Machine Learning Project Final Report

Mohammad Sagor
Texas A&M University
msagor@tamu.edu

Jonathan Grimes
Texas A&M University
nsa@tamu.edu

Abstract

Wireless communication, especially WiFi communication is prone to many attacks. Due to the nature of WiFi protocol, its easier for an attacker to hijack a victim's MAC and sink all traffics to the attacker machine. To detect MAC spoofing attack, our solution provides a novel way that samples client's signals strength at the access point, and uses statistical analysis and machine learning approach to detect an attack. Our proposed solution can detect a MAC spoofing attack with a 95 % Precision.

1. Introduction

Wireless communication security now poses a big challenge in communication technology. Due to the nature of wireless communication, such as WiFi communication, and the protocols behind WiFi, allows malicious users to take advantage of other benign clients. One of the very common types pf attack is called MAC spoofing attack,in which an attacker mimics a victim's MAC address and uses it in the attacker machine to sink or sniff all data traffic into attacker machine. This problem is even aggregated when the wireless network does not use authentication. Even using authentication token, MAC spoofing attack can still take place by retrieving or bypassing authentication which was shown in studies such as KRACK [18]. Mac filtering is another populate solution in which wireless access points allows connection to only manually authenticated users.

Unfortunately, MAC Filtering can also be bypassed by exploiting the wireless access point's trust in connecting devices [14] and [**?**]. In addition to bypassing MAC Filtering, MAC spoofing allows a malicious actor to frame other clients for malicious actions while allowing the malicious actor to stay anonymous.

Most solutions either have loopholes to evade authentication, or they simply deny untrustworthy users to use the wireless network. Also, Current solutions cares more for protecting the users from an attack, instead of detecting the attacker. Moreover, as discussed above, previous works shows it is evident that using authentication is not enough to completely prevent MAC spoofing attack; it is also crucial to analyze the nature of signal to formulate a different attack measurement metric. Little to no work has been done on utilizing the nature of WiFi signal distribution on a real environment and analyzing how signal distribution changes over time when an attack happens. In this way, we can reliably detect when a MAC spoofing attack is happening by collecting data from access point. Additionally, works such as, such as [2], [10], and [1], have been used to better help with detection of such tasks.

The proposed solution in this paper can be incorporated as a feature in wireless access points as a security feature. Now-a-days, given the fact that wireless access points have robust processing unit, running this tool for each connected client, would not be an issue. Our final product can be found at [6].

2. Related Work

Previous works to detect MAC spoofing attack ranges from using authentication tokens, to analyzing signal strength. Since our solution involves utilizing receivers signal strength at access point, we dived into current works involving signal distribution of wireless network.[7] showed that some MAC Spoofing attacks could be detected by validating the fingerprint of connected devices was consistent. [5],[3],and [17] all have planned to detect when spoofing attacks happen by measuring how signal strengths differ between different clients between multiple access points. These solution s require direct access to the access points and requires multiple monitoring stations to be connected to the access points, which takes precious network bandwidth and computational power.

Since we are focused on detecting MAC spoofing using machine learning approach, we surveyed previous works which incorporated machine learning to solve this problem. [11] provided motivation to our work which incorporated clustering frameworks to detect outliers and produce result as outliers in a separate list. However, their proposed "two-phase" is actually a repetitive process and their modified k-means clustering algorithm deviates from the original k-means clustering algorithm in terms of performing clusterization. Also, their proposed modified k-means clustering algorithm chooses centroids randomly so in each different experiment, the result can be different, even using same dataset. So, one of our major goal was to formulate our algorithm in such way that the result is consistent over multiple simulations over same dataset.

3. Theoretical Basis

It is crucial to understand the nature of wireless signal distribution pattern in WiFi, as we are using the signal strength and its distribution pattern to identify an attacker. Contrary to popular believe, several studies [8], [15], [4], [19], and [16] shows that wireless signals does not not travel as a perfect sphere, rather the signal sphere is distorted by physical objects. In a common environment, where interference such as electronic machines, furniture, even walls exist, each emitted signal shape varies to a great and measurable extent [8], [15], [4], [19], and [16]. Hence, signal sent from a non-moving client connected to an access point usually maintains a particular signal pattern, given that the room's objects are not frequently moving. Access points can set its receiving strength to a particular level, and measure signal strength as RSSI(Received Signal Strength Indicator) values for each received signal. For a typical home router, each RSSI value can range between 0 to -120 dBm. However, it is to mention that, even for a static client, an access point will collect different RSSI values for the same client, if the access point switches between receiving strengths. Our proposed solution exploits this idea of switching receiver strength among multiple dBm values, to collect data.

So, we can agree that, If two machines, sitting distant from each other and are connected to an access point using same MAC address, for each connected machine, the emitted signal pattern will be different. Hence, the RSSI values collected at the access points, will be measurably different for each connected machine. Our proposed solution analyzed the change of signal distribution for a client over time, and identifies if the signal pattern has a substantial change over time.

4. Assumptions

Although our proposed solution was not formulated based on any strong assumption, there are some basic conditions under which the proposed solution is expected to perform the best. Our solution requires to build a *normal profile* in which it must be assured that there is no attack happening and only the true client is connected to the access point. Also, during the time normal profile is being generated, it is assumed that both the client and the access point is static.

5. Architecture

5.1. Normal and Test Profiles

The proposed solution can run either as a standalone tool or can be added as a feature to the access point, for providing better security. For the scope of this project, we used a Linux machine with Kali distribution to act as an access point, to which other client machines will be connected and performing networking tasks. The access point can change its antenna's strength and can listen for client signals in different dBm levels.

The solution runs in two phases to generate two different types of profiles. In the first phase, namely *Learning Phase*, a *normal profile* is generated. The access point will keep constant connection to the client and continuously alternate its receiver strength to collect different RSSI values for different receiver strength. The normal profile is needed to understand the current signal pattern of client's connectivity. During this phase, it is assumed that no attack is taking place. After generating a normal profile, the data is then passed into our four core evaluation metric in which valuable information is parsed. Collected information is then passed into an objective function as input to come up with a single scoring system which denotes current condition of client connectivity.

The second phase is called *Detection phase*, in which the access point will generate a *Test Profile* for a client. A test profile is generated exactly the normal profile was generated in Learning phase, and after each iteration, test profile also produces a connectivity score. If the score is not within the acceptable range, it is considered as a red-flag. After certain number of continuous red-flags,it is then confirmed that an attack has taken place and the alarm is raised.

If the test profile appears to produce connectivity score that is within acceptable range, we include the data points from test profile into the normal profile and recompute the normal profile. Hence, the normal profile is continuously updated with more updated connectivity information.

5.2. Understanding Sample Data

As mentioned above, access point collects RSSI values of client, by periodically altering receiver strength(measured in dBm unit). RSSI values of typical WiFi ranges between 0 to -120. So, basically our collected data is one dimensional integer values ranging between 0 to -120. If a client maintains constant connectivity with an access point, and the signal is not interfered, then an agreeable range of a good wireless connectivity is between -20 to -80, -20 being the connection is excellent, and -80 being barely connected.

Since we collect enormous amount of RSSI data points, we first normalize the data to fit our proposed solution model. We separate noise signals from actual signals when

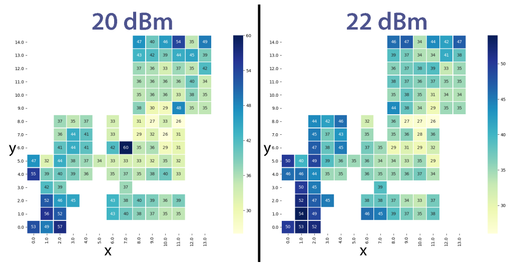

Figure 1. This shows how even with a small change in the signal strength of the wireless access point, there can still be large changes to RSSI values

sampling any data points, and take an RSSI value only if it is within [-20,-80] range. We collect data as sets of timestamps. In each timestamp, we collect data points for several different receiver strengths. For example, in timestamp#1, we collect RSSI values for listening strength of 0, 5, 10, 15, 20 dBm, then in timestamp#2 we collect RSSI values for the same iterations of listening strengths. We continue collecting data for a certain number of timestamps, until when we decide there is enough data points to generate either of normal or test profile.

5.3. Evaluation Metrics

As mentioned above, we have four core evaluation metrics using which we formulate normal and test profiles. Since, collected RSSI values are directly related to the signal distribution pattern, our core evaluation metrics analyzed collected data by looking from different point-of-views to understand current client connectivity. We analyze changes and deviations of data points in both intra and inter timestamps, to have knowledge about how signal pattern changed over time for a statically connected client. Below are the detailed explanations for each of the core evaluation metrics-

5.3.1 *Intra-Timestamp Correlation:*

In this behavior profiler, our solution will attempt to determine the relationship between each of the samples within a single timestamp. In one timestamp, there are RSSI data taken for multiple different dBm values, and the RSSI values possess a correlation among them. The intuition behind this is, all RSSI values for all dBm levels in one timestamp must maintain a consistent numeric difference among them. As an example, [Fig 2],shows that within the same timestamp, when a sample taken at 10 dBm gives an RSSI value of 45, and a samples taken at 5 dBm and 25 dBm give an RSSI value lower than 45. This pattern should continue to persist among following timestamps.

For each timestamp, to find the correlation among data points for different dBm levels, we find possible permutations of RSSI values, and compute their differences. As mentioned above, it is expected that the difference of RSSI values for each pair of dBm levels, should be consistent in following timestamp sampling.

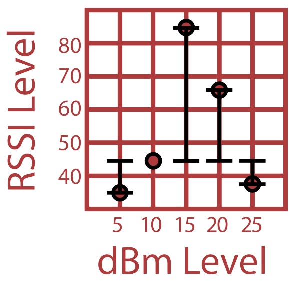

Figure **2: Intra-Timestamp Correlation**

```
Algorithm 1: Intra-timestamp
Input: timestamp
Output: outputList
Initialization: outputList = []
1 for a,b in Permutations(timestamp) do
      // permutations of all size 2 subsets
2     append a-b to outputList
```

5.3.2 *Inter-Timestamp Correlation:*

For Inter-Timestamp Correlation, our solution will observe the relationships among all combinations of RSSI values from timestamp *n-1* and *n*. In this way, we continuously examine collected data and look for sudden and noticeable deviation of RSSI values between two adjacent timestamps. For instance, [Fig 3], we might be able to determine that when a sample taken at 5 dBm increases, we should also expect a sample taken at 15 dBm to increase. Inter-Timestamp Correlation section works similar way as intra-timestamp correlation, except it finds the differences of RSSI values in two adjacent timestamps, instead of one. This process is necessary to continuously evaluate client's connectivity and regenerate the normal profile with more updated RSSI values.

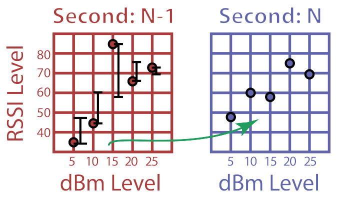

Figure **3: Inter-Timestamp Correlation**

```
Algorithm 2: Inter-timestamp
  Input: $timeA$,$timeB$
  Output: $outputList$
  Initialization: $outputList$ = []
1 for $a$,$b$ in $CartesianProduct(timeA,timeB)$ do
      // Cartesian product between two
         timestamps
2     append a-b to $outputList$
```

5.3.3 *dBm-Correlation:*

For dBm Correlation, our solution will determine how spread out the data is for each dbm for all timestamps. In this evaluation metric, we try to find the overall spread of RSSI values for one dBm level, over multiple timestamps. For instance, in [Fig. 4], it might actually be normal to see a very spread out result from samples taken at 25 dBm while also normal to see samples taken at 20 dBm to not be so spread out.

We take all RSSI values for each dBm level, and generate a box-plot. The box-plot gives us information such as first-quartile, median and third-quartile.

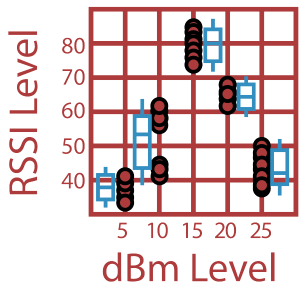

Figure **4: dBm-Correlation**

```
Algorithm 3: dBm-Correlation
  Input: $Timestamps$
  Output: $outputList$
  Initialization: $outputList$ = []
1 for $n$ in $Timestamps$ do
      // list of samples taken at nth dBm
         level
2     append 1st, median, and 3rd Quartile of $n$ to
        $outputList$
```

5.3.4 *dBm-Clusterization:*

In this section we run a modified clustering algorithm on all RSSI values for each dBm levels and see the change of clustering results over time. If no attack is happening, and client maintains a constant connection, all collected RSSI values for each dBm level should form a particular pattern of clusterization. The pattern should continue to persist unless an attack happens, in which case, the clustering results in detection phase will give noticeably different result, compared to normal phase.

We had to take extra precaution to choose the best clustering algorithm that fits the nature of our dataset. Moreover, given the fact that, collected RSSI values are within a narrow margin, that is [-20,-80], clustering parameters must have to be chosen in such way that, even slightest anomaly in data distribution can be identified. Many clustering algorithm has the idea of clustering insignificant data points as Noise and we tried to incorporate a clustering algorithm that has that feature.

Our survey over clustering algorithm for the purpose of detecting MAC spoofing attack showed little previous works. [11] used modified *K-means Clustering Algorithm* to detect outlier, which gave us motivation to try K-means. Although, our initial choice of clustering algorithm was K-means, this clustering algorithm performed poorly with our dataset. For large numbers of simulation, K-means algorithm generated different clustering results which affected the overall decision metric for our proposed solution. The reason being, K-means consider only distance among data points to be the major criterion for clustering, instead of density of data-points. Also, choosing a random centroid every time K-means algorithm is run is in the nature of the algorithm, so the same dataset would produce different results in multiple simulations.Lastly, we came up with a clustering algorithm that ad up with a adaptively chooses its own parameters based on current dataset, unlike K-means, which requires user to pass number of clusters the data needs to be clusterized into.

We took a famous clustering algorithm, *DB-SCAN*, short for **Density-based spatial clustering of applications with noise** and modified it to fit our particular goal. DB-SCAN is a density based clustering algorithm that decides number of clusters based on the density of data in a region, instead of euclidean distance like k-means. DB-SCAN considers region of data as a cluster where they are densely located, and considers data as noise if it is not within any of densely located regions. One of the major advantage of DB-SCAN over K-means is that, DB-SCAN algorithm can safely clusterize arbitrary shape of clusters, and can provide consistent results over multiple simulations, which is a big missing feature in K-means.

From each iteration of clustering, we collect several information such as- number of clusters, number of data points in each cluster, average number of points in all clusters, and mean for each cluster.

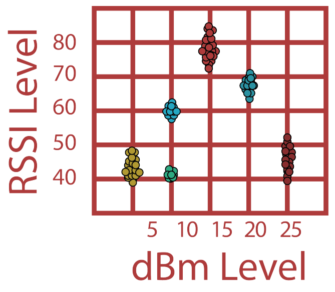

Figure **5: dBm-Clusterization**

Algorithm 4: dbm-Clusterization

```
Input: $numberOfDBMs$
Output: $outputList$
for $n$ in $numberOfDBMs$ do
    load data for nth dbm value;
    normalize(data);
    clusters =DBSCAN_Clustering_Alg(data);
    if clustering succeeded then
        collect #data points of each cluster;
        collect average for all cluster;
        collect mean for each cluster;
```

6. *Choosing DB-SCAN Parameters*

Vanilla DB-SCAN algorithm takes two parameters to perform clustering- episode(in short, eps), and minimum sample count (denoted as $minSampCount$). eps refers to the minimum distance between two points for which they can be considered as part of one cluster. $minSampCount$ denotes the minimum number of data-point needs to be present in one region,for that to be considered a cluster. Although DB-SCAN is different that most clustering algorithms in terms of logical sanity, choosing optimum eps and $minSampCount$ values are one of the non-trivial decisions for DB-SCAN algorithm. [12], [13], shows a good collective list of several modified DB-SCAN algorithm in which some of them does the parameter selection automatically. [20] converted the challenge into an optimization problem and solves it by dividing all data into multiple grids, and computes local minimum points using bitmap mapping and S-P tree. However, the key to choosing most effective set of eps and $minSampCount$ lies in what type of data we are dealing with.

For the nature of our data, we came up with an effective way to choose eps and $minSampCount$ parameters. For eps we take the standard deviation of all RSSI values of in each dBm levels. For $minSampCount$, we divide total number of data points, divided by eps. The logic behind such choice of parameters is intuitive for our particular dataset, since our data range is small, that is [-20,-80]. We have to consider an eps value that maintains a general distance from each other, and standard deviation does just that. Standard deviation denotes to the average distance of each data points from mean. A common question may arise, *"why not using mean of dataset itself?"* Although using mean of all data points in dataset seems intuitive, mean simply is not the right type of value for eps parameter. eps parameter relates to the distance and location of data points in the dataset, not the numeric value itself of data points.

Furthermore, our choice of $minSampCount$ relates directly to the method we used to choose eps parameter. The issue with choosing effective $minSampCount$ lies on not choosing a number that affects the true number of total clusters. If a value for $minSampCount$ is chosen too high, then one or more densely populated locations will merge together and result into one cluster. We do not want a valid data point to be clustered in a wrong cluster, or be assigned as noise due to poor choice of parameters. So $minSampCount$ must relate to the overall distribution of data points within the whole domain. Since we already computed the standard deviation, that is the average distance for each data point from the mean, and we also have total number of data points, a good estimation for $minSampCount$ is to find how many data points there are in one unit length standard deviation. That way, we allow each clusters to be big enough to hold all its densely located data points, but not small enough that some data points are mis-clusterized.

Algorithm 5: DBSCAN_Clustering_Algorithm

```
Input: $RSSIvalueList$
Output: $ClustersList$
eps = Standard_Deviation($RSSIvalueList$);
$minSampCount$ = len($RSSIvalueList$) / eps;
if eps < 1 then eps = 1;
if $minSampCount$ < 1 then $minSampCount$ = 1;
for $o$ in $RSSIvalueList$ do
    if o is not yet classified then
        if o has $minSampCount$ objects within eps then
            choose o as core object;
            collect objects in eps distance from o;
            assign objects to a new cluster;
        else
            assign o to NOISE;
```

6.1. DB-SCAN Parameters Optimization

In this section, we explore several key aspects of the above modified DB-SCAN algorithm and special checks used to avoid failures in corner cases. It is to mention that, both eps and $minSampCount$ parameters require positive values to be passed before the actual algorithm is run.

Since eps parameter is basically a distance value and equals to standard deviation, it cannot be a negative value. If the computed standard deviation is smaller than 1, it is likely that most of the data points are very close to the mean and they are most likely to converge into one cluster. That is why, if computed eps is less than 1, we assign it 1.

if computed $minSampCount$ is less than 1, that means the standard deviation value is greater than the total number of data points. In this case, the eps value will automatically be selected to be 1. Having $minSampCount$ less than 1 also means most data points are scattered and there are some data points which have very large values. In this case, it is expected to have more than one clusters. When we have more than one clusters, most often there are some clusters which contains singular data points. Especially, when eps has been selected 1, some data points have so much weight that they themselves contain a lot of gravity, hence they form their own cluster only by themselves. in traditional DB-SCAN algorithm in which parameters are selected manually, these data points are most likely to be considered as noise and discarded from the clustering process. In our case, since we are sure that our collected RSSI values are not noisy, we do not consider them as noise.

6.2. Example Cases

As an example, suppose, we collect large number of RSSI values as data points after listening into a particular listening strength(dBm level). If the data points are clumped into one region, then the standard deviation, i.e. the eps value would be small. In this case, the $minSampCount$ would be large. So, even the eps value would be small, requirement for $minSampCount$ would go up and maintain the true number of clusters.

Again for above example, if standard deviation, i.e. the eps value goes up, $minSampCount$ would mathematically go down. This is to maintain the fact that, higher the eps value is, the cluster is more stretched out and there are distant data points which are also part of the same cluster. If there are any small group of data points that have enough weight to be added as part of a cluster, but they are far away from any densely located region, they deserve to form their own cluster. In this case, a smaller $minSampCount$ would enable them to form a cluster of their own, despite being a smaller population.

7. Evaluation

7.1. Connectivity Score

In this section, we examine the connectivity scoring algorithm, as well as give real experimental results. We formulated a connectivity scoring algorithm based on the information that we collect from four core evaluation metrics. The connectivity score is computed each time new set of data is added to it. If the new set of data happens to be normal, i.e. no attack is taking place, then the increment is very insignificant. On the other hand, if new sample of data shows significant difference in all four core evaluation metrics, then the score is incremented by a large value, in this case 1. The alarm is raised when the score hits a certain threshold. Since the score has been updated and incremented over a long period of time, if the score hits the threshold, it is safe that an attack is taking place.

7.2. Setup

The evaluation of this system was done with a laptop that is setup to host a wireless hot-spot, which acts as the wireless access point in this experiment. The laptop was placed at (13,7) as shown in **Fig. 1**). The client is another laptop that was placed at (5,5). We kept both our client and wireless access point stationary during the evaluation. It is important to understand that, the choice of location where to put the client and attacker machine, in respect to each other matters. If the attacker is located very close to the victim machine, signal pattern created by both victim and attacker tends to be of similar pattern. On the other hand, if the attacker keeps distance from the victim, their individual signal patterns should vary to a measurable difference. In the former scenario, it is hard to detect an attacker, hence we name it *Hard attack*, and the later scenario is *Normal attack*.

Before each test, we took 480 samples to ensure that each test was not using stale training data. Three experiments were done to test the validity of this tool. The first was a control where an attacker was not present at all. This is labeled as *Control* in **Fig. 6**.

The second scenario had an attacker that was placed in an area where the attacker and client were close to each other. For this experiment, the attacker was placed at (6,1). This is the worst case scenario and is labeled as *Hard Attack* in **Fig. 6**.

The third scenario contained an attack in which the victim and attacker were located distant from each other. For this experiment, the attacker was placed at (3,8). This is the most common scenario since in real life, an attacker is not likely to sit next to the victim and this scenario is labeled as *Normal Attack* in **Fig. 6**.

Algorithm 6: Scoring

```
    Input: $TestProfile$, $NormalProfile$
    Output: $Score$
    Initialization: $Score = 0$
1   for $Method$ in $TestProfile$ do
2       for $Score$ in $Method$ do
            // Get the 1st, medium, and 3rd quartile from the NormalProfile
3           $L_{Q1} = 1stQ(NormalProfile[Method][Score])$
4           $L_{Q3} = 3rdQ(NormalProfile[Method][Score])$
5           $L_{Med} = med(NormalProfile[Method][Score])$
6           if $Method$ is dBm-Clusterization then
7               if number of clusters != number of learned clusters then
8                   $Score$ += 1
9                   $continue$
10          if $L_{Q1} < Score < L_{Q3}$ then
11              if $Score$ not within $L_{Med}$ +- $(L_{Med})/2$ then
12                  if $Method$ is not dBm-Clusterization then
13                      $Score$ += 0.25
14                  else
15                      $Score$ += 0.01 // avoid over-weighting
16          else
17              if $Method$ is not dBm-Clusterization then
18                  $Score$ += 1
19              else
20                  $Score$ += 0.3 // avoid over-weighting
```

7.3. Results

In **Fig.6** , we can see that when an attacker is located not near to the actual client.. Unfortunately, in the case that there is large signal pattern overlap between the attacker and client, the attack becomes much harder to reliably detect. But even in the worse case scenario, the attack still looks much different than when there isn't an attack occurring at all.

In **Fig. 6**, we can can see that when there is a high similarity in RSSI-pockets between the attack and client, our solution has trouble differentiating between an attack and outliers in normal behavior. But, when this is not the case, we can see that there is a very large distinction between an attack and outliers in normal behavior.

Intuitively this makes sense. our solution relies on a client and attacker having different signal pattern at different dBm values (associated with the wireless access point's antenna gain strength). When this is not the case, our solution does not work as well as it could. But, even in the worse case scenario, as shown in **Fig.7** , it is still able to detect the presence of an attack with high precision.

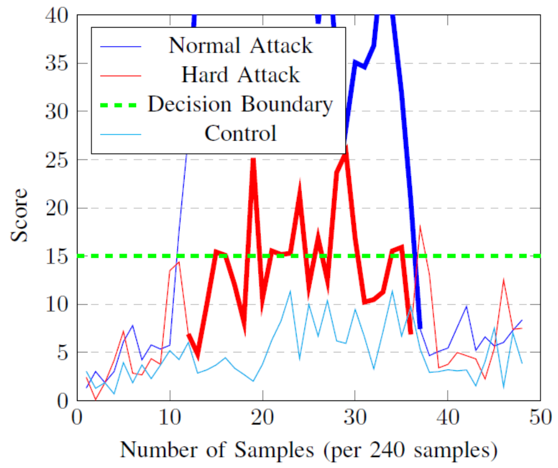

Figure **6:**A bold line indicates an attack is actually happening, while a non-bold line indicates no attack. Any point above the *Decision Boundary*, which at at y=15, is interpreted as an attack.

	Control	Hard Attack	Normal Attack
Accuracy	100%	72.92%	95.83%
Recall	N/A	52.0%	100%
Precision	N/A%	92.86%	92.59%
FPR	0%	4.35%	8.70%

Figure **7:**This table shows that even in the worse case scenario, this is still able to detect a MAC spoof attack with low false positives and high precision.

8. Future Work

While exploring the potentials of this tool, we discovered future works that could be implemented based off of the ideas presented in this paper.

8.1. Localize Disturbances

One finding was that moving objects, such as people, can cause disturbances to connectivity score. With this in mind, we could potentially use the disturbances and the concepts presented in this paper to track where those disturbances.

This means that it could be possible to track where people are simply by using an access point, or for more accuracy, multiple access points. Using concepts presented in this paper, it may be possible to make techniques such as those presented in [9], much more accurate.

8.2. Use multiple monitor stations

We can also utilize the ideas presented in this paper to further increase the ideas presented in other papers, such as [8], [15], [4], [19], and [16], by introducing multiple access points that vary their signal strengths . By doing this, we can allow for localization of the attack and client to better increase reliability of the system.

9. Conclusion

This work presented a tool that uses a single access point to determine when deviations occur in a client's signal strength at different antenna gain strength levels to determine when a MAC spoofing attack has occurred.

References

[1] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007. 1

[2] Armin Daneshpazhouh and Ashkan Sami. Entropy-based outlier detection using semi-supervised approach with few positive examples. *Pattern Recognition Letters*, 49:77–84, 2014. 1

[3] Murat Demirbas and Youngwhan Song. An rssi-based scheme for sybil attack detection in wireless sensor networks. In *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*, pages 5–pp. IEEE, 2006. 1

[4] Eiman Elnahrawy, Xiaoyan Li, and Richard P Martin. The limits of localization using signal strength: A comparative study. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 406–414. IEEE, 2004. 2, 8

[5] Daniel B Faria and David R Cheriton. Detecting identity-based attacks in wireless networks using signalprints. In *Proceedings of the 5th ACM workshop on Wireless security*, pages 43–52. ACM, 2006. 1

[6] Jonathan Grimes. Alt-rssi. 1

[7] Fanglu Guo and Tzi-cker Chiueh. Sequence number-based mac address spoof detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 309–329. Springer, 2005. 1

[8] Brian Ferris Dirk Hähnel and Dieter Fox. Gaussian processes for signal strength-based location estimation. In *Proceeding of robotics: science and systems*, 2006. 2, 8

[9] B Heredia, Manuel Ocaña, Luis M Bergasa, MA Sotelo, Pedro Revenga, R Flores, Rafael Barea, and Elena López. People location system based on wifi signal measure. In *2007 IEEE International Symposium on Intelligent Signal Processing*, pages 1–6. IEEE, 2007. 8

[10] Victoria J Hodge and Jim Austin. An evaluation of classification and outlier detection algorithms. *arXiv preprint arXiv:1805.00811*, 2018. 1

[11] Mon-Fong Jiang, Shian-Shyong Tseng, and Chih-Ming Su. Two-phase clustering process for outliers detection. *Pattern recognition letters*, 22(6-7):691–700, 2001. 1, 4

[12] Amin Karami and Ronnie Johansson. Choosing dbscan parameters automatically using differential evolution. *International Journal of Computer Applications*, 91(7):1–11, 2014. 5

[13] Kamran Khan, Saif Ur Rehman, Kamran Aziz, Simon Fong, and Sababady Sarasvady. Dbscan: Past, present and future. In *The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014)*, pages 232–238. IEEE, 2014. 5

[14] Guenther Lackner, Udo Payer, and Peter Teufl. Combating wireless lan mac-layer address spoofing with fingerprinting methods. *IJ Network Security*, 9(2):164–172, 2009. 1

[15] Andrew M Ladd, Kostas E Bekris, Algis Rudys, Lydia E Kavraki, and Dan S Wallach. Robotics-based location sensing using wireless ethernet. *Wireless Networks*, 11(1-2):189–204, 2005. 2, 8

[16] Vahideh Moghtadaiee and Andrew G Dempster. Wifi fingerprinting signal strength error modeling for short distances. In *2012 International conference on indoor positioning and indoor navigation (IPIN)*, pages 1–6. IEEE, 2012. 2, 8

[17] Yong Sheng, Keren Tan, Guanling Chen, David Kotz, and Andrew Campbell. Detecting 802.11 mac layer spoofing using received signal strength. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 1768–1776. IEEE, 2008. 1

[18] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1313–1328. ACM, 2017. 1

[19] Wei Wang, Alex X Liu, Muhammad Shahzad, Kang Ling, and Sanglu Lu. Understanding and modeling of wifi signal based human activity recognition. In *Proceedings of the 21st annual international conference on mobile computing and networking*, pages 65–76. ACM, 2015. 2, 8

[20] Chen Xiaoyun, Min Yufang, Zhao Yan, and Wang Ping. Gmdbscan: multi-density dbscan cluster based on grid. In *2008 IEEE International Conference on e-Business Engineering*, pages 780–783. IEEE, 2008. 5