

IP Address filtering and reverse DNS checking

Executive Summary

This project was designed to help the Network Security Engineers working for The Church of Jesus Christ of Latter Day saints. The church has a large network that requires the protection of an entire team of Network security professionals. This team of professionals works 24/7 to protect all of the churches assets, personal information and other sensitive information from getting into “bad guys” hands.

This team of professionals is given the daunting task of reviewing thousands of log files per day looking for the proverbial needle in the haystack. This takes considerable amounts of time. While many of the tasks take a large amount of skill, some of the tasks are quite simple, and could easily be automated.

Problem Description

When I was interning at the church as a Network Security Engineer one of the most time consuming tasks that we did was reviewing what IP addresses the Point of Sale machines in the distribution centers from around the country were contacting. We reviewed them to determine if the IP address was valid (from a known payment processing service), or invalid (facebook, malware, etc).

My co-workers had developed a spreadsheet that listed valid IP Addresses. However, all of the comparisons were manually done, which took forever. Most addresses were expected, it was simply a matter of sorting through all of the other addresses to determine if yours really was in the list.

This final project will attempt to fix the tedious process of doing this manually. Specifically it includes these functions.

- Easy importing for IP addresses that are being tested via CSV file, because this was the only way other than straight copy paste to get data into excel.
- Automate the removal of duplicate entries, to reduce calculation time.
- Remove the “expected” Ip addresses from the list of accessed IP addresses. Leaving us with the “unexpected” or “dangerous” Ip addresses and their resolved domain names.
- Automate logging into the reverse dns checking website to determine what the hostname of the unexpected IP address is.
- For testing purposes I Generated a random list of valid routable IP addresses to test with.

I spent an hour or two a day (at least) manually looking up information that should have been automated. I haven’t checked with the church to see if they still need this, but this was my first ever realization that I should probably learn VBA, because if I had automated that, I think I would have easily gotten a raise, and a job offer. This task is what prompted me to take this VBA class.

Solution Creation

I began this project by taking the list of required functionalities that the final project would require, and started knocking them down one at a time.

The primary features of the project are actually quite simple. At its core this workbook and its imbedded macros do six things.

- 1) Randomly generates IP addresses to be used in the testing of this workbook
- 2) Allow easy importing of IP addresses by importing CSV files
- 3) Remove duplicate IP addresses from the list of IP addresses that is either provided or generated
- 4) Compare provided IP addresses against a list of IP addresses that have been previously approved
- 5) Remove approved IP addresses from the list, leaving behind a list of unapproved IP addresses
- 6) Reverse DNS check the unapproved IP address list for their resolved hostnames

Task 1

The first thing that I did was learn how to randomize numbers in order to complete the first task that this workbook was supposed to do. This task was to randomly generate a number of IP addresses. Because previous assignments had introduced us to the concept and methodology of creating random numbers this portion of the assignment was fairly straightforward. This was done first because the rest of the project required IP addresses to work with, and since I don't have countless log files containing hundreds of thousands of IP addresses, generating them was the next best thing I could do.

An IPv4 address is essentially a single number composed of four three digit numbers separated by decimal places. No individual octet (one of the four groupings containing up to three digits) can have a value of more than 255. This means that when I was generating random numbers, I needed to limit the numbers being generated to being between 0 and 255.

After this was done I figured out how to make sure that I didn't create IP addresses that were non-routable. This was done to make the IP addresses that I was generating as realistic as possible.

Task 2

Removing duplicate entries was quite simple. Excel has built in functionality to remove duplicates from columns. The only real input here was how many rows to compare to each other.

After removing the duplicates, the next step was to take the list of generated IP addresses and comparing them to the IP addresses in the expected IP addresses column. This proved to be the hardest part of the entire project because excel's built in functions do not allow it to play nicely with wildcards. And with wildcards being a very instrumental part of this plan it took a bit of doing to get things working nicely. This was actually the hardest portion of the project for me, and these difficulties are all documented in the "difficulties" portion of the document.

Task 3

Once this feature was working I began work on allowing the program to import CSV files. Now we had done work in class before about how to open directories and import data. However, if I used that method things were opened in their own workbooks, and I was having issues getting all of the information out of them. Instead I found another method. This is documented in the difficulties section.

Task 4

This task is what took me the longest out of all of the different tasks that I had to complete. The issues that I had with being able to compare IP addresses to an IP address that included wildcards are well documented in the difficulties encountered section.

This portion of the real life project is also what took the most time to do, and therefore that makes it the single most important part of this project to complete. I guess it is fitting that the most important part was also the most difficult.

Task 5

Removing the IP addresses that were expected was simple. It was done in conjunction with task 4. I used a Boolean value called “nomatchfound”. As the IP addresses were compared to the expected addresses an if statement was used to see if a match was found. If a match was found, then the value that was being compared iterated onto the next one. If no match was found, then the value that was being compared was inserted in order into the column labeled “Unexpected IP Addresses”.

Task 6

The last portion of the project turned out to be both easier AND harder than I expected. It was remarkably easy to set up the web query. The issue was that the problem with all of the websites that I was using allowing me to work with their site in this manner. All of the issues that I had with this are documented in the difficulties section of this paper.

Finally, for ease of use, I added buttons to each main function that has been discussed in this document to allow the user to easily use all of the functions that it presents.

Here is a screenshot of the completed projects user interface complete with test data.

19								
1	A	B	C	D	E	F	G	H
2	IP Addresses	Generate IP	Expected IP Addresses	Remove Expected	Unexpected IP Addresses	HostNames	Find Hostnames	
3	82.91.60.152		8.*		82.91.60.152	Italy Roma Telecom Italia Wireline Services		
4	180.69.147.148		199.271.*		180.69.147.148	Korea, Republic Of Seoul Hanaro Telecom		
5	254.238.64.207		9.*		254.238.64.207	Reserved Rfc3330		
6	197.104.161.126		165.*		128.34.248.45	United States Virginia Beach Navy Network Information Center		
7	128.34.248.45		167.198.*		151.174.28.167	United States Arlington Federal Deposit Insurance Corporation		
8	151.174.28.167		4.*		189.200.110.21	Mexico Mexico City Mexico Red De Telecomunicaciones S. De R.L. De C.v		
9	189.200.110.21		205.*		128.147.15.106	United States Pittsburgh Upmc		
10	128.147.15.106		12.*		30.37.81.254	United States Columbus Dod Network Information Center		
11	30.37.81.254		16.*		155.70.107.120	United States Denver Qwest Corporation		
12	155.70.107.120		175.*		70.44.130.88	United States East Stroudsburg Penteledata Inc		
13	16.117.112.242		197.*		131.102.66.17	Switzerland Bern Swiss Federal Government		
14	70.44.130.88		228.*		108.152.97.15	United States Atlanta AT&T Mobility Llc		
15	131.102.66.17		33.*		216.132.205.252	United States Costa Mesa Megapath Networks Inc.		
16	164.93.140.226		43.*		134.17.160.84	Belarus Minsk Mobile Telesystems Jllc		
17	108.152.97.15		164.*		71.175.103.98	United States Philadelphia Verizon Online Llc		
18	28.37.37.195		186.*		53.224.39.103	Germany Stuttgart Cap Debis Ccs		
19	68.35.11.177		200.*		156.141.187.32	United States Santa Clara Agilent Technologies		
20	216.132.205.252		48.*		83.2.201.236	Poland Warsaw Telekomunikacja Polska S.a.		

Difficulties Encountered

I ran into difficulties a number of times while working on this project. This is the summary of the main difficulties that I encountered during the course of this project, and it documents their solutions as well.

Task 1: IP Address generation

As I mentioned in the previous section, I randomly created hundreds of IP addresses for this project. However, using a random number generator to create IP addresses is not entirely foolproof. This is because there are IP addresses that simply do not exist on the internet, so to increase the level of realism I filtered out what are known as “Private” IP addresses.

The main problem that I ran into while attempting to solve this problem was trying to figure out how to keep private IP addresses from being generated at random. Attempting to control which random numbers were being generated was fruitless, and it also reduced the randomness of the function. This solution was clearly a problem.

After a number of attempts at solving this problem I resorted to a semi-complex if statement that tested the entire string of the IP address. This if statement tested if any of the private IP addresses were in the IP address that was being created (with or statements), and if ANY of them were found the process of creating an IP address was restart. While this solution might not be the absolute best solution out there, it is simple to maintain, and worked very well.

Here is a snippet of code showing what I did.

```
'Looping through to make the IPAddress from all of the individual octets
For j = 0 To 3
    octet = ((Rnd * (MaxNumber - MinNumber + 1)) + MinNumber)
    octetstring = Str(octet) & "."
    IPAddress = IPAddress + (Mid(octetstring, 2, Len(octetstring)))

    'checking to see if the IP address is a private IP address. If it is, it resets it.
    If ((IPAddress = "10.") Or (IPAddress = "127.") Or (IPAddress = "172.16.") Or (IPAddress = "169.254.") Or (IPAddress = "192.168.")) Then
        IPAddress = ""
        j = -1
    End If
Next j
'end looping through to create the ipaddress from the octets
```

Task 3: Importing IP Addresses with CSV Files

Importing CSV files caused me a bit of pause, but the delay wasn't anywhere as substantial as finding out a way to correctly compare the provided IP addresses with the expected list of IP addresses.

I found a function online that allowed me to easily import IP addresses. And after that I recorded myself importing the CSV files. The only problem this really caused me was that when importing the csv file it would constantly overwrite the first column in the workbook. To get around this I simple turned off screen updating, imported the CSV files contents into a newly created first row, copy and pasted the contents of this new row in the correct range, and then deleted the now un-needed extra column.

Task 4: IP Address Filtering

Another problem that I ran into was that of trying to compare the listed IP addresses to the list of expected IP addresses. The problem occurred in the form that the expected list of IP addresses used

wildcards to denote whole ranges of IP addresses. While this makes reading and writing the expected IP addresses much easier for us humans, it was causing excel a considerable amount of grief.

The problem was that I could easily use the instr function to find if the exact string was present. However, if the string was NOT found, then it would error. If this was being done manually in excel you would have just seen an error indicating that the sought after value had not been found. In VBA however this generated an error code that halted the program. Even after attempting many different methods of manipulating the error codes (on error resume next, using the worksheet.search function etc) I could not get around this problem with the current set of tools that existed in excel.

Not to be stopped in my progress, I searched the internet for a few hours before I found the solution. Some kind soul out there has created an improved instr function (named aptly “instr2”). Essentially the only thing this changed about the initial instr function was that it worked with wildcard characters (*) and would not throw errors when searching on strings that included them.

After working with this new instr2 function I did manage to get things to work. The solution is a bit tedious, but it works fine. I ended up having to compare each IP address to all of the expected IP addresses one at a time. This means that I wrote a loop within a loop. So while it works fine for my current implementation, this would not be the ideal solution for an extremely large number of records. Used within its current set of operating criteria though, it works just fine.

Here is a screenshot of the core of the IP address filtering subprocedure.

```
'loop through the comparison
Do Until IsEmpty(ActiveSheet.Cells(comparison, 4).Value) Or nomatchfound = True

    If (Instr2(ActiveSheet.Cells(row, 1).Value, ActiveSheet.Cells(comparison, 4)) <> 1) Then
        nomatchfound = False
    Else
        nomatchfound = True
    End If

    'iterate the comparison
    comparison = comparison + 1
|
Loop
'end looping through the comparison
```

Task 6: Reverse DNS checking web queries

This turned out to be a very difficult portion of the project, but for a different reason than I initially thought. The websites that I was using was easily queried for the desired results, and even signing in wasn't a problem. I ran into a problem for a while with the connection settings and tried all sorts of settings trying to get rid of the “could not download file” error that I was getting. I eventually found that the problem was that the “.background” setting was not set. The “.background refresh” setting was set, but the “.background setting” was what I needed. Once that was set everything was just peachy, and the queries worked exactly how I was wanting.

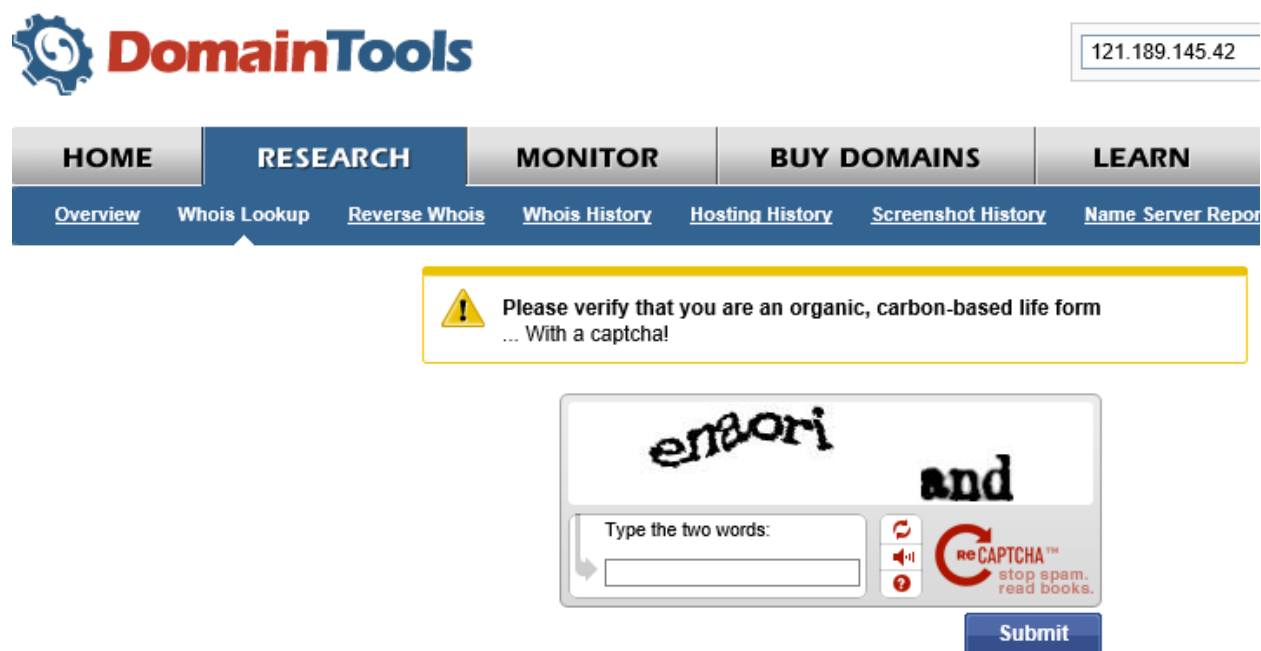
However, the success was short lived as nearly the same instant that I got the “could not download the file error” to go away that a new problem appeared. The problem with the web query came with the fact that while these websites are largely free to use, they highly discourage automating the use of their services. The way that this appeared in my project was that the requests would simply stop at random. After a large amount of troubleshooting I went in and edited the query manually out of frustration to check and see what the problem was.

It turns out that the problem was that every 20 requests it would throw up a captcha that would only appear when you edited the connection manually. So whenever the macro was called, it would work for exactly 19 queries. Then I had to edit the query and defeat the captcha before the subprocedure would work again.

I presented this problem to Professor Allen and asked how to proceed. After this I met with Professor Allen we determined that the best course of action here would be to present the captcha to the user to authenticate and press forward. To do this we needed to use Doctor Allens web agent instead of excels built in web query agents.

After re-writing this portion of the assignment with this web agent to get the queries’ working again everything started to work flawlessly. The captcha would appear to the user after every 19 queries I would then prove that I was a carbon-based life form (the sites wording, not mine) and then the agent would continue.

See, I told you it asked if I was a carbon based life form.



The screenshot shows the DomainTools website interface. At the top left is the DomainTools logo. At the top right, the IP address 121.189.145.42 is displayed. Below the logo is a navigation bar with tabs: HOME, RESEARCH (selected), MONITOR, BUY DOMAINS, and LEARN. Under the RESEARCH tab, there are links: Overview, Whois Lookup, Reverse Whois, Whois History, Hosting History, Screenshot History, and Name Server Report. A yellow warning box with a triangle icon contains the text: "Please verify that you are an organic, carbon-based life form ... With a captcha!". Below this is a CAPTCHA challenge box. It contains two words, "enari" and "and", which are distorted. Below the words is a text input field with the placeholder "Type the two words:". To the right of the input field are icons for a refresh button, a volume icon, and a help icon. Below the input field is a blue "Submit" button. The CAPTCHA logo "reCAPTCHA" is also visible, with the tagline "stop spam. read books."

The real problem appeared when the site blacklisted my login, and my IP address. This blacklisting stays in affect for a minimum of 24 hours, and seems to affect other users of the site coming from the same

location. What this means is that I might have inadvertently locked all of BYU's network security engineers out of this particular industry tool for an entire day.

On the upside, after some additional research I found that the site would allow such types of automated queries to be run, but that they charge for the type of account that can use the service in that way, plus an additional amount per query. After this point in time after consulting with Doctor Allen we both decided that I had done my due diligence in making the web query work, and that the website itself was stonewalling my advance. It wasn't worth the investment for a class project, particularly when it was clear that the proof of concept project worked.

Assistance Received

I received some assistance from Dr. Allen at the very end of my project. The captcha issue has been discussed above in the "difficulties" section of this paper. The remainder of the work I did entirely myself, and I'm dang proud of it.

Conclusion

The end result of this whole experience is that Dr. Allen gave me a pat on the back, and said that I had done my part and simply had to do my writeup. He told me that my project was good, and that I had done well. I felt good about that because I have spent a considerable amount of time on this project, and it's good to feel validated.

If this was a real business application, I would have gladly paid the \$40 a month to automate this task. This is because the task as it was when I worked there took two engineers an hour or two to do this task three times a day (every 8 hours all log files were reviewed). At that rate, the monthly charge would be saved the very first time that the task was performed in a month. The next 89 times that the task was completed that month would be pure savings. As a two hour task for two people would be reduced to a 10 minute task for one person. And that is exactly why I decided to learn this skill, even though I am not the best coder out there, nor do I particularly enjoy it. It is textbook ISYS, taking a process that took many man hours to do, and simplifying it into a single easy task.

I feel that I learned a lot during the course of this project. I also feel quite a bit of closure as well, because it was nearly two and a half years ago that I first realized that I should learn this skill, and it has been taunting me ever since then.

It will no longer.