Machine Learning Engineer Nanodegree

Capstone Project

Kenneth R. Farr III

July 28th, 2018

I. Definition

Project Overview

Cyber Security is the domain that focuses on cost-effectively reducing losses in and around digital assets. This includes accidental losses caused by mistakenly emailing confidential information, and malicious losses such as a stolen mobile device or a breached database. A popular modeling technique for computing the risk (dollars lost per year) of an asset is the Factor Analysis of Information Risk (FAIR) ontology. This model accepts as input the frequency of a loss event (how often a loss is expected to occur) and the loss magnitude (when a loss occurs, how much does it cost the organization).

A Cyber security industry standard exists to catalog all known vulnerabilities: Common Vulnerabilities and Exposures (CVE). The CVE list details every known vulnerability and applies attributes to them, such as level of involvement to exploit, closeness to the asset (physical presence, adjacent network, internet) etc. These elemental attributes are used to derive a Common Vulnerability Scoring System (CVSS) score.

The annual Verizon Data Breach Investigations Report (VDBIR) is highly regarded by Cyber Security Professionals and in the 2015 report, page 21 stated that "...we agree with RISK I/O's finding that a CVE being added to Metasploit is probably the single most reliable predictor of exploitation in the wild". Metasploit DB (MDB) contains publicly available exploits and is freely searchable.

As a member of a Research and Product Development team for the leading Cyber Security Risk Quantification Company, this kind of research (predicting exploits) is both interesting and important to our work.

This project attempts to predict whether a vulnerability (CVE) will eventually have an exploit created for it.

Problem Statement

In a business landscape, prioritizing capital investments is critical to maintaining a competitive advantage. A portion of all capital expenditures include risk reduction, and cyber security is no exception. It would be cost-prohibitive for a business to attempt to effectively patch every system against every vulnerability, therefore a business must prioritize which vulnerabilities get patched.

With over 5000 new CVE entries per year, a business must determine which vulnerabilities affect them, and by predicting which of these new vulnerabilities will most likely result in a usable exploit, the business can prioritize their patching efforts.

Utilizing predictive machine-learning algorithms, this research attempts to pinpoint which vulnerabilities are most likely to become usable exploits, helping prioritize which systems need priority patching. When a vulnerability is discovered and a CVE is created for it, attributes are assigned to the CVE that help determine the type and severity of the vulnerability. These attributes are:

- *Attack/Access Vector*: Level of access to vulnerable system, includes Physical Access, Local Access, Adjacent Access, or Network (Remote) Access
- *Attack Complexity*: Whether or not extenuating circumstances are required to exploit the vulnerability, includes Low or High
- *Authentication/Privileges Required*: Whether or not access credentials are required before the vulnerability can be exploited, includes None, Low, and High
- *Confidentiality*: Measures the impact to underlying system if the vulnerability is exploited, includes None, Low, and High
- *Integrity*: Measures whether an exploit would affect the system's level of trustworthiness, includes None, Low, and High
- *Availability*: If exploited, would the general availability of the system be impacted, includes None, Low, and High

It is hypothesized that these attributes provide enough variability and insight into the nature and exploitability of a vulnerability to provide a preemptive prediction and allow priority patching to reduce exposure to exploits.

Metrics

Much like cancer detection (general direction, not scale) predicting that a vulnerability will become an exploit has the same class imbalance (more negatives to positives) and the same desire to catch all or as many as possible Positives, allowing for an increase in False Negatives. When used as a first-level scanner, reducing any True Negatives while catching as many True Positives as possible is the goal. As such, the model will be designed to allow for a lower precision with a higher recall. This will be accomplished by using a F-Beta Scoring method. The level of F-Beta will be evaluated to provide as much meaningful predictability as possible without sacrificing too many False Negatives. It is expected that Beta will need to be rather high to capture enough of the positive results. Our dataset is heavily skewed to negatives results (60:1).

The data will be broken into train and test data sets by selecting randomly, and use k=5 cross-fold validation. While perhaps naïve, the model will not assume that any earlier CVEs will affect any later CVEs.

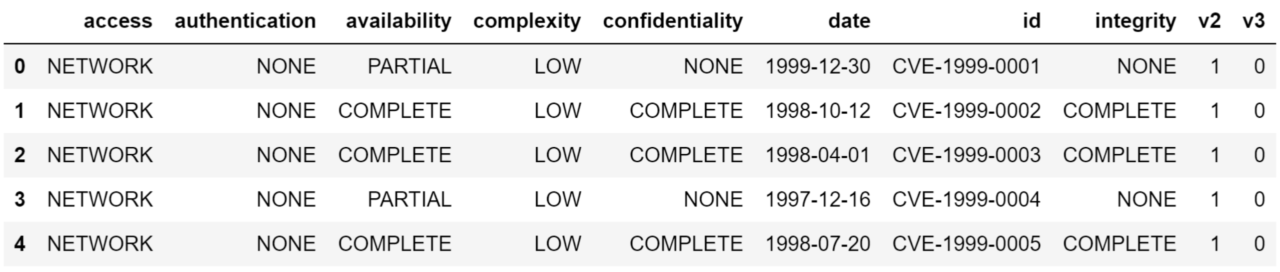
Three different methodologies will be tested: linear regression, adaboost/decision tree, and support vector machines. For adaboost/decision tree and the SVM, hyper-parameter tuning will be employed to determine the best training parameters.

II. Analysis

CVEs

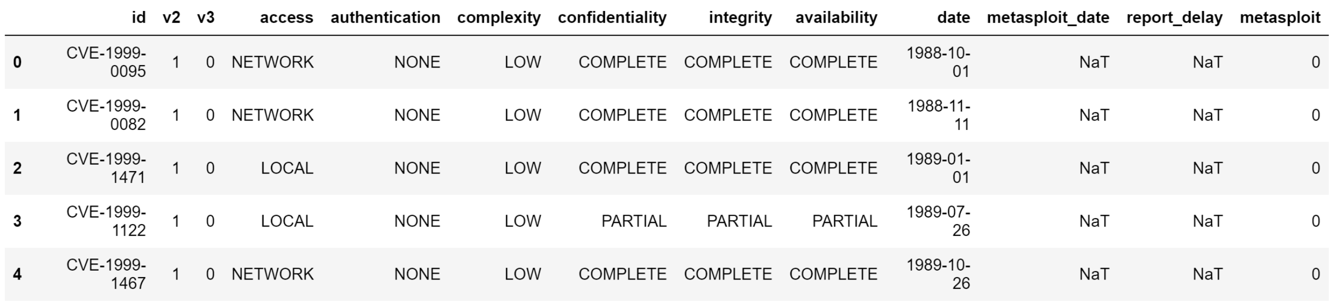
The CVE dataset provided by Mitre contains quite a bit of information for each vulnerability. Some of these values may have impact on whether an exploit is created for the vulnerability. In addition to the CVSS categorical values that are used in this project, the platform and platform versions affected by the vulnerability may be a good indicator of whether the vulnerability is turned into an exploit. However, due to the free-form nature of these entries, the amount of work, and potentially Natural Language Processing required pushes that portion of the dataset outside the scope of this project. As such, only the categorical attributes that compose the CVSS score will be used as a way to limit the scope of the project and still provide meaningful results.

The following image shows the raw CVEs as extracted from Mitre's JSON format with only the variables being inspected.



	access	authentication	availability	complexity	confidentiality	date	id	integrity	v2	v3
0	NETWORK	NONE	PARTIAL	LOW	NONE	1999-12-30	CVE-1999-0001	NONE	1	0
1	NETWORK	NONE	COMPLETE	LOW	COMPLETE	1998-10-12	CVE-1999-0002	COMPLETE	1	0
2	NETWORK	NONE	COMPLETE	LOW	COMPLETE	1998-04-01	CVE-1999-0003	COMPLETE	1	0
3	NETWORK	NONE	PARTIAL	LOW	NONE	1997-12-16	CVE-1999-0004	NONE	1	0
4	NETWORK	NONE	COMPLETE	LOW	COMPLETE	1998-07-20	CVE-1999-0005	COMPLETE	1	0

The Metasploit database was merged in, and the report-delay was calculated between the time the CVE was published and the Metasploit Exploit was published. Data was then sorted by CVE year.



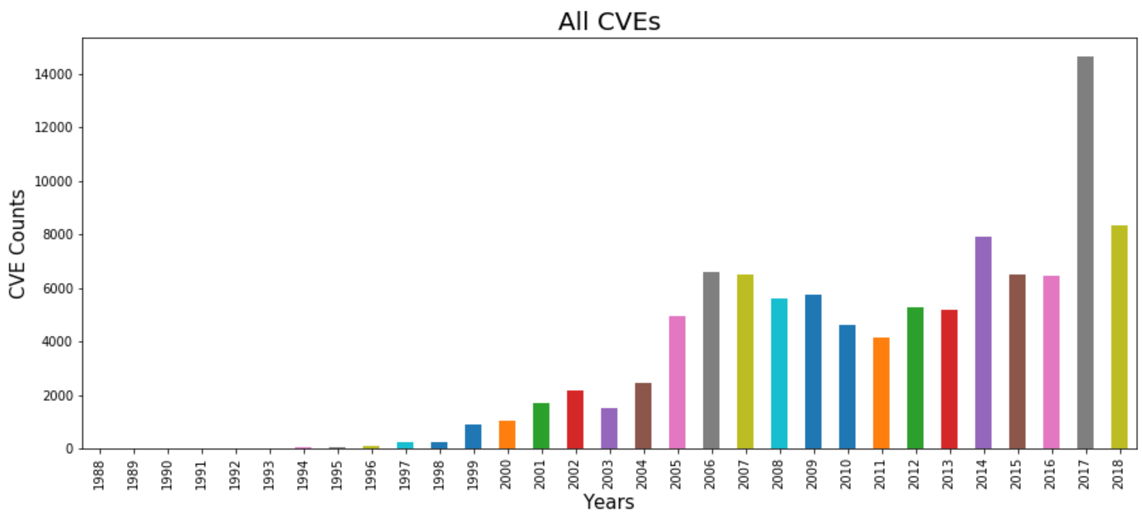
	id	v2	v3	access	authentication	complexity	confidentiality	integrity	availability	date	metasploit_date	report_delay	metasploit
0	CVE-1999-0095	1	0	NETWORK	NONE	LOW	COMPLETE	COMPLETE	COMPLETE	1988-10-01	NaT	NaT	0
1	CVE-1999-0082	1	0	NETWORK	NONE	LOW	COMPLETE	COMPLETE	COMPLETE	1988-11-11	NaT	NaT	0
2	CVE-1999-1471	1	0	LOCAL	NONE	LOW	COMPLETE	COMPLETE	COMPLETE	1989-01-01	NaT	NaT	0
3	CVE-1999-1122	1	0	LOCAL	NONE	LOW	PARTIAL	PARTIAL	PARTIAL	1989-07-26	NaT	NaT	0
4	CVE-1999-1467	1	0	NETWORK	NONE	LOW	COMPLETE	COMPLETE	COMPLETE	1989-10-26	NaT	NaT	0

CVEs may have one or both CVSS V3 and CVSS V2 attributes. Where appropriate these were merged together taking the V3 when available and using V2 when not. It is expected that V3 is superior by the nature of being newer. Select older CVEs have had V3 attributes applied, but it wasn't until 2015/2016 did CVEs really start consisting of V3 attributes.

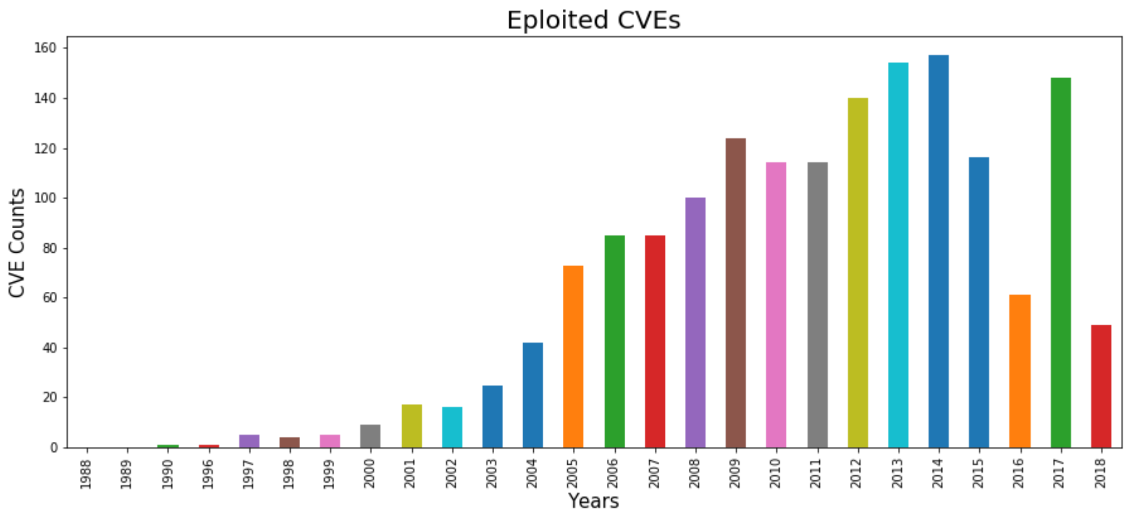
Of the 111,520 CVEs available when data was collected, 102,947 (92%) of them have either V2 or V3 attributes. The 8,573 (8%) that were missing V2/V3 attributes were sometimes REJECTED CVEs or just incomplete. These observations were omitted from the population since there is no predictive ability or exploits for these vulnerabilities.

Of the 102,947 kept observations, 29,392 (28%) had V3 attributes and every one (100%) had V2 attributes. There were no cases where a V3 attribute was available and no V2 attribute.

In the rest of this report, when the CVE population is mentioned as a whole, and without any additional qualifiers, it is expected to be the CVE population that has V2 and/or V3 attributes, the 102,947 mentioned above.



The above figure shows how the number of reported vulnerabilities has increased up to around 2005/2006 and tapered off. There appears to be an influx in 2017 and it is unknown whether that will carry over into 2018 or not.



The above figure shows the number of exploits per year. It looks like the number of exploits increases along with the number of vulnerabilities. Running a correlation test against these datasets shows that there is a strong correlation (0.758) between the number of vulnerabilities in a given year and the number of vulnerabilities.

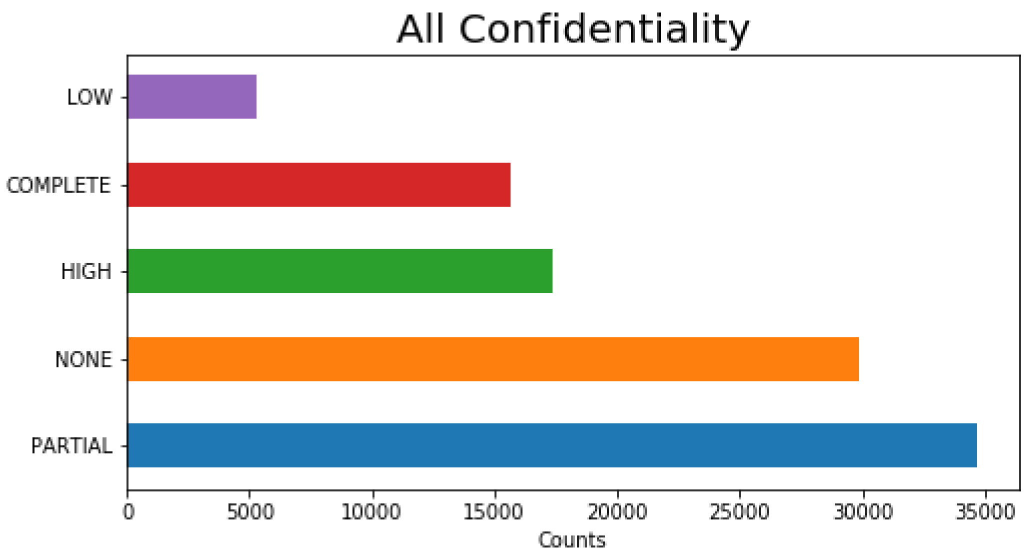
CVSS Attributes

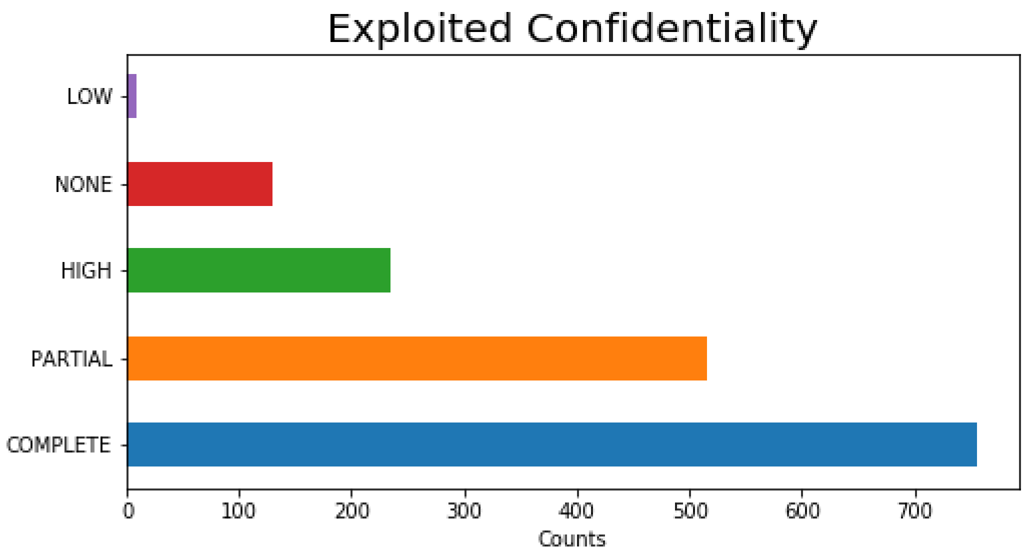
There are 6 CVSS Attributes that have categorical properties:

- *Attack/Access Vector*: Level of access to vulnerable system, includes **Physical Access**, **Local Access**, **Adjacent Access**, or **Network (Remote) Access**
- *Attack Complexity*: Whether or not extenuating circumstances are required to exploit the vulnerability, includes **Low** or **High**
- *Authentication/Privileges Required*: Whether or not access credentials are required before the vulnerability can be exploited, includes **None**, **Low**, and **High**
- *Confidentiality*: Measures the impact to underlying system if the vulnerability is exploited, includes **None**, **Low**, and **High**

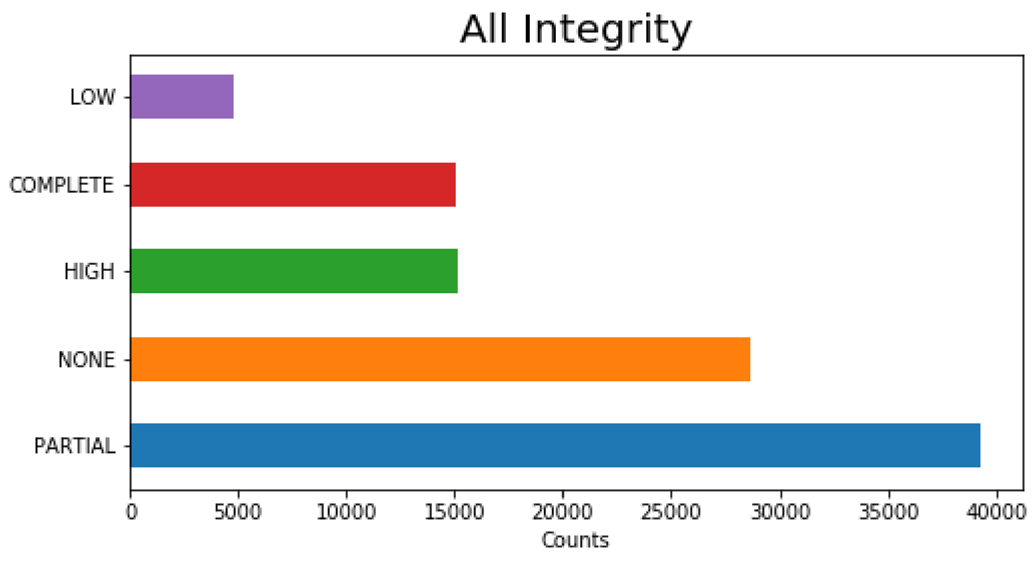
- *Integrity*: Measures whether an exploit would affect the system's level of trustworthiness, includes **None**, **Low**, and **High**
- *Availability*: If exploited, would the general availability of the system be impacted, includes **None**, **Low**, and **High**

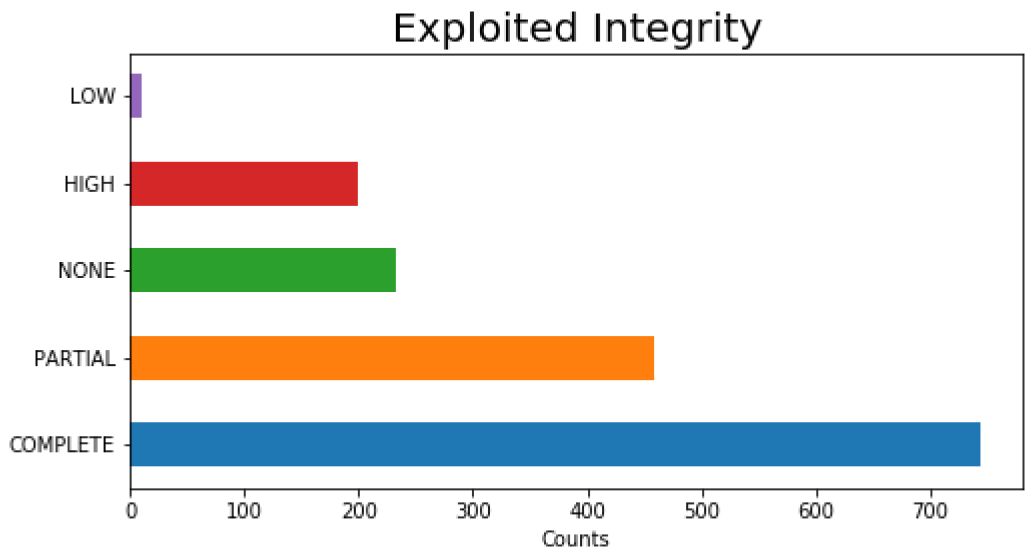
Of these 6, only Confidentiality, Integrity, and Availability differ greatly between exploited vulnerabilities and non-exploited vulnerabilities.



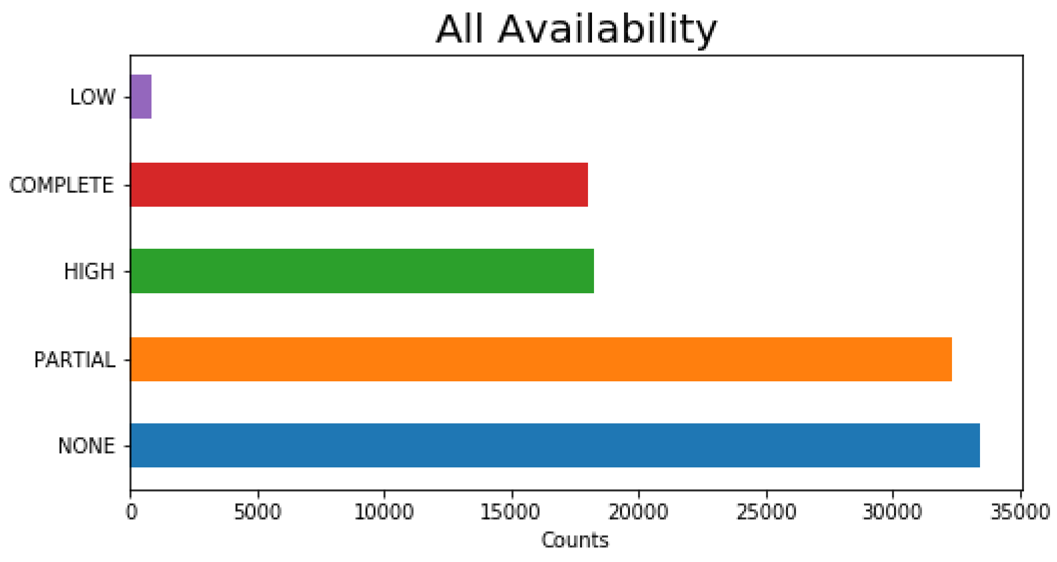


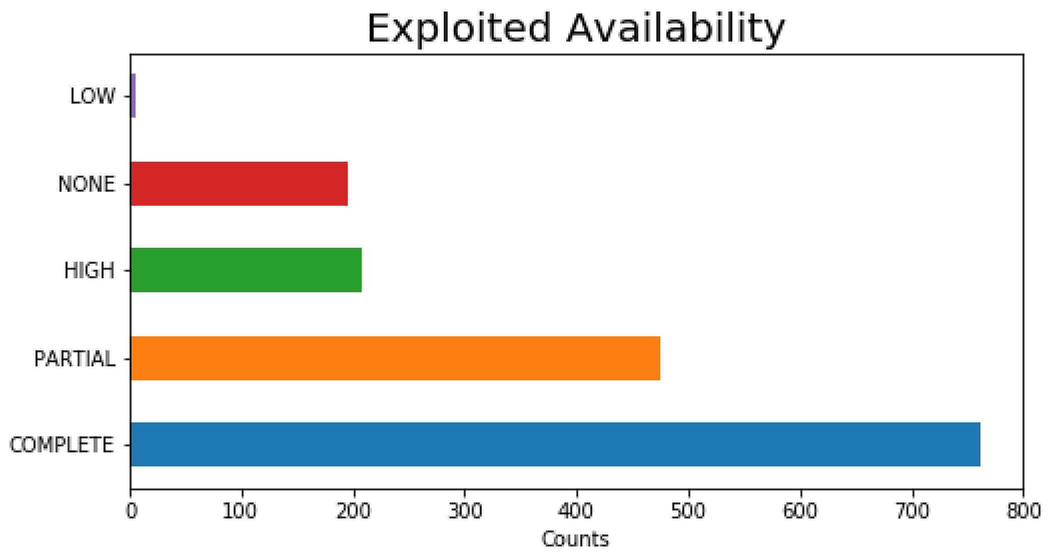
For confidentiality, it appears as if **Complete** and **Partial** confidentiality result in more focus for exploits, which seems to reason that attackers are interested in vulnerabilities that produce the most confidentiality loss as opposed to benign exploits.





In the same vein as confidentiality, exploits targeting vulnerabilities where integrity is listed as **Complete** and **Partial** are also of increased focus.





The same applies to availability for vulnerabilities listed as **Complete** and **Partial** loss of availability. Exploits targeting **Complete** and **Partial** losses in the **CIA** triad (Confidentiality, Availability, and Integrity) categories are not at all surprising. Vulnerabilities often require extensive resources to exploit, going after vulnerabilities that do not disrupt or product data would be of little use.

Exploit Date

One thought going into this project was that the time between when a vulnerability was published and an exploit was published could be used to gauge the potential lag-time and remove any of the more recent vulnerabilities that are within the expected lag time. This turned out to be more difficult than expected since there appears to be many exploits that are published before the vulnerability that they attack was published.

Lag Statistics	
Count	1536
Mean	-66 Days
Std-Dev	398 Days
Minimum	-3326 Days

Lag Statistics	
25%	-37 Days
50%	-5 Days
75%	-1 Days
Maximum	4465 Days

One interpretation for this could be that some exploits are being repurposed for new vulnerabilities as they are released and the exploit date is not updated to reflect this. It's also possible that the data was not captured correctly, though this seems less plausible. In any event, the lag time is not usable as originally expected and as such is left for potential future work.

Algorithms and Techniques

There are many machine-learning techniques available to this problem. Before deciding which algorithms should be applied, it's best to identify the key characteristics of our problem that will help narrow down the available methods.

- Predictive: we need to predict an output
- Binary Classification Output: our predictive output is one of two categories, Exploited or Not-Exploited
- Supervised Learning: we have training data and are trying to predict an output
- Categorical Input: all input parameters are categorical
- Biased Data: we have a 60:1 in favor of negative training cases
- Explainable: when spending money, it's best to have explainable predictions

In the realm of simplest, categorical predictive, and explainable, **Logistic Regression** will be the first algorithm employed, followed by an **Adaboosted Decision Tree**, and finished with a **Support Vector Machine**. These three were chosen for their flexibility and explainability.

For each method hyperparameters will be tuned using GridSearchCV. Weighting will be **balanced** for all, which will aid with the biased dataset (60:1 negative:positive).

For Linear Regression the liblinear algorithm will be utilized due to our smaller dataset and lack of multi-classification output. Regularization will be tuned in the range [0.1, 0.25, 0.5, 0.75, 1, 10, 100, 1000].

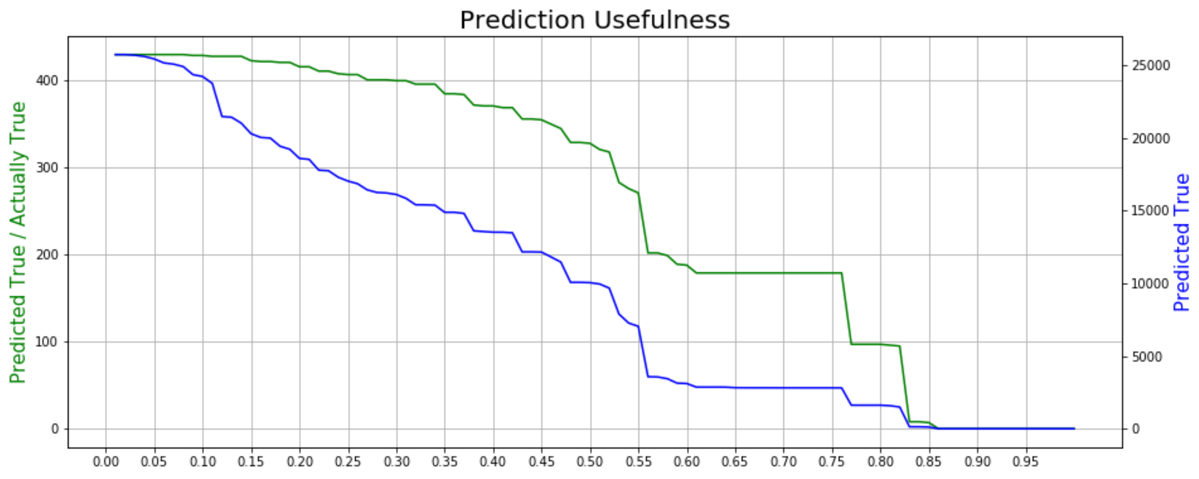
For the Ada Boosted Decision Tree the number of estimators will be checked from 1-101. Both criterion options will be tested: **gini** and **entropy**. Additionally, both splitter methods will be tested: **best** and **random**.

The Support Vector Machine will test different kernels which each have their own parameters. For the three kernels tested, regularization will consist of [1, 10, 100, 1000]. Degrees [1, 2, 4, 8, 10] will be test for the **poly** kernel, and gamma [0.001, 0.0001] will be tested for the **rbf** kernel.

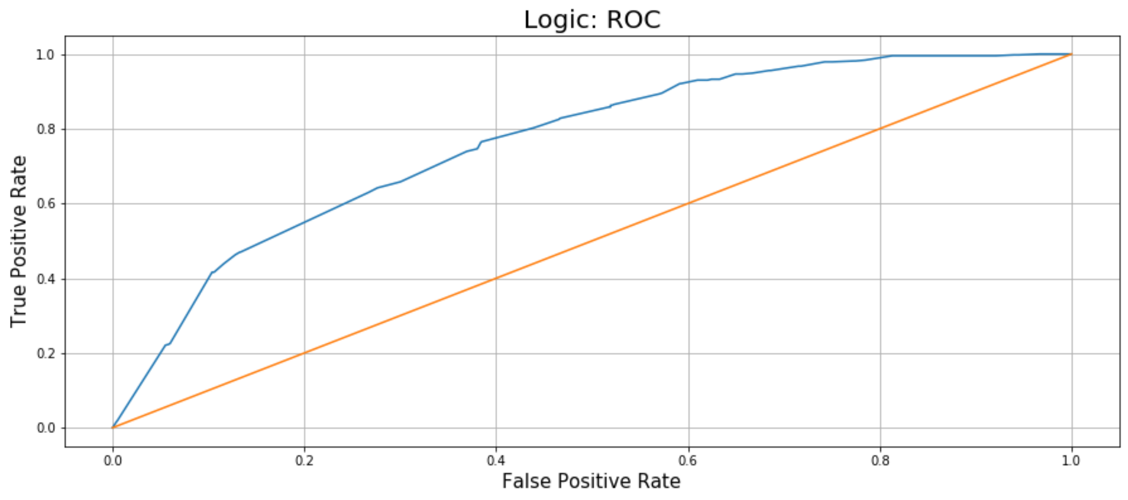
Benchmark

For a biased dataset like this one, determining whether to focus on recall or precision was important. After implementing a simple, default logistic regression, prior to any hyper-parameter tuning, the following threshold graphs were compared.

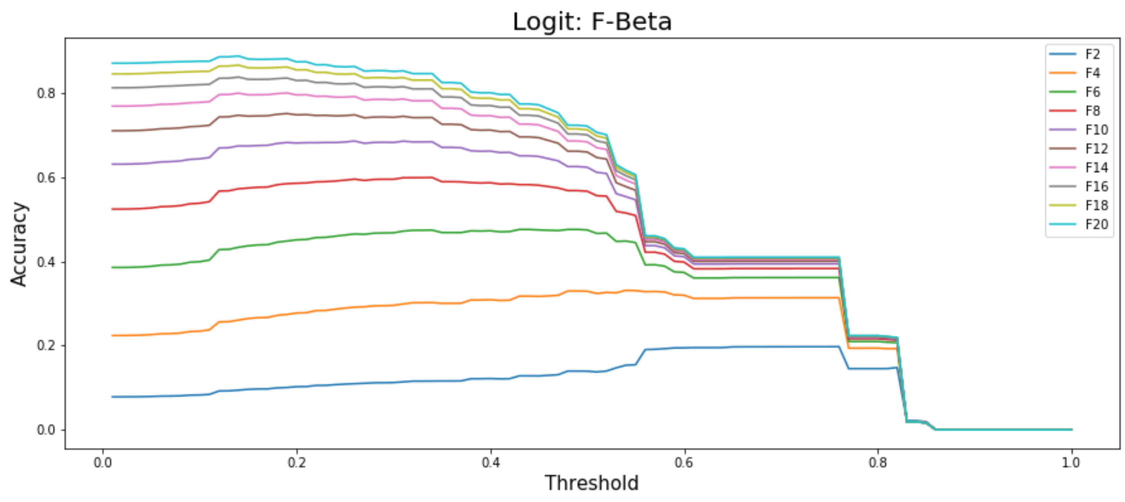
The following graph charts the tradeoff in False Positives and True Positives. With a low threshold, more observations are labeled Positive, increasing both our False Positive rate and our True Positive rate. As the threshold increases so does the False Positive and True Positive rate. While interesting, it's difficult to gauge what threshold here provides the best gain.



Next an Receiver Operating Characteristic (ROC) curve was charted. This provided enough guidance to believe that there is at least some usable predictive power in this data.



Wanting to reduce the number of False Positives and increase the number of True Positives, one has to focus on recall. It was deemed best to implement an F-Beta Scoring system where Beta can be selected such that an appropriate recall is achieved. Prior to hyperparameter tuning for Logistic Regression, a baseline was achieved and scored using the following F-Beta scores: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20.



Moving forward with hyperparameter tuening, An F10 Scoring system will be utilized for all training methods. This intuitively feels like it provides enough recall to reduce False-Positives but retain enough True Positives. There is no free lunch here.

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:

- *If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?*
- *Based on the **Data Exploration** section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?*
- *If no preprocessing is needed, has it been made clear why?*

Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:

- *Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?*
- *Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?*
- *Was there any part of the coding process (e.g., writing complicated functions) that should be documented?*

Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- *Has an initial solution been found and clearly reported?*
- *Is the process of improvement clearly documented, such as what techniques were used?*
- *Are intermediate and final solutions clearly reported as the process is improved?*

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*

Future Work

CPEs, exploit-cpe lag publish

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?