A toolkit for testing domains and their webpages

Posted on October 9, 2020



In the present blog, we demonstrate how to perform a variety of technical and security tests against a domain by using WhoisXML API's Domain Reputation API. It is a RESTFul API that can be used in a broad range of popular programming environments, including e.g., BASH shell scripts, Windows PowerShell, Python, Java, C++, to name a few. It can be seen as a toolkit performing many tests ranging from DNS checks through revealing e-mail and web server configuration shortcomings to safe web browsing issues such as SSL problems or the presence of the domain in blacklists. The API has recently been updated to provide numeric codes for various tests and warnings; let us see what they can be good for.

The proper operating of Internet domains and web pages involves a number of aspects. The domain has to be duly configured in the Domain Name System (DNS), including its WHOIS information. The web server should also be properly configured, including its SSL settings. If the domain has an IP service, the mail servers have to be set up properly, and the respective DNS settings should be correct, etc. Looking at it from the reverse side, when visiting a web page there may be a number of indicators that suggest that there is a risk involved in the visit.

Checking all these aspects of a top-level domain, either to decide if your own domain is well-configured or to assess the risk of visiting a domain can require a lot of effort. WhoisXML API has an API and lookup service, the Domain Reputation API, which will do a number of tests at once. A result for of a lookup, e.g., for whoisxmlapi.com would look like this:


```
{
 "mode": "fast",
 "reputationScore": 98.54,
 "testResults": [
 {
   "test": "SSL vulnerabilities",
   "testCode": 88,
   "warnings": [
  "HTTP Strict Transport Security not set",
  "Heartbeat extension disabled",
  "TLSA record not configured or configured wrong",
  "OCSP stapling not configured"
   ],
   "warningCodes": [
  6015,
```

```
    6016,
    6019,
    6021
     ]
  }
  ]
}
```

Apart from some minor warnings which are due to configuration settings made at will, this is a properly configured domain hosting a page that is safe to be visited.

But what if you are interested in the verification of certain aspects only? Say, for instance, that you pick a subset of criteria that you want to check before visiting a page hosted on that domain? In our example, we go for the aspect of safe browsing and we want to check the following points:

- We want to get warned if the domain is recently registered or its SSL certificate was obtained recently.

- We want to make sure that the page contains no links to executables (aka .exe or .apk files), has a valid certificate, and is not blacklisted.

To implement a utility to help us with that, we opt for a BASH shell script as a tool for implementation for the sake of simplicity. (Even though it can also be done in Python, Java, etc.)

The following little code will do the job:

```
#!/bin/bash

API_KEY="YOUR_API_KEY"

echo "-> Checking "$1
api_result=$(curl -s --get "https://domain-reputation.whoisxmlapi.com/api/v
retval=0
for warning_code in $(echo $api_result| jq ".testResults[].warningCodes" |
 case $warning_code in
  2001)
   echo -e "\033[33;1;1mRecently registered domain. \033[0m"
```

```
    retval=1
    ;;
  6001)
    echo -e "\033[33;1;1mRecently obtained ssl certificate. \033[0m"
    retval=1
    ;;
  3003|3004)
    echo -e "\033[31;1;1mLinks to executables found. \033[0m"
    retval=2
    ;;
  6002|6004)
    echo -e "\033[31;1;1mSSL certificate invalid. \033[0m"
    retval=2
    ;;
  4001|6022)
    echo -e "\033[31;1;1mBlacklisted domain. \033[0m"
    retval=2
    ;;

  *)
 esac
done

echo "-> Conclusion:"
case $retval in
 0)
  echo -e "\033[32;1;1mNo issues. \033[0m"
  ;;
 1)
  echo -e "\033[33;1;1mBe careful. \033[0m"
  ;;
 2)
  echo -e "\033[31;1;1mNot recommended. \033[0m"
  ;;
esac

exit $retval
```

To give it a try, first you need to get an API key (free subscription is also available). Replace

"YOUR_API_KEY" with your actual API key at the beginning of the script to give it a try.

Let's see how it works. We use curl to get the API response as usual. The output is parsed by the jq utility which is a useful API parser, you can install it easily on most platforms (e.g. by doing "sudo apt install jq" on Debian-flavor systems). The loop will go through all the warning codes the API has returned.

Next, we go through all these codes, and give warning messages upon the codes we find as relevant. In particular, codes 2001 and 6001 will result in less severe warnings, whereas 3003, 3004, 6002, 6004, 4001, and 6002 will be considered a more serious problem.

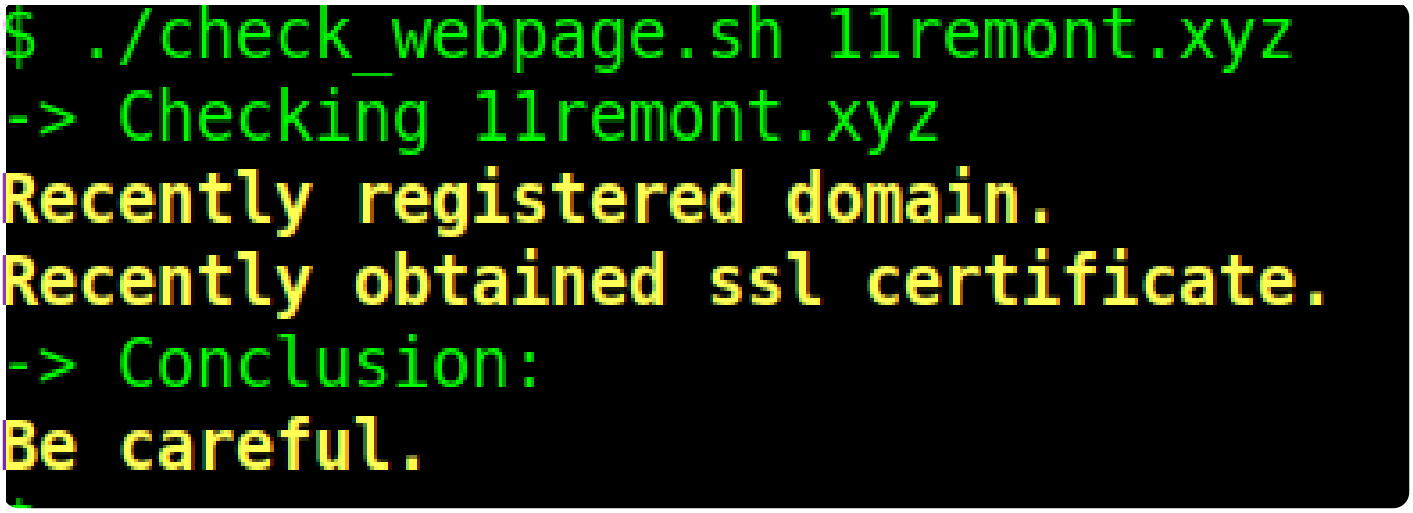
From the code it is clear what problems these codes reveal. A full list of warning codes is to be found in the description of the output format of the API. We echo a message if the relevant codes appear to describe the situation verbosely. (The control characters in the strings are there as we want to have a fancy color output; less severe issues will be reported in yellow, more severe ones in red.)

Meanwhile, we set the value of the variable retval: it will be 0 if there were no problems, 1 if there are warnings but no severe problems, and 2 if there is at least one severe issue. Depending on the value of retval we draw a conclusion in the last conditional, and we use this value also as a return value of our script. Let's see it in action.

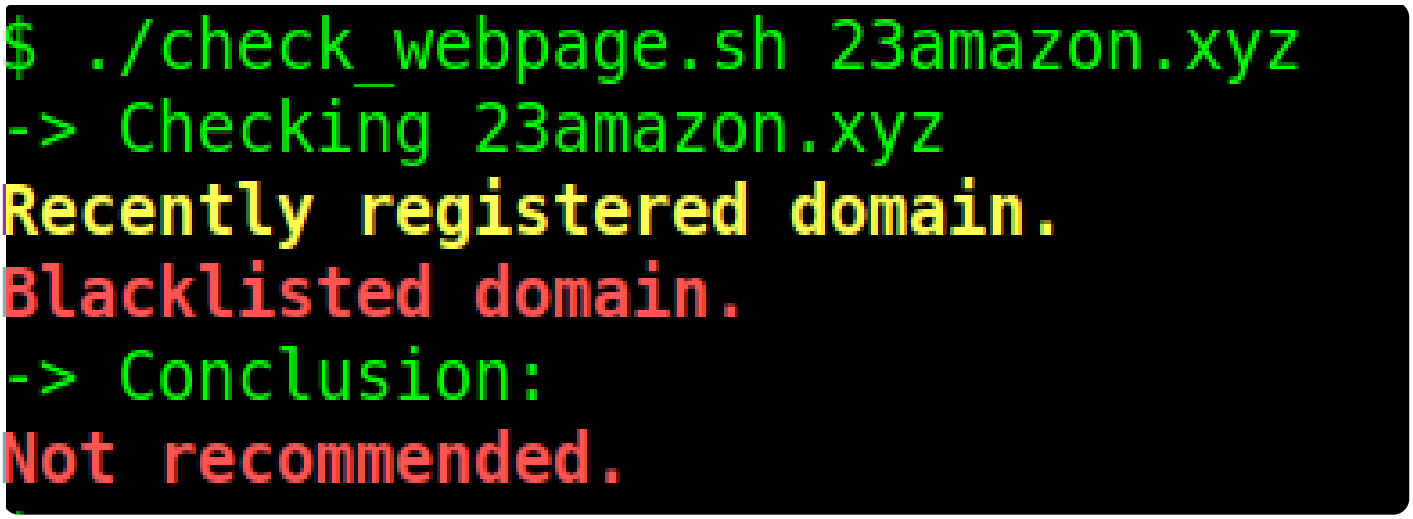
First let's confirm again that whoisxmlapi.com is safe to visit:

```
$ ./check_webpage.sh whoisxmlapi.com
-> Checking whoisxmlapi.com
-> Conclusion:
No issues.
```

If, however, the domain we check a domain which has been recently registered, we get:



This one is actually a parked domain at the moment. Finally, let us check a domain which is on PhishTank's list at the time of writing of this blog:



Apart from being recently registered, it is also on a blacklist. Of course we can identify the actual blacklist from the details in the full API response (the API key is stored in the API_KEY environment variable):

```
$ curl -s --get "https://domain-reputation.whoisxmlapi.com/api/v1?apiKey=$API_KEY&domainName=23amazon.xyz" | jq
{
  "mode": "fast",
  "reputationScore": 75,
  "testResults": [
    {
      "test": "WHOIS Domain check",
      "testCode": 93,
      "warnings": [
        "Registered 24 days ago"
      ],
      "warningCodes": [
        2001
      ]
    },
    {
      "test": "Malware databases check",
      "testCode": 82,
      "warnings": [
        "Listed on Phish Tank"
      ],
      "warningCodes": [
        4001
      ]
    }
  ]
}
```

However, now we just wanted to have a quick check of a subset of tests rather than go into detail.

Our little example can be useful as a utility for checking a domain before visiting a webpage hosted there. In addition, the idea can be adopted to cover many other applications for different professionals and organizations, including:

- **Web server administrators** can check the webpages they operate against relevant criteria such as SSL configuration issues, expiring certificates, etc.
- **Hosting providers** can implement an automated alerting system to notify their clients about improper DNS settings of their domains, web server configuration issues, etc.
- **E-mail server operators** can check domains against the validity of the e-mail service

configuration in the DNS and the sanity of the actual web servers.

- **Security experts** can filter domain lists to identify certain custom-defined attack vectors defined on the basis of the available tests.

- **Domainers** can batch process domain name lists to evaluate customized criteria in support of domain appraisal, or just to identify domains that will soon expire.

This list of users could also be significantly extended, and so can be the complexity of the implementation. For instance, you may have observed that the API also provides a cumulative "reputationScore" to quantify the results in a single numeric parameter.

If these values do not meet your customized expectations, the test and warning codes make it really easy to implement your own cumulative quantitative measures. It is time now to go ahead, register your account, and implement your own approach.