# CU HinDom IP-domain-IP Matrix

Kent Campbell
*CYBR-5320*
*University of Colorado*
Boulder, USA
jaca3214@colorado.edu

***Abstract*—CYBR 5320 class has created the beginnings of a heterogeneous information network-based malicious domain classification system for application on the CU-Boulder network. This paper describes the work surrounding the generation of the IP-domain-IP matrix. IP addresses and communication pairings are extracted from Netflow logs. Input from DNS logs filters these addresses to determine which IPs map back to common domains, and the resultant product is a constrained sparse row matrix containing IP pairs and the number of times they communicated.**

***Index Terms*—HinDom, Netflow, IP, domain, malicious activity, sparse matrix**

## I. Introduction

The CYBR 5320 class is creating a python-based implementation of the malicious domain detection system, HinDom [1], for potential deployment on the CU-Boulder network servers. HinDom models the Domain Name System (DNS) scene as a Heterogeneous Information Network (HIN) and applies a classification method to determine malicious domain activity. The creation of individual components of the CU-HinDom effort are split amongst several students and the portion described here is the IP-domain-IP matrix. This matrix describes the counts of repeated source IP and destination IP address communications of CU Boulder network traffic captured in the Netflow data and filtered by the domain criteria from a related HinDom pruning function, $prun.py$ [2].

## II. Related Work

Everything has a cost and there are no infinite resources. While the IP landscape is expanding from IPv4 to IPv6 in terms of number of addresses available to claim, they have a cost inherent in acquisition. Similarly, in terms of cybercrime, defenders and attackers are limited by the available tools they can deploy and afford and fortunately, for defenders, this resource limitation - the reuse of IP addresses, can be exploited.

The HinDom [1] authors incorporate source IP address and destination IP address in the DNS schema for malicious domain detection. This is an explicit relationship incorporated in their workflow based on the notion is that 'limited resources likely force the reuse of IP addresses' and as such IPs can be mapped back to common domains. Chang, Mohaisen, Wang and Chen [3] utilize a similar assumption in their botnet modeling; IP addresses can uniquely identify botnets. This is with caveats, as they acknowledge IP address can change due to Dynamic Host Configuration Protocol (DHCP). They document 14% botnet reuse and this implicit IP address reuse should be a beneficial characteristic to leverage. Blacklists can defend against this reuse and simplistic one-to-one mapping.

Chiba, Tobe, Mori and Goto [4] hold that IP addresses are a more stable metric to address when detecting malicious behavior and these authors developed a malicious website detection methodology based on IP address features focusing specifically the name space structure.

Coskun [5] and Umer and Sher [6] both employ clustering approaches on IP addresses to detect malicious activity. Coskun documents successful clustering of malicious IP addresses with weak blacklists. He observes that similar behavior such as malicious web accounts, email spam campaigns and botnet command-and-control traffic can be identified with clustering and then even a weak blacklist will identify some of the malicious IP addresses and thus expose the other IP addresses as likely malicious IPs. Using a different approach, Umer and Sher present a successful approach of using IP addresses, ports, packet information, duration and protocol with domain knowledge to label malicious activity and then unsupervised methods such as k-mean, SOM, and DBSCAN to identify similar clusters of unidentified malicious activity.

With increasing sophistication, Khalil, Guan, Nabeel and Yu [7] capture the fact IP addresses can be either public or dedicated. Public IP addresses can have a many-to-one relationship while as dedicated IP addresses have a one-to-one mapping for a set amount of time. Malicious domains on a dedicated IP address should have more samples of activity to capture and model, but unfortunately public IP addresses can be assumed to be more fleeting in nature.

While the HimDom matrix IP-domain-IP does not explicitly capture the nuances of these IP address-focused models or models that extract more information content from Netflow data, it does capture the communication 'degree' and these relationships are leveraged elsewhere in the HinDom approach. These authors [8] extend this HinDom approach in a subsequent paper utilizing heterogeneous graph convolutional network, DeepDom.

A key takeaway from these papers are that inherently,

external information is needed such as domain information, blacklists, etc to identify IP addresses when only using Netflow or even DNS logs are used to identify malicious behavior.

## III. Approach Taken

The initial approach taken to create the IP-domainp-IP matrix was first an attempt to create lists of the source and destination IPs from for loops after reading from converted Netflow data. Upon research, a more effective approach came to the forefront to use a Pandas dataframe and methods assigned to this object type. While in the Pandas dataframe, initially internally consistent unique IP dictionaries were created and the number of times matching pairs were counted and then input into a constrained sparse row matrix. This temporary approach of creating a Neflow derived dictionary of IP addresses was superseded by integration of the $prun.py$ script's IP dictionary with E. Goodman's implementation. The dictionary derived from the function $GenerateWL$ within $prun.py$ is the official list of IPs used for the HinDom project. $prun.py$ filters the data down with it's domain selection criteria $\mathrm{K}_d\%, K_a and K_c\% parameters$.

Unit testing was performed on limited aspects of code and code peer review performed. Creating appropriate unit tests was a challenging step with limited success due to the necessity of integrating the $prun.py$ inputs.

## IV. Sparse IP-to-IP Matrix

Netflow can be exported to ASCII, CSV format on the CU server using the command: flow-export -f2 $<$ ft-v05.202... $>$ converted/ft-v05.202...csv.

Running $ip\_to\_ip.py$ in standalone mode can be performed with the command: python $ip\_to\_ip.py$ –dns_files /data/examples/dns1000000.log –netflow_files /data/examples/netflow1000000.csv

Data is then processed through the following steps in the python code: $ip\_to\_ip.py$ [2]:

- Input DNS log data first passes through $prun.py$'s $GenerateWL$ function to create a filtered list of IP addresses that meet the domain selection criteria. This creates the project wide list of IP addresses that will be similar to all HinDom components.
- Input Netflow data is read in with source and destination IP addresses parsed from file.
- Unique source and destination IP addresses are calculated.
- IP addresses are compared against $prun.py$'s GenerateWL master and dropped if matching address not found.
- Unique IP addresses are mapped to integer values in source and destination dictionaries.
- Constrained sparse row matrix function called.
- Counts of repeated pairings of source to destination IP communications are summed.
- The integer representation of source and destination IP addresses are combined with value counts in a compressed sparse row matrix. Columns are set to be source IP, rows set to be destination IP and value counts set to be the value.
- The columns, rows and values are combined into the constrained sparse row matrix and returned. This is the matrix IP-domain-IP of the HinDom paper.

Summary statistics have been tabulated for an early beta version of the code (TABLE I and TABLE II). Sampling of intake rows/volume and processing time was tested with 15 minute Netflow files and concatenated to hourly Netflow files. These results offer a glimpse of the challenge of the size of data to be consumed and a ballpark estimate of efficiency of reading data without interaction and querying of $prun.py$ function.

TABLE I
Preliminary Performance Metrics

| CU Server | 15 Minute Netflows | | |
|---|---|---|---|
| | ***Number Samples*** | ***Average*** | ***STD*** |
| Rows (Netflows) | 5 | 1,349,638.40 | 338,986.01 |
| Unique SRC | 5 | 40,612.60 | 2,705.22 |
| Unique DEST | 5 | 57,479.80 | 8,105.41 |
| Unique pairs | 5 | 348,641.00 | 108,757.26 |
| Seconds | 5 | 4.614 | 1.46 |
| Rows/Second | 5 | 300,779.13 | 27,293.51 |

As of 03/02/2021

TABLE II
Preliminary Performance Metrics

| CU Server | Hourly Netflows | | |
|---|---|---|---|
| | ***Number Samples*** | ***Average*** | ***STD*** |
| Rows (Netflows) | 3 | 6,156,757.33 | 1,516,293.49 |
| Unique SRC | 3 | 69,808.67 | 5,045.28 |
| Unique DEST | 3 | 115,271.00 | 10,167.03 |
| Unique pairs | 3 | 1,045,401.67 | 303,590.68 |
| Seconds | 3 | 20.34 | 5.83 |
| Rows/Second | 3 | 305,703.21 | 16,088.09 |

As of 03/02/2021

Note, these measurements exclude time required for binary Netflow to Ascii (CSV) file conversion as this step is assumed to be a shared action for all CU-HinDom components and/or performed in advance as a pipeline process since CU HinDom would likely be implemented in near-real time and this step would be automated.

Fig. 1. Preliminary Performance Metrics

| Test File Size | Time |
|---|---|
| 1000 | 1.1s |
| 10 000 | 2.07s |
| 100 000 | 12.80s |
| 1 000 000 | 2m 26s |

## V. Performance

Performance of $ip\_to\_ip.py$ with $prun.py$ criteria applied, as per command line implementation previously illustrated

the performance on test files created for code evaluation (Fig. 1). Note, this is single sample summary, time did not permit a more rigorous sampling approach. With $prun.py$ applied, the performance is dramatically slower than with the beta/standalone implementation. However, this is a necessary step for coordination of IP addresses across the project and the necessity of having the external knowledge of domain information from the DNS logs.

## VI. Findings

The current state of the $ip\_to\_ip.py$ code does not allow for any definitive prediction of the scalability of this component of the CU-HinDom project to a full volume of network surveillance data. E. Goodman stated that current processing time as a whole is slow with the current CU-HinDom code on test dataset. This was attributed to other components of algorithm and not the IP-domain-IP matrix, though, the time spent ingesting Neflow data is not zero.

With application of the $prun.py$ criteria, Netflow sourced IP pairs are dropped in accordance from external domain information criteria (from DNS logs) since we are interested in finding the pairs of IPs that map back to a common domain. This filtering can also be considered a data enhancement step, adding value to the IP information, and is also positive in regards to data reduction.

However, upon inspection of the output results with this integration of the domain criteria, it was noticed that there was a large drop out of IPs captured from intake of the Netflow. The retention of matching IPs ranged from extremely poor to poor.

To understand why the code behaved this way, test code was created to track data as it flowed through $ip\_to\_ip.py$ and while calling the $prun.py$ function. Using the test data, counts of unique IPs and communications of repeated pairs were run. The results are in Table III and the behavior is as expected.

TABLE III
READING FROM NETFLOW ONLY

| **Example Data Lines** | **Netflow** | |
|---|---|---|
| | ***Netflow Unique IPs*** | ***Repeated Pairs*** |
| 1,000 | 1221 | 5 |
| 10,000 | 6,611 | 46 |
| 100,000 | 24,068 | 258 |
| 1,000,000 | 59,434 | 1775 |

However, upon calling in the $prun.py$ functionality, the amount of matching IPs sourced from the Netflow vs. $GenerateWL$ using DNS logs shows how poorly the two IP dictionaries compare, ranging from less than 2% for 1,000 rows to only 17.50% for 1 million rows of both Netflow and DNS log input data as seen in Table IV.

This discrepancy in the amount of matching IPs initiated a review of the input test data. Using the Unix commands $head$ and $tail$, the Netflow and DNS log Unix second timestamps were compared. Converted to human-readable format, the time windows compared show a mis-alignment of time. Review of Table V show the only contemporaneous times (with this capture approach) occur with sample size of 1,000,000 rows.

TABLE IV
APPLICATION OF $prun.py$

| **Example Data Lines** | **Prun** | |
|---|---|---|
| | ***Prun Valid IPs*** | ***Percent Remain*** |
| 1,000 | 257 | 1.90% |
| 10,000 | 2,730 | 9.20% |
| 100,000 | 16,216 | 13.30% |
| 1,000,000 | 61,583 | 17.50% |

TABLE V
BEGINNING AND ENDING UNIX SECOND TIME STAMPS

| **Netflow** | **DNS Logs** | | |
|---|---|---|---|
| **Net T0** | ***Net T1*** | ***DNS T0*** | ***DNS T1*** |
| 9:00:00 | 9:00:00 | 8:57:31 | 8:57:51 |
| 9:00:00 | 9:00:03 | 8:57:31 | 8:59:55 |
| 9:00:00 | 9:00:45 | 8:57:31 | 8:59:43 |
| 9:00:00 | 9:08:07 | 8:57:31 | 9:06:16 |

For date 04/05/2021

It is acknowledged that this paints a bleak picture, but further delving into the test data shows that this an exception rather than the rule because of the following reasons. First, this was a crude beginning and ending of file analysis using the above mentioned Unix commands, so this does not reflect the full contents of the files. Upon review of the converted Netflow logs, the time stamps are rounded to integer Unix seconds and sorted/written incrementally increasing in time. This implies that cutting down Netflow-derived data files will be well 'behaved', a beginning and ending review of the file will provide a correct view of the contents. However, in review of the DNS log example data files, the timestamps are much less orderly. The $head$ and $tail$ approach fails to provide correct insight into the contents. Using $egrep$ to track a representative integer Unix timestamp, this singular integer timestamp was spread throughout almost 1 million rows of DNS log file. Interspersed within this approximate 1 million rows of data were other timestamps incrementally interleaved and equally unsorted. In whole, one can interpret DNS logs to be heavily interleaved and poorly sorted, so cutting file sizes down comes with risk of having an unknown or unpredictable time window.

Additional comments should be made about this finding. First and foremost, this should be relegated to the 'exception rather than the rule' bucket. There is likely more overlap and rigorous analysis would have entailed sorting on timestamp proper, not using $egrep$ and visually scanning the result as applied here. Also, Netflow and DNS logs are voluminous, being truly within the realm of 'big data' and what this investigation into the test data highlights the importance of correct sampling procedures of 'big data.' In a production setting, with likely much larger batches of input data, this potential mis-match of temporal windows would heal, as

hinted in the results of the 1 million row data. The degree of communication amongst IPs is quite large, recalling the the protocol of the 'three-way handshake' of TCP and the amounts of communications in delivering data. Taken as a whole, this issue will likely affect a minuscule amount of data and also only manifest itself at the boundary conditions of the beginning and ending conditions of the input files.

## VII. Recommended Next Steps and Future Work

First and foremost, renaming the Python file from $ip\_to\_ip.py$ to $ipDip.py$ would more inline with the naming convention of the paper and other components of the project. Granted, this is a trivial name change requiring some effort, but future consumers of this code might appreciate the consistency of labeling.

A more complicated next step is to remove the dependency on the Pandas Python library. Following the logic of the current code should allow for a more streamlined and perhaps efficient solution to a refactoring for this aspect. Other CU-HinDom components have similar logic, so these could be leveraged.

While a refactoring is ongoing to remove the Pandas dependency, investigation of the counts of pair communications should be taken. At time of writing, a question has arisen if the sum of the occurrences of repeated IP-IP communications is accurate. Current implementation of code separates IP's explicitly into source and destination IPs and thus, in the Pandas table, these manifest themselves as distinct SRC and DEST columns. Presently there is doubt that the count is correct in the sense that the pair SRC IP x communicating with DEST IP y is being treated the same as SRC IP y communicating with DEST IP x. Clearly, this is the same communicating pair, but it it in now doubt that this example will assign 2 to the number of occurrences to the communicating pair and not 1 for each direction of the communication. For communications like the 'three-way handshake', would this implementation correctly count the 3 times or will it be 2 for SRC x with DEST y and 1 for SRC y with DEST x?

In light of the findings of small sample window sizes in the test data, a pre-processing step should be undertaken to ensure contemporaneous time windows of Netflow and DNS logs. This is likely only an issue while code development and testing is underway, but it can distort the veracity of the results. Awareness of this issue is a must especially if a streaming approach of the CU-HinDOM approach is to be applied. With the streaming approach, some mechanism should be in place to ensure, with as much fidelity as possible that the Netflow and DNS logs are 'seeing the same thing.' Implementation of this might perhaps prevent a streaming implementation from even being a viable approach. In a production implementation, mis-alignment of time windows will likely only manifest itself as a boundary condition at the beginning and ending periods of the input data and is likely have negligible affect in terms of total data percentage.

The preceding recommended next steps and future work can be considered tactical changes to the code. A more strategic effort should be undertaken to evaluate if more meaningful information can be gleamed from the Netflow captured on the CU network. Currently the focus is on the counts of communicating pairs, either the pairing has a large number of communications or very few or somewhere in the middle, in the time window of analysis. Value gained from this information is top communicating pairs, bottom, etc and also perhaps that malicious domains seem to be within a certain range of values. Can there be more information value gained in capturing other headers from the Netflow? Tests should be undertaken to see if the sum of 'dpkts' or sum of 'doctets' provides more meaningful information, following the lead from Umer and Sher. The quantity of data communicated might be indicative of payload, intent, etc. This information might better differentiate the 'type' of communications between the communicating pairs, much like the flags for TCP or UDP do. Additionally, and the implementation of this would require much effort, perhaps duration of communication could be of additional value. It is acknowledged that this might be unrealistic in terms of the appropriate amount of time that should be allocated in creation of the IP-domain-IP matrix.

## VIII. Summary

An IP-domain-IP constrained sparse row matrix was created as a part of the class CU-HinDOM project. This matrix ingests CU network Netflow and counts the time IP pairs communicate within the analysis time window and is filtered by Domain criteria from DNS logs. This matrix is a part of a heterogeneous information network that is created as input into a classification system to identify malicious domain activity. This was an excellent learning opportunity to learn how to code in Python and integrate code and understanding within a larger coding effort using GitHub.

## References


[1] X. Sun, M. Tong, J. Yang, L. Xinran, and L. Heng, "Hindom: A robust malicious domain detection system based on heterogeneous information network with transductive classification," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 399–412.

[2] J. K. Campbell, "Cu_hin," https://github.com/elgood/CU_HIN, 2021.

[3] W. Chang, A. Mohaisen, A. Wang, and S. Chen, "Measuring botnets in the wild: Some new trends," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 645–650.

[4] D. Chiba, K. Tobe, T. Mori, and S. Goto, "Detecting malicious websites by learning ip address features," in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*. IEEE, 2012, pp. 29–39.

[5] B. Coskun, "(un)wisdom of crowds: Accurately spotting malicious ip clusters using not-so-accurate ip blacklists," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1406–1417, 2017.

[6] M. F. Umer and M. Sher, "Automatic clustering of malicious ip flow records using unsupervised learning," in *Enterprise Security*, V. Chang, M. Ramachandran, R. J. Walters, and G. Wills, Eds. Cham: Springer International Publishing, 2017, pp. 97–119.

[7] I. M. Khalil, B. Guan, M. Nabeel, and T. Yu, "A domain is only as good as its buddies: Detecting stealthy malicious domains via graph inference," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 330–341.

[8] X. Sun, Z. Wang, J. Yang, and X. Liu, "Deepdom: Malicious domain detection with scalable and heterogeneous graph convolutional networks," *Comput. Secur.*, vol. 99, p. 102057, 2020.