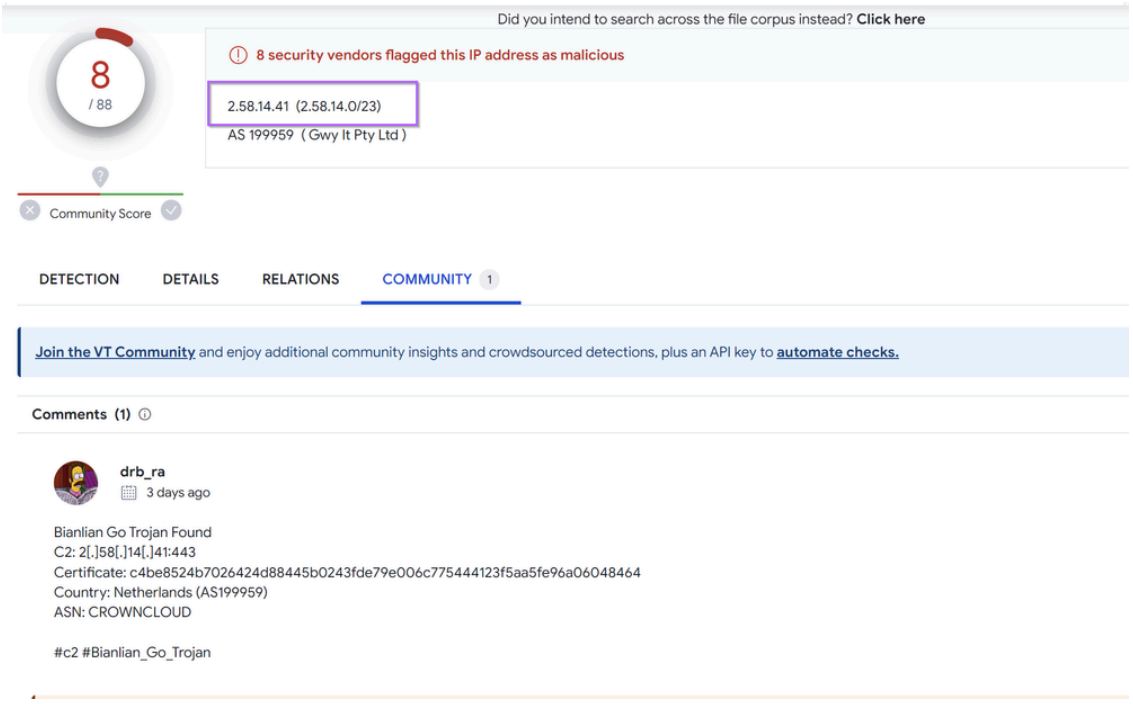
`services.banner_hashes="sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855" and services.tls.certificates.leaf_data.subject_dn:/C=.{16}, O=.{16}, OU=.{16}/`

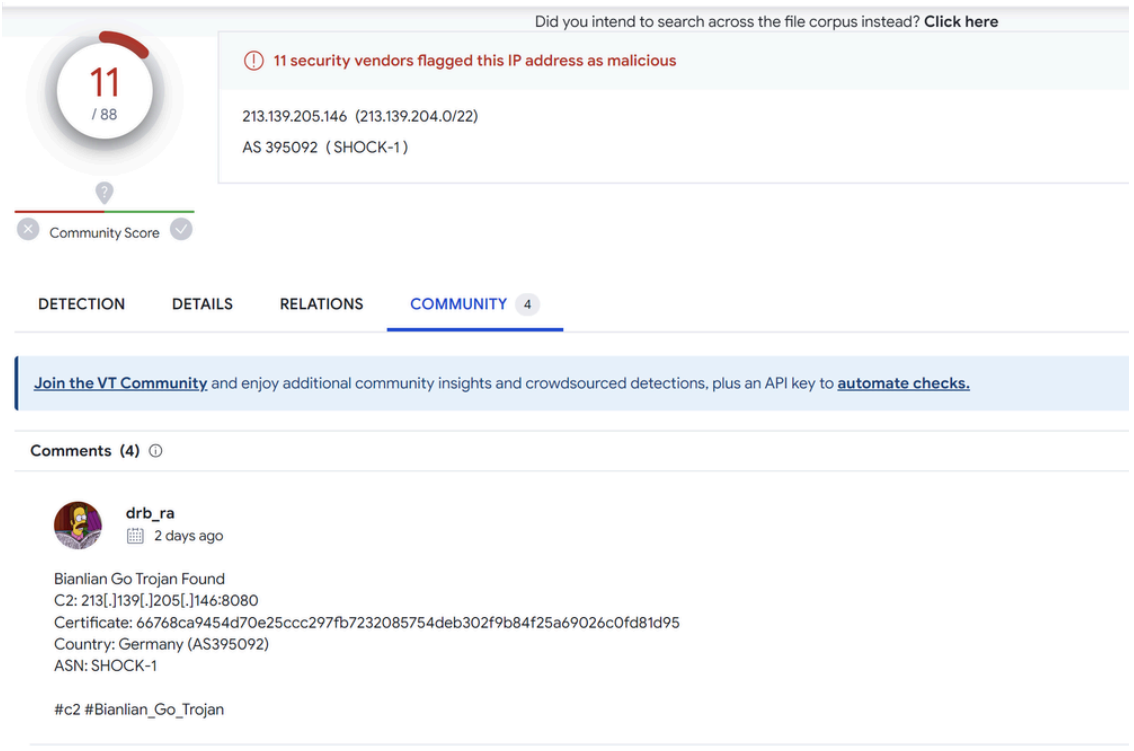
This results in 53 returned servers.



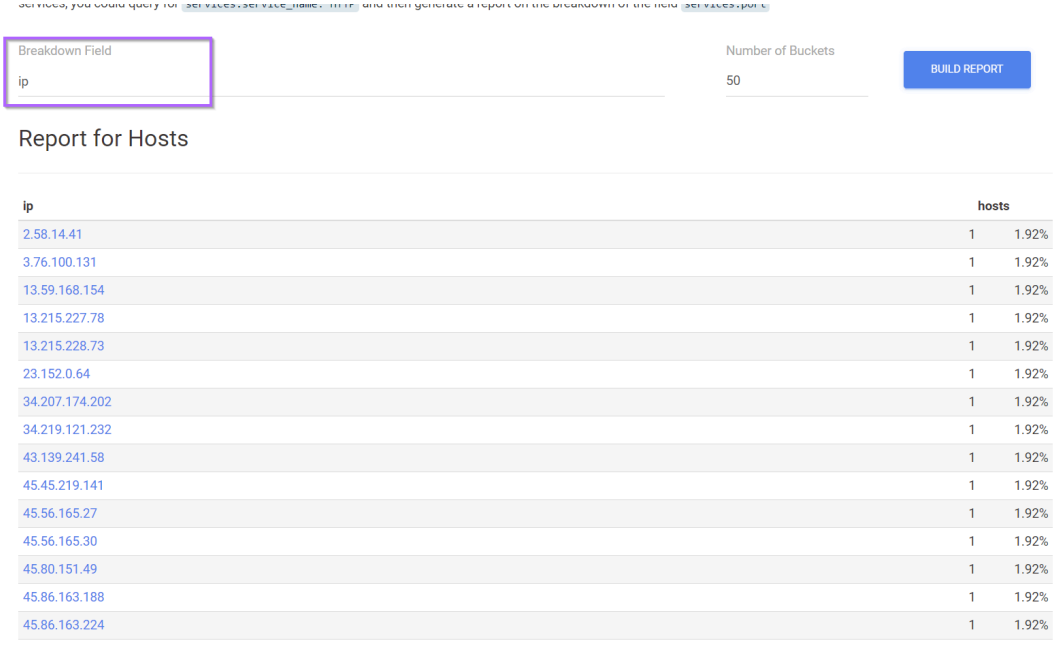
Investigating Hits With VirusTotal

Investigating the first 2 hits (that aren't our initial IP) with VirusTotal, we can see that both have been marked as BianLian trojan.





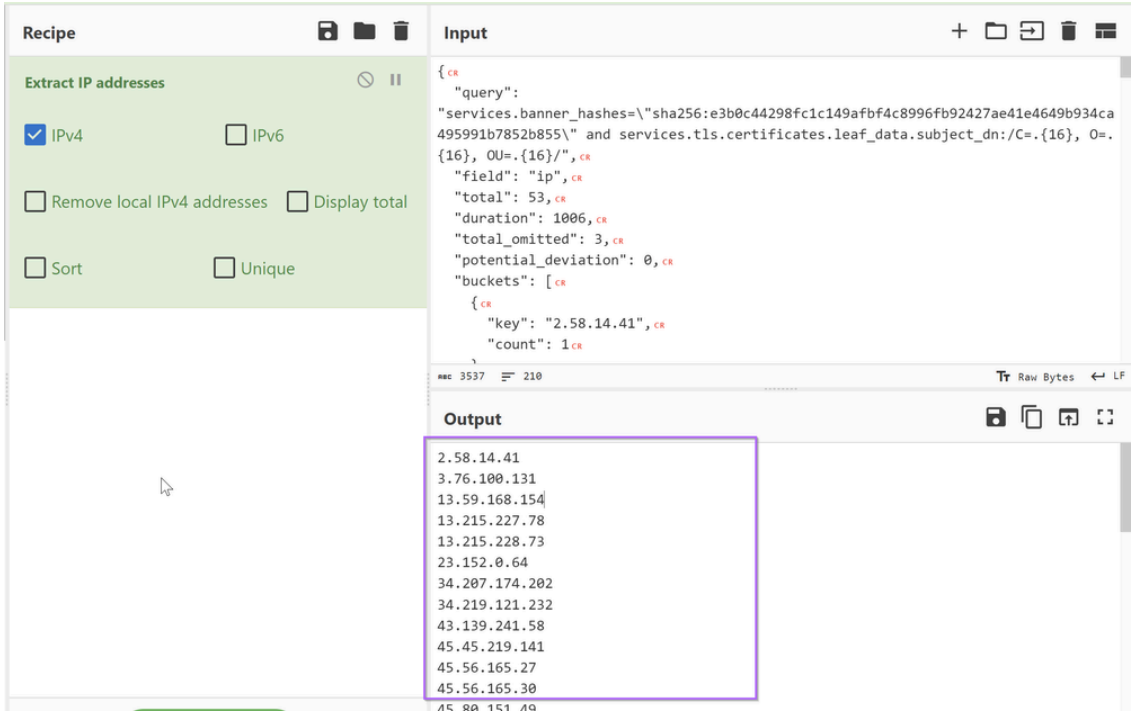
To save time scrolling through the page, we can use the `Build Report` feature again to obtain an easy list of all returned IPs.



We can also scroll down to the JSON output and combine it with CyberChef to obtain a text-based list.

JSON Report


```
{
  "query": "services.banner_hashes=\"sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855\" and services.tls.certificates.leaf_data.subj
ect_dn:/C=.{16}, O=.{16}, OU=.{16}/",
  "field": "ip",
  "total": 53,
  "duration": 1006,
  "total_omitted": 3,
  "potential_deviation": 0,
  "buckets": [
    {
      "key": "2.58.14.41",
      "count": 1
    },
    {
      "key": "3.76.100.131",
      "count": 1
    },
    {
      "key": "13.59.168.154",
      "count": 1
    },
    {
```

Taking that list and validating servers with VirusTotal, we can that every result was marked malicious, and all had at least one comment referencing BianLian.

Final Results Checked Against Virustotal

```
 2[.]58[.]14[.]41 - BianLian 8/88 VT
3[.]76[.]100[.]131 - BianLian 11/88 VT
13[.]59[.]168[.]154 - BianLian 11/88 VT
13[.]215[.]227[.]78 - BianLian 11/88 VT
13[.]215[.]228[.]73 - BianLian 11/88 VT
23[.]152[.]0[.]64 - BianLian 11/88 VT
34[.]207[.]174[.]202 - BianLian 11/88 VT
34[.]219[.]121[.]232 - BianLian 9/88 VT
43[.]139[.]241[.]58 - BianLian 13/88 VT
45[.]45[.]219[.]141 - BianLian 6/88 VT
```

```
45[.]56[.]165[.]27 - BianLian 12/88 VT
45[.]56[.]165[.]30 - BianLian 11/88 VT
45[.]80[.]151[.]49 - 4/88 BianLian
45[.]86[.]163[.]188 - 9/88 BianLian
45[.]86[.]163[.]224 - 13/88 BianLian
54[.]193[.]91[.]232 - 11/88 BianLian
65[.]109[.]3[.]80 - 6/88 BianLian
66[.]29[.]155[.]44 - 6/88 BianLian
85[.]13[.]118[.]11 - 6/88 BianLian
87[.]247[.]185[.]109 - 6/88 BianLian
91[.]102[.]162[.]229 - 9/88 BianLian
94[.]131[.]98[.]34 - 14/88 BianLian
94[.]198[.]50[.]195 - 11/88 BianLian
103[.]57[.]250[.]152 - 14/88 BianLian
104[.]36[.]229[.]15 - 13/88 BianLian
104[.]194[.]11[.]252 - 6/88 BianLian
104[.]238[.]34[.]130 - 6/88 BianLian
104[.]238[.]35[.]163 - 11/88 BianLian
104[.]238[.]60[.]64 - 9/88 BianLian
104[.]238[.]61[.]150 - 9/88 BianLian
104[.]243[.]32[.]53 - 5/88 BianLian
104[.]243[.]33[.]83 - 6/88 BianLian
104[.]243[.]33[.]84 - 7/88 BianLian
108[.]174[.]60[.]151 - 8/88 BianLian
120[.]48[.]110[.]233 - 5/88 BianLian
139[.]59[.]40[.]48 - 8/88 BianLian
143[.]198[.]46[.]29 - 9/88 BianLian
149[.]154[.]158[.]34 - 10/88 BianLian
149[.]154[.]158[.]199 - 11/88 BianLian
149[.]248[.]14[.]201 - 6/88 BianLian
168[.]119[.]88[.]236 - 6/88 BianLian
173[.]254[.]235[.]30 - 10/88 BianLian
185[.]240[.]103[.]195 - 9/88 VT BianLian
185[.]248[.]100[.]118 - 6/88 BianLian
188[.]34[.]130[.]46 - 4/88 BianLian
192[.]52[.]166[.]233 - 6/88 VT BianLian
192[.]236[.]192[.]207 - 6/88 VT BianLian
194[.]213[.]18[.]45 - 9/88 VT BianLian
195[.]128[.]235[.]20 - 7/88 VT BianLian
198[.]199[.]76[.]216 - 6/88 VT BianLian
```

Note on Searching Without Regex

The regular expression (Regex) feature is only available within the Paid version of Censys.

If you wish to obtain the same results on the free version, you can use the following query. This works if you know the specific length of each field you want to search.

It won't work in other cases where you want to specify a length range (eg something like `.{14,16}`. But for hardcoded lengths, it can work and is functionally equivalent to `.{16}`

```
services.banner_hashes="sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855" and services.tls.certificates.leaf_data.subject_dn:"C=??????????????, O=??????????????, OU=??????????????"
```

Source: https://embee-research.ghost.io/building-advanced-censys-queries-utilising-regex-bianlian/