Pythonic *Magna Carta* Of Cyber Security

By Jacob H. Barrow

Dedication

I would like to dedicate this book to my family.

Thank you for helping start something beautiful. This is a culmination of all our testaments. A tribulation of sheer will and perseverance. A time told tale and testimony of strength and blessings. A constance of God’s grace and love.

Title Explanation

Cyber has been deemed the new Wild West Frontier! Currently, there is no transnational cyber framework that governs how each country collectively deals with cyber criminals and digital crimes. To make matters worst, most of the companies add on security instead of baking it in from the start; leaving huge cyber holes that allow Advanced Persistent Threats (APT) to march right on through. This book helps tackle this issue by addressing all of the relevant skill sets needed to success in cyber security and cyber warfare.

Labeled as the Magna Carta of Cyber Security, this book reflects the true homitage of that background. Cemented as a foundational documentation for the United States Constitution, rights of citizens, extent of government regulation or collection, separation of Church from government, and the right to trial before the people. Without these inalienable rights, our nation is no different than the ones our military fights against. So, this book is coined off our American history's beginnings because we must navigate these same principles in the cyber world. Unlike other malicious nations, we are bound by the privacy rights of individual and the company, we must uphold a valid, undoubtable chain-of-custody, our work may be scrutinized by the American people, and lastly, our work should not influence the religious ideologies of those it's meant to serve.

Story Behind Python

Python was catchy! The designer of Python, Guido Van Rossum, saw an opportunity to intrigue others by creating a vague name for a programming language based off the hit 1970s British comedy series, " Monty Python's Flying Circus" (Monty Python's Official Website, 2021). Yes you read that correctly, what better way to learn cybersecurity than through the testimonies of Monty Python. To proceed within this book, it is required reading to go watch these movies. Doing so has absolutely nothing to do with programming but will get you in the spirits to develop. There is something to be said about creativity while designing, if you go in flat footed then your designs will also be unoriginal.

Table Of Contents

Introduction

Description

Security analysts can decipher cybersecurity requirements, capture key stakeholder input, design a Unified Modeling Language (UML) diagram for these key performance parameters (KPI), and then implement a fully functional solution within an acceptable turnaround. Junior enlisted, and junior industry members, should be able to understand the threat environment and produce useful tools to augment the enterprise after reading through this.

Who Should Read It?

This was designed for United States individuals in the industry, government, and military that have one month of Python under their belt with ZERO cyber experience. For the industry, this would be persons with two months' experience on the job. For the military, this would be those fresh out of military bootcamp with an aptitude for cyber or programming.

Python Basics

Taking the journey through Python Object Oriented Programming (OOP) requires some initial investment on the reader's part. This book was designed to make it easier and faster to learn OOP, but you should still set enough time to learn. It is the expectation of the author that you have one month of Python programming under your belt, but for those who do not, we will go through the essentials of Python. Please remember that the first sections within this book were meant for a quick review. With that said, let us now dive into the wonderful world of Python.

Here we will briefly explain primitive data types, or those data types that are fundamental to Python. You can consider primitive data types as the necessary building blocks to compose the rest of the language, or you could think of them as the needed ingredients to make more complex dishes. However, you view primitive data types, there are a few of interest to get us started: integer (int), floating point (float), Boolean (bool), and strings (str).

Isn’t everything related to a number somehow? If you ask the designers at the Puzzle Factory in Maryland, they could probably find some odd mathematical pattern with 100 different ways this book was written. Therefore, numbers are one of the most useful items in Python. You have two different types of numeric value: int and float. Integers will be the whole numbers, all stored as 32-bit signed integers in python (Code Bless U, 2021). This means that we have the flexibility of specifying our variable size ahead of time, a feature called duck typing. Python will automatically allocate an integer from -2^31 to 2^31. If you have ever calculated the Fibonacci sequence or tried to crack an RSA cipher in Python, you may have realized that Python will run into an overflow eventually. Those problems are great reason for storing numbers within running lists, which will be covered later. Other problems, like calculating the expected base change or getting the degree to radian conversion, are better suited to floating point, or decimal numbers. These are usually stored with a minimal of 24 bytes in Python (Code Bless U, 2021). Below are a few examples of integers versus floating points. Note that floating points can be stored with scientific notation. Also, within the examples below, you will see data type casting from integer to floating point and vice versa. Data type casting is performed by wrapping the numeric data type in the desired format with that data type’s keyword name. You could truncate information if you cast from a floating point to an integer, with the cast rounding down or truncating all the information to the right of the period.

```
  # Integer
variable =   10

  # Integer casting
variable = int(  10.0  )

  # Floating point
variable =   1.0

  # Floating point stored with scientific notation
variable =   1.15e10

  # Floating point casting
variable = float(  1  )

  ## Output ##

The item   is     ""     is     False  :

The item []   is     False  :

The item {}   is     False  :

The item ()   is     False  :

The item   10     is     True  :

The item Hello   is     True  :
```

Working with numbers is useful, but there are times that a simple true or false value using Booleans will work just fine. Anything that is zero or empty is false. Conversely, anything with information is true. There are numerous data types we will cover such as strings, lists, dictionaries, tuples, etc. If they do not have information then they are false, and if they have anything stored they are true. Even though this is a simple concept, the complexity of such grows exponentially when you start building truth tables, Boolean logic gates, and more. Even though these examples will seem trivial, Cybersploit LLC will be releasing a book later next year on how Boolean logic is the crux for abstract Algebra, ring theory, and advanced cryptology. Also, implementing half Adders in circuitry would blow your socks off if you knew the fundamentals basics were derived from the following Boolean logic.

```
container = [  ""  , list(), dict(), tuple(),   10  ,   "Hello"  ]

  for   item   in   container:
    baseStr =   f'The item   {item}   is'
      if   item:
        print(baseStr +   f' True:   {type(item)}  '  )
      else  :
        print(baseStr +   f' False:   {type(item)}  '  )
```

Strings are special as they are immutable objects that store human-like language for us in our scripts, minus those items that are stored in bytes. Understanding all of the ins and outs of strings in Python cannot be done in a few pages, the number of methods, or class functions, associated with the string data type is mind boggling. A few of these methods include, find, upper, lower, capitalize, isalnum, and much more. This book is not meant to be a complete tutorial on the Python language for beginners, so I encourage you to explore string methods within Python. Instead, we will briefly focus on string conversion, indexing, slicing, and the lower method. This section will use for loops and if statements that will be covered later, but readers who are intuitive, will figure it out as we go along. Learning is about stumbling until you can run.

Indexing and slicing strings is not difficult, as the former is a single spatial position while the latter is usually multiple positions. You can think of indexing as a singular example of slicing. Both concepts require the use of brackets to the right of the string or variable. The formula for slicing is to include the starting point, the finishing point, and the stride, which are all separated by colons. Let us now get hands on to observe this!

```
var =  "Puzzle"

print(var[ 1 ])
 ## Result will be 'u'

print(var[ 1 :])
 ## Result will be 'uzzle'

print(var[ -1 ])
 ## Result will be 'e'

print(var[ -2 : -4 : -1 ])
 ## Result will be 'lz'

print( "Factory" [ 3 :])
 ## Result will be 'tory'
```

You may have noticed that we could index from the front and back of the string, either stored as a variable or as a raw string. The syntax does not require a stopping point or stride. If the stopping point is omitted, then you will be going to the rightmost item. On the other hand, if the stride is omitted you will inherently have a stride of positive one. From left to right starts at zero and goes to the length minus one of all the characters in the string. From right to left, starts and negative one and goes to the negative length of all the characters, but requires a stride of negative one. Now let’s do things with string methods, specifically the isspace method. Also, will use the chr and ord built-in Python methods to convert decimal to ASCII and from ASCII to decimal. ASCII is a specific code page that can be looked up off Wikipedia for your own situational awareness. The code below will shift each of the characters by nine. This algorithm should not be used for any information that should be of importance, nor should it be used in a production environment. Again, do not use homemade cryptology ciphers in production networks for concern of failure to provide confidentiality. If it was not made clear, below is representative of a modified Caesar cipher. We will not cover cryptology extensively in this book; it will be expanded upon in a future book. The point was that strings are very useful, but you need to know how to wield the power of their might!

```python
text = "hello there lets learn python"
ciphertext = ""
shift = 9
mapping = {num: chr(((num + shift) % 26) + 97) for num in range(26)}

for letter in text:
    if not letter.isspace():
        ciphertext += mapping[ord(letter) - 97]
    else:
        ciphertext += letter

print(ciphertext)
```

Conditional logic is one of the most crucial aspects associated with programs. A signature of novice programmers includes a lot of if else statements, but even advanced programmers use conditional logic in some form or fashion to get tasks done. Understanding how to create an effective conditional test is a skill that is acquired over time. But know there are a few ways in which the Python student can leverage conditional statements. The first entails understanding how to set up the right conditional tests. Sometimes you one if statement, multiple elif statements, and a default else statement, whereas other times you may only want a few of these. In the code below we want to check even versus odd, if it's odd and a multiple of 5 do something special with it!

```
target =   15

 if   target %   2   ==   0  :
   print(  f'The target number of   {target}   is even!'  )
 else  :
     if   target %   5   ==   0  :
       print(  f'You are a very special odd number:   {target}  '  )
     else  :
       print(  'You are only kind of special: {target}'  )
```

Code fails, and when it does, it does so badly! Make sure you wrap and protect your code so that it doesn't compromise the rest of your system(s). Understanding the faults in your code is a great start, but sometimes you may need a generic default catch to make sure that the error is accounted for even if you did not expect it! In Python we use the try except block syntax to try code first, then catch that code if it raises an exception. Our example below exemplifies that if we get input that is unvalidated, such as a string zero instead of an integer zero, we won't have the proper exception to catch that error. It becomes extremely important to allocate a default exception to catch these future unknown errors.

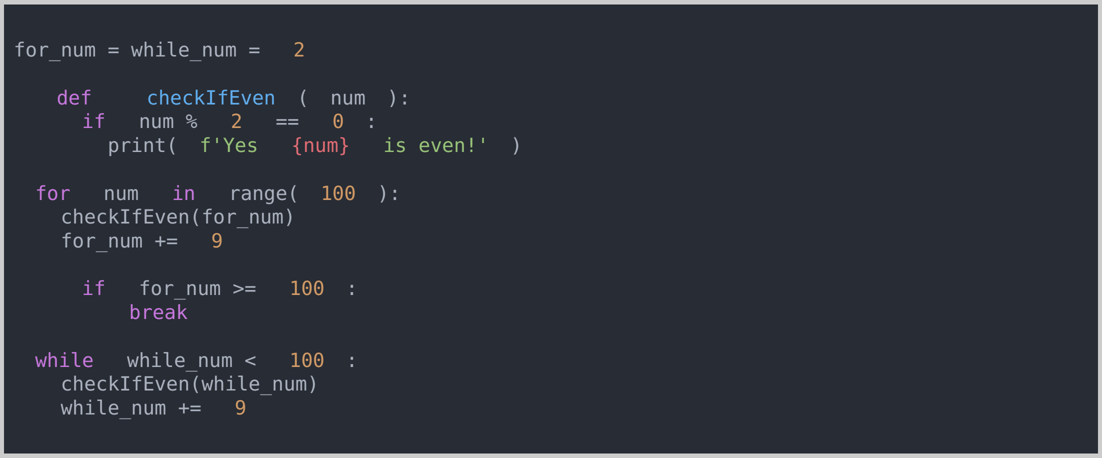
```
 try  :
   print(  10   /   "0"  )
 except   ZeroDivisionError:
   print(  "Don't divide by zero silly!"  )
 except  :
   print(  "Here is what we really catch because the devisor is of string type"  )
```

Functions have numerous features associated with them depending on what version of Python you are working with. All functions have a signature type to include the keyword def, the name of the function, the parameters, and the body of the function. Some may or may not have a return type. In recent versions of Python, we can dynamically type check the parameters and return type through a similar methodology as input and output validation. In the code below, we see that we can perform input validation to make sure the parameter is an integer. We can also perform output validation to make sure the return type is of type Boolean.

```
def     checkFive  (  num: int  ) -> bool:
  if   num ==   5  :
      return     True
  else  :
      return     False
```

Runtime Complexities and Looping

Iterations within Python can be equated to the infinite time loop that Dr. Strange put on Dormammu in the Dark Dimension. Even if one iteration resulted in his death, he could relive another simulation. Using for and while loops we ourselves can put a time loop in our Python scripts. A for loop uses a range keyword and has almost the same syntax that we used for slicing strings: start, stop, and stride. You can also use negative indexing, using negative strides, that will iterate backwards for you. Again, you need not incorporate a stop or stride, and will take defaults of the entire length of characters minus one and of one. If you want to include a stride, you must also include a stopping point. One of the times you want to use iteration is when you are continuously searching for a target value within a non-sorted array or matrix. On the other hand, when you are counting down from a value or condition, or are searching through a sorted array or matrix, you probably want to use a while loop. There are some nuisances associated with while loops that are different, for example we are not concerned with ranges but with whether a test condition is still true. Since we do not have do-while loops in Python, the while loop will only enter and continue the loop if the test condition is true. Below is code showing the differences between using a for and while loop for the following problem set count the numbers that are even and less than one hundred, as we add nine to our initial number of 2. Within each iteration, whether that is a for or while loop, we must have the logic to test if the number is even and then print that number out to the screen. Using a modulus operator within an if condition statement is helpful to provide us the ability to check if the number is odd or even. Once that is determined, we can print the number out to the screen and increment the number by nine.



```
for_num = while_num =   2

    def     checkIfEven  (  num  ):
      if   num %   2   ==   0  :
        print(  f'Yes   {num}   is even!'  )

  for   num   in   range(  100  ):
    checkIfEven(for_num)
    for_num +=   9

      if   for_num >=   100  :
          break

  while   while_num <   100  :
    checkIfEven(while_num)
    while_num +=   9
```

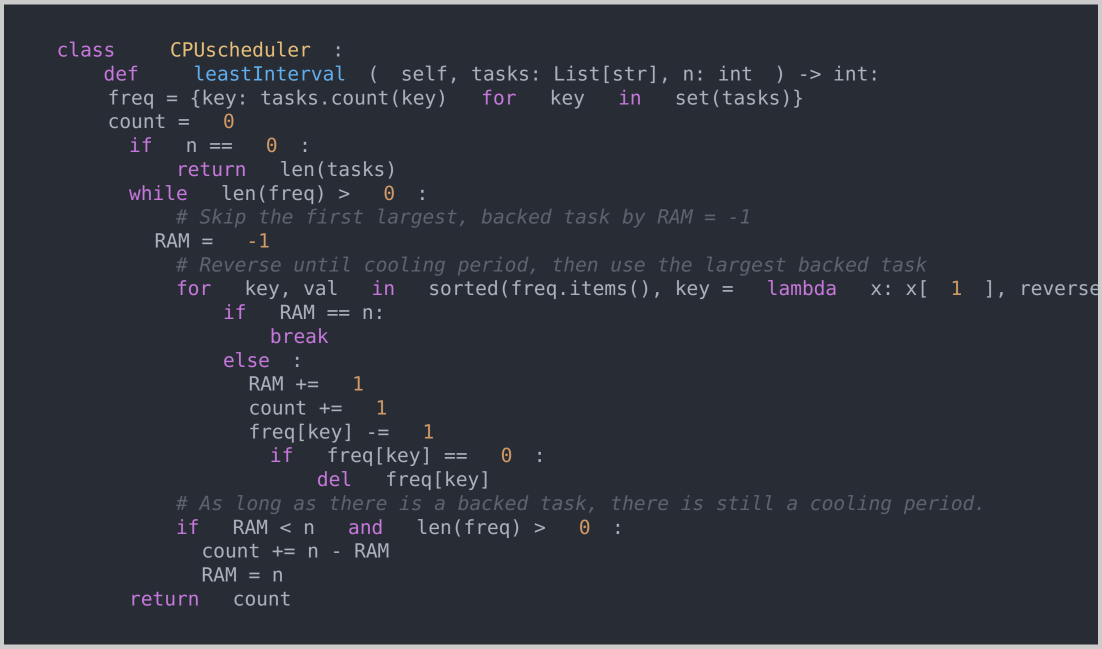
Yes it should be obvious that the while loop is the better solution here for several reasons. First, the while loop does not need to use a break statement, it is inherent in the test condition. Second, there is no need to keep track of the iteration that the while loop is on because it will run forever until the test condition is false. Lastly, based on the previous reasons, the while loop is more efficient because it requires less code. Does less code mean more efficiency? No, it does not, just because code is more concise and takes up less space in the source code does not guarantee it is faster. Even though the while loop is probably more efficient, we need to use standard methods to compare the runtime complexity. Since this is an informal book on Python objects, we should take an informal approach to runtime complexity. Disregard everything but the main Big-O notation. What is meant by this is that we typically do not care if there is a linear addition to our formula, just capture the following aspects of runtime complexity: constant time O(1), logarithmic time O(log n), linear time O(n), quasilinear time O(n log n), quadratic time O(n^2), exponential time O(2^n), and factorial time O(n!) (Prado, 2019). Understanding and comparing these runtime complexities is crucial for delivering actionable and useful code within your enterprise. Before I give you examples of each, heed to the warning that IBM gave about quantum computing. They used an old wise tale of how the game of chess was invented to explain computational complexity. Here is the main part of that story, "I would like one grain of rice for the first square of the board, two grains for the second, four grains for the third and so on doubled for each of the 64 squares of the game board" said the mathematician (Syed, 1999). Do not be like a king that could not understand computational complexity, instead be like the mighty mathematician and leverage Python effectively through appropriately written programming logic. Below is an example of each of these categories. Do not harp on the fact that we do not cover these extensively here. This is a starting point. There will be a separate book on computational structures in relation to data structures.

```
timeLoop = [  1  ,   1  ,   2  ,   3  ,   5  ,   8  ,   13  ,   21  ,   34  ]

  ## Constant Time - O(1) ##
print(timeLoop[  3  ])

  ## Logarithmic Time - O(log n) ##
    def     binarySearch  (  _list, target  ):
    left, right =   0  , len(_list) -   1

      while   left <= right:
        pivot_idx = (left + right) //   2
        pivot_element = timeLoop[pivot_idx]

          if   target == pivot_element:
              return     True
          else  :
              if   target < pivot_element:
                right = pivot_idx -   1
              else  :
                left = pivot_idx +   1
      return     False

print(  f'The target 5 is in the timeLoop:   {binarySearch(timeLoop,   5  )}  '  )

  ## Linear Time - O(n) ##
    def     linearSearch  (  target  ):
      for   num   in   timeLoop:
          if   num == target:
              return     "Found target"
      return     "Did not find target"

  ## Quasilinear Time - O(n log n) ##
  for   num   in   [  5  ,   8  ,   17  ,   21  ,   35  ]:
    print(  f'Target   {num}   is in the timeLoop:   {binarySearch(timeLoop, num}  '  )

  ## Quadratic Time - O(n ** 2) ##
  for   num   in   [  5  ,   8  ,   17  ,   21  ,   35  ]:
    print(  f'Target   {num{   is     in   the timeLoop: {linearSearch(num)}  '  )

  ## Exponential Time - O(2 ** n) ##
    def     fib  (  num  ):
      if   num <=   1  :
          return   num
      return   fib(num -   1  ) + fib(num -   2  )

  ## Factorial Time - O(n!) ##
  for   factorial   in   range(  1  ,   10  ):
      for   nums   in   range(factorial,   0  ,   -1  ):
        print(num, end =   " "  )
```

Containers

There are four types of containers within Python that are useful for storing objects: lists, tuples, dictionaries, and sets. Each of these has pros and cons to their design, with lists and dictionaries being used most often. We will briefly cover those aspects which are pertinent to demonstrate why you would use a particular container. If you are familiar with containerized software, containers within this section essentially act as standalone environments where objects live. Having compartmentalization and segregation allots designers to maintain separate object types within each standalone environment.

Arrays are implemented as lists within Python and are great for storing a collection of objects. Unlike pure arrays, Python lists can store various data types all in the same container. In so doing, lists provide the perfect avenue for designing stacks of various objects. One of the best examples seen here is in networking where you have multiple queues tagged with certain types of quality of service (QoS). If we have a priority value, whether that is Differentiation Services Field Codepoints (DSCP) or Internet Protocol Precedence (IPP) to determine the criticality, we can run a heap sort to prioritize critical applications without having to check out the payloads involved every time. All these payloads can be separate objects without causing too much concern. Eventually we will cover this book in our Coding for Networking book, but here let’s focus on a single threaded computer. This problem was taken from Leetcode’s 237 Weekly Contest that I participated in (Leetcode, 2021). Below is the code I came up with during that contest. At this point in the book the code is a little advanced for the average reader. The main point to take away from the code snippet is that single-threaded CPU systems need to minimize the wait time by choosing the shortest processing task possible, assuming no prioritization here. Within the problem, we were asked to associate the shortest processing time with the smallest index, to permit the CPU to run without stopping (Leetcode, 2021). Setting up a process queue and some way of categorizing or sorting that queue was necessary to select the smallest index. As stated with DSCP and IPP, here QoS was provided to make the single-threaded CPU function as originally intended. All of this was accomplished using lists.



```
class    CPUscheduler  :
    def    leastInterval  (  self, tasks: List[str], n: int  ) -> int:
    freq = {key: tasks.count(key)   for   key   in   set(tasks)}
    count =   0
      if   n ==   0  :
          return   len(tasks)
      while   len(freq) >   0  :
          # Skip the first largest, backed task by RAM = -1
        RAM =   -1
          # Reverse until cooling period, then use the largest backed task
          for   key, val   in   sorted(freq.items(), key =   lambda   x: x[  1  ], reverse
              if   RAM == n:
                  break
              else  :
                RAM +=   1
                count +=   1
                freq[key] -=   1
                  if   freq[key] ==   0  :
                      del   freq[key]
          # As long as there is a backed task, there is still a cooling period.
          if   RAM < n   and   len(freq) >   0  :
            count += n - RAM
            RAM = n
      return   count
```

Dictionaries are extremely useful and tend to be the container I use first. Their structure is representative of a NoSQL database, structuring the database through key-value pairs. There are numerous useful methods with dictionaries, to include keys, values, items, and more. Using NoSQL with these methods we can compose the phone collection database you see below. I created this when I was studying for NSA’s CISSP certification. What you see is that we can create a phone collection class to run us through the entire lifecycle of mobile collection. Pair this with appropriate data retention policies and put representational state transfer (REST) application programming interfaces (API) into the code, and you could have a legally adhering signals collection database. In our future book, Crypto Bash, we will be focusing on how to turn both Python and Bash scripting into useful tools for the lifecycle of cryptology and signals intelligence. This is expected to be released before RSA 2022.

```
import   time
import   random
  class     PhoneCounter  :
      def     __init__  (  self  ):
       self.phones = {}

      def     hit  (  self, number, timestamp  ):
        if   number   not     in   self.phones.keys():
          self.phones[number] = [timestamp]
        else  :
          self.phones[number].append(timestamp)

      def     getHits  (  self, timestamp, selector = None, duration =   300    ):
        if   len(self.phones.keys()) ==   0  :
            return     0

        if   selector ==   None  :
          hits =   0
            for   key, val   in   self.phones.items():
              ptr = len(val)  -1
                while   ptr >=   0     and   abs(timestamp-val[ptr]) <= duration:
                  hits +=   1
                  ptr -=   1
            return   hits
        elif   selector   in   self.phones.keys():
          hits =   0
            for   val   in   self.phones[selector]:
                if   abs(timestamp-val) <= duration:
                  hits +=   1
            return   hits

      def     getSelector  (  self, number, timestamp = None, duration =   300    ):
       number = str(number)
        if   number   in   self.phones.keys():
            try  :
                if   abs(timestamp-self.phones[number][  -1  ]) <= duration:
                    return   [x   for   x   in   self.phones[number]   if   abs(timestamp-x)
                else  :
                    return   self.phones[number]
            except  :
                return     "Error"
        return     "No selector here!"

      def     database  (  self, number = None  ):
        if   number ==   None  :
            return   self.phones.items()
        elif   number   in   self.phones.keys():
            return   self.phones[number]
        else  :
            return     "Number not found!"
```

Sets are extremely useful when performing mathematical set operations with dictionary keys, list items, and found targets within searches. In the case below, we are interested in identifying all possibilities in each scenario. There is no limit to what this could be about. For instance, we could be interested in a sports team, related stock trends, political news, and much more. With set theory, useful operations could be union, difference, symmetric difference, etc. Below are all unique cases to get our super set of the situation. That means that for all related stock trends we have knowledge of the entire environment, which will be important in our future book, Bayesian Statistics.

```
def pairwiseDisjoint ( listOfSets ):
superSet = set()

  for _set in listOfSets:
    superSet = superSet.union(_set)

  return superSet
```

Record keeping requires accurate and precise books, and this is where tuples come in. You can use these to ensure that the cybersecurity principle of integrity is followed. Leveraging this will give you an immutable container that cannot be changed. These are rarely used and will not be covered here. I implore the readers to go get more resources on this if they need more information.

Stack Tracking

No one on Earth, no matter how experienced or proficient in the language, writes perfect code every time on the first try. If you don’t believe me then why did Tony Stark have multiple artificial intelligence programs? Even Ironman required multiple software iterations to work out the bugs and improve on the code. The lesson that can be learned from this fictional character is that you should expect your code to fail. As you write more advanced programs, if you do not account for these error conditions it will be extremely hard to debug. For example, when I was writing the case study for this book I came across a few physical and logic bugs across the components. The physical bugs could easily use try except blocks to catch undesired input or functionality. However, it became very difficult to narrow down and identify the logical bugs across class inheritance. Baking in both physical and logical errors will help make your life easier down the road, trust me!

So, what is a traceback? Think of this as part of Python, that helps deliver you an error of what failed (Geeks for Geeks, 2020). Even better is the fact that this error information comes with the following information: what type of error, addition information about the error, traceback statements for the line and code of interest, and finally, the line at which this occurred (Geeks for Geeks, 2020). Regarding the types of errors that can be generated, Python allocates 64 different errors in a hierarchical fashion (Quackit, 2021). As you can see there are tons of examples we could go through, but let us just show one error, then write code to catch it. In the code directly below, we see that the code, on line one from the zeroError.py script, generated a ZeroDivisionError. This error is described as dividing by zero. In the second piece of code, we go ahead and run a try except to catch our ZeroDivisionError!

```
print(  10   /   0  )

  ## Result ##
Traceback (most recent call last):
    File   "zeroError.py"  , line   1  ,   in
        print(  10   /   0  )
ZeroDivisionError: division by zero

  ## Caught Exception ##
  try  :
    print(  10   /   0  )
  except   ZeroDivisionError:
    print(  "We can't divide by zero!"  )
```

Cryptology - Case Studies

Overview

Cryptology is broken down into two main categories: cryptography and cryptanalysis. Both are equally important but have different objectives in mind. First, let us start with the former. Cryptography protects our information through confidentiality and integrity. So far, or at least as far as the author is aware, there has not been cryptography developed to protect the availability of systems. Through subcategories like symmetric, asymmetric, and hashing algorithms, we help protect our customer data. On the opposite spectrum, cryptanalysis is an adversarial approach to compromising the confidentiality and integrity of control measures in place. There are different ways to compromise these blue side mitigations, to include side channel attacks, mathematical attacks, social engineering, etc. The point of this section is not to be a comprehensive explanation on the history, techniques, and attack vectors within cryptology. Instead, we will cover a few statistical techniques that are helpful in cryptanalysis, one cipher, and then we will break that one cipher. In the field of cryptology, this is a simple example that helps rely the message on how cryptanalysis is difficult, and how much computational complexity it takes to break a cryptosystem.

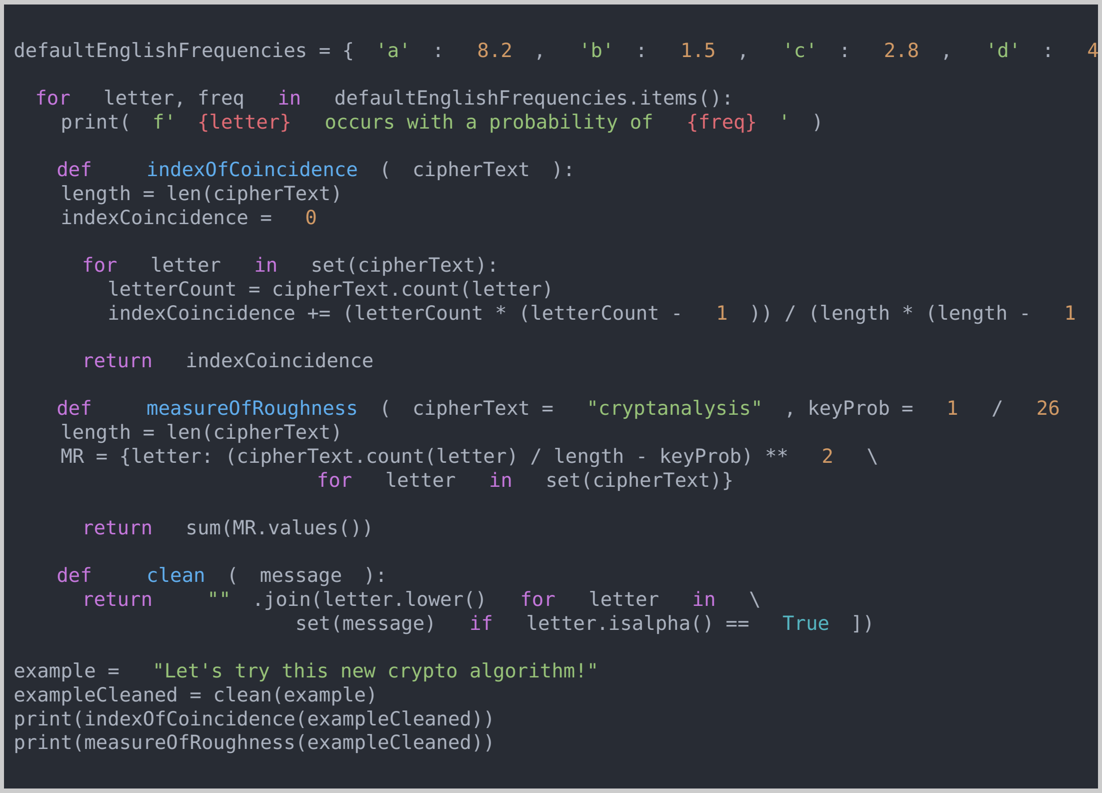
Cryptography and cryptanalysis are a game of cat and mouse, sometimes the attackers have the edge while other times the network defenders have the advantage. One of the weaknesses of both fields entails understanding the time at which an attacker can compromise the system. Even today, an algorithm like the American Encryption Standard (AES) is difficult to account for since network defenders do not know all the attacks against the system. To understand such information would require possession of complete information about the computational complexity associated with the system. In the theoretical and academia world, attackers have unlimited resources, computing power, and knowledge of the system. Using the information at hand, they then derive how long it should theoretically take to break the system. Unfortunately, this does not help for law enforcement and military operations that only need information secured for a certain shelf life. National Institute of Standards and Technology (NIST) holds a more pragmatic view of cryptology for shelf-life operations. They started an ongoing competition to find our next AES, Rivest-Shamir-Adleman (RSA), and Secure Hash Algorithm (SHA) algorithms of the future. Within the competition rules, NIST informally states that whoever owns a quantum computer can break AES 256, SHA3-256, and indirectly, RSA (NIST, 2016). To design such algorithms that defeat these standard algorithms, with known computational complexity times associated with each, NIST handicapped the attackers by limiting the chosen plaintext attacks to two to the power of 64 (NIST, 2016). To limit the blue side development, NIST forced competitors to limit their architectures to a maximum of two to the power of 96, achieved not even three years later by a company called IonQ, who developed a 160-qubit processor (NIST, 2016). Cybersploit LLC will dive deeper in quantum cryptanalysis in a future series, as this was just a teaser to show you that real-world research and development is not driven by theoretical limits, but instead by pragmatic environmental influences.

Moving towards the actual cryptology problems, implementations, and attacks, we need to understand that you cannot do cryptology without some background in mathematics. Just imagine watching the matrix, math and cryptology are a pair and Mr. Anderson will tell you its inevitable. Unless you are Neo and can see the code being deciphered in real time, then you need some informal math to get the job done. The three techniques of cryptanalysis that we will use to help in our analysis of the Affine cipher will be index of coincidence and measure of roughness, chi-squared, and n-gram probability. Each of these techniques introduces us to something new. Index of coincidence (I.C.) and measure of roughness (M.R.) helps to identify the frequency, chi-squared covers frequency and types, and finally, n-gram probability helps get both in addition to order (Barter, 2018). Frequency helps identify the individual letter amounts, while type analyzes the proportions associated with the text of interest (Barter, 2018). Finally, using n-gram probability helped crack Enigma and bring us into the modern age of cryptanalysis by using the order, or the letters that should likely proceed and follow it (Barter, 2018). After learning these statistical methods, we can introduce the Affine cipher and immediately understand how to crack the implementation.

Before diving into statistical methodology, a little history on the subject might be well received. It is said that Turing helped change the face of modern cryptology by championing statistical analysis in both cryptanalysis and cryptography. Turing, the teacher, traveled back and forth between the US and Britain to help us in statistical measures applied to cryptology. From the author's understanding, sources online say his manual is still considered to be classified property of the British government. No one cared about that little bit of trivia, instead the reason why Turing is so well known, is that he was considered the father of the computer. Remember that statement earlier that cryptography and cryptanalysis is a cat and mouse game? Well Turing had to develop a computer to break the TUNNY, or Lorenz, computer that was providing encryption. So, both countries developed computers in WWII so that they could do opposite things associated with confidentiality, one to make and one to break. The FISH, TUNNY/Lorenz, and the Schlusselgerat (Hagelin encryption) helped create COLOSSUS, or the allies' first computer (NSA, 2021). Some may say that they were their Bill Gates of their time, so who are our next generation's Gates?

Index of Coincidence and Measure of Roughness

I.C. and M.R. are related techniques to derive statistical frequencies associated with the text of interest. The main difference lies in the comparison of frequency distribution, with the former using a uniform distribution and the latter using a flat distribution (Practical Cryptography, 2012). Using this information, code breakers can gather insight into how the expected frequency differs from the distribution measured against. Probability theory and statistical identification is an important part in Cybersploit's master plan. In a future edition, Cybersploit will cover the ins and outs associated with each significant statistical methodology. For now, just gather that M.R. is the differences between the expected probability of each letter to the probability from that uniform, with which you perform the sum of squares associated with each section (Practical Cryptography, 2012). The chi-squared technique we know is directly related to the M.R. if the uniform distribution is the English language, or 26 characters (Practical Cryptography, 2012). The code below helps demonstrate how you would implement this in Python. I got the letter frequencies off Wikipedia.



```
defaultEnglishFrequencies = {  'a'  :   8.2  ,   'b'  :   1.5  ,   'c'  :   2.8  ,   'd'  :   4

  for   letter, freq   in   defaultEnglishFrequencies.items():
    print(  f'  {letter}   occurs with a probability of   {freq}  '  )

    def     indexOfCoincidence  (  cipherText  ):
    length = len(cipherText)
    indexCoincidence =   0

      for   letter   in   set(cipherText):
        letterCount = cipherText.count(letter)
        indexCoincidence += (letterCount * (letterCount -   1  )) / (length * (length -   1

      return   indexCoincidence

    def     measureOfRoughness  (  cipherText =   "cryptanalysis"  , keyProb =   1   /   26
    length = len(cipherText)
    MR = {letter: (cipherText.count(letter) / length - keyProb) **   2   \
                          for   letter   in   set(cipherText)}

      return   sum(MR.values())

    def     clean  (  message  ):
      return     ""  .join(letter.lower()   for   letter   in   \
                        set(message)   if   letter.isalpha() ==   True  ])

example =   "Let's try this new crypto algorithm!"
exampleCleaned = clean(example)
print(indexOfCoincidence(exampleCleaned))
print(measureOfRoughness(exampleCleaned))
```

In the code above we take the mathematical principles associated with I.C. and M.R. and use it against an actual piece of text. If it is not apparent, the clean function is used to only take characters from the English alphabet, convert them to lowercase and return a long string without punctuation. Doing so allows use to call both functions, the indexOfCoincidence and the measureOfRoughness, without concern and as explain earlier.

Chi-Squared

Looking beyond just frequency, we use the chi-squared statistical measure to include both frequency and type in our analysis. With the help of the chi-squared statistic, we can compare the degree to which two categorical probability distributions are similar, with the lower result being better (Probably Cryptography, 2021). Once again, we got the letter frequencies from Wikipedia. For this section, we will not go through the defaultEnglishFrequencies variable again. If you need to review that variable, go back to the last section. Later in the n-grams probability section, we will derive these probabilities for bi, tri, and quad grams from any English text we desire. If you are deciphering ciphertexts from another language, you must use n-gram frequencies from a text written in the language of interest.

```
   def     chiSquared  (  decryptedCipherText =   "cryptanalysis"  , \
            alphabet = sorted(  [chr(    97  +i  ) for i in range(    26    )]  ), \
            expectedFrequencies = defaultEnglishFrequencies  ):

    expectedFreqMapping = {alphabet[idx]: (expectedFrequencies[alphabet[idx]] /  100  ) \
* len(decryptedCipherText)   for   idx   in   range(len(alphabet))}

    cipherTextFreqMapping = {letter: decryptedCipherText.count(letter)   for   \
letter   in   sorted(set(decryptedCipherText))}
    cipherChiSquared = {}

      for   letter   in   sorted(set(decryptedCipherText)):
        cipherChiSquared[letter] = (cipherTextFreqMapping[letter] - \
expectedFreqMapping[letter])**  2   / expectedFreqMapping[letter]

      return   sum(cipherChiSquared.values())

  from   random   import   randint

Texts = [  ""  .join([chr(  97  +randint(  0  ,  25  ))   for   i   in   range(  1000  )])   fo

legitimateText =   """Hello there. We are starting this section on statistics
to make everyone aware of the struggles and difficulties of cryptanalysis, as
well as the tools. Enjoy this lab!"""

Texts.append(  ""  .join([letter.lower()   for   letter   in   legitimateText   if   letter.is
dictOfChiSquared = {text: chiSquared(text)   for   text   in   Texts}

print(  "Decrypted Ciphertext"   +   "."  *  35   +   "Chi-squared"  )
print()
  for   text, chi   in   dictOfChiSquared.items():
    print(text[:  50  ] +   " ... "   + str(chi))
```

Grams

Using n-gram probabilities to crack ciphers seemed like magic when it was first introduced by Turing in WWII, so for our examples moving forward we will use the Harry Potter text as the source for tetragraphs (quadgrams). As we looked at early, n-gram probabilities opened a range of possibilities with cryptanalysis by considering the statistical characteristic of each deviation from the expected frequency, both frequency and type, and tracking the order in which these normally occur in the language of choice. For example, bigraphs use two characters that are likely to occur, as defined by the three dimensions of frequency, type, and order. Trigraphs added an additional character, with tetragraphs adding two more characters on top of bigraphs. Following the advice of Practical Cryptography, we can take our text of interest, making sure it is in the language we are trying to crack, and generate all tetragraphs associated with it (Practical Cryptography, 2012). Finally, tetragraphs are given a fitness value by finding the log probability of it occurring. As stated earlier, this may seem tricky, so we hope to break down cryptanalysis and quantum cryptanalysis in a future endeavor.

```
  # Calculate the quadgrams of Harry Potter and the Sorcerers Stone
HP = open("Harry-potter-sorcerers-stone.txt", 'r', encoding='utf-8')

    def clean(word):
      return "".join([letter.lower() for letter in \
                    word if letter.isalpha() == True])

HP = "".join([clean(word) for word in HP.read().split()])

  from math import log
    def ngrams(size = 4, text = ""):
    count = 0
    gramFreq = {}

      for num in range(len(text)-size+1):
        gram = text[num:num+size]
        count += 1
          if gram not in gramFreq.keys():
            gramFreq[gram] = 1
          else:
            gramFreq[gram] += 1

      for key, val in gramFreq.items():
        gramFreq[key] = log(val/count)

      return gramFreq

harryPotterQuadgrams = ngrams(4, HP)
```

Review the code and realize that we can take any text of interest, in any language, and create a tetragraph probability code book for that language. Wow, this book may even be made into a code book one day! After generating the Harry Potter tetragraph probability code book with the ngrams function, we use that code book to determine the fitness value associated with an unknown ciphertext.

Affine Cipher

```
################################################################
# Math from
# Practical Cryptography
# 2012
# http://practicalcryptography.com/ciphers/classical-era/affine/
################################################################

 class    affineCipher  :
     def     __init__  (  self, a, b, m  ):
        if   a <   1     or   a > m   or   b <   1     or   b > m:
            raise  (  "Try again with different values!"  )

      self.a = a
      self.b = b
      self.m = m

     def     modInverse  (  self, a, prime  ):
      a = a % prime

        for   x   in   range(  1  , prime):
            if   ((a * x) % prime ==   1  ):
                return   x

        return     -1

     def     encrypt  (  self, plainText  ):
      cipherText =   ""

        for   letter   in   plainText:
            if   letter.isalpha() ==   True  :
              cipherText += chr(((self.a * (ord(letter.lower()) \
                              -   97  ) + self.b) % self.m) +   97  )
            else  :
              cipherText += letter

        return   cipherText

     def     decrypt  (  self, cipherText  ):
      inverse = self.modInverse(self.a, self.m)
      plainText=   ""

        for   letter   in   cipherText:
            if   letter.isalpha() ==   True  :
              plainText += chr(((inverse * (ord(letter) \
                          -   97   - self.b)) % self.m) +   97  )

        return   plainText
```

Affine Cryptanalysis with Examples

Using what we now know about statistical cryptanalysis techniques, we can in fact break the cipher above to conclude our case study. Since we know that the modulus is going to be 26 for the English langue, and with the use of tetragraph probability code book, we can map the fitness values of all a and b values within the key space of the cipher. Once mapping all the plaintext fitness values of ciphertext within the given key space, we can select the likely text of interest. For our ciphertext of interest it will be the lowest fitness value. However, it is pertinent to warn the reading that the lowest fitness value will not always be the cleartext pair. If we do not use another methodology in conjunction with this technique, then we will have to manually scan the results to verify that our algorithm works as perceived. In the real world of cryptanalysis, we would probably use multiple techniques to generate multi-dimensional validity, or a truthness value, for each suspected plaintext-ciphertext pairing. Also, note that we will be using all the code derived in the tetragraph example. Using that Harry Potter probability code book, we can generate a function that will statistically crack our ciphertext. Lastly, note that we must go through the key space with a quadratic computational complexity. You can consider a and b to both be n here since they have the same inclusive bounded range: one to 26. If we were to have a varying modulus, it would increase the computational complexity significantly.

```
    def     breakAffineCipher  (  cipherText  ):
    allPlainTexts = []

      for   x   in   range(  1  ,   27  ):

          for   y   in   range(  1  ,   27  ):
            plainText =   ""

              for   letter   in   cipherText:
                  if   letter.isalpha() ==   True  :
                    plainText += chr(((x * (ord(letter) \
                                      -97  -y)) %   26  ) +   97  )
                  else  :
                    plainText += letter

              if   len(set(plainText)) >   3  :
                allPlainTexts.append((plainText, \
                                      calculateTextFitness(plainText)))

      return   allPlainTexts

print(sorted(breakAffineCipher(affineCipherText), \
             key =   lambda   x: x[  1  ], reverse =   True  )[  0  ])
```

Object Oriented Programming (OOP)

OOA, OOD, OOP

Learning OOP principles is a whole lot of fun! It’s one of those topics that once you learn it, it is very difficult to unlearn it. The best way to review this section is to take the concepts you are about to learn and apply them in your daily routines. After applying them in your routines, the skills you develop should transition to your job and make you a better programmer. Do not expect to be an expert in OOP by the end of this book. Becoming an expert in OOP requires years of practice and hands-on skills, but you can start OOP anytime!

There are three concepts in the book Python Object Oriented Programming that break down OOP for beginners: object-oriented analysis (OOA), object-oriented design (OOD), and object-oriented programming (OOP) (Lott & Philips, 2021). All these terms relate and complement each other but are accomplished in sequential order. As we go through the rest of this book keep this process in mind: analyze the situation to see what items useful objects could be and how they would interact, formulate the design on paper or with UML, and then implement your work with actual code (Lott & Philips, 2021). It’s as easy as one, two, three. Remember it will take some practice to truly get the hang of it, and since you must do the process to learn, why not start with a few examples.

Encapsulation, Interface, Abstraction, Composition

Encapsulation is the process of enclosing some payload or data in a container. In networking, we consider encapsulation to be the payloads, headers, and trailers of each layer of the Open System Interconnection (OSI) model. De-encapsulation would be extracting those payloads, headers, and trailers from each layer as deemed appropriate. For programming, encapsulation means that the information is hidden from us. The process of hiding information on purpose is abstraction. Most of the information and details of programming do not or should not be shown, so we abstract and encapsulate it. To provide useful programs, we need to access that abstracted data. In other words, we provide an interface to give us access to encapsulated data to de-encapsulate it for us to use it. Creating great interfaces is an art and is usually categorized under User Interface (UI) Design. When we cover HTTP REST methods at the end of this book, you will start becoming familiar with how to design interfaces with industry standards in mind.

Composition is the ability to combine objects or types to make other types (Rodriguez, 2021). We could look to astronomy for examples of this. When have the whole universe that’s composed of the visible universe, that’s composed of galaxy, with ours being the Milky Way galaxy. The Milky Way galaxy contains 100-400 billion stars, which may or may not have life (Wikipedia, 2021). This can be further broken down forever, but we do not need to do that when we say our universe. Using composition, when we say our universe we mean all the things that contribute to the definition of the universe.

Example

Instructors tend to use how cars work as a student's first exposure to OOP principles. What's even more interesting, is that the De Bono Thinking Course took the same approach with the USG when trying to teach people how to think about thinking. The reason why using cars are easy examples is because almost everyone has a license and drives to work each day. Since they both use this as an introductory example, and the fact that most individuals should have familiarity with the concept, explaining the process of driving will also be our first example. Moving to the example, when you take your driver's license test in the state of California you are required to check 17 items before starting your test, a few of these include: checking that both the turn signals and brake lights work, the horn is loud enough, and that all tires have enough tread (CA DMV, 2021). Within these pre-checklist items, the driver does not need to know how the turn signals and brake lights work, just that they turn on. These objects are abstracted so that the driver can focus on the road instead of worrying about how the electrical is working in your car. Now the driving part of the test has begun, the driver does not need to know how an engine works to pass the driving test. Instead, the engine components are encapsulated so that when the gas pedal is pushed down, the engine does its magic and the car springs forward. The pedal is the interface to the driver, allowing drivers to push harder on the gas pedal for more speed. Similar explanations can be delivered for the brake, the radio, etc. Everything that has been covered in this example is not unique to one type of car, but instead is common across cars. So, if we wanted to encapsulate the object of car further, we could do more composition, or has a relationship. For instance, all cars have taillights, headlights, turn signals. All of these have an electrical system that manages their input and output. Car electrical systems have energy derived from a fuel source. As you can see the list goes on.

Implementation

We have covered objects, but what are classes? Think of classes as the actual code associated with objects, with every class having a different type. In Python, we use the key word class to define the code block associated with a class. Some may ask what is bare bones implementation of a class, what is needed and how do we create examples? There is a lot to that question. Here is code demonstrating the what the minimal requirements are to run a class in Python.

```
    class     Terminator  :
    print(  "I will be back to learn!"  )

Terminator()
```

As you may have gleaned, to create a class and run an instance only requires three lines. Yes, only three lines! You have just created your first class. The Terminator told us that he will be back to learn, so let us create more “stuff” inside to make this class useful. So, what are class variables? Now these are variables that are local to the class, which can be accessed at runtime. However, without some sort of method, which is what a function is called within a class, we can’t do much with a class variable. So, how do we add methods? To define a method, which is a class function, you must use the keyword def, the name of the method, then include the self key word as the first parameter. Before continuing there are a few rules, the self key word is always the first parameter to a method but does not have to be called self. The standard naming convention recommends that you use self to refer to things within the class. Also, if you define or reference a class variable within a method you must use that key word present within the first parameter, usually self.

```
    class     Terminator  :
    target =   "John Connor"

        def     mission  (  self  ):
        print(  f'I will be back to learn more about   {self.target}  '  )

humanoid = Terminator()
humanoid.mission()
```

Above you should see that we extended our Terminator class with a class variable, called target, and a class method. After defining the class, we instantiated the class. This means that we have a running copy with all the details inside the variable humanoid, allowing us to call the mission method. The result of this program will print the statement with the class variable target injected within the middle.

What if we wanted to initialize the target variable when the class is first created? You can do that through a Python's class's __init__ method. Changing the class's internals around is rather easy, so let us have a look at the code

```
    class    Terminator  :
        def    __init__  (  self, name  ):
        self.target = name

        def    mission  (  self  ):
        print(  "I will be back to learn more about "   + self.target +   "!"  )

humanoid = Terminator(  "John Connor"  )
humanoid.mission()
```

In the code above, we added the __init__ method to our Terminator class so that we could initialize the target. To do this, we needed to take in a name when instantiated. In the future, we could also include input validation to ensure that the parameter is a string. Also, know that there is a lot more to classes, but this should suffice until we cover inheritance design patterns.

Inheritance

Teaching students about inheritance for the first time can be exhilarating. Learning this crucial OOP principle changes a student’s perspective, opening the vast world of modern programming practice and design. Inheritance is one of those concepts that once you learn, you cannot unlearn. You can remind the students that every time you program from that moment forward you will be wondering whether your design or system would be better split into object hierarchies. Within the following pages, we will cover the necessary types of inheritance that are pertinent to Cybersploit’s future cybersecurity curriculum. We are not computer science majors looking to write a thesis on the notions and aspects surrounding OOP; instead, we want to be able to talk and walk intelligently around Development and Operations (DevOps) in as little as nine weeks. In doing so, we will spend significant time understanding and applying OOP. If you continue down our future path, from Software Engineering to Cyber Warfare, you will experience more exposure to inheritance design patterns and implementations. However, the knowledge you learn in the following section will be enough to understand requirements, decompose key elements, and finally, structure and compose a solution.

The types of inheritance we will survey now include single, multiple, multilevel, hierarchical, and hybrid (Python Geeks, 2021). Each one has it benefits, advantages, and cons. Understanding when to use one design pattern over another is an important skill you acquire as time processes. Remind the students that OOP may seem complex, abstruse, and nonsensical at times, but that there are three general reasons why we focus on OOP: abstraction, composition, code reusability, large-team collaboration, and design pattern standardization. Each of these benefits can be a subject in and of themselves, so let us break down the generalities associated with these benefits. When managing a large corporate team, whether that be as a project manager or as a DevOps manager for a Security Operations Center (SOC), we need to all follow our sets of orders to fulfill the unified mission or objective. There are times this may mean you are not actually interacting with other teams because of geographical proximity, clearance levels, or mission task. Making sure that our components fit into the overall solution is key to the whole team concept. If we design something outside of the set design pattern standardized, then we may run the risk of compromising mission assurance with another team's product, or somewhere in the future with our own team. Those key design patterns on what type of inheritance to use, what classes to inherit from, what methods or functionality should be included, what libraries are allowed to be called, or how the interface should look to key stakeholders should be set by your organizational and mission team policies. Following your organizational guidelines, you are then able to decompose customer requirements and perform abstraction and composition locally by creating and fusing together what is necessary. This is where your OOP thinking cap comes into play, there will be times where you may need to inherit from a single class, and other times you will inherit from multiple classes. In the process, make sure you are only combining, or composing, the necessary elements from other classes to ensure minimal dependency. Also, provide a friendly interface, whether that is an Application Programming Interface (API) or documenting properly, so that there is little confusion about what was written and how it works. Finally, following these guidelines will was written and how it works. Finally, following these guidelines will help ensure that you have code reusability. Yes your job is to accomplish your mission, but wouldn’t it be even better if you could leave a repository of well-structured and documented code so that your tasks in the future are even easier. Code reusability is why OOP blew the C programming out of the water because we were able to develop classes, modules, and packages that could facilitate faster time to market, or time to mission accomplished!

There is a warning that you need to adhere to before continuing forward: OOP should not be used for everything! I repeat, do not use OOP for every single stakeholder requirement. OOP is a very useful tool, but it is common for novice programmers to use OOP for all their solutions. There are times when OOP is overkill, would slow down the system, does more harm than good, or cannot be used. A very simple example of OOP overkill is when you are designing a single function to do one specific thing. If you then create a whole class out of a function that multiples every third number by seven, then you create more work for yourself, you generate more overhead, and the interface will be more difficult than using a singular function. To that same regard, generating overhead could slow down the system. Singular functions that do not perform a lot of action, or are extremely transient, would slow the operations down if implemented in OOP by allocating extra space and time to make that class. Lastly, it might be clear by now that Python points directly to the C programming language in the background. C is estimated to be greater than 45,000 times faster than Python but is unable to use OOP design patterns. When using C, you are limited to non-OOP solutions.

Our first example will be over single inheritance. As with this example, all the following examples will be based on the popular movie called John Wick. Here we may be inheriting from an abstract class or a regular parent class. An abstract class, or abstract methods, provide a structure that should be overridden, but do not permit the actual implementation of such. Here is the implementation:

```
    class    InformationSecurity  :
        def    __init__  (  self, name, email, phone, company  ):
         self.name = name
         self.email = email
         self.phone = phone
         self.company = company

        def    insecurePII_Leak  (  self  ):
         print(  "Here is "   + self.name +   "'s information: "   + self.email \
                        +   " "   + self.phone +   " "   + self.company)

        def    dataBreach  (  self  ):
         self.insecurePII_Leak()

    class    employee  (  InformationSecurity  ):
        def    __init__  (  self, name, email, phone, company, loyalty  ):
         super().__init__(name, email, phone, company)
         self.loyal = loyalty

        def    loyalty  (  self  ):
         print(  "This employee is loyal to "   + self.loyal)

wick = employee(  "wick"  ,   "johnwick@continental.com"  ,   "1(800)2232774"  , \
                          "self-employed"  ,   "Daisy"  )

wick.dataBreach()
wick.loyalty()
```

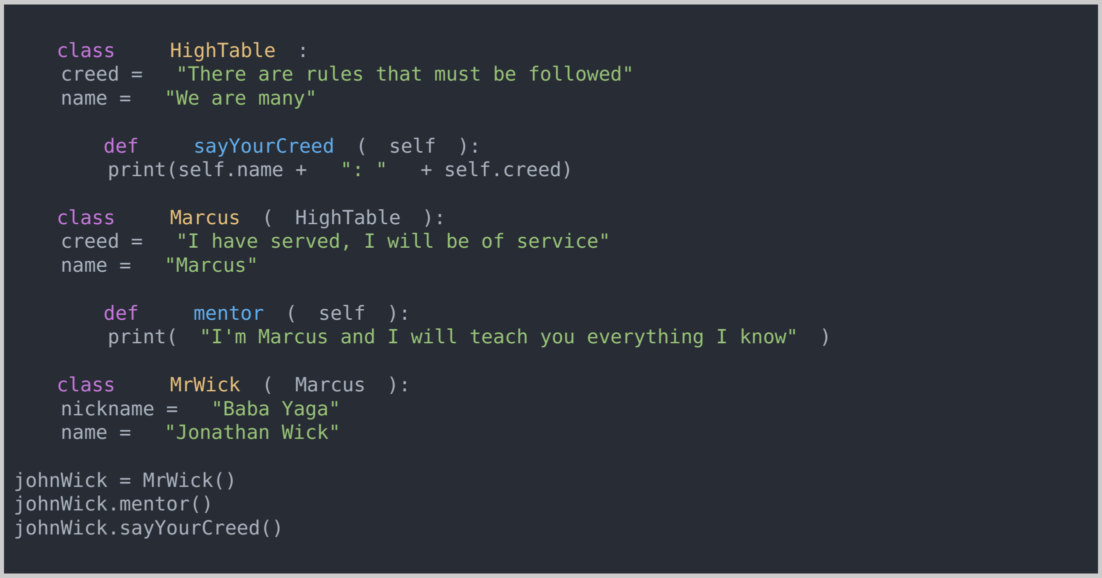
Seen in the single inheritance implementation above, wick is instantiated, or initialized, through the employee class. He is self-employed, usually sub-contracted through the Continental, and so inherits from their information security class. Within this information security class, the Continental saves their name, email, phone number, and company information. In this example, the Continental maintains their Personally Identifiable Information (PII) and would be required to tell Mr. Wick if there was a data breach. After coming back from a normal life, Mr. Wick's information is compromised in the second movie by Italian crime lord Santino D'Antonio after betrayal. In our second course, the Digital Analyst, you will be able to emulate this data breach with Metasploit. For now, just realize that wick was instantiated from employee, which inherited from the InformationSecurity class. The parent class held all the information stored from the child class, and so was able to leak the PII associated with Mr. Wick. The child class, employee, was abstracted so that the end user's interface did not have to deal with the implementation details associated with most of the parameters.

For those of you that are not John Wick fans, it appears that Marvel and DC Comics put aside their differences and united this Thanksgiving. Some in the community have called this the multiverse of multiverses! Without any more absurdity, mention to the students that there is a possibility of inheriting from more than one class. Here is the implementation for this multiverse multiple inheritance example:

```
    class    MarvelUniverse  :
    MCU_Heroes = [  'Thor'  ,   'Iron man'  ,   'Hulk'  ,   'Black Widow'  ,   'Hawkeye'  ,

    class    DC_Universe  :
    DC_Heroes = [  'Batman'  ,   'Superman'  ,   'Wonder Woman'  ,   'Aquaman'  ]

    class    Multiverse  (  MarvelUniverse, DC_Universe  ):
        def    thanksgiving  (  self  ):
          return    "It’s not a Thanksgiving without "   +   " "  .join(self.MCU_Heroes) \
                        +   " "   +   " "  .join(self.DC_Heroes)

party = Multiverse()
print(party.thanksgiving())
```

If Wonder Woman dancing with Captain America around a turkey is not patriotic enough for you then I don’t know what is! Using multiple inheritance, we were able to create a multiverse of multiverses through the Multiverse class. This class took in the left parent of MarvelUniverse, and the right parent of DC_Universe. Now it is not important to know that behind-the-scenes Python is formalizing MRO. You will learn about this in a later section since both the parent classes have disjoint variables that do not cause confliction. Leveraging this bare-bones and silly example, demonstrates that the thanksgiving method within the multiverse class has access to both parents’ variables. In a later section you will learn the differences between private, protected, and public methods, as well as the way to implement private methods in Python. Strangely, in Python there is a philosophy that we are all adults and know how to use things. However, just know that in this Python example we are not limited to private classes. To sum up the scenario, our party variable has instantiated the multiverse class, which has two parent classes called MarvelUniverse and DC_Universe. We then called on the thanksgiving method in our child class to pull variables available to us in our parent classes.

Imagining Wonder Woman swirling Captain American around the dance floor with her lasso maxed out my capacity for Marvel and DC for a while. So let us transition back to John Wick for our multilevel inheritance example. Within the first movie, we got acquainted to Mr. Wick’s mentor: Marcus. Although both are independent contractors, both must adhere and follow the rules set forth by the high table. Using this knowledge, we could in fact set up a grandfather, father, child class structure. Others would call this a potential abstract class, inherited by the Marcus class, inherited by the MrWick class. Any way you want to label it is fine for these first nine weeks, but as time progresses, you will need to formalize the way in which you discuss OOP designs. In our Software Engineering course, we strive to get you familiar with OOP. Later in subsequent courses, we aim at honing your vernacular for professional acceptance. Moving on, let us now go through this implementation:



```
    class    HighTable  :
    creed =   "There are rules that must be followed"
    name =   "We are many"

        def    sayYourCreed  (  self  ):
        print(self.name +   ": "   + self.creed)

    class    Marcus  (  HighTable  ):
    creed =   "I have served, I will be of service"
    name =   "Marcus"

        def    mentor  (  self  ):
        print(  "I'm Marcus and I will teach you everything I know"  )

    class    MrWick  (  Marcus  ):
    nickname =   "Baba Yaga"
    name =   "Jonathan Wick"

johnWick = MrWick()
johnWick.mentor()
johnWick.sayYourCreed()
```

Through this multilevel inheritance example, we can see three classes: the HighTable, Marcus, and MrWick. It is pertinent to note that the high table class could be abstracted here but was not as to not cause confusion. Later we will cover abstracted classes. Observing that each of the child classes takes in their parent class within the first line of that class is important: Marcus takes in the HighTable class, and MrWick uses the Marcus class. There are extra variables in here for fun, but the ones used include name and creed. The methods used are mentor and say your creed. Going through the source code we see that johnWick was instantiated from the Mr. Wick class. We then use the variable to call Marcus's method, or the mentor function, to print Marcus' statement: "I'm Marcus and I will teach you everything I know". Now the tricky part is going to be the next call to your variable johnWick. Understanding this part of OOP will be difficult, but it is the cornerstone to making more advanced design patterns. When we call johnWick's method sayYourCreed, we are using Marcus' creed class variable to call the HighTable's sayYourCreed method. Wow, that was a lot! We just used a parent's class variable within our grandparent's method. All of this demonstrates abstraction, by which we separated the implementation details of the sayYourCreed method and the creed class variable. By using these two concepts, we eliminated several details associated with the implementation so that the end-user could use their name class variable only. There was no need for the MrWick class to know anything else besides his name, everything else was taken care of through multilevel inheritance. That is the true power of using OOP that was lost on most first, second, and third generation programming languages. Now you have the knowledge to powerfully program reusability throughout your projects moving forward.

Before allocating time to go over a large program, let's review hierarchical inheritance first. As the name suggests, this component of OOP permits a design pattern of one parent class with several child classes. The goal here is to provide a common method(s) and variable(s) that would be used by numerous other classes. As mission operators, managers, or leaders we do not want to spend the day continuous checking off the same task or requirement. In an ideal world you would get a stakeholder validated requirement and find the right operational asset or solution to collect on a target or perform some defensive action. Once that task is completed by customer A, you do not want to go back and allocate those same assets to fill that same requirement for customer B. No that would be absurd, a huge waste of funds, and a waste of your time! So then why would you want to create and bake-in the same functionality across multiple classes? The answer is you don't, and instead want to create one parent class that can service all the generic requirements across all customers, or classes. That leads us to Mr. Wick, Winston, and the High Table. The High Table sets the rules that would be silly to reimplement within each class, so let us make Winston and Mr. Wick inherit those generic rules from the High Table. In this case, Mr. Wick and Winston are not friends, but allies 90 percent of the time. Movie three took a turn on their friendship, that is probably permanently severed. Coming back to the example at hand, let us implement two solutions. The first will be without abstracted methods, the second will include an abstracted method.

Example 1 - Non-Abstracted

```
    class     HighTable  :
    adjudicator =   "We enforce the rules for those that follow the High Table"
    oath =   "We want complete servitude"

        def     pledgeYourService  (  self  ):
        print(self.name +   " is agreeing to the rules set forth by the High Table."  )

    class     Winston  (  HighTable  ):
    name =   "Winston"

    class     MrWick  (  HighTable  ):
    name =   "John Wick"

winston = Winston()
wick = MrWick()

winston.pledgeYourService()
wick.pledgeYourService()
```

Implemented above is an example of two adversaries, once friends, inheriting from the same parent class: the HighTable. Unlike the prior example where Mr. Wick inherited from Marcus, the third movie showed Winston shooting Mr. Wick after a parlay. It doesn’t always end pleasantly for the puppet, but usually only for the puppeteer. Refocusing back on our example, as mentioned previously, we always greatly appreciate reducing our workload. We especially take that stance when we are facing the same tasks and requirements we covered with another customer. Therefore, both MrWick and Winston take in the HighTable to use the HighTable’s method called pledgeYourService. Leveraging this hierarchical inheritance design pattern ensures that code reusability is followed throughout the enterprise. All members of the HighTable, whether they are direct Continental hotel employees or independent contractors, need to understand, inherit from, and follow these rules. Once these classes are instantiated, either MrWick or Winston can use their name and pledge their loyalty to the organization. Next we will finally get to abstracted methods by redesigning the code above. The first of these examples will be incorrect.

Example2 - Incorrect Abstraction

```
  from   abc   import   ABC, abstractmethod

    class     HighTable  (  ABC  ):
    adjudicator =   "We enforce the rules for those that follow the High Table"
    oath =   "We want complete servitude"

      @abstractmethod
        def     pledgeYourService  (  self, name  ):
        print(name +   " is agreeing to the rules set forth by the High Table."  )

    class     Winston  (  HighTable  ):
    name =   "Winston"

        def     pledgeYourService  (  self  ):
        super().pledgeYourService(self.name)

    class     MrWick  (  HighTable  ):
    name =   "John Wick"

        def     pledgeYourService  (  self  ):
        super().pledgeYourService(self.name)

winston = Winston()
wick = MrWick()

winston.pledgeYourService()
wick.pledgeYourService()
```

It is not necessary to rehash the principles associated with this hierarchical design pattern, but instead take the time to add to our understanding of abstracted methods. These are methods that provide guidance for derived classes, pertaining them to override the lame functionality in place of one that is more tailored to the derived class’s purpose. In so doing here, we see that we should be reimplementing the pledgeYourService method within each class. The problem with Python is that following these rules are optional, and in this case, will only be enforced when it is instantiated at the command line. Calling the abstract method from a derived class will side skirt the whole reason of using abstract methods. Don’t do this, it is bad code! Instead, we should be reimplementing these methods within our derived classes but should still follow the same pattern. This way, we do not have to continuously tell customers what the method to reimplement should look like, instead observing the abstract method should give them indication on the answer. Here is the correct solution.

Example 3 - Correct Method Abstraction

```python
  from   abc   import   ABC, abstractmethod

    class     HighTable  (  ABC  ):
    serviceStatement =   "is agreeing to the rules set forth by the High Table."

      @abstractmethod
        def     pledgeYourService  (  self, name  ):
          pass

    class     Winston  (  HighTable  ):
    name =   "Winston"

        def     pledgeYourService  (  self  ):
        print(self.name +   " "   + self.serviceStatement)

    class     MrWick  (  HighTable  ):
    name =   "John Wick"

        def     pledgeYourService  (  self  ):
        print(self.name +   " "   + self.serviceStatement)

winston = Winston()
wick = MrWick()

winston.pledgeYourService()
wick.pledgeYourService()
```

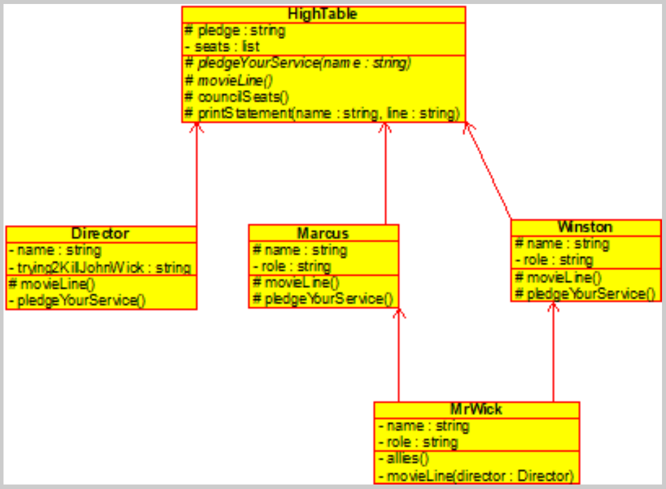
Last of all, let us now focus our attention to combining all these features together through a hybrid inheritance pattern. The first section will contain the imports with the base class. This base class will have two abstracted methods: pledgeYourService and movieLine. Remember an abstracted method means that you need to implement the same function within a derived class. If you choose not to and instead try to implement the abstracted method when the class is instantiated there will be an error. The confusion in Python lies in the fact that you can bypass abstracted methods within the code and call the abstracted method within the source code. Such a design validates the concept of abstraction and should be avoided. Here, being able to do something does not mean you should; try to avoid circumventing abstracted and private methods in the future. Now focusing on the internals of this HighTable class. It should be very familiar as we used it in our previous example. We have extended the HighTable class to include two class variables and four methods, two of which are abstracted: pledgeYourService and movieLine. Each of these should be reimplemented within derived classes since they have the information pertinent for these methods to work. Also, the pledgeYourService class should be reimplemented in the child class as a pseudo form of nonrepudiation or making sure that the child class can’t deny they pledged their service to the High Table. The other two methods, councilSeats and printStatement, do not require reinventing the wheel. The former helps keep track of who we know to be put of the High Table, as of movie three, while the latter implements the principle of OOP code reuse by allowing all child classes to generically print either movie or pledge lines.

```
from   abc   import   ABC, abstractmethod

  class     HighTable  (  ABC  ):
  seats = [  'The Elder'   ,   'The Adjudicator'  ,   'The Administrator'  ,   'The Director'
  pledge =   "is agreeing to the rules set forth by the High Table."

    @abstractmethod
      def     pledgeYourService  (  self, name  ):
        pass

      def     councilSeats  (  self  ):
      print(  "This is what we know so far: "   +   " "  .join(self.seats))

    @abstractmethod
      def     movieLine  (  self  ):
        pass

      def     printStatement  (  self, name, line  ):
        try  :
          print(name +   " "   + line)
        except  :
          print(  "You provided bad input!"  )
```

In the next parts of the code for the hybrid inheritance example, we will be reintroducing revamped classes and adding one additional class. For our revamped classes, Winston, Marcus, and MrWick, we have incorporated the role and name class variables, and the following methods: pledgeYourService and movieLine. These methods are reimplemented because of our choice to abstract them in the base class, or the HighTable. To add some more insight into the movie, MrWick also has an allies method that determines who his ally is at a given point in time. As a side note, this would be an interesting dynamic method to design in the future. Within the Wick movies, it seems who his allies are shift between and within movies. For example, sometimes Winston is his friend, like when he gave him an extra hour be excommunicato, or his enemy, like when he shoots him after parlay. Wow, Winston really is quite the politician! Some say that this was meant to happen to make another three movies. It makes you wonder if the John Wick series will follow the main conclusion of Underworld. That conclusion being that Selene was hunted by her own for political reasons to maintain the order of their tavern. Only at the end of the movie did they vote her to be an elder and maintain the northern tavern. Such a conclusion makes you think that John Wick will inherit the New York City Continental at the end of the sixth movie. Now that would be poetic justice! Without digressing into further fan theory, let us get back to our examples covering hybrid inheritance. Remember there are different types of inheritance covered in this section, but hybrid inheritance takes that perspective that we will be combining multiple types of techniques learned to fuse a better overall implementation. Using multiple inheritance types, we leverage the best of each type, while excluding cons that could inhibit our effectivity. Here is the code, of which we will cover after you see it.

```
  from   abc   import   ABC, abstractmethod

    class    HighTable  (  ABC  ):
    seats = [  'The Elder'  ,   'The Adjudicator'  ,   'The Administrator'  ,   'The Director'
    pledge =   "is agreeing to the rules set forth by the High Table."

      @abstractmethod
        def    pledgeYourService  (  self, name  ):
          pass

        def    councilSeats  (  self  ):
        print(  "This is what we know so far: "   +   " "  .join(self.seats))

      @abstractmethod
        def    movieLine  (  self  ):
          pass

        def    printStatement  (  self, name, line  ):
          try  :
            print(name +   " "   + line)
          except  :
            print(  "You provided bad input!"  )

    class    Winston  (  HighTable  ):
    name =   "Winston"
    role =   "Owner of the Continental."

        def    movieLine  (  self, role  ):
        print(role)

        def    pledgeYourService  (  self, name  ):
        self.printStatement(self.name, self.pledge)

    class    Director  (  HighTable  ):
    name =   "Director"
    trying2KillJohnWick =   "The High Table wants your life. How can you fight the wind?"

        def    movieLine  (  self  ):
        print(self.trying2KillJohnWick)

        def    pledgeYourService  (  self  ):
        self.printStatement(self.name, self.pledge)

    class    Marcus  (  HighTable  ):
    name =   "Marcus"
    role =   "John Wick's Mentor"

    class    MrWick  (  Winston, Marcus  ):
    name =   "John Wick"
    role =   "Baba Yaga"

        def    pledgeYourService  (  self  ):
        self.printStatement(self.name, self.pledge)

        def    allies  (  self  ):
        print(  "My allies are "   + Winston.name +   " and "   + Marcus.name +   "."  )

        def    movieLine  (  self, director  ):
        print(  "This is what the high table has said: "  )
        director.movieLine()
        print(  "Guess we need guns. Lots of guns."  )

wick = MrWick()
wick.pledgeYourService()
wick.movieLine(Director())
wick.allies()
```

I admit, it is difficult to see the hybrid inheritance with so much code. We will see later a handy tool in the industry called Unified Modeling Language (UML). For now, just know that UML exists and can help you with visualizing code. The real power of UML allows us to visualize code that is obscured from view.



Based on the UML, you may notice that the following classes inherit in a multilevel fashion from the HighTable class: Winston, Director, and Marcus. These implement abstracted methods locally and provide any extra features for MrWick's class's use. The MrWick class then inherits from both Winston and Marcus in a multiple inheritance design. That is the beauty of hybrid inheritance models, you can tailor what design is appropriate for each class and encapsulate what the end-user should have no right seeing. Designing UML models and OOP interfaces is an artform, you will get better with practice.

Method Resolution Order (MRO) of Inheritance

When you have multiple inheritance, you sometimes can get a diamond problem as seen below. A diamond problem is when you inherit from multiple classes where a class variable or method's signature is found more than once. If you run the code below you will be surprised to see that it works, but not as expected. Run the code and find out the result since I only gave you half of the output.

```
   class    MarvelUniverse  :
   Heroes = [  'Thor'  ,   'Iron man'  ,   'Hulk'  ,   'Black Widow'  ,   'Hawkeye'  ,   'Cap

   class    DC_Universe  :
   Heroes = [  'Batman'  ,   'Superman'  ,   'Wonder Woman'  ,   'Aquaman'  ]

   class    Multiverse  (  MarvelUniverse, DC_Universe  ):
       def    thanksgiving  (  self  ):
         return    "Its not a Thanksgiving without "   +   " "  .join(self.Heros) + \
  " "   +   " "  .join(self.Heros)

party = Multiverse()
print(Multiverse.__mro__)
print(party.thanksgiving())

  >>>   (  ,   ,   ,…)
```

If you ran this in your Linux terminal you would have noticed that the MarvelUniverse Heroes variable was called twice in the derived class's thanksgiving method. To avoid this diamond problem in the future, we should be aware of Python's Method Resolution Order (MRO) algorithm that takes our diamond issue and makes it easier for us by putting the calling order into a tuple. As seen in the code above, you can use the internal mro method to find out the calling order of that class (Lott & Philips, 2021). Sometimes you don't always know that you are inheriting multiple of the same names across either a class variable or method, but you should. Since you should know what you are inheriting in your child class, linearly search through the internal mro method of your class to verify you are using the correct class to make sure there are no surprises when you interpret and run your code.

Taking this a few steps further, we can see that MRO is an incredibly important principle to understand. Without understanding how classes, methods, or variables will be called could blind the programmer to critical errors. Thus, allocating proper security controls and functionality into the code to account for MRO will help mitigate any surprises that pop up.

Go above and beyond MRO, there is still a lot to learn about Python OOP implementations. For instance, we could wrap methods in decorators, we could solve MRO issues by leveraging metaprogramming with the __new__ method, and much more. This section was not an all-encompassing treatise about Python OOP. In the future, we plan to help alleviate advanced Python OOP understanding with examples. Through theoretical and practical knowledge of the subject, we can break down misunderstanding of advanced Python OOP.

Intermediate Cryptology

Encoding

Although encoding, such as Base64, is extremely useful for transmitting code in an efficient format, it is not truly encryption. Encoding is included here because people commonly mislabel encoding as encryption; on the PenTest+ exam if you see this as an option in an encryption question that answer usually can be crossed out. The purpose of encoding is to reformat the data into a useful representation. Sometimes useful can mean more efficient or digestible, but almost always means a different encoding scheme. For instance, Base64 encodes ASCII (seven bits) into six bits using only a-z, A-Z, 0-9, +, and /. Below is code to represent switching back and forth from Base64.

```
 import base64

password = "dXNlcnRlc3Q6cGFzc3dvcmQ="
passwordBytes = password.encode( "ascii" )
string_bytes = base64.b64decode(passwordBytes)
ans_string = string_bytes.decode( "ascii" )
print( f"Decoded string: {ans_string} " )
```

Another important one is hex encoding. To understand that we must first understand number systems. Hex is base 16, decimal is base 10, octal is base 8, and binary is base 2. What they means is that each slot can only incorporate up to, but not included, the base number (also referred to as modulus of the base number). Let us observe hex to int, then hex to Base64 encodings.

```
  def    decodeHex2Int  (  hexcode  ):
   dictionary = {  'a'  :  10  ,   'b'  :  11  ,   'c'  :  12  ,   'd'  :  13  ,   'e'  :  14
   ints =   ""
     for   x   in   range(  0  , len(hexcode),   2  ):
       i = hexcode[x:x+  2  ]
         if   i ==   '00'  :
           ints +=   " "
         else  :
           first = i[  0  ].lower()
           second = i[  1  ].lower()
             if   first   in   dictionary.keys():
               first = dictionary[first]
             if   second   in   dictionary.keys():
               second = dictionary[second]
           ints += str(int(first)*  16   + int(second))

     return   ints
```

```
import   base64

  def    hex2Base64  (  hex_string  ):
   decodedString =   ""
     for   num   in   range(  0  , len(hex_string),   2  ):
       temp =int(hex_string[num:num+  2  ],   16  )
       decodedString += chr(temp)
   sample_string_bytes = decodedString.encode(  "ascii"  )
   base64_bytes = base64.b64encode(sample_string_bytes)
     return   [base64_bytes, decodedString]
```

Sometimes it is nice to incorporate both python and bash. You can integrate all of the concepts above using this xonsh script.

```
import   sys
import   base64
# Put in custom base64 encoder/decoder later

  def     hex_to_chr  (  hexcode  ):
  dictionary = {  'a'  :  10  ,   'b'  :  11  ,   'c'  :  12  ,   'd'  :  13  ,   'e'  :  14
  word =   ""
    for   x   in   range(  0  , len(hexcode),   2  ):
      i = hexcode[x:x+  2  ]
        if   i ==   '00'  :
          word +=   " "
        else  :
          first = i[  0  ].lower()
          second = i[  1  ].lower()
            if   first   in   dictionary.keys():
              first = dictionary[first]
            if   second   in   dictionary.keys():
              second = dictionary[second]
          word += chr(int(first)*  16   + int(second))

    return   word

  def     encoded_text  (  cipher_text: str, encoding: str  ):
     if   encoding ==   "base64"  :
      ![echo @(cipher_text) | base64 -d]
      string_bytes = base64.b64decode(cipher_text)
      print(string_bytes.decode(  "ascii"  ))
    elif   encoding ==   "hex"  :
      ![echo @(cipher_text) | xxd -r -p]
      print(hex_to_chr(cipher_text))
        # OR
      print(cipher_text.decode(  "hex"  )

if   __name__ ==   "__main__"  :
  encoded_text(sys.argv[  1  ], sys.argv[  2  ])
```

Identifying Hashes

Sometimes the hardest part of cracking a program is the identification of the hashing algorithm you are working with. There are great tools for this, such as Hashcat, John The Ripper, Hashid, and more. Although, hashcat and JTR normally prefer that you provide the mode/format, they will sometimes tell you what the algorithm is suspected to be. Hashid is another beast entirely, as it's sole purpose is to identify the algorithm that made the hash provided. The structure of the hex string is the key identifier when determining the algorithm used. Leveraging regular expressions can help narrow down the algorithm relatively quickly by matching the hex strings against pre-defined patterns, which could be compiled. Below is a quick example that re-implements hashid for the common user.

```
  import   re
  import   sys

  # Shadow file: https://www.cyberciti.biz/faq/understanding-etcshadow-file/
  # Regexes derived from: https://github.com/psypanda/hashID/blob/master/hashid.py
regexes = [ [   r'(^[a-f0-9]{32}(:.+)?$)|(\$1\$.*)'  ,   "MD5"   ],
            [   r'^(\$NT\$)?[a-f0-9]{32}$'  ,   'NTLM'   ],
            [   r'^[a-f0-9]{40}(:.+)?$'  ,   'SHA-1'   ],
            [   r'(^[a-f0-9]{64}(:.+)?$)|(\$5\$)'  ,   'SHA-256'   ],
            [   r'(^[a-f0-9]{128}(:.+)?$)|(\$6\$)'  ,   'SHA-512'   ],
            [   r'^\$2[ay]\$'  ,   'Blowfish'   ]
          ]

    def     hash_identifier  (  hash_of_interest: str  ):
      for   regex_pattern, hash_algorithm   in   regexes:
          if   re.search(regex_pattern, hash_of_interest, re.IGNORECASE):
              return   $(echo   "Found your hash: @(hash_algorithm)"  )
      return     f'Could not found a hashing algorithm for   {hash_of_interest}  '

  if   __name__ ==   "__main__"  :
    response = hash_identifier(sys.argv[  1  ])

    print(reponse)

      if   int($(echo @(response) | grep -o   "Found"   | wc -c)) !=   0  :
        print(  f'Let\'s crack   {sys.argv[  1  ]}   with the Ripper now!'  )
```

Identifying Readable & Writeable Shadow Files

After getting onto the victim's machine, whether through ssh, netcat, or something else, you have identified that the /etc/shadow file is readable and writeable. Below is how you would check then grab that information. The script is a little tricky, but all it's doing is using a long listing on the file of interest. Then, it cuts on the permissions (the first field) and cuts on the read and writeable characters for others. If you understand this, it is not difficult to check the group permissions as well. That assumes you were able to be pivot towards a group that you have inclusion towards!

```
info=$(ls -l /etc/shadow | cut -d   " "   -f 1 | cut -c8-9);

  if   [ $(  echo     ${info:0:1}  ) =   "r"   ]
  then
      echo     "Shadow is readable"  ;
  fi

  if   [ $(  echo     ${info:1:1}  ) =   "w"   ]
  then
      echo     "Shadow is writeable"  ;
  fi
```

Unshadowing

If you have a readable shadow file go ahead and use the unshadow method as seen below. Using John The Ripper, Hashcat, or another cracking software suite, we can attempt to break the password stored. Now the cryptographic strength, chosen wordlist, and resource/time allocation will determine if the passwords are broken. Typically, the passwords in Linux are stored using bcrypt/SHA-512. According to Cyberciti.biz, the /etc/shadow file holds encrypted password in the second field and the algorithm is identified by the following formula: $id$salt$hashed (Gite, 2022). On Linux, you will usually see $1$ (MD5), $2a$ (Blowfish), $2y$ (Blowfish), $5$ (SHA-256), $6$ (SHA-512) (Gite, 2022). Although not required, you could provide John The Ripper with the hash identifier using the format option. Determining the hashing algorithm is rather easy, as it relies on regular expressions. Go ahead and use hashid on the command line with the hash of interest to determine what algorithm was used!

```
  # Pass users/accounts of interest as command line args
  # Simple; order will break this code snippet

  declare   -A importantAccounts=  "$*"  ;
  #printf 'Arg: %s\n' "${importantAccounts[@]}"

mkdir shadowing
  cd   shadowing
touch passwds
touch shadows

IFS=$  '\n'  ;
  for   line   in   $(cat /etc/passwd)
  do
        account=$(  echo     $line   | cut -d   ":"   -f 1)

          for   user   in     "  ${!importAccounts[@]}  "
          do
                  if   [[   $user   ==   $account   ]];
                  then
                          echo     $line   >> passwds;
                          break  ;
                  fi
          done
  done

  for   line   in   $(cat /etc/shadow)
  do
        account=$(  echo     $line   | cut -d   ":"   -f 1)

          for   user   in     "  ${!importAccounts[@]}  "
          do
                  if   [[   $user   ==   $account   ]];
                  then
                          echo     $line   >> shadows;
                          break  ;
                  fi
          done
  done

unshadow passwds shadows > passwordCrackingTime.txt
john --wordlist=/usr/share/wordlists/rockyou.txt passwordCrackingTime.txt
```

Custom Cracking - Brute Force

Brute forcing is not ideal, but can be useful in certain situations. For example, the techniques you are about to see works in a similar manner to Hydra, Medusa, Pratator, etc. Once you have a wordlist that fits the socially profile of the target (individual or company), then you can blend social engineering with password cracking. The scripts differ from Hydra by the fact that we are offline cracking here. However, if you tie these scripts to a custom wordlist and target service, then you will be online password cracking.

```
IFS=$  '\n'  ;
  for   i   in   {0000..9999}
  do
        _hash=$(  echo     $i   |   $2   | cut -d   " "   -f 1);

          if   [   "  $_hash  "   =   "  $1  "    ]
          then
                  echo     "Found, its   $i  "   ;
                  break  ;
          fi
  done
```

```
  import   hashlib
  import   sys

    def     hashIt   (  algorithm: str, data: str  ) -> str:
      if   algorithm ==   "sha1sum"   :
        _hash = hashlib.new(  'sha1'  )
      elif   algorithm ==   "md5sum"   :
        _hash = hashlib.new(  'md5'  )
      elif   algorithm ==   "sha256sum"   :
        _hash = hashlib.new(  'sha256'  )
      else  :
        _hash = hashlib.new(  'sha512'  )

    _hash.update(data.encode())
      return   _hash.hexdigest()

    def     simpleCrack   (  hashFound: str, algorithm: str, ranger: int = [  1   ,    9999   ]  )
      if   algorithm   not     in   [  "md5sum"   ,    "sha1sum"   ,    "sha256sum"   ,    "sha512sum
          raise   Exception(  "Algorithm desired is not supported!"   )

    padLength = len(str(ranger[  -1  ]))

        def     pad   (  val: str, length: int =   3    ):
          while   len(val) != padLength:
            val =   "0"   + val
          return   val

      for   i   in   range(ranger[  0  ], ranger[  -1  ]):
        potentialHash = hashIt(algorithm, pad(str(i)))
        print(i, potentialHash)
          if   potentialHash == hashFound:
              return     f'Found hash value:   {potentialVal}  '

      return     f'Could not find the hash value for   {hashFound}  '
```

Putting It Together - Case Study

Putting it Together - Case Study

Automated identification and cracking can easily be programmed; however, the source or method you use to retreive those encrypted hashes can vary. Completing the rest of the code below will be to the reader as a case study. Integrate the previous code snippets by including methods to identify a few attack paths that may be of interest. Then check what the hashing algorithm used was. Finally, send the result to be cracked! It's as easy as 1, 2, 3!

```
import   re
import   sys

  class     Passwords  :
    # Shadow file: https://www.cyberciti.biz/faq/understanding-etcshadow-file/
    # Regexes derived from: https://github.com/psypanda/hashID/blob/master/hashid.py
  regexes = [ [   r'(^[a-f0-9]{32}(:.+)?$)|(\$1\$.*)'  ,   "MD5"   ],
                [   r'^(\$NT\$)?[a-f0-9]{32}$'  ,   'NTLM'   ],
                [   r'^[a-f0-9]{40}(:.+)?$'  ,   'SHA-1'   ],
                [   r'(^[a-f0-9]{64}(:.+)?$)|(\$5\$)'  ,   'SHA-256'   ],
                [   r'(^[a-f0-9]{128}(:.+)?$)|(\$6\$)'  ,   'SHA-512'   ],
                [   r'^\$2[ay]\$'  ,   'Blowfish'   ]
              ]

      def     __init__  (  self, hash_of_interest: str, mode: int =   0    ):
        try  :
          self.run(hash_of_interest, mode)
        except  :
          print(  "There was a problem with cracking this hash!"  )

      def     run  (  self, _hash: str, mode: int  ):
      algo = self.id_hash(hash_of_interest)

        # Remember "" strings are false in Python
        if   algo !=   False  :
          print(self.crack(hash_of_interest, algo, mode))
        else  :
          print(  f'Could not found a hashing algorithm for   {hash_of_interest}  '  )

      def     id_hash  (  self, hash_of_interest: str  ) -> str:
        for   regex_pattern, hash_algorithm   in   self.regexes:
            if   re.search(regex_pattern, hash_of_interest, re.IGNORECASE):
                return   hash_algorithm
        return     ""

    # Modes: 0 for JTR, 1 for Hashcat
      def     crack  (  self, hash_of_interest: str, algo: str, mode: int  ) -> str:
        # HERE
        # SCRAP BOTH JOHN THE RIPPER AND HASHCAT
        # COULD MODIFY HASHCAT DEPENDING ON GPU
        # RETURN PASSWORD OR ""

if   __name__ ==   "__main__"  :
    if   len(sys.argv[  1  :]) >   1  :
      mode = int(sys.argv[  2  ])
    else  :
      mode =   0

  Passwords(sys.argv[  1  ], mode)
```

Database Case Study

Modify the code below to create a sustainable Initialization Vector (IV)

(iv)

```
import   time
import   os
import   json
from   random   import   randint
from   Crypto.Cipher   import   AES
import   hashlib
from   random   import   randint

  def     generateHmac  (  tok, salt = False, rounds = False  ):
    if   salt ==   False  :
      salt = str(randint(  10000  ,  1000000  ))
    if   rounds ==   False  :
      rounds = randint(  100000  ,  200000  )

  _hash = hashlib.pbkdf2_hmac(  'sha512'  , json.dumps(tok).encode(), salt.encode(), rounds

    return   str(_hash), salt, rounds

  class     EncryptionManager  :
      def     __init__  (  self, key=  "@cipher"  +str(  int(  time.time(    )  )  )[:  8  ],
      self.key = self.pad(key)[:  16  ].encode(  "utf8"  )
      self.iv = self.pad(iv).encode(  "utf8"  )

      def     pad  (  self, mes, length=  16    ):
        while   (len(mes)%(length))   and   len(mes) !=   0  :
          mes += chr(randint(  97  ,  122  ))
        return   mes

      def     remove  (  self, mes  ):
      backPtr =   -1
        while   mes[backPtr] !=   "}"  :
          backPtr -=   1

        return   mes[:backPtr+  1  ]

      def     encrypt  (  self, mes  ):
        return   AES.new(self.key, AES.MODE_CBC, self.iv).encrypt(self.pad(mes).encode(  "u

      def     decrypt  (  self, ciphertext  ):
      imm = self.remove((AES.new(self.key, AES.MODE_CBC, self.iv)).decrypt(ciphertext).dec
        return   json.loads(imm)

## Pass a token with a username and password, then decrypt it to verify!!!
```

RSA Token Generator Case Study

Using the database code above, go and finish the RSA token case study below.

```
class AuthenticationManager:

    def __init__(self, timeToLive: int):
        self.map = {}
        self.timeToLive = timeToLive

    def generate(self, tokenId: str, currentTime: int) -> None:
        self.map[tokenId] = currentTime

    def renew(self, tokenId: str, currentTime: int) -> None:
        if tokenId in self.map.keys():
            if (currentTime - self.map[tokenId]) >= self.timeToLive:
                del self.map[tokenId]
            else:
                self.map[tokenId] = currentTime

    def countUnexpiredTokens(self, currentTime: int) -> int:
        if len(self.map) > 0:
            count = 0
            items = list(self.map.items())
            for item in items:
                if item[0] in self.map.keys():
                    if (currentTime - item[1]) >= self.timeToLive:
                        del self.map[item[0]]
                    else:
                        count += 1
            return count
        return 0


# Your AuthenticationManager object will be instantiated and called as such:
# obj = AuthenticationManager(timeToLive)
# obj.generate(tokenId,currentTime)
# obj.renew(tokenId,currentTime)
# param_3 = obj.countUnexpiredTokens(currentTime)

def getRsaToken():
    pass
```

Deeper Understanding

After identifying some of the key characteristics of cryptology, explain how you would modify the code below for a reliable lattice-based encryption algorithm using only numpy! This code is an open-source implementation of OpenStego using Numpy.

Once that is accomplished, figure out how you would implement this solution using ffts to hide the information in sound.

Finally, Coagula is a blue force tool that is used to combat both of the techniques above. Go forth and reimplement Coagula so that we may better understand our blue force counterparts!

```
  import   base64
  import   hashlib
  from   Crypto.Cipher   import   AES
  from   Crypto   import   Random
  import   numpy   as   np
  from   numpy.random   import   default_rng
  import   matplotlib.pyplot   as   plt
  from   Crypto.Cipher   import   AES
  import   matplotlib.image   as   mpimg
  from   PIL   import   Image

  ### AES CODE FROM https://www.delftstack.com/howto/python/python-aes-encryption/ ###
BLOCK_SIZE =   16
pad =   lambda   s: s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * chr(
    BLOCK_SIZE - len(s) % BLOCK_SIZE
)
unpad =   lambda   s: s[: -ord(s[len(s) -   1   :])]


    def     encrypt  (  plain_text, key  ):
    private_key = hashlib.sha256(key.encode(  "utf-8"  )).digest()
    plain_text = pad(plain_text)
      # print("After padding:", plain_text)
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
      return   base64.b64encode(iv + cipher.encrypt(plain_text.encode()))


    def     decrypt  (  cipher_text, key  ):
    private_key = hashlib.sha256(key.encode(  "utf-8"  )).digest()
    cipher_text = base64.b64decode(cipher_text)
    iv = cipher_text[:  16  ]
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
      return   unpad(cipher.decrypt(cipher_text[  16  :]))


  ### AES CODE FROM https://www.delftstack.com/howto/python/python-aes-encryption/ ###


mapping = {idx: chr(  65   + idx)   for   idx   in   range(  26  )}
  for   idx   in   range(  26  ):
    mapping[  26  +idx] = chr(  97   + idx)
  for   idx   in   range(  10  ):
    mapping[  52  +idx] = str(idx)
mapping[  62  ] =   "+"
mapping[  63  ] =   "/"
mapping[  64  ] =   "="

base642int = {chr(  65   + idx): idx   for   idx   in   range(  26  )}
  for   idx   in   range(  26  ):
    base642int[chr(  97   + idx)] =   26   + idx
  for   idx   in   range(  10  ):
    base642int[str(idx)] =   52   + idx
base642int[  "+"  ] =   62
base642int[  "/"  ] =   63
base642int[  "="  ] =   64

    class     Pictures  :
        def     __init__  (  self, imagePath, key, message =   ""  , operation =   "Encrypt"
        key = key.split()
        print(key)
          if   len(key) !=   4     or   sum([  1     for   x   in   key   if   len(x) <   6  ])
              raise   Exception(  "Key needs to be four words, with each being at least 6 cha

        self.destination = destination
        self.save = save
        self.  print   = _print

        self.imageArr = np.array(mpimg.imread(imagePath))

          if   operation ==   "Encrypt"  :
            self.cipherArr = [base642int[x] ^ self.words2int(key)   for   x   in   encrypt(me
            self.msgLen = len(self.cipherArr)
            self.encode()
            print(self.replacements)
            print(self.offsets)
          else  :
            self.decode(key, replacements, offsets)

        def     words2int  (  self, words  ):
        integerTotal = sum([sum([ord(letter)   for   letter   in   word])   for   word   in
          return   integerTotal %   255

        def     encode  (  self  ):
```

```
    def encode ( self ):
        rng = default_rng()
        self.x, self.y, self.z = self.imageArr.shape

        self.offsets = []

            def setupLocalFields ( indices ):
              # (b - a) * random() + a
              # Range avoids the boundaries for now...less bounds checking
            newIndices = []
            newIndices.append(indices[ 0 ] if indices[ 0 ] < (self.x - 10 ) and
            newIndices.append(indices[ 1 ] if indices[ 1 ] < (self.y - 10 ) and
            newIndices.append(indices[ 2 ])

            localField = 0

              for x in range( -2 , 2 ):
                  for y in range( -2 , 2 ):
                      # Skip z for now (means we arent crossing RGB channels)
                    localField += self.imageArr[x,y,indices[ 2 ]]

            self.offsets.append(localField // 16 )

              return newIndices

        self.replacements = []
          for indices in zip(
                            rng.choice(self.x, size=self.msgLen, replace= False ),
                            rng.choice(self.y, size=self.msgLen, replace= True ),
                            rng.choice(self.z, size=self.msgLen, replace= True )
                          ):
            self.replacements.append(list(setupLocalFields(indices)))

        self.imageArr[tuple(np.transpose(self.replacements))] = [(x + y) % 255 for x,
          for idx, item in enumerate(self.replacements):
            x, y, z = item[ 0 ], item[ 1 ], item[ 2 ]

          if self. print :
            plt.imshow(self.imageArr)
          if self.save:
            im = Image.fromarray(self.imageArr)
            im.save( "PeaceEncoded.png" )

    def decode ( self, key, replacements, offsets ):
        encoded_b64 = ""
        idx = 0

          for x, y, z in replacements:
            val = ((int(self.imageArr[x][y][z]* 255 ) - offsets[idx]) % 255 ) ^ self.wor
            encoded_b64 += mapping[val]
            idx += 1

        print(decrypt(encoded_b64, "" .join(key)).decode())
```

Project - Develop And Design An Intrusion Detection System (IDS)

Problem Statement

An open-source, functional cloud computing attestation tool that can be used in the cloud, on your local system, or in an embedded system. Hence with the examples listed in the previous section, we have set up an intelligence gap of how to attestify systems to protect us from APTs in the cloud, while at the same time providing systems reliability to protect the cloud vendor from loss of revenue due to systems failure. To do this, there are numerous questions that need to be answered, some of which are listed here. What important aspects needs to be accounted for when considering VM escape? How does the cloud environment contribute to a continuously challenging, dynamic environment? What has Linux operating systems done to attestify on local systems? How are vendors containing and adhering to security principles of attestation while the client could be switched between servers in the cloud? Are there any open-source solutions to help gain back trust in the cloud, and analyze real-time security profiles?

Introduction and Background

A historical need for cloud computing attestation was once again demonstrated in October of 2018 when China infiltrated the entire industry, to include FBI, CIA, and DoD (Robertson & Riley, 2018). The Bloomberg article pointed out that there is a huge security gap in the cloud computing industry to verify that least-privilege across the enterprise. Furthermore, last December contributed to another example, as mentioned by the NSA, that the veil of secrecy was broken when a Russian VMware escape attack that affected a large part of the industry (Goodin, 2020). We need to start analyzing the situation to develop a solution that fits both vendor-neutral and intelligence agencies. There are numerous promising candidates, but only time will tell which one is going to be a universal solution.

Another extreme historical example from 2017, where Amazon lost $150 million due to faulty engineering of enterprise downtime, led to a profitable interest in designing better automated log architectures (He et al, 2021). Pure academia and the study of cybersecurity is not supposed to be about money, but the CISSP teaches otherwise. For instance, the CISSP says that IT is meant to support business objectives. We as IT professionals need to map our roles and responsibilities with C-suite objectives, and if we can’t than we are not doing our job correctly. Each process and task done within an IT role should be related to policy set forth by the executives. Not only are we aligned to business goals, but IT is not traditionally a revenue generating department: IT does not make money but spends it. Even in this Amazon example, we are expected to provide a service that is reliable. Understanding the nuisances pertaining to failures were a subject of interest as well during their background literature scan. Two issues that popped up was that 60% of failures from software faults did not leave evidence of such in the logs, and that 70% of logging patterns designed to find errors were skewed towards the end of a block of instruction (He et al, 2021). Hence demonstrating that our job does not get paid extra for things working, they are expected to work. Shaping your individual-organizational mindset to remember that IT is traditionally there to support and not generate should help when mapping quantitative cyber incidents, like a CVE, to qualitive assessments within the risk management framework process for C-suite members.

Objectives Of The Project

1. Hashing special files and directories on the system

2. Having the end-user leverage RSA tokens

3. Using RSA tokens, clients could provide JSON Web Tokens (JWT) for real-time authentication

4. Application profiling of client software could help mitigate zero-day attacks using machine learning

5. Only use secure protocols with open-source validation like DNS over HTTPS or DNS over TLS, IP and file integrity APIs, and IP geolocation

6. Monitor for open and unsecured ports

7. Verify your log files to ensure only the right profiles are accessing the right resources, or in other words, least privilege is verified

8. Check that no unknown/unverified process is running on your system

Requirements

1. There are three expected deliverables: UML code, source code, and a 10-page written report

2. All classes must follow best OOP design principles

3. You must provide UML code for all classes

4. If any exists, you must describe each classes inheritance. Going through these OOP principles should help solidify cybersecurity concepts, while at the same time permitting learning opportunities that could help them market their skillsets. Some skillsets that can be derived from this exercise includes DevOps, threat hunting, cloud engineering, and much more

5. Use REST HTTP fundamentals for cloud-ready programmability. API programmability of user profiles within the profiles share is the crux of the JSON REST server. Keeping accountability and integrity of user directories across platforms, even if the server swap server racks, can be done at a reduced cost. When users enroll, they can attestify their directories, files, network connections, and FQDNs

6. Every malicious share should be updated at least bi-weekly, to include indicating the sources you collect the threat intelligence from. The greater the number of sources you pull from, assuming legitimate intelligence, the better your cloud-ready solution will be

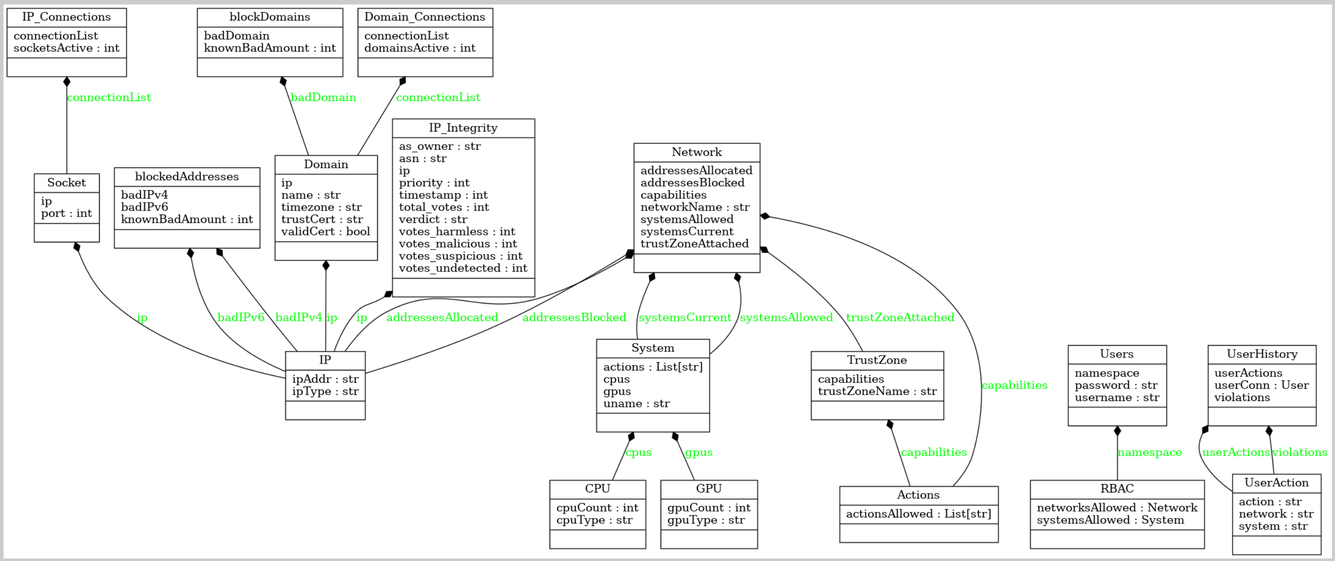
7. Users also have the option to de-enroll should they choose

8. Incorporate RFC 7519 as a baseline for JSON Web Token (JWT) class.

UML Basics

The Unified Modeling Language (UML) is an excellent way to communicate your source code with others. Understanding how to effectively communicate through UML takes practice, but here we will dive into what types of diagrams are available and what we will be using later.

Diagrams within UML is split into a dichotomy, either you are creating a structural UML diagram or a behavioral UML diagram (Smart Draw, 2021). Within the structural UML category, you have the following diagrams: class, package, object, component, composite structure, and deployment (Smart Draw, 2021). For the behavioral UML diagram, you have the following diagrams: activity, sequence, use case, state, communication, interaction overview, and timing (Smart Draw, 2021). We will not be covering most of these types of diagrams, and instead will only be focusing on the backbone of every object-oriented method: the class diagram. It's not that the other diagrams are not important, it is just the fact that the class diagram will meet all our needs as we start down the road to Python OOP. Within this, the class diagram usually has the following types included within each UML model: class, interface, data types, and component (Bell, 2004). Later in a book specifically for UML, we will be covering the associations present. For now, let us take an informal approach by pointing to the class that we will be inheriting from.



UML Explanation

Within the UML you can see numerous types of inheritance, to include single inheritance, multiple inheritance, hierarchical inheritance, multilevel inheritance, and hybrid inheritance. I implore the reader to go through every detail of this UML code but doing so here would take up too much time and space. What is important on the other hand is that the UML code minimalized code redundancy and maximized code reusability by using OOP principles. Using base classes, the UML code used class variables and methods once, but permitted access to numerous derived classes. In essence, child classes did not have to redesign the wheel and implement those same class variables and methods. Ultimately, this means that the IDS and user classes can inherit a multitude of operational capabilities through abstracted interfaces. Both IDS and user do not need to know everything that is happening behind the scenes. The best part of using abstraction is that the user does not need to know anything besides how to set and remove users, and then display their metrics.

Extension Recommendations

Regarding software implementation and development, the code has already been written for the UML presented, but is left to the reader for practice. The reason why it was kept out of this book is to help you foster the necessary skills to decipher cybersecurity requirements, capture key stakeholder input, and then design and implement the key performance parameters (KPI) for a fully functional solution. A few recommendations regarding the code would be the following: write these classes using the principles of concurrency, address the frontend JSON server look and API documentation, and integrate the user class with RSA hardware tokens.

Extending this solution so that the user class can take in two-factor authentication (2FA), a subset of multi-factor authentication (MFA), would help provide defense in-depth against sophisticated attacks in today’s cyber realm. Ultimately, this would have to be taken as a parameter with the user class and passed through the hashManager class, which would inherit from the jwtlib class. Implementing the core methods in jwtlib would abstract any APIs required with trusted USB or TPM keys. The hashManager class could then utilize these core methods that were inherited to then perform useful security attestation, even when the cloud vendor swaps the OS to a different server stack. Within this implementation, we can use 2FA through SMS with username and passwords or RSA tokens with username and passwords; in essence, we would be leveraging 2FA through something you have and something you know (Pearson IT Certification, 2011). Each of the previously mentioned examples use two different types of authentications. For example, whether it is an RSA token, a Yubikey, or a smart card, all these falls into the category of something you have. A username and password are a traditional example of something you know, but now a pin count as well! Therefore, whether you enjoy Titan or YubiKey products as your preference. The point is that they provide key pairs that allow dynamic attestation of certificates, and sometimes EV code and document signing (SSL, 2020). This would relate to one of features we tried to implement with the Red Phone in the cold war, which is now at the tips of everyone’s fingers: one-time pads (OTP). Using dynamic OTPs through NTP and RSA/ECC tokens can provide similar encryption standards that the USG relied on for years at the above top-secret level (Cimpanu, 2019). Incorporating these into this solution would ensure fidelity and reusability of security principles across cloud vendors.

Due to the lack of transparency and accountability in times of a geopolitical crisis, it can be tough to get authorization for activities. With the revamp of Pegasus, it is pertinent to incorporate real-time authorization for uncleared individuals, systems, and organizations after a validated NSC request (CITE). To do so without adding member or organizational enrollment may be conducted by embedding permissible and permeable JWT claims within Personal Assertion Tokens, and further structuring the information with dictionary-based substitution. Ultimately, this would permit action on behalf of the member, system, or organization in real-time situations. This would thus minimize enrollment overhead, latency of certification validation per client, and increase security. It should not be unbeknownst to the observer that machine-to-machine (M2M) collaboration in a multi-domain environment would benefit the most within a contingency-based kill-chain scenario due to the alacrity of the process. Without glancing too far into how DARPA's killchain attestifies and dynamically changes in multi-domain warfare environments, it should be obvious that JWTClaimNames within PASSporTs were specifically designed for autonomous warfare (Housley, 2021). Two examples of such can be seen as follows: beyond line-of-sight missile handling authorization and HVT target reprioritization. For these examples, target reauthorization and prioritization for M2M warfare could autonomously take into account a certificate that authenticates to assets after intruding on the killchain targeting cycle; such an authentication could include a certificate announcing an Enhanced JWT Claim Constraints certificate extension that contains keypairs with the following permitted values JWT Claim Names: {"OPERATION": “BERRYCOOL”, “NSC-Authority”: “ECHOCHARLIE”, “TARGET”:”LOCATION”,“PRIORITY”:"CRITICAL"}. To wrap up the novel techniques that could be incorporated within resilience Global Information Grid (GIG) networks, the observer of such claims should also be aware of one more built-in operations security feature (OPSEC): inherent message security through Enhanced JWT Claim Constraint restructuring. Open standardization of the PASSporT structure is expected and encouraged due to the built-in OPSEC extensions of situational dependent Enhanced JWT Claim [illegible] extensions of situational dependent Enhanced JWT Claim Constraints, with such would allow native filtering without deduplication or sensory overload. Lastly, these situational extensions should be seen as a closely held secret protected to the same level as the classification levels of the information, assets, and operations. In the end, leveraging PASSporT’s native Enhanced JWT Claim Constraints extensions could be useful for attestifying all processes and assets, to include both cloud-based environment and the multi-domain environment.

During the survey of academia and the industry, these researchers were able to uncover four problems with modern day log analysis systems. These four areas included that systems engineering was traditionally based around the developer’s domain knowledge, that the spread of big data analytics has percolated to gigabytes of data per hour, that global interdisciplinary teams has led to a global version control issue where team members were not involved in the original planning, and that modern AGILE software engineering practices has required a continual update cycle to established logging architectures (He et al). Let us break these thoughts down one by one. It is incredibly difficult to step outside of our mindset and perspective and account for not just your specialty, but to provide a concerted, cross-disciplinary solution. Breaking out of the continual thought process is the reason why De Bono’s Thinking Course or Copi’s Informal Logic course is so critical. As analysts, whether that is cybersecurity, machine learning, geopolitical analysis, or whatever you are providing predictive analytic reasoning on, breaking away from your traditional habits to look freshly at a subject can’t be understated. Within De Bono’s Thinking Course, in the Digital Analyst course we will cover various techniques that allow us to look at the subject or topic anew. No longer will you be stuck from a mental transition or have the inability to decipher new possibilities to a problem. Instead, with his tools and techniques, you can liberate your mind to observe the possible. Seeing more possibilities does not guarantee a successful prediction but will help create alternative lines of analysis that could guide your strategic thinking towards the right conclusion. With the fusion of informal logic, you will also facilitate a methodology to disregard alternative lines of analysis to parse what is both possible and plausible.

A greater explanation and expansion upon log compression could be helpful. Log compression is one of the most pertinent aspects associated with the Global Information Grid (GIG) but is the least researched subject when discussing automated logging (He et al, 2021). The problem today is not that we don’t have enough data, but that we are overtaxed by the information that we do have. The sheer volume of information is difficult to parse for answers, and extremely troublesome to send back to all the key stakeholders. More appropriate compression solutions are the answer we need when talking about big data analytics associated with the GIG. Various techniques listed here included buckets, dictionaries, statistics, machine learning, and parallelization (He et al, 2021). Using a concerted effort from all these techniques may offset any weaknesses and amalgamate data extractors for a better solution. In the model that will be designed by every student in the veteran bootcamp, we will first focus on parallelizing bucket-based compression through rigid JSON schemas. Then we will use log parsing to identify common keywords and phrases that are mapped to domain specific code pages. These code pages will act as the cipher and decipher for more efficient encoding of information. Through Bayesian statistical techniques we can further consolidate the logs that have been generated. To play the advocate, if we needed to collect assets in real-time it would be extremely difficult without their BSON code pages. Beyond just the capabilities, understanding operational modes of BSON categorization is one of the most significant aspects when considering blue side capabilities in real-time. Without these operational modes of BSON categorization, our systems of systems would be blind in a real-time sensor fusion analysis.

Planning And Scoping - Understanding the Enterprise

RMF

Starting with a hard concept like Risk Management Framework (RMF) is usually discouraged because it tends to be concepts even mid-level managers have a problem following. So why teach it with individuals who only have six months of vulnerability management experience? The answer is quite clear: do it right the first time! Our culture tends to baby those in the workforce by easing them into the job and partially giving sense to their position, only to find that a few years down the road they say that’s not how that works. Now you must go back and relearn how vulnerability management is done at the 5-year mark. Let us stop the nonsensical way of learning and just do it right the first time!

So, then what is RMF exactly? It’s a strategic, enterprise framework to help all companies take a holistic approach for their security posture, this includes a lifecycle of incorporating and monitoring stakeholder guidance and equity to make decisions about the security direction of organization, through rigorous categorization, selection, and implementation. The following visualization was taken from NIST and helps provides a nice graphic to digest the information.

As you can see, NIST provides an excellent suite of functions that contribute to the lifecycle of your enterprise security posture. The first is preparing the organization for their security and privacy consideration (NIST, 2021). For junior-level vulnerability management professionals this may seem a little scary. Unless you work at a start-up or small business, you probably have never interfaced with an executive before. These executives, or C-suite members, make the decisions on behalf of the company for it’s livelihood. Nevertheless, it is pertinent to understand that everything you do in IT at your job, whether administrating Group Policy Objects (GPO) or developing software patches for the latest vulnerability, must be tied back to business objectives that these C-suite members set. In most companies in the world, IT is a consumer and not a producer of funds within the organization. If your daily tasks as an IT member cannot be linked back to those business objectives then you are not performing the job as seen fit by the executives of your company. There are times these ideas get lost from C-suite to system administrator and is representative of the game we play as children called telephone. Reorganization or reprioritization is then required to better align the strategic vision and mission of the company. It is also important to note here that most of the time these same C-suite members must address the concerns and needs of key stakeholders. This will be covered later in this section, but just know that key stakeholders can vary from time to time and vary based on what they view as important. Also, there are certain regulatory and legal frameworks that must be followed within certain industries to be able to operate your business. For example, if you are within the banking industry, then you may be required to attest that your following legal requirements when it comes to customer data, firewall standards, etc. If instead you are a defense contractor, you will be required to maintain your information systems according to FISMA standards.

If C-suite members analyzed their market environment and aligned business objectives to maximize on these trends, these objectives are now passed to mid-level managers to capture and categorize what information systems will be impacted (NIST, 2021). These could be software, hardware, or people. Understanding the entirety of your corporate enterprise, to include both on-premises, in the cloud, and remote work, can be a struggle and requires great corporate insight. Not only do you have to identify and capture the necessary systems that you can see and own, but you need to identify upstream and downstream information systems, possibly in other companies, that would impact the systems you oversee. Once that part is tackled, you have the task of performing the impact analysis to determine where resources should be allocated. Not even the United States Government (USG) has unlimited resources, there needs to be prioritization on what is important so that we can move onto the next step. This prioritization can be conducted by qualitative and quantitative analyses. Most of the time, we analyze this through the ideas of threat and vulnerabilities to determine risk. The next section will cover the notion of risk more, just know that based on a system's vulnerability, like a Common Vulnerabilities and Exposures (CVE), and the likelihood of that happening, or threat, we can categorize the necessary information systems and processes appropriately. The common methodology used in the industry to deliver this to mid to senior managers is through the use of a risk acceptability matrix. A risk matrix fuses qualitative and quantitative risk to deliver strategic situational awareness on information systems without being technically inclined or knowledgeable about the system. For instance, a senior level manager does not need to understand SQL injection against their PHP Admin Lite page. Instead, security and cyber analysts leverage open-source vulnerability databases, like MITRE, to parse scoring metrics that would compare this CVE, using the Common Vulnerability Scoring System (CVSS), for the manager to contrast with against other vulnerabilities. In so doing, the manager can decipher the security threats and allocate proper resources towards the assets and information systems that pose the greatest risk to the organization.

Remember that we most likely do not have unlimited resources to tackle every issue, so there will be some that are prioritized, while others will not be touched. More on this is covered in the types of risk section.

After parsing out and categorizing cybersecurity items of interest, we perform the selection of NIST SP 800-53 controls (NIST, 2021). Last September, NIST provided an excellent tool for stakeholders to improve their implementation of the RMF process by way of accessing the controls and control baselines (SP 800-53B) in real-time (NIST, 2021). In the real world, analyzing the threat environment and understanding what vulnerabilities are associated with what is not enough. There needs to be action done on these vulnerabilities, or else why did we do this in the first place. Gathering blue side mitigations using NIST SP 800-53 allows our managers tangible courses of actions (COA) to ensure our security posture. Some may even find the reverse process of identifying the security baselines within NIST SP 800-53B first to be fruitful. After identifying the baselines, you could then go and find the controls necessary to securely deploy it within your enterprise. This reverse approach is great for new development areas, while the former is usually done for information systems that are already in place.

After selection you must go through the hard work of implementing these information systems and controls. Although this may seem like the most laborious and frustrating task, if everything was done properly in the prior steps then this should be rather easy. Though it is true that everything done on paper is easier, or that it is easier said than done. Look to common installation guides, company manuals, to coworkers, and if all else fails, go check out online forums. Remember to do it right the first time so that you do not need to waste time and must implement it a second time.

The next two steps can be very similar, so I will lump them together here: assess and authorize. Assessing that the information systems and controls are properly in place is a necessary evil to be able to allow senior level managers make a determine on the worthiness of the system (NIST, 2021). In the military you will hear terms like Air Worthiness, Authority to Operate (ATO), Certification and Accreditation (C&A), System Validation, etc. All these things really boil down to the same thing, you have verified that the information systems are deployed appropriately and that the controls are in-place providing a certain security profile that is acceptable. As soon as that ATO or C&A is given, the information systems are said to be fully operational and deployed.

Once there is an ATO for the information system, the tangible and intangible assets are passed down to the operational team, like mission control managers, system administrators, intelligence analysts, security officers, etc. It is the job of these professionals to ensure that the ATO is continuously monitor for any violation or infraction. The information system will now remain in this state for the remain of its lifecycle, except in the instance that the asset is reutilized or decommissioned.

Types of Risk

There are five main types of risk you should be aware of moving forward in your career: mitigation, acceptance, avoidance, aversion, and transfer. Each of these types of risk contribute to your organization’s security posture, whether that be to strengthen your security profile or to open vulnerabilities in your network. Before continuing, risk aversion is really a state of mind and not a decision state. Instead of considering risk aversion as a decision state, it is really the tendency at which you choose a higher or lower security process. This is derived from your C-suite member posture on security but needs to be understood going into the risk analysis process. Separating from risk aversion, I tend to do everything in reverse because it gives you a different perspective. In honor of that thought process, let us start with the last option to transition to the beginning.

The very last option you should consider is knowing the risk and accepting it, also called risk acceptance. Unwilling to deal with the risk, or vulnerabilities and threats, at hand while continuing forward to operate can be extremely dangerous if not accounted for. It is not uncommon for a customer to say, “Yes, yes we accept the risk! Just let us operate already, I have deadlines you know!”. If your job is to certify and accredit the information systems, then you must maintain integrity and resist the temptations to give into inpatient customers. Security sometimes is a thankless job, but it is a job which we as Americans must held to the highest standards. Hence, before you permit customers to legally operate their information systems, you must relay all the warnings of what is known and what is unknown about the situation/system. Only then can the customer make an educated decision on how to proceed forward: go with it or not. For those who read the introduction, there is a time and place for accepting risk. Today the military and USG fights hard each day to achieve mission success in a continual degrading electronic warfare and cyber environment. The official stance today, as backed by the RAND corporation, is that we need to have both adaptive and dynamic countermeasures for real-time cyber resiliency across the entire lifecycle of the mission (Snyder, 2020). This means that as mission managers, we accept certain risks regarding cyberspace and electronic warfare that are out of our control. Integrating these considerations is extremely vital to the survivability of weapon systems but should at the same time not be an escape clause for failure to address security principles.

With acceptance being the very last option, we should identify our security vulnerabilities and come up with solutions or mitigations. These mitigations should be prioritized based on what vulnerabilities we deemed as significant in our risk acceptability matrix. Assuming we did our prioritization, we can observe vulnerabilities that are deemed critical to our business survivability and determine what mitigation strategies are at our disposal. It is important to know that not all mitigation strategies are worth doing. I repeat, not all mitigation strategies are feasible. Either they may cost too much, the solution requires multiple parties to consent, there may not be enough time, etc. When considering mitigation strategies, make sure to incorporate all stakeholders that would be affected by the decision. I can’t tell you how many times I have seen a decision being made without key stakeholder consideration, only to have that solution rejected as infeasible by a third-party agent months down the road. Moving forward, if the mitigation is acceptable and able to be done at a reasonable cost then we can move on. If not then we have three more options: avoid the risk, transfer the risk, or as a last option, accept the risk. The last one we have discussed, so let us now dive into avoiding or transferring the risk.

For discussion’s sake, let us consider a threat mitigation to cost too much. So much so, that the solution is considered infeasible, which is quite common across large enterprises. That gives the organization a few COAs through risk avoidance, stop the service or develop a countermeasure that leverage another system path or service. The first instance stops the service, process, or functionality completely if the mitigation is found to be too costly. On the flip side, ingenuity can prevail if the company allots the opportunity for the team’s systems engineers to take a crack at the problem. General Patton used to say, give people a task and let them solve how the order should be fulfilled. What General Patton was saying is that people surprise the heck out of you if you let them, so if permitted the time and resources, try to think of various ways to accomplish this threat mitigation.

Finally, transferring risk is a normal process that happens when an organization is unable to implement the mitigation. Sometimes this also occurs when the company cannot implement the mitigation themselves for cheaper than a competitor or supporting company. Don’t be afraid to admit that. You are not admitting defeat or showing weakness, instead you have the strength and bravery to put organizational needs above your own ego.

Now is a perfect opportunity to go through a simulated example. We have a large corporation hosting sites for publicly available, sensitive information, let’s say for the 2021 Virginia Elections (Virginia, 2021). Even though that information is publicly available, and non-classified, the state of Virginia needs to assure that all citizens always have access to this site. Right before the major elections day, a nation state group from the far east decided to hire non-nation state actors to conduct a Distributed Denial of Service (DDoS) attack against the company hosting these servers. As this was occurring, the hosting company had a few options. A few factors affect their decision-making process, first they are required by contact to maintain 100 percent availability, they are required to use secured/validated servers following FIPS 140-2 only, and they must stay within a budget. Being that this was a small-business, veteran-owned cloud hosting company, they could not mitigate the risk because they did not have the elasticity to provision more servers. By contract, they could not avoid or accept the risk. Also, they were under strict agreement to find servers with similar FIPS 140-2 security profiles. Eventually the small-business reached out to a rival small-business cloud hosting company and signed a temporary Service Level Agreement (SLA) to alleviate the DDoS attack. This is one of many examples that you will see throughout this book. Applying risk decision states across technical operations help facilitate knowledge, understanding, and transparency across the enterprise’s security operations.

Whether you are wearing a badge, a military uniform, or in the commercial sector, there are a lot of procedures and methodologies set in place for cyber first responders. There are actually so many different ways to perform triage that there are consulting agencies that specialize in Business Impact Analysis, Disaster Recovery and Business Continuity Planning (BCP). The ISC2 uses terms such as digital forensics, electronic data discovery, and cyberforensics to associate incident handling, with methodologies guided by frameworks to include the Scientific Working Group on Digital Evidence (SWGDE), Motive, Opportunity, and Means (MOM), and NIST's Special Publication 800-61 (Harris & Maymi, 2019). Looking for Indicators of Compromise (IOCs), containing them, remediating the attack, and delivering lessons learned is formalized by the EC-Council through the following steps: preparation for incident response, incident discovery, incident recording and assignment, incident triage, incident containment, eradication, recovery, and post-incident activities.

During the initial phase of incident handling, the following items are critical for success: planning, recording, and assignment. There is a high correlation between planning and successful execution. Rigid, formalized security procedures help individuals know what to do, when to do them, and whom to report to. Within BCP, it is crucial to deliver proper employee expectation and responsibilities to ensure safety of personnel. The most important aspect of BCP is human life, identifying the systems, allotting the appropriate force/response, setting up the remediation controls, and ensuring that the infrastructure can support these actions is more than half the battle. Once the organization’s vision is mapped out, execution comes second nature with monthly fire drills. The second most important step within BCP, after planning, is recording and assignment. Identifying the correct information, that a cyber breach has occurred, and notifying the proper chain-of-command, in a timely manner, cannot be understated. If information does not get to the right person then vital decision making cannot take place. After all of these initial processes are achieved, the Certified Network Defender (CND) can begin the incident triage stage, which is split into three components: incident analysis and validation, classification, and prioritization. The incident triage stage is the implementation of the BCP policies set in place and should be second nature due to continuous security training set forth by the organization. Key Stakeholders

There is no secret sauce for who your key stakeholders are, they could be your investors, your C-suite members, your board of trustees, your mission managers, your customers, and the list goes on. In 2016, NIST identified their key stakeholders as other government agencies, national cybersecurity excellence partners, integrators, vendors, reference design users, and information technology users (NIST, 2016). As you could probably glean, an organization can identify their key stakeholders as anyone they desire. Practically, the list of key stakeholders will only be a handful or two of people/organizations. The size is dependent on the revenue and influence of the organization, the scope of the project, the impact of the solution, who is the intended target, etc. Whenever you get involved in a project, or cybersecurity operation, understand the following items: who has equity, what resources are required, who's going to maintain this, what impact will it have on the organization, and finally, who is the intended target. If you can pinpoint, address, and account for all those items then your project has a high likelihood of succeeding. For projects that have a long-term implementation, such as the B-52, which was operational for over 65 years, you may need to reidentify and validate those key stakeholders. You may also need to reidentify and validate those key stakeholders if there is a change in the requirements or operational deployment. Without input from these key stakeholders, your efforts may be for waste because you are not capturing the right RFIs, or requirements. We will cover those in the next topic.

Requirements

Request for Information (RFI) permits those from the government information on the cyber capability or technological solution from the industry. Capitalizing on industry human resources, the government can get time to market faster than if they used in-house technology experts. By identifying known information, research trends, and intelligence gaps, cybersecurity programs can narrow down what’s important to the project, what can be done, what a reasonable timeframe would be, and much more. After soliciting for RFIs on capabilities or technologies, the cybersecurity program may decide to produce a Request for Proposal (RFP). This is taking the RFI process one step further by asking what the industry can offer as a solution. Sometimes this can be a small add on or fix, or a completely new program. Once the RFP process has completed, determining source selection, project milestones, deliverables, and much more would be next. Basically, we take RFIs to inform us of what is out there, then we use RFPs to ask for what can be provided.

Business Continuity

It is easy to look at an organization when it is performing day-to-day business operations and conclude that it is a well-oiled machine, operating at peak efficiency. However, like the age- old colloquialism that a diamond is made by pressure, you cannot observe how elastic a company is until it experiences a business interruption. Having a well-defined, senior leader approved plan of action in place is critical for understanding and executing the mission in times of duress. The industry leaders in Business Continuity Planning (BCP) routinely define, and redefine, policies, set tabletop exercises, perform simulated events, and have full-scale disaster interruptions to test the respond of their organization. To implement this process, it starts with the BCP team. According to the CISSP, members of the team can be from various departments but must account for the following six critical business processes: required roles, required resources, input and output mechanisms, workflow steps, required time for completion, and interfaces with other processes (Harris & Maymi, 2019). It is essential to review and analyze the results against corporate objectives.

This book simplifies BCP by explaining that it is a way to protect critical assets and personnel by being resilient against threats. The categories of threats that need to be accounted for vary by the organization's structure, however, there is one intersection among all organizations: the people. The employees are the most important asset of any business and need to be thoroughly accounted for in all Business Impact Analysts (BIA) and BCPs. There are six goals listed for BCP, with the most important of these being the prioritization and welfare of the organization and its staff. The other five are the identification of potential risks and losses, active use of the risk management process, remediation of the threat for operations continuity, good record keeping, and enabling disaster readiness through adequate disaster preparedness. To reiterate, all of these goals can be achieved through proper, senior leader approved corporate policy. Setting personnel first, capturing all the enterprise resources, and maintaining adequate response times through continuous training is achieved through the guidance set forth by the BCP team.

Enterprise Risk Management

Below are the goals that the ERM framework sets out to perform and accomplish. Leveraging these goals will help ensure that the enterprise follows a defined, continuous security strategy that will protect enterprise assets. If the below are accomplished thoroughly, a better Capability Maturity Model Integration (CMMI) can be achieved. Contributing to organizational assets and building from policy feedback can help the company sustain an acceptable level 3 rating; while adding optimization with a focus on continuous security monitoring and improvement can give the organization a maturity level 5 rating (ISACA, 2021).

1. Bake-in the organization’s performance management with the ERM framework

2. Translate the improvements of using a risk management framework

3. Establish roles and responsibilities that the need to manage the organization’s risks

4. Prepare written policy and guide lines on the chain-of-command’s risk-reporting procedures

5. Provide support in the vulnerable, threat-centric areas of the enterprise

6. Ensure continual audits of security controls, as defined by the corporation and regulation of your industry

7. Deliver senior-leader guidance on policies, approaches, and the risk appetite of the enterprise

8. Hold departments and security professionals accountable to their roles and responsibilities

Mission Assurance

Mission assurance, based on military mission acceptance in the face of hostility, has bred an environment for Information Assurance (IA) in the industry. A relatively new term, IA is making headway by setting standards of protection and defense of information and information systems despite hostile intent; IA ensures that information and information systems are secured and hardened using confidentiality, integrity, availability (CIA), authentication, and non-repudiation principles (NIST, 2021). Through the lifecycle of protecting, detecting, and reacting, we can achieve IA, which will inherently facilitate mission assurance (NIST, 2021). After understanding each of these components, we will be thoroughly examining in the reminder of this book how each situation applies to these concepts. Furthermore, understanding the basic tenants of security, whether that is cloud security or local security, helps drive decisions on the type of deployment, whether this is on-premises or hosted by a third party, and the security controls revolving around the technologies and solutions employed. It is necessary to understand these basic security tenants to defeat and defend future penetration attacks against the enterprise. Leveraging standard frameworks and governance documents can help C-suite members adequately prepare their cyber professionals for normal day-to-day operations and for all types of incidents. Facilitating effective communication, through policy, is the first step in helping the network defender against data breaches. Leveraging Information Assurance (IA) principles are becoming more and more necessary to maintain to defeat sophisticated cyber threat agents and nation-state actors.

For network defenders to be equipped to defend the homeland, they need to first be enabled to do so. Adequate resources, doctrinal support, and training must be in place if the enterprise is to be well protected. Starting this cycle off right requires sign off from C-suite members; whether your organization is private, public, or a federal agency, having the proper documentation in place cannot be understated. The best way to ensure that the network defenders can do their jobs properly is to take a top-down security management approach, where the C-suite members, probably the Chief Information Officer (CIO), approves and champions security conscious policies (Chapple & Stewart & Gibson, 2018). These policies include a Business Impact Analysis (BIA), Business Continuity Plan (BCP), Disaster Recovery Plan (DRP), Standard Operating Procedures (SOPs), an approved chain-of-command list, and much more. When senior leaders create expected behavior, through guidance frameworks, procedure lists, and acceptable user policies (AUP), the organization is more cyber resilient to not only perform their day-to-day operations, but also to protect themselves against threat agents. Failing to have these policies and directives in place are disastrous. Cyber defenders more than likely followed some sort of methodology incident response: discovery of the incident through various Indicators of Compromise (IOCs), incident recording and assignment, triage and remediation, business recovery, and post-incident activities like lessons learned. There are numerous frameworks to use when basing your corporate response, some of these include Scientific Working Group on Digital Evidence (SWGDE), Motive, Opportunity, and Means (MOM), and NIST’s Special Publication 800-61 (Harris & Maymi, 2019). We will cover this more later.

Information assurance through the identification of the appropriate threats and vulnerabilities, the proper network security controls, and proper data protection through cryptographic means has always been important. Security of the network has been such a high priority that half of the No Such Agency’s (NSA) archives relate to mitigation and protection, or National Security Agency for those not as familiar with its history. Their organization has been such a positive influence on the cybersecurity community for decades. Since 2005 they have made a significant contribution and positive image at the RSA conference (NSA, 2005). One important document in their archives relating to the security of cloud enterprises is their Cloud Security Basics, which explains how to set the stage for adequate information assurance principles in your environment. Managing enterprise risk can be achieved by employing security controls for the following concepts: access control, cloud patching, multi-tenancy, encryption, proper cloud security services, reliability and denial of service, and data spillage (NSA, 2018). Recommended security implementations of NSA suggestions can be seen through NIST’s Special Publication (SP) 800-53, or better known as the Security and Privacy Controls for Federal Information Systems and Organizations (NIST, 2013). Each of the recommended security controls by NIST SP 800-53 is still based on the inherit enterprise risk policies set by C-suite members. Usual security controls seen are data encryption with the Advanced Encryption Standard (AES), integrity through the Secure Hashing Algorithm (SHA), proper key escrow and key exchange protocols like the Elliptic Curve Cryptography (ECC) Diffie-Hellman key exchange (Barker et al, 2018). Each of these cryptographic means of security establishes information assurance through data at rest and data in transit by achieving different objectives associates with the Confidentiality, Integrity, and Availability triad.

Since the inception at ARPAnet, network technology has continuously improved and changed the enterprise model. There are too many different networking devices to consider within this paper, as most serve a special function. There was star- based topologies working with token rings, framed relays working with dedicated, permanent virtual circuits (PVCs), layer 3 switches combining the functionality of routers and switches, and more (Harris & Maymi, 2019). Moving from cloud networking technologies like storage area networks (SANs), load balancers, or web application firewalls (WAFs), it is relevant to consider the models that describe how a network device functions and which layers are affected. Today it is common to use the seven layers of the Open Systems Interconnection (OSI) model: application, presentation, session, transport, network, data link, physical. However, there is another that was created to serve the purpose of the military and ARPAnet: the TCP/IP or DoD model. There is a bridge between the OSI and TCP/IP model, but over 95 percent of people do not use the TCP/IP model anymore. Therefore, understanding how your cloud enterprise's networking technology fits into these models can help you achieve secure networking and API integration. One example is Cisco's Network-based Application Recognition (NBAR) engine. This intrusion prevention system (IPS) capability within the Fire Power services on Adaptive Security Appliances (ASA) cover layers three through seven to identify and regulate layer three and four network traffic from layer seven applications. Finally, when integrating all of your network capabilities across the enterprise, it is important to understand which cloud service models you are providing to your end-users: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). Leveraging knowledge of what your organization has, what they will need in the future, and the cost associated with processing elasticity, will provide your IT department the direction, whether they are looking at a sole private cloud, a public, or a community deployment. Based on the company's responsibilities, regulations may eliminate multitenancy options and dictate single tenancy solutions.

These recommendations were derived from NSA’s cloud security basics IA directive. Actual implementation best practices from NSA’s higher leadership directions came from NIST’s SP 800-53. Within that three-page gold mine, the following topics were covered: shared responsibility, having a threat model, federal and DoD requirements, managing risk, access control, cloud patching, encryption, multi-tenancy, reliability and denial of service, and data spillage (NSA, 2018). Standards set forth were holding the CSP to security standards and policies of the company, understanding who the enemy is and their TTPs, ensuring security are implemented according to industry and government regulations, having CND members follow the risk appetite of the enterprise, proper data controls through encryption and accountability, and finally, making sure to patch and update production networks (NSA, 2018).

In summary, CNDs defend the corporate resources against penetration attacks but require C-suite support to maximize their security posture through sanctioned policies and procedures. Leveraging these guidelines and frameworks help identify security issues relating to information assurance, helps the CND in the supportability of network devices and Application Programming Interfaces (API), and ensures that the right information and cryptographic bit-level strength is applied. Security should be at the forefront of every CND’s mindset, preventing breaches and attacks from happening at the source of the problem. If a more proactive security posture is done, by way of NSA’s Cloud Security Basics principles and NIST’s SP 800-53 implementations, data breaches would occur less frequently. It is our job as CND, and as cybersecurity professionals, to adopt these teachings to better equip our organizations against threat agents.

Having situational awareness and being knowledgeable on the types of investigations (Administrative, Criminal, Civil, and Regulatory), how legal proceedings accept electronic discovery, to include the types of evidence (real, documentary, best evidence rule, and parole evidence rule), how such evidence can be admissible in court, the chain of command, and the accepted evidence collection and forensic procedures (Chapple & Stewart & Gibson, 2018). To even understand the legality of what is admissible or not, depending on the circumstances and collection of that information, requires legal solicitation. It is not advisable for cyber first responders, nor managers, to navigate the legalities of the letter of the law; the procedures are cleared and set forth by the Board. Over the last few years there has been a trend to hold the CEO more accountable for federally mandated regulatory information. The NSA, Cybersecurity and Infrastructure Security Agency (CISA), and more have been significantly modifying and adding regulations around how organizations handle data breaches associated with critical infrastructure-related systems (Cyber Hedge, 2020). What this new trend has been saying is that the first-line supervisor is not responsible for the procedures set forth by the Chief Information Officer, they are only responsible for implementing those procedures. The CIO and CEO are being held accountable for proper forensic investigation procedures when a data breach occurs.

This book expresses that a typical forensics investigation methodology has nine steps: obtaining the search warrant, evaluating, and securing the scene, collecting the evidence, securing the evidence, acquiring the data, analyzing the data, preparing the final report, and testifying as an expert witness. Once again, there are whole books written on each of those subjects. Perceived unethical collection, that violates our Constitutional privacy, fuels bad perception against federal organizations (such as the NSA). Evaluating and securing the scene can take many forms and may even be unnecessary if the cyber group with Title 10 orders is already in their networks. The NSA is also very cognizant of legal collection of evidence against foreign and US persons, through USSID SPO018 and others (NSA, 2011). Securing the information and acquiring the data can either be easy, as the data is in-house, or could be quite challenging, like when 1000 programmers touched Solar Winds. Preparing the report with enough technical details, but not too much, is pivotal for higher leadership to understand and act on that information. Testifying on behalf of a data breach may never happen, but the requirement to be ready when it does is real.

Threat Intelligence

Also, this book suggested three broad encompassing types of intelligence: strategic, tactical, and operational. Strategic threat intelligence is the highest-level awareness and information regarding threat details, cyber hygiene, and Business Impact Analysis (BIA) concerns. Tactical is described as the enterprise-level information that can help mitigate cyber threat agents on a large scale. Finally, operational threat intelligence provides Cyber Network Defenders (CND) with the knowledge of specific Techniques, Tactics, and Procedure (TTPs) against specific Advanced Persistent Threats (APTs). To understand the differences, you only need to know the following political analogy. Capitol Hill issues are strategic, states lines are tactical, and local districts are operational. One more analogy used in the military is with the warfighter. If the information is kicked back stateside it is strategic, if the information is used in the deployed theater it is operational, and if the information is used at the deployed location then it is tactical.

After diving into the types of intelligence used, strategic versus tactical versus operational, the way the information was collected is next. There are numerous ways that threat intelligence can be produced, but the book lists the main categories as the following: internal intelligence, open-source intelligence (OSINT), counterintelligence, human intelligence (HUMINT). Using google hacking techniques can deliver great reconnaissance and cyber footprinting of an attacker, often giving indications on where to perform another category of intelligence collection. However, there were two types of intelligence collection missing from our textbook that are applicable in defeating APTs in the cyber realm: signals intelligence (SIGINT) and measurement and signature intelligence (MASINT). Both the NSA and NRO would be disappointed to be left out as main contributors within the intelligence community, as they are equally as important as all the other 14 intelligence agencies. In any case, to defeat highly tailored APTs SIGINT, or collection of foreign intelligence from communication systems, guides the company's TTPs of an APT (NSA, 2021). SIGINT is great in combating exotic threats against communications equipment, maybe like an antenna. Finally, MASINT, or scientific and technical intelligence from technical sensors, should not be lost as a viable way to defend the enterprise from a cyber-attack (Pike, 2000). One example is the NASA breach that gave cyber attackers NASA's Deep Space Network (DSN) specifications (Cimpanu, 2019). Having these large, phased array specifications could allow for tailored exploits.

IA Tenets

Confidentiality covers how you keep information secret and has shown to be problematic throughout history, hence leading to intense study on the subject. Over the course of time, confidentiality has branched into two main mathematical categories: symmetric versus asymmetric systems. The former uses block and stream ciphers like DES, AES, and RC4 as implementations of symmetric ciphers. The importance of symmetric ciphers is that key is known by both parties ahead of time and provides for more efficient algorithms for encryption. Unfortunately, the speed of symmetric algorithms is met with the inability to distribute keys in a reasonable fashion. With symmetric encryption, you would have to transmit private keys to every channel you want set up: (n*(n-1))/2. Fortunately, asymmetric systems, like RSA or ECC, solved this problem by setting up public and private keys. Public keys are stored on a central database, while private keys remain with the end user. To encrypt information to someone else, you would use their centrally available public key. If, however, you wanted to digitally sign something for integrity and non-repudiation you would use your private key. Although this seems like the best of both worlds, asymmetric systems are slow and require lots of space/more bits. So, protocols like SSL and TLS set up channels with asymmetric ciphers and then use symmetric ciphers for session keys. Using that approach truly gives you the best of both worlds.

Historically, besides perhaps certain cellular devices, data in use was virtually non-exist. Even today if your company uses encryption in use your organization is a rarity. In our text, it uses a pseudo-data in use encryption example with oracle databases, but this was still data in transit and data at rest. True in use data encryption is one of the hardest problems in cryptography because the data must remain encrypted throughout the entire process, in other words the data remains encrypted from storage to use to transit to use to storage. Even in 2016, Microsoft's cryptonet (possibly an early representation of SEAL) was only able to provide 51,000 estimates per hour (Thomson, 2016). It was only last year that data in use may start being a Federal Information Processing 140-2 standard. This was accomplished with IBM's fast, opensource HELib toolkits (Bergamaschi & Daniel & Levy, 2020). Extending the end-to-end encryption notion to client devices and databases is a necessity moving into a more hostile threat environment.

Encrypting Data at rest has been the most famous example, with national competitions ranging from the 1999 American Standard to the 2016 Post-Quantum Algorithm NIST competition. There was no shortage of competitors, all the way from Tokyo to Britain. Even the RSA conference brings out some good designs for data at rest. Understanding that algorithms change is a requirement in the crypto business. For example, DES was great in 1970 but terrible in the 1990s. AES was unbeatable for 20 years, but NIST/Maryland experts are looking for something past k-ary C* schemes, with schemes like Supersingular Isogeny Key Encapsulation (Smith-Tone, 2019). Basically, whenever corporate data is stored somewhere it should be encrypted. Using the right algorithm's bit strength complexity and time use cost depends on the application and needs of the organization.

Often touted in vendor products are data in transit encryption. Network defenders should be extremely sensitive to Internet Protocol Security (IPsec) for tunnel endpoint encryption. Setting up tunnels from the enterprise to the branch offices are never enough. Tunnels do not provide encryption themselves. Once again there has to be a broker, like IPsec, that manages an encrypted pipe over a tunnel because all a tunnel does is create a dedicated virtual path from one site to another. Utilizing Generic Routing Encapsulation (GRE) with Elliptic Curve Diffie Hellman (ECDH) is pivotal for secured communications. As data is moving from one place to another, encrypting that process can be challenging. For years WiFi Protected Access (WPA) failed to provide this with their faulty Transient Key Integrity Protocol (TKIP) implementation.

Integrity is keeping the information the same regardless of where it was stored, transmitted, or used for. Using integrity, we can cryptographically validate that the information has not changed. Some of the famous hashing algorithms include MD5, SHA, and Tiger. Based on the bit-strength required to break these, assuming known mathematical vulnerabilities in the algorithm, we can adequately provide security profile protection for the desired level of protection we want. For instance, whether I am sending financial account activities or nuclear launch codes over the wire, we want to make sure that the information sent has not changed. Inability to verify incoming information can be quite disastrous.

There is no known cryptographic method to validate availability yet. In the industry, we use queuing algorithms, load balancing systems, and service level agreements (SLA) to help provide availability to the customer.

Non-repudiation is the act of attesting that you put your name behind some activity. This means that that person or service is unable to deny that they performed this activity later. Using asymmetric systems, like RSA or ECC, you can leverage digital signatures to provide non-repudiation.

IA Case Study

One real world example of why it is important to set up proper network policies can be seen through the Anthem data breach. The United States found that Anthem’s 2015 data breach resulted from poor security policies, and if set up before the incident there might not have been a breach (Dhillon, 2016). They were not able to adequately provide confidentiality. Using this event as an example has motivated corporations to put security first, instead of trying to add it on later. For Anthem, once the data breach occurred their first responders went into full incident response mode.

Health of the Mission Manager

Transitioning to another related subject, how cybersecurity responsibility affects the perception of mission assurance. There is an old saying that says that everything smells like roses when things work, but when they don't they go bad very quickly. Looking at reality versus theory, understanding how important it is for a cybersecurity mission manager's mindset on the overall team's morale contributes to mission assurance. Few will go into this area because it humanizes the cyber operation process but helps gather better indications and warning for the team. Hence, according to Infosecurity Magazine, 51 percent of cybersecurity professionals do not sleep well because of the stress that feel from their job (Muncaster, 2021). Isn't it interesting that the appearance of control stresses managers out? As a manager, whether in cybersecurity or construction, you should always do the best you can do and let the chips fall. You should not take it personal if you follow all the STIGs and could not stop the attackers. Learn from it and do better next time. Without being too philosophical, life is too short to worry about situations or activities that are out of your control. For a more pragmatic interpretation, the mental disposition of the cyber operations manage has limitless tangible affect on your subordinates. Keep that glass half full attitude that STIGs and formal guidance documentation will protect your enterprise. The act of believing in the process will bear fruit on those within your cybersecurity cell, and ultimately transfer to complete mission assurance.

Threats Agents and Vulnerabilities

Laying out the threat landscape is important in today's cyber age. With threat intelligence you can identify and remediate threat agents before they accomplish their goals. Failure to fuse threat intelligence into cyber operations leaves you vulnerable to attacks against your enterprise. Here we will walk through threat categories, threat agents, and how to map threat agent tactics, techniques, and procedures (TTPs) with blue side mitigations.

Threat categories

One of the threat categories listed above was threat agents. These agents vary in sophistication, motive, and end objectives. The five threat agents to be concerned about, from least sophisticated to most, are script kiddies, hacktivists, insider threats, organized criminal syndicates, and nation states or APTs (Sentinel One, 2019). Script kiddies are nearly harmless and rely on others to create their exploits so that they can blindly run them. Hacktivists go a step further and hack for a cause. Whatever political objective they have, they can leverage common vulnerabilities to attack their target. This can come in the form of defacing public facing websites, hijacking social media accounts, or conducting a distributed denial of service (DDoS) attack. Contrary to popular belief, hacktivists are not as sophisticated as you may believe. What about that terrorist group anonymous or shadow brokers? Those fall more under the activities of malicious insiders coerced by nation state actors. Instead of thinking of them as hacktivists, they are financially motivated insider individual and groups backed by nation state resources. Moving on you have criminal syndicates that have commercialized the exploitation of cyber space. They have significant financial motivation and have adequate resources at hand. These organizations structure mimic those of legitimate corporations. Lastly, nation state actors have almost unlimited power, influence, and resources to accomplish their mission. Their motivation ranges from financial to technological espionage. In future books, we will identify and demonstrate TTPs, and campaigns conducted by APTs. MITRE has provided great resources on the subject to enable the ability to categorize and map APT TTPs within their ATT&CK framework. Once again, this will be shown in a later book.

Attack Surface

An organization’s attack surface can be quite extensive and require a dedicated workforce or contracting agency to monitor. The primary attack surfaces are devices and people, and can dramatically affect small businesses each year, which make up 43% of the targets (Avast, 2021). It has been noted that small businesses are targeted often because they do not have the workforce and expertise necessary to defeat more advanced threats. Within our text, the attack surface is described as the sum of all vulnerabilities associated with the enterprise, including known and unknown. Knowing your vulnerabilities can be quite daunting, but the larger the enterprise, the more variables to consider.

It is not unreasonable to state that as an enterprise’s span of influence and size increases, there is a larger uncertainty about the threat vectors that will threaten business activities. Categorizing threats and performing a rigid methodology is the only hope with which the company expects to defeat them. Attack surfaces can be categorized into five areas: network, software, physical, human, and system. Certified Network Defenders (CND) should continuously leverage these categories by understanding their enterprise attack surfaces, identifying the Indicators of Exposures (IoEs), walking through how the attack would occur, and then reducing the potential attack surface. Reducing the attack surfaces can promote a healthy cyber hygiene for the company, thus limiting successful attacks. One example could be scanning your enterprise, identifying that the edge point router is not hardened, fixing this problem with better encryption, and logically shutting down unused physical ports. After performing these actions, the attack surface is reduced.

Penetration Testing Lifecycle

Having a good penetration testing methodology in place will help provide a standardized way to approach new organizations, networks, and systems. Beware of marketing campaigns because there are more penetration testing methodologies than you can shake a stick. Most of the ones on the market follow very similar plans of action, so let us use an industry standard: Lockheed Martin’s Cyber Kill Chain. Within this Cyber Kill Chain, there are seven phases: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives (Lockheed Martin, 2021). Reconnaissance is the first phase because it's purpose is to collect open-source intelligence (OSINT) information passively or actively about an organization or target. Passive OSINT entails gathering information on targets from publicly available information, like Google Dorking or LinkedIn. Active OSINT requires you to interact with the target systems in some form or fashion, such as port scanning. After gathering enough OSINT, whether passive, active, or a combination of both, you should be able to statistical identify an attack vector. Go ahead and create a payload, or weaponize an exploit, tailored to that attack vector. Find a way to deliver that weaponized data to the target to exploit the target system, account, or protocol of interest. After exploiting, make sure to install hooks through a rootkit to maintain continued access. This assumes you want to continue performing intelligence collection operations through command-and-control centralization. The final step is vague as it relates to actions on objectives, or what you want to accomplish on the command line in support of your original goals (Lockheed Martin, 2021). Actions here may different between APTs, and even within the APT.

Patch Management

Keeping up to date with in-house, Software Development (Dev) and IT Operations (Ops) called DevOps, or national databases, like the National Vulnerability Database is essential to thwart cyber actors. Just as happens in computer firmware, a cyber actor leverage of vulnerabilities falls into a Time of Check Time of Use (TOCTOU) scenario. If the DevOps' team falls short on preparedness and response time on checking and implementing the vulnerability patch, then the threat agents could gain access to the system, account, or data. The book gives the definition for application patch management to be the monitoring of application security on end nodes by way of routine maintenance, or patches. Important processes of an application patch management solution are to scan the network, flag vulnerable systems and applications, then automate the respond.

Automation versus manually delivering patches is almost an irrelevant conversation in today's environment. Assuming your enterprise is larger than three devices and a router, manually testing and deploying patches takes too much time between TOCTOU of vulnerability exploitation. Thus, automating IT operations has become the standard and absorbs more of an IT professional's life each day. CISSP recommends that large companies perform a blue versus green deployment of patches to mitigate any adverse side effects patches may have on production networks. A DevOps manager may require that the software and network engineers test on the blue deployment before rolling over to the active blue production network (Bigelow, 2016). For example, a Customer Relation Management (CRM) solution may be incapable of working with the latest and greatest security fix. The company may require the CRM software be run on a virtual machine to interact appropriately with the rest of the network. A lot of headaches could be mitigated if the company has a standing policy to test on a green network first before implementing on a blue network. Once the patch is determined to be safe, the engineers could allow the Microsoft Windows Server Update Services (WSUS) to make changes to the blue production network.

Cloud Computing

It was 10 years ago that NIST came out with their final version of what cloud computing was, the five characteristics, the three service models, and the components that facilitates proper inter-operation (NIST, 2021). Government specifications usually take a few years, and it was referenced that NIST started in 2009 and took 15 drafts to get to the final stage (NIST, 2011). We should go through the basics to consider how we can extend the original cloud computing standard. The Industry and NIST view 5G, smart city virtualization, and edge intelligence to be next generation technologies today, essentially extending the five essential characteristics may include at a minimum these three concepts.

The original NIST standard called for the following features: measured service, rapid elasticity, network access, resource pooling, and on-demand self-service (NIST, 2021). Our textbook extended these five characteristics and included distributed storage, automated management, and virtualization technology. A further extension of the cloud computing features may, in the future, incorporate and differentiate Virtual Network Functions (VNF) and Network Function Virtualization Infrastructure (NFVI). Even these are general simplifications of what is in the market, and the trends moving forward. For example, UK has looked at how the 5G underlay (VNFs and NFVIs depending on the deployment) drives smart city cloud computing, or NFVI for large geographic areas (Buni, 2019). The reason 5G and smart city cloud computing could be cataloged in VNF versus NFVI depending on the deployment is because of the four models of communication we talked about two weeks ago for IoT communication: Device-to-Device, Device-to-Cloud, Device-to-Gateway, and Cloud-to-Cloud. There is not a one size fits all description for how smart city virtualization works, and is instead a combination of all the virtualization that summarize to the whole. Cisco has been working diligently on virtualizing the entire smart city deployment model into one unified platform, but such is not common.

To complicate matters more, when the business needs to consider the cost of operations and licenses, security of data, regulation requirements of data accountability and privacy, and much more. These needs drive organizations to choose a combination of service models, Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service, and deployment models, private cloud, community cloud, public cloud, and hybrid cloud. An organization’s decision on which service model or deployment model to use is derived from their policies and guidelines set in place. Some enterprises are early adopters which entails more risk or are required by law to store the data in a particular country. Even with these considerations, cloud computing benefits to an organization are; fewer staffing requirements, less investment in security, reduced capital expenditure through increased operational expenditure, and the ability to meet customer demands through cloud bursting.

The types of cloud service modules seem well fixated and have held over the years. Cloud service providers have stuck true to the definitions of Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) and use them in marketing religiously. However, with the boom of IoT devices and open source projects for Radio Access Networks (RAN) will these definitions remain true? Almost certainly not. In the next couple of years NIST needs to revamp what IaaS means, and more specifically, what is underneath IaaS. Currently around the world, a good percentage of the world permits data sharing for network traffic among Service Providers (SPs). Without going into the somewhat complicated roaming agreements between large US carriers, there is a good deal of sharing among the US wireless industry (Dano, 2019). Facebook even promotes data sharing at the network traffic layer among SPs in different countries: look at Peru (Dano, 2019). Also, most of Europe shares similar data agreements on shared equipment on RANs (Dano, 2019). US funded dollars, led by DARPA, are going towards building Google's Anthos platform for Open RAN virtualization, which, subsequently is being shared with Finnish's Nokia's 5G operations (Google, 2021). The point has been made, RAN-as-a-Service (RaaS) is prevalent and does not appear to affect data privacy rights in today's environment. To throw a wrench in the mix, smart city interconnection points should eventually be passing data back to the IoT layer, in the city, without ever having to get backhauled to the data center. Therefore, if in the future edge intelligence is shared and transmitted as multicast traffic across multi-vendor applications, then IaaS does not describe that activity (Gloukhovtsev, 2020). Essentially, NIST needs to remap the cloud service modules as we move towards true 5G edge intelligence in smart cities. Sometimes NIST comes up with mobile terminology first, sometimes it is the 3GPP. The session breakout standards being initialized in the 3GPP exemplifies RaaS being a lower level cloud service module compared with IaaS (Suskovics et al, 2020).

Recently, a company from Salt Lake City, Venafi, an adamant proponent of accountability, has proposed Machine Identity Management. Their solution is not unheard of but is not even required material in the CISSP. Their principle argument is to make sure that all devices can perform attestation with valid certificate management lists, especially in the cloud (Venafi, 2021). Apparently, lack of verified hardware certificates plague the cloud computing environment, with self-signed certificates being the most disastrous (Howe, 2019). As we rely more on Cloud Service Providers and Cloud Security Brokers, establishing the verified, trusted protocols for Machine Identity Management is becoming more and more critical.

NFV

Before the revamp of the CCNA and CCNP two years ago, NFV was not a well-studied topic. I went through CASP, Sec+, CEH, and CCNA before ever crossing paths with Software Defined Networks (SDNs), let alone anything that closely resembled NFV. The progression of NFV at Cisco before the revamp and prioritization of DevOps were TCAM modifications for switches, then stackwise software, then L3 switches, then Policy-based Routing and the Application Centric Infrastructure (ACI) design. There are almost certainly missing parts to the NFV evolution at Cisco, but that was just a few examples demonstrating that NFV has taken time to get here and has come a long way. There are numerous aspects and architectural designs when considering NFV, but the three basic components that build all these designs is the NFV Infrastructure (NFVI), Virtualized Network Functions (VNFs), and Network Management Orchestration (MANO).

Instead of focusing on very specific virtualization models and software, we will be highlighting just the main variances among the unification of network resources under a single architecture. While the obvious benefits of scalability, speed, and logical administration are clear, security segmentation is not. Just as logical segmentation aided internal networks with port security, through community and private ports (both being leveraged by promiscuous ports), NFVI components have facilitated the next layer of traffic segmentation. Security and functionality can be designed ahead of time by choosing where and when to incorporate a given tool. Two categories to choose from when designing a NFVI is the difference between External Virtual Networks, or the combination of LANs, and Internal Virtual Networks, or all on the same system. It is critical to address the subparts of the NFVI: hardware resources, the virtualization layer, and all virtual resources going to be used. The book gives an example on how to use a type-1 hypervisor on an internal network design. While Cisco demonstrates extending external virtual networks through their Overlay Transport Virtualization (OTV) VNF software capability. Both key elements of VNF, VNF and Element Management System (EMS), is seen through OTV extensions of L2 broadcast domains past the router, and over the Internet, to other campuses (Cisco, 2021). The OTV is the glue that holds together a single L2 broadcast domain over an L3 underlay, perhaps through the nearest vSmart controller.

Lastly, the most complicated, and often proprietary, implementation of NFVs are MANO. The three components MANO uses are Virtualized Infrastructure Manager (VIM), VNF manager, and an orchestrator. The VNF manager takes care of the life cycle of the VNF, while the orchestrator deals with NFVI and software resources. One example of a MANO architectural design and implementation is Cisco’s Enterprise Network Functions Virtualization (ENFV). The ENFV utilizes a Linux-based x86 platform to unify all VNFs onto a single platform, examples include load balancers, firewalls, and more (Cisco, 2020).

Logging

Logging

Understanding the nuisances pertaining to failures were a subject of interest as well during their background literature scan. Two issues that popped up was that 60% of failures from software faults did not leave evidence of such in the logs, and that 70% of logging patterns designed to find errors were skewed towards the end of a block of instruction (He et al).

During the survey of academia and the industry, these researchers were able to uncover four problems with modern day log analysis systems. These four areas included that systems engineering was traditional based around the developer’s domain knowledge, that the spread of big data analytics has percolated to gigabytes of data per hour, that global interdisciplinary teams has led to a global version control issue where team members were not involved in the original planning, and that modern AGILE software engineering practices has required a continual update cycle to established logging architectures (He et al). Let us break these thoughts down one by one. It is incredibly difficult to step outside of our mindset and perspective and account for not just your specialty, but to provide a concerted, cross-disciplinary solution. Breaking out of the continual thought process is the reason why De Bono’s Thinking Course or Copi’s Informal Logic course is so critical. As analysts, whether that is cybersecurity, machine learning, geopolitical analysis, or whatever you are providing predictive analytic reasoning on, breaking away from your traditional habits to look freshly at a subject can’t be understated. Within De Bono’s Thinking Course, we will cover various techniques that allow us to look at the subject or topic anew. No longer will you be stuck from a mental transition or have the inability to decipher new possibilities to a problem. Instead, with his tools and techniques, you can liberate your mind to observe the possible. Seeing more possibilities does not guarantee a successful prediction but will help create alternative lines of analysis that could guide your strategic thinking towards the right conclusion. With the fusion of informal logic, you will facilitate a methodology to disregard alternative lines of analysis to parse what is both possible and plausible.

Also, the researchers were able to uncover the main challenges associated with automated logging: what-to-log, how-to-log, and where-to-log. Differentiating and thoroughly understanding these questions are of utmost importance when designing a new automated logging enterprise solution. The what-to-log section expressed concern for accounting for dynamic content, relying static text, and capturing the appropriate verbosity level. Static text is thought to have a rigid structure that can be pre-determined with statistical methods, like Bayesian approaches, or with deep learning convolutional neural networks. After your static text schema is determined, you must decide how to take in and identify dynamic information that may not fit into a set category. This can and will be shown later within the capstone paper through NoSQL methodologies. Lastly, one of the hardest issues with the what-to-log is defining the verbosity level. Too much verbosity and you tax the network defenders, but not enough could threaten missing the advanced persistence threat (APT) or maintenance problem within your enterprise. Moving towards the how-to-log problem, the researchers surmised two problems: no unification of design patterns nor central gravity of control. Regarding control, some corporations preferred centralized, while others required decentralized agents for automated logging. There is no right answer for either of the problems mentioned, your pattern and control preference should be based on a multitude of factors. Some of these factors could pertain to your IT environment, your corporate structure and work locations, the type of data you are collecting, what your stakeholders and customers prefer, and finally, the laws and regulations you may be required to follow to operate. Finally, the where-to-log problem cannot be underestimated either. You need to find your company’s sweet range of solicitating services and database to accurately diagnose the problem without missing out on critical information. The best examples of this are predictive maintenance studies for software and hardware of automobile and weapon systems.

Throughout this literature review, a greater explanation and expansion upon log compression could be helpful. Log compression is one of the most pertinent aspects associated with the Global Information Grid (GIG) but is the least researched subject when discussing automated logging (He et al, 2021). The problem today is not that we don’t have enough data, but that we are overtaxed by the information that we do have. The sheer volume of information is difficult to parse for answers, and extremely troublesome to send back to all the key stakeholders. More appropriate compression solutions are the answer we need when talking about big data analytics associated with the GIG. Various techniques listed here included buckets, dictionaries, statistics, machine learning, and parallelization (He et al, 2021). Using a concerted effort from all these techniques may offset any weaknesses and amalgamate data extractors for a better solution. Future directions of this company’s literary works include focusing on parallelizing bucket-based compression through rigid JSON schemas. Also, we want to use log parsing to identify common keywords and phrases that are mapped to domain specific code pages. These code pages will act as the cipher and decipher for more efficient encoding of information. Finally, through Bayesian statistical techniques we will further consolidate the logs that have been generated for predictive analytics.

Apple has designed an efficient and effective log monitoring application programming interface (API) architecture on the Mac OS. After running the console process, a terminal is presented with all the console messages. Since that is not very user friendly, Apple also designated where the logs can be stored within the file system by the service it provides. The destination of a log file within the Mac OS depends on the type of log, security logs go to /private/var/log, firewall logs go to /private/var/log/appfirewall.log, user-specific logs go to ~/Library/Logs, command line logs go to the .bash history file, shared application logs go to /Library, crashes go to /System/Library /CoreServices/Crash Reporter.app, as well as numerous other log types. Whether you are in Red Hat, CentOS, Ubuntu, or Mac OS, knowing ahead of time where these logs are stored cuts down the incident response process. During a breach notification, an analyst may need to verify where and when to determine the how and why. One of the most significant components of any logger is the timestamp, Mac follows the Unix log format: MMM DD HH:MM:SS Host Service: Message. Verifying the integrity of the timestamp can help a defender from going down an unnecessary path.

Securing Macs entails more than just logging, it requires acting on the information of the logger! So far, we have covered the details associated with inherit Mac logging, but how is this incorporated within the corporate enterprise? In our labs this week we covered two important central logging and management applications: AlienVault and Splunk. Last year, Gartner named Splunk’s Security Information and Event Management (SIEM) technology as a leader in the industry, with Splunk maintaining the best postured software to execute its mission for two years in a row (Regalado, 2020). As a Certified Network Defender (CND), we are expected to know the Splunk Universal Forwarder operates, but there is much more to Splunk. Some of the components useful for Mac logging is the universal forwarder, load balancer, heavy forward, indexer, search head, deployment server, and license manager (Comodo, 2021). Unlike a monitoring a single Mac OS within the /private/var/log/system.log program, being able to leverage these components effectively across the enterprise to deter and mitigate threats makes Mac logging useful. Extending past the system logging itself, centralizing a robust Splunk architecture is becoming more of a requirement each day.

Logging Systems Case Studies

Let us now review our first case study tying in both the logging and JWT concepts we have already covered. Due to the lack of transparency and accountability in times of a geopolitical crisis, it can be tough to get authorization for activities. With the revamp of Pegasus, it is pertinent to incorporate real-time authorization for uncleared individuals, systems, and organizations after a validated NSC request. To do so without adding member or organizational enrollment may be conducted by embedding permissible and permeable JWT claims within Personal Assertion Tokens, and further structuring the information with dictionary-based substitution. Ultimately, this would permit action on behalf of the member, system, or organization in real-time situations. This would thus minimize enrollment overhead, latency of certification validation per client, and increase security. It should not be unbeknownst to the observer that machine-to-machine (M2M) collaboration in a multi-domain environment would benefit the most within a contingency-based kill-chain scenario due to the alacrity of the process. Without glancing too far into how DARPA’s killchain attestifies and dynamically changes in multi-domain warfare environments, it should be obvious that JWTClaimNames within PASSporTs were specifically designed for autonomous warfare (Housley, 2021). Two examples of such can be seen as follows: beyond line-of-sight missile handling authorization and HVT target reprioritization. For these examples, target reauthorization and prioritization for M2M warfare could autonomously take into account a certificate that authenticates to assets after intruding on the killchain targeting cycle; such an authentication could include a certificate announcing an Enhanced JWT Claim Constraints certificate extension that contains keypairs with the following permitted values JWT Claim Names: {"OPERATION": “BERRYCOOL”, “NSC-Authority”: “ECHOCHARLIE”, “TARGET”:”LOCATION”,“PRIORITY”:"CRITICAL"}. To wrap up the novel techniques that could be incorporated within resilience Global Information Grid (GIG) networks, the observer of such claims should also be aware of one more built-in operations security feature (OPSEC): inherent message security through

Enhanced JWT Claim Constraint restructuring. Open standardization of

the PASSporT structure is expected and encouraged due to the built-in OPSEC extensions of situational dependent Enhanced JWT Claim Constraints, with such would allow native filtering without deduplication or sensory overload. Lastly, these situational extensions should be held as a closely held secret appropriate with the classification level of the information, assets, and operations. In the end, leveraging PASSporT's native Enhanced JWT Claim Constraints extensions could be useful for attestifying all processes and assets, to include both cloud-based environment and the multi-domain environment.

The second case study includes determining how Remote Direct Memory Access (RDMA) networks could be leveraged to provide more cost-effective and elastic cloud-native databases. Lately, one group of researchers have been trying to redesign the database layer to bypass the Linux I/O stack, which would allow RDMA flows per data pattern or template, ultimately resulting in an efficient, novel memory management architecture controlled by the backend systems (Zhang et al, 2021). This relates to the main endeavor with GRAPE optimization at Argonne laboratories and Pegasus 16 at Lockheed included adequate compensation for eigensolver workloads on quantum platform architectures. That same year, the Japanese released their first designs of what a quantum server should look like. Although I researched, designed, and presented my proposed RFCs and ideas to only one CCIE member, these concepts still stand for the Chinese. Implementing the researcher's RDMA network epiphany, whereby pushing memory constraints of onto quantum servers, could attestify to the best practices NASA set for hybrid systems in respect to noisy, intermediate scale quantum computers. While the Chinese's design would have significant speed up for hybrid systems, there still needs to be a drastic shift to which Los Alamos achieved that same year. Instead of only focusing on mapping classical to quantum memory regions, using quantum eigensolvers like GRAPE or Pegasus, it would be pertinent to incorporate these variational quantum algorithms within distributed quantum modes or states. What the article suggests is possibly segmenting different regions on the backend depending on the data determined to be most interesting. In that same year, I was reading that Germany was able to manipulate the optical pathways to maintain three dimensionality associated with affixed data-driven quantum control paths, to which was only announced by NIST in Colorado in 2020. With such being said, the current research at Los Alamos is attempting to provide methodologies to decouple continual variational quantum states through tunable optical sites/regions across the graphene quantum dot medium. Ergo, it would seem likely that they should not only decouple the memory for RDMA across hybrid architectures but should also implement semi-affixed neural pathways for localized memory computation within that same medium. Obviously RNN during the training phase could attesify which neural pathways to cement. Such was the way in 2019 and remains true going into 2022. In the end, I would say that I had a very productive six-month Active-Duty military deployment.

MDM

Today’s vendors for mobile enterprise security have taken lessons learned and now offer enhanced coverage through Enterprise Mobility Management (EMM). Traditionally, like Internet of Things (IoT), mobile devices did not provide enough protection and were an easy target for hackers because they were an emergent technology and were representative of bare bones devices. Unlike IoT, mobile computing capacity has accelerated dramatically making the once bare bone bullseyes formidable defenders in the enterprise. To the causal reader, including myself, background knowledge of G technology is helpful to understand this transition. Stanford gives an easy representation through the evolution of G’s: 1G is 2.4 kbps, 2G is 64 kbps, 3G is 2 mbps, 4G is 100 mbps, and 5G is 1-10 gbps (Sound, 2017). With this increase in network transmission, and parallel computing capacity, vendors have been able to offer greater security capabilities. Advanced capabilities seen through the EMM can be seen by examining the components of the unified suite solution: Mobile Email Management (MEM), Mobile Device Management (MDM), Mobile Threat Management (MTM), Mobile Application Management (MAM), and Mobile Content Management (MCM).

Applying MDM solutions to an enterprise network helps protect devices no matter the device policy, such as Bring Your Own Device or Company Owned, Personally Enabled. Standard mobile device policies that can be enforced through MDMs are management and configuration, tracking and inventory, legitimate software store downloads, encryption, and remote wiping. There are various types of MDM delivery methods, such as premise-based, Software as a Service (SaaS), or managed services. One such recommended solution is AirWatch.

Phishing has plagued the corporate networks for years. Spear phishing, or targeted phishing, tailors their material towards the target audience, with whale phishing targets C-suite members. MEM helps combat phishing by data loss prevention, storage and transit encryption, and policy compliance. Using Simple Certificate Enrollment Protocol (SCEP) and S/MIMEMDM defeats and defends against some of the phishing attempts. The book suggests using 1Mobility MEM or 42Gears MEM solution

Although the EMM components help the enterprise network defenders stay one step ahead of threat actors, having next-generation protection requires dedicated threat intelligence with MTM. Fireeye realized the benefits of partnering with Zimperium to defeat threat actors through data-driven, tailored threat intelligence, “collaborating with Zimperium... provides real-time, on-device protection against both known and unknown Android and iOS threats” (Unick, 2020). Helix’s MTM solution employs Zimperium’s machine learning algorithms, implemented in the z9 detection engine, to fuse features to predict unknown attacks on mobile devices before they happen (Zimperium, 2021).

Functionality of the MAM is intricately intertwined with MDM capability but extends the certificate-based features. The MAM attests to device identity through activation, enrollment, and provisioning to deliver secure access to legitimate enterprise application stores. Also, the MAM helps the enterprise legally by checking application authorization, software licensing, version control, and much more. Microsoft's Intune software aids organizations in accomplishing the previously mentioned features.

Finally, MCM, or Mobile Information Management (MIM), helps assure network defenders and managers that employees are only viewing authorized, encrypted corporate data. Increasing both productivity and security, MCM guarantees that the right information is properly viewed by employees. To follow confidentially principles and policies, MCM also provides remote data wiping in case the mobile device is lost. A few MCM solutions include Vaultize, MobileIron, and ContentBox.

RAID

Storing information for later use is engrained in every aspect of our lives, even down to our very genetics. Whether it is a qubit, background cosmic radiation, or Redundant Array of Independent Disks (RAID) technology in datacenters, everything has information stored that can be later retrieved and put to good use. Today we are currently pursuing many alternatives of RAID material that will lead us into the future. One of which is DNA-based disk arrays that will emulate how bioinformatics work (Langston, 2019). Another Microsoft project is called Project Holographic Data Storage that uses optical light infraction inside a crystal to pack lots of data inside (Miller, 2020). Whatever the RAID material is they still follow some basic guidelines of benefits and negatives.

Copying data to identical disks, called mirroring or RAID 1, was created in the 1970s by Tandem NonStop Systems and the term was coined by UC Berkeley in the 1980s (eRacks, 2021). RAID is the implementation of storing data across multiple disks to increase the input/output performance. There are a vast number of options that have been devised over the years, with each implementation depending on one’s environment. A few types of RAID are RAID 0, RAID 1, RAID 3, RAID 10, and RAID 50. Going through an understanding of each methodology, and how it applies to a particular enterprise’s need is beyond the scope of this discussion. Instead let’s summarize the pros and cons. Each RAID type is judged against fault tolerance, performance, and competence to score whether that is the right design for your needs. Some organizations may require fast performance, while others like the banking industry may need the most reliable fault tolerance. Although not universal across all RAID designs, like RAID 0, the following advantages provide better metrics when analyzing the right RAID deployment: performance and reliability, parity check, data redundancy, disk striping, and system uptime. Striping is the process of dividing the data into smaller segments to spread over multiple disks. While, parity checks are only achieved within certain RAID devices, like RAID 5, to ensure that the disks can recover after a crash. When researching the market for which RAID to choose, be aware of the negatives: writing network drivers, noncomputability, loss of data, time consumed rebuilding, and cost. Two negatives to highlight are the cost and time consumed after a crash. Since IT supports businesses, make sure that the RAID implementation fits within the Recovery Time Objective (RTO) within the Business Impact Analysis (BIA) for the company. Sometimes this may mean spending more money on double parity RAIDs.

Networking Basics

Addressing

A background in cyber is assumed moving forward, but let's highlight some important concepts about networking first using the Open System Internetworking (OSI) model. Layer one, or the physical layer, works with the actual signals. Most of the time this information will be abstracted. Layer two, or the data link layer, deals with frames and MACs through the use of bridges and switches. Attacks at this layer include, but aren't limited to, ARP poisoning, STP BPDU spoofing, and more. Using wireshark or tcpdump can help monitor data going across "the wire" to detect this nefarious activity. At the network layer, or layer three, logical addressing using IPv4 or IPv6 is accomplished. There are plenty of attacks, like the ping of death or a smurf attack, but tend to all leverage the ICMP protocol. Important coding concepts we will cover here relate to IP integrity and more.

Port Scanning

After getting access to the network, it is important to consider what ports are open. You can accomplish this many ways, such as using nmap or hping3, but we will build our own custom scanner here!

```
import sys
import os
import socket
import multiprocessing as mp

class PortScanner:
    # Timing same as nmap timing
    # Different scan types later
    def __init__(self, ip: str, ports: list = [], timing: int = 3, scan_type:
        self.ip = ip
        self.ports = ports if len(ports) else [i for i in range(1, 65536
        self.timing = timing
        self.open_ports = []

    def probe(self, port: int):
        # is important? sys.stdout.flush()
        try:
            # Scan types later
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(0.5)
            r = sock.connect_ex((self.ip, port))
            if r == 0:
                result = r
            sock.close()
        except:
            pass

        return port if result == 0 else False

    def __call__(self):
        with mp.Pool(os.cpu_count()) as pool:
            processes = [pool.apply_async(self.probe, args=(self.ports[idx],)) for idx
        results = list(filter(  lambda   x: x !=   False  , [p.get()   for   p   in   proc
          return   results
```

Pivoting Through Network - Host Scanning

Let's say you get onto the network and now need to determine what other assets are inside the company. You could easily determine immediate services in the proximity by observing what connections the host is talking to!

```
  declare   -A protos
  declare   -A foreignIps

privateRegex=  "(0.0.0.0)|(^127\.)|(^192\.168\.)|(^10\.)|(^172\.1[6-9]\.)|(^172\.2[0-9]\.)|(^

  # Only care about proto and foreign IP right now
IFS=$  '\n'  ;
  for   connection   in   $(./ipstat | sed -E   's|\s+|,|g'   | cut -d   ','   -f 1,5)
  do
        proto=$(  echo     $connection   | cut -d   ','   -f 1);   # ${proto::-1}; # variable
        foreignIp=$(  echo     $connection   | cut -d   ','   -f 2);
        boundPort=$(  echo     $foreignIp   | sed -E   's|.*\:||g'  );
        foreignIp=$(  echo     $foreignIp   | grep -oE   ".*\:"  );   # No positive lookahead
        foreignIp=  ${foreignIp::-1}  ;

          if   [   ${protos[$proto]+_}   ]
          then
                ((protos[  $proto  ]++));
          else
                protos+=([  $proto  ]=1);
          fi

          if   [[ !(  $foreignIp   =~   $privateRegex  ) && !  ${foreignIps[$foreignIp]+_}   ]
          then
                foreignIps+=([  $foreignIp   ]=  $boundPort  );
          else
                  echo   -n   ""   ;  # echo "$foreignIp is running a NAT overload!";
          fi

  done

./threatApi <<<   echo     "  ${!foreignIps[@]}  "

  ############ Testing Below ################

<<  '### BLOCK-COMMENT'
  for   key   in     "  ${!activeConnections[@]}  "
  do
          echo     "  $key   has the following active connections:   ${activeConnections[$key
  done
  for   key   in     "  ${!foreignIps[@]}  "
  do
          echo     "  $key   is on the following port   ${foreignIps[$key]}  "  ;
  done
  ### BLOCK-COMMENT
  # echo "${!foreignIps[@]}"; # ! means keys, without it means values
  # for key in "${!foreignIps[@]}"; do echo -n "$key "; done
```

Load Balancers - Case Study

SDN Priority Queue

Load balancers are fun! There are numerous algorithms, such as round-robin or class-based, to consider. Here we will model our software off of a class-based routing model.

```
import   heapq
from   multiprocessing   import   Lock

  class     ClassBasedQueue   :
      def     __init__   (  self, classes =   3   , quantumValues: list = [  3   ,   2   ,   1   ]
        if   len(quantumValues) != classes:
            raise     "Lengths don't match!"

      self.classes = classes
      self.length = self.classes
      self.totalRoundLength = sum(quantumValues)
      self.quantumValues = quantumValues

      self.queues = {idx: [[], Lock()]   for   idx   in   range(self.length)}

      def     add   (  self, classIdx: int, value  ):
        with   self.queues[classIdx][  -1  ]:
          self.queues[classIdx][  0  ].append(value)

      def     pop   (  self, classIdx: int  ):
        with   self.queues[classIdx][  -1  ]:
            return   heapq.heappop(self.queues[classIdx][  0  ])

      def     getNext   (  self, classIdx  ):
        try  :
            return   self.pop(classIdx)
        except  :
            return     None

      def     nextRound   (  self  ):
      idx =   0
      roundVals = []
      totalLength = self.totalRoundLength
      notFound =   0

        while   idx < self.classes:
          y =   0
          y_length = self.quantumValues[idx]
          found =   False

            while   y < y_length   and   totalLength >   0  :
              val = self.getNext(idx)
                if   val !=   None  :
                  y +=   1
                  totalLength -=   1
                  roundVals.append(val)
                  found =   True
                else  :
                  y = y_length

            if     not   totalLength:
                break

            if     not   found:
              notFound +=   1
              idx = (idx +   1  ) % self.classes

                if   notFound == self.classes:
                    break
                elif   idx ==   0  :
                  notFound =   0

      roundVals = sorted(roundVals, key =   lambda   x: x.score())

        return   roundVals, self.firstGoodIp(roundVals)

      def     heapifyQueues   (  self  ):
        for   classIdx   in   range(self.length):
          heapq.heapify(self.queues[classIdx][  0  ])

      def     firstGoodIp   (  self, vals  ) -> int:
      length = len(vals)
      left, right =   0  , length -   1
      last =   -1

        while   left < right:
          mid = (left + right) //   2
            if   vals[mid].isMalicious() ==   True  :
              left = mid +   1
              last = mid +   1
            else  :
              right = mid

        if   vals[right].isMalicious():
```

```
if   vals[right].isMalicious():
    return    -1
elif   last ==   -1  :
    return    0
else  :
    return   last
```

Firewall Rundown

Concepts

The five types of firewalls go in this order: packet filtering, circuit-level gateway, application-level gateway, stateful inspection, next-generation firewall (NGFW) (Decarlo and Ferrell, 2021). First Generation – Stateless. Focuses only on stateless data associated with src/dst IP, src/dst port, and protocol (Mojidra, 2020). It is only pertinent here to consider the five criteria above. We can easily wrangle the data to use and display only this data. Second Generation - Circuit-Level Gateway. Uses network protocol session initiation messages, such as the TCP handshake, to monitor whether the connection should be accepted or not (Decarlo and Ferrell, 2021). Third Generation - Application-Level Gateway (Proxy Firewall). Looks at the application layer in the Open System Interconnection (OSI) model, such as the HTTP request strings, to determine whether to accept the connection (Decarlo and Ferrell, 2021). Here we could add the L7Protocol and ProtocolName columns. Since the researchers did not provide the actual data within the packets, we can't check the HTTP request strings within this review. Third Generation - Application-Level Gateway (Proxy Firewall). Looks at the application layer in the Open System Interconnection (OSI) model, such as the HTTP request strings, to determine whether to accept the connection (Decarlo and Ferrell, 2021). Here we could add the L7Protocol and ProtocolName columns. Since the researchers did not provide the actual data within the packets, we can't check the HTTP request strings within this review. Fifth Generation - Next Generation Firewall (NGFW). Provides the most granular level of control over previous firewall generations as it fuses both stateful and deep packet inspection (DPI), from layer 2 to layer 7 of the OSI model, to provide enough granularity to check whether a particular HTML response was formatted correctly or not (DeCarlo and Ferrell, 2021). With the lack of data provided within the packets, I thought it would be best to look at a NGFW dataset populated from Cisco's Adaptive Security Appliance (ASA). The important things to note here is that Cisco's FirePower provides important threat-driven firewall intelligence for firewall analysis, such as credibility, IPgeo recognition, severity, etc. The lack of flexibility, scalability, and speed within multi-tenancy environments, or environments with heterogenous IIoTs like Naval weapon systems, is concerning against stronger near-peer adversaries.

A comprehensive firewall topology discussion would require a firewall drill of definitions! There are numerous topologies to consider when determining effective measures against your threat environment. A firewall topology is not a one size fits all solution. The banking industry may have different threat actors that are more sophisticated with better tools than the retail industry. The government may be concerned with Advanced Persistent Actors (APT), while the small business may be concerned with script kiddies. First, establishing an organizational policy determining what the potential threats are and what security level the organization wants is most important, such as FIPS 199’s security category (Stine et al, 2008). Analyzing the threat environment drives the technological deployment of an organization’s firewall topology. Once establishing threat agents, both their capabilities and intents, knowing the necessary technical integration is important. Technical integration falls into the planning stage of firewall deployment: technical objectives, does it fit into the existing network, what traffic will be inspected, deciding on hardware versus software, and which OS to use. This could encompass ten pages of analytic rigor towards one case. Instead, let us focus on a fictional case study of the Poseidon & Sons mid-sized business branch office firewall deployment.

A new Poseidon & Sons branch office is opening at Camp Half-Blood, with connectivity back to the headquarters in Olympus. Known threat actors are very TCP/IP savvy monsters that can tailor exploits to affect buffer overflow vulnerabilities to SQL injection attacks. Therefore, a simple packet filtering firewall, one that only looks at IP headers like destination and source addresses, are not enough to keep these attackers outside of the camp. The camp leaders also realize that the demigods enrolled in the academy need to remain anonymous from all outsiders, requiring a circuit-level gateways proxy the original sessions and application-layer gateways to proxy social media accounts. Since there has been concern of masquerading IP addresses, there is a need to implement stateful multilayer inspection to track geolocated IP addresses with inbound requests. Finally, incorporating tailored intelligence towards Kronos' minions would be necessary to combat white-hat based network attacks. With all of these requirements listed within the Request for Proposal, Athena's Next Generation Firewall (NGFW) seems to be the best system to fit all needs.

The Poseidon & Sons branch office at Camp Half-Blood must consider the actual deployment of Athena's NGFW. Zesus' Thundercloud is offering Firewall as a Service (FWaas) which will incorporate Web-based Application Firewalls (WAFs) and Athena's NGFW as a service with his Managed Service Provider subscription (Ingalls, 2020). The branch office must understand latency concerns with this option. Soon Percy realizes that the cloud FWaaS is not enough protection and instead lobbies to the board that there should be a buffer between the private network and the Internet. Percy recommends a screened subnet, with a Demilitarized Zone (DMZ), protecting the web servers (bastion hosts). The FWaaS can be applied to the dual homed external router, while the internal router employs a proprietary Camp Half-Blood deep packet inspection firewall. The FWaaS can facilitate Peerlock to ensure safe BGP transit back to the Headquarters office, ensuring trusted network communication (McDaniel et al., 2021).

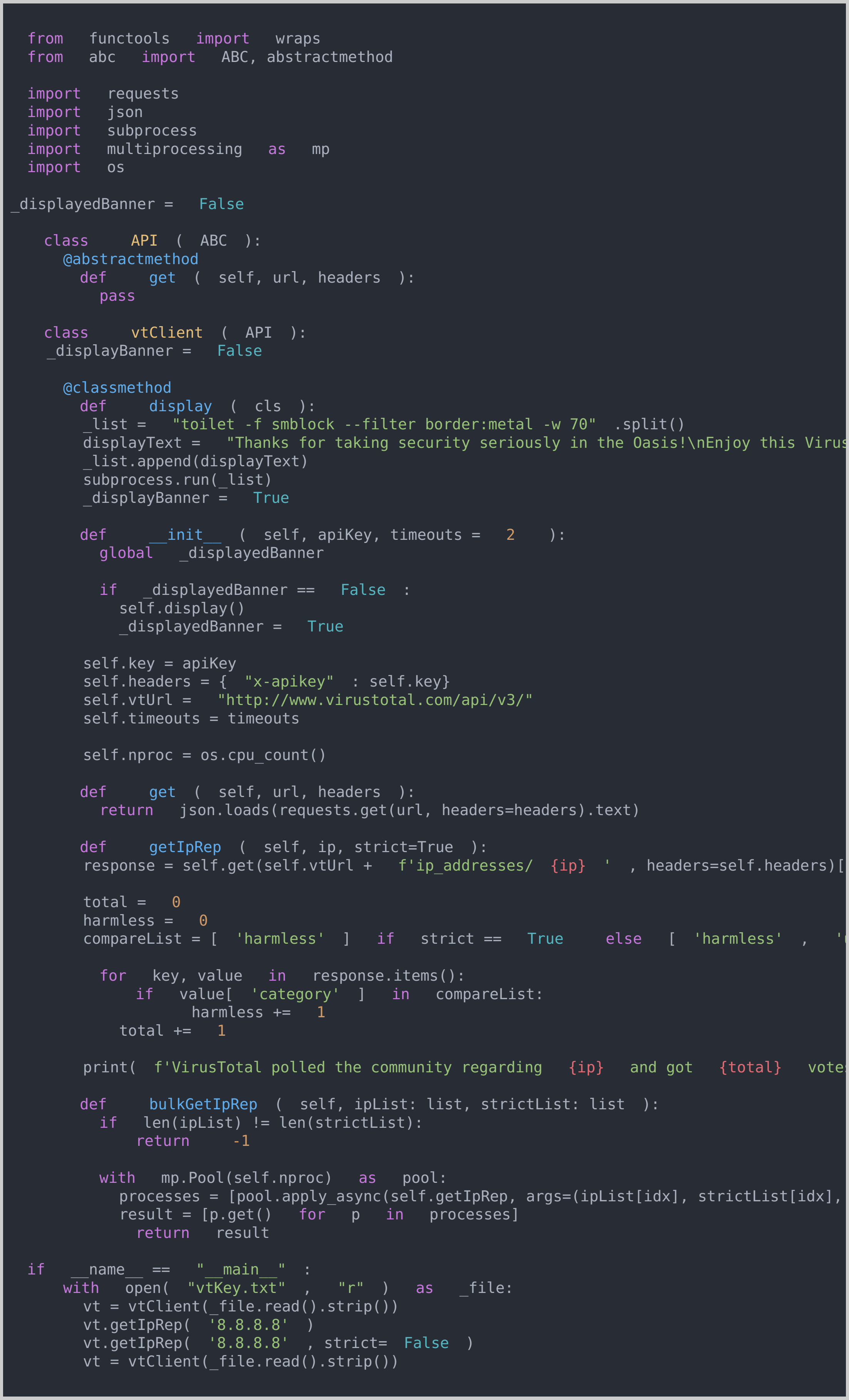
IDS/IPS

Both intrusion detection systems (IDS) and intrusion prevention systems (IPS) leverage information in multiple ways to do some action. These categories of leveraging information fall into signatures and heuristics/anomaly, with both being split into subcategories. For example, anomaly detection can use Bayesian statistics, neural networks, or deep learning to identify abnormal behaviors of a system or user to defend against zero-day attacks. On the other hand, signatures use Indicators of Compromise (IoC) like shell code or nmap scanning techniques to determine if there are malicious activities occurring. In later books we will focus specifically on these topics in-depth, however for now, it is important to understand the differences between the two. An IDS generates an alert, whereas an IPS generates and alert and actively tries to stop the attack. Also, it is important to know that there are different types of these IDS/IPS programs depending on the application at hand: network IDS, host IDS, protocol-based IDS, application protocol-based IDS, hybrid IDS (Security Trails, 2021).

Practical Threat Intelligence Integration - Case Study

Here is a case study of setting up distributed cluster processing each section of a majority space-based architecture: the China High-Resolution Earth Observation System (CHEOS) (Jones, 2021). Within this architecture we have several Gaofen generations, some of which span to at least five generations since 2015 (Jones, 2021). The interest of this discussion is not to expand on these architectures, to include both their orbits and capabilities, as I'm sure there are some that can augment for those pieces. Instead, it is of utmost importance to understand the data and information warfare associated with these multi-domain assets. However, to glean insight into such, it should not be unaware that each generation of assets are composed of different components that would augment specific mission sets. Based on the findings from NASA in 2013, and the fact that it is likely that data will double every two years, you would have roughly 16 terabytes per generational series deployment (Jones, 2021). Fusing this information with their five-year plan, then corresponding it to the technical and operational capabilities, could give a better estimate for the CHEOS. With such knowledge it would not be unreasonable to assume daily data acquisition in the 100s of terabytes. Once again, the anticipated number of satellites would have to be speculated from doctrine and policy, but it would not be unreasonable to assume that data may in fact be closer to the petabyte scale soon. If this information is further fused with other terrestrial and airborne assets, the Chinese could be looking at an exabyte scale. Ergo, information management is key.

Now after gaining insight into a NGFW, we can code up software that allows up to leverage threat-driven IP information. Go through the code below and integrate with a load balancer!



```
  from   functools   import   wraps
  from   abc   import   ABC, abstractmethod

  import   requests
  import   json
  import   subprocess
  import   multiprocessing   as   mp
  import   os

_displayedBanner =   False

    class     API  (  ABC  ):
      @abstractmethod
        def     get  (  self, url, headers  ):
          pass

    class     vtClient  (  API  ):
    _displayBanner =   False

      @classmethod
        def     display  (  cls  ):
        _list =   "toilet -f smblock --filter border:metal -w 70"  .split()
        displayText =   "Thanks for taking security seriously in the Oasis!\nEnjoy this Virus
        _list.append(displayText)
        subprocess.run(_list)
        _displayBanner =   True

        def     __init__  (  self, apiKey, timeouts =   2    ):
          global   _displayedBanner

          if   _displayedBanner ==   False  :
            self.display()
            _displayedBanner =   True

        self.key = apiKey
        self.headers = {  "x-apikey"  : self.key}
        self.vtUrl =   "http://www.virustotal.com/api/v3/"
        self.timeouts = timeouts

        self.nproc = os.cpu_count()

        def     get  (  self, url, headers  ):
          return   json.loads(requests.get(url, headers=headers).text)

        def     getIpRep  (  self, ip, strict=True  ):
        response = self.get(self.vtUrl +   f'ip_addresses/  {ip}  '  , headers=self.headers)[

        total =   0
        harmless =   0
        compareList = [  'harmless'  ]  if   strict ==   True     else   [  'harmless'  ,   '

          for   key, value   in   response.items():
              if   value[  'category'  ]   in   compareList:
                    harmless +=   1
            total +=   1

        print(  f'VirusTotal polled the community regarding   {ip}   and got   {total}   vote

        def     bulkGetIpRep  (  self, ipList: list, strictList: list  ):
          if   len(ipList) != len(strictList):
              return     -1

          with   mp.Pool(self.nproc)   as   pool:
            processes = [pool.apply_async(self.getIpRep, args=(ipList[idx], strictList[idx],
            result = [p.get()   for   p   in   processes]
              return   result

  if   __name__ ==   "__main__"  :
      with   open(  "vtKey.txt"  ,   "r"  )   as   _file:
        vt = vtClient(_file.read().strip())
        vt.getIpRep(  '8.8.8.8'  )
        vt.getIpRep(  '8.8.8.8'  , strict=  False  )
        vt = vtClient(_file.read().strip())
```

Vulnerability Scanning

CVEs

Understanding the entirety of your corporate enterprise, to include both on-premises, in the cloud, and remote work, can be a struggle and requires great corporate insight. Not only do you have to identify and capture the necessary systems that you can see and own, but you need to identify upstream and downstream information systems, possibly in other companies, that would impact the systems you oversee. Once that part is tackled, you have the task of performing the impact analysis to determine where resources should be allocated. Not even the United States Government (USG) has unlimited resources, there needs to be prioritization on what is important so that we can move onto the next step. This prioritization can be conducted by qualitative and quantitative analyses. Most of the time, we analyze this through the ideas of threat and vulnerabilities to determine risk. The next section will cover the notion of risk more, just know that based on a system's vulnerability, like a Common Vulnerabilities and Exposures (CVE), and the likelihood of that happening, or threat, we can categorize the necessary information systems and processes appropriately. The common methodology used in the industry to deliver this to mid to senior managers is through the use of a risk acceptability matrix. A risk matrix fuses qualitative and quantitative risk to deliver strategic situational awareness on information systems without being technically inclined or knowledgeable about the system. For instance, a senior level manager does not need to understand SQL injection against their PHP Admin Lite page. Instead, security and cyber analysts leverage open-source vulnerability databases, like MITRE, to parse scoring metrics that would compare this CVE, using the Common Vulnerability Scoring System (CVSS), for the manager to contrast with against other vulnerabilities. In so doing, the manager can decipher the security threats and allocate proper resources towards the assets and information systems that pose the greatest risk to the organization. Remember that we most likely do not have unlimited resources to tackle every issue, so there will be some that are prioritized, while others will not be touched. More on this is covered in the types of risk section.

After parsing out and categorizing cybersecurity items of interest, we perform the selection of NIST SP 800-53 controls (NIST, 2021). Last September, NIST provided an excellent tool for stakeholders to improve their implementation of the RMF process by way of accessing the controls and control baselines (SP 800-53B) in real-time (NIST, 2021). In the real world, analyzing the threat environment and understanding what vulnerabilities are associated with what is not enough. There needs to be action done on these vulnerabilities, or else why did we do this in the first place. Gathering blue side mitigations using NIST SP 800-53 allows our managers tangible courses of actions (COA) to ensure our security posture. Some may even find the reverse process of identifying the security baselines within NIST SP 800-53B first to be fruitful. After identifying the baselines, you could then go and find the controls necessary to securely deploy it within your enterprise. This reverse approach is great for new development areas, while the former is usually done for information systems that are already in place.

After selection you must go through the hard work of implementing these information systems and controls. Although this may seem like the most laborious and frustrating task, if everything was done properly in the prior steps then this should be rather easy. Though it is true that everything done on paper is easier, or that it is easier said than done. Look to common installation guides, company manuals, to coworkers, and if all else fails, go check out online forums. Remember to do it right the first time so that you do not need to waste time and must implement it a second time.

The next two steps can be very similar, so I will lump them together here: assess and authorize. Assessing that the information systems and controls are properly in place is a necessary evil to be able to allow senior level managers make a determine on the worthiness of the system (NIST, 2021). In the military you will hear terms like Air Worthiness, Authority to Operate (ATO), Certification and Accreditation (C&A), System Validation, etc. All these things really boil down to the same thing, you have verified that the information systems are deployed appropriately and that the controls are in-place providing a certain security profile that is acceptable. As soon as that ATO or C&A is given, the information systems are said to be fully operational and deployed.

Once there is an ATO for the information system, the tangible and intangible assets are passed down to the operational team, like mission control managers, system administrators, intelligence analysts, security officers, etc. It is the job of these professionals to ensure that the ATO is continuously monitor for any violation or infraction. The information system will now remain in this state for the remain of its lifecycle, except in the instance that the asset is reutilized or decommissioned.

```
import   urllib3
from   bs4   import   BeautifulSoup
import   re
import   requests

  class     OSVDB_Converter   :
  osvdb_to_cve_url =   "https://cve.mitre.org/data/refs/refmap/source-OSVDB.html"
  osvdb_pattern =   r'OSVDB:[0-9]{1,}'
  cve_pattern =   r'CVE-[0-9]{4}-[0-9]{1,}'

      def     __init__  (  self  ):
      self.parseUrl2()

      def     parseUrl2  (  self  ):
      self.mapper = {}
      http = urllib3.PoolManager()
      response = http.request(  'GET'  , self.osvdb_to_cve_url)
      soup = BeautifulSoup(response.data,   "html.parser"  )

        for   td   in   soup.find_all(  'tr'  ):
          data = str(td)
          osvdb = $(echo @(data) | grep -oE @(self.osvdb_pattern)).strip()
          cves = $(echo @(data) | grep -oE @(self.cve_pattern)).strip().split()
          self.mapper[osvdb] = cves

      def     get_data  (  self  ):
        return   self.mapper

if   __name__ ==   "__main__"  :
    with   open(  "osvdb_translation.json"  ,   'w'  )   as   writer:
      converter = OSVDB_Converter()
      data = converter.get_data()
      data = data.items()
      length = len(data)
      count =   0

        for   key, values   in   data:
            if   key   and   values:
              writer.write(  f'  {key}     {  " "  .join(values)}  '  )

                if   count != length -   1  :
                      writer.write(  '\n'  )

          count +=   1
```

Ping To Nikto

Once you find an IP address of interest...

```
if   [[ $(  echo   $(ping   $1   -c 5) | grep -oE   "bytes from"   | wc -l) -gt 0 ]]
then
        echo     "  $1   appears to be up!"
      nikto -h   $1
fi
```

Ping To Vulnerability Exploitability - Case Study

Time to integrate all of the above techniques together, building a ping to vulnerability package.

```
#!/usr/bin/xonsh

import   sys

mapper = {}

with   open(  "osvdb_translation.txt"  )   as   reader:
    for   line   in   reader.read().split(  "\n"  ):
        if   len(line):
          line = line.split()
          osvdb = line[  0  ]
          osvdb = osvdb.replace(  ":"  ,   "-"  )
          mapper[osvdb] = line[  1  :]

  def     ping2vuln   (  host  ):
    if   int($(echo $(ping @(host) -c   5  ) | grep -oE   "bytes from"   | wc -l).strip()) >
      print(  f'  {host}   appears to be up'  )
      count =   0

        for   osvdb   in   $(echo $(nikto -h @(host)) | grep -oE   "OSVDB-[0-9]{1,}"  ).spl
            try  :
              print(  f'Exploitable with   {mapper[osvdb]}  '  )
            except  :
              print(  f'Exploitable with   {osvdb}  '  )
          count +=   1

        if   count ==   0  :
          print(  f'  {host}   does not appear to be up!'  )
        else  :
          print(  f'  {host}   has   {count}   vulnerabilities!'  )

if   __name__ ==   "__main__"  :
    ping2vuln(sys.argv[  1  ])
```

Beyond PenTest+ - Protecting the Networking Enterprise from APTs

BGP Modernization Introduction

A continuously growing threat environment has stimulated the growing need to secure the Internet backbone’s Border Gateway Protocol (BGP) through cryptographic mechanisms. Last year, President Trump and the Department of Justice’s Attorney General William P. Barr unexpectedly agreed with banning the China Telecom Corporation due to vast BGP hijackings. To make matters worse, last year also saw Russia’s Rostelecom hijack BGP traffic across hundreds of content delivery networks. Internet Service Providers, technology industry giants, and federal agencies are taking a stand and saying enough is enough. Within this paper we analyze current standards and regulations, deployments, operational successes, future add-ons, and tailored technological implementations of cryptographically secured BGP, or RPKI. Leveraging robust Route Origin Authorization and Route Origin Validation functionality through X.509 certificates is, and will continue to be, the inherit cornerstone of cybersecurity that will fight back against threat actors and defend our homeland against any nation state defined as Advanced Persistence Threats.

Imagine a freeway where the exit signs were changed by an evil machine. Sometimes you would get the right exit, other times you would be directed to an unplanned destination. Although this is a modification on Rene Descarte’s evil demon concept in the 1600s, Border Gateway Protocol (BGP) hijacking does occur multiple times a year. For at least two decades, BGP hijacking has affected the industry and has been well documented in the media. In anticipation of the new Resource Public Key Infrastructure (RPKI) rollout, Forbes laid out famous historical cases that have plagued Internet Service Providers (ISPs). Sometimes it is a country’s own government that shuts down perceived propaganda, such as Turkey’s BGP route withdraws against governmental protesters, and other times it is a foreign government redirecting traffic for intelligence purposes (Crabtree, 2021). The Department of Homeland Security (DHS) and the National Institute of Standards and Technology (NIST) have been working on securing the Internet’s backbone protocol, BGP, for almost a decade (Durand, 2020 & NIST, 2016). It was only after Amazon, and other large technology companies, jumped on board that the security of BGP became a priority in the industry. To understand the industry’s environment today, and into the future, we need to understand how BGP is being secured. Therefore, correct implementations of RPKI variants will enable and facilitate globalized, hardened security in the cybersecurity world. BGP security can further support the Confidentiality, Integrity, and Availability (CIA) triad to ensure safe, responsible routing services such as Domain Name System Security. To understand industry and government-led next-generation BGP RPKI it is necessary to explore X.509 public key certificate themes, Route Origin Authorization (ROA) and Route Origin Validation (ROV) functionality, attacks against RPKI, decentralized Edge Intelligence optimizers, and post-quantum encryption replacements.

ASN Requirements

Predicting where the future is going sometimes involves knowing what has happened in the past. Although every internet user relies on BGP for their services, BGP within the Information Technology (IT) world is a very specific subset of skills. System administrators can go a whole career within network administration and never have to be concerned on how the BGP connection is routed through an Internet Service Provider (ISP). As a business grows, network engineers become more intricately hands-on with the principles and implementations of BGP. A simple example is demonstrated through Cisco's certification pathways. A thorough knowledge of what drives BGP, how to set up BGP, and what aspects are important to secure, and monitor are not introduced until three to five years after on-the-job experience with enterprise networking. After five years of a specific job skillset, the 350-401 ENCOR exam, which is the most prestigious technical IT exam, requires you to know BGP for only 6% of the material (Cisco, 2020). It is therefore fair to say that BGP is not an expected topic to know in the IT world; going even further to say that enterprise BGP networking configuration may never be touched by a majority of IT professionals. Let us dive into a small introduction with simplification in mind. First, knowing that as networks grow, they get wrapped up into their own Autonomous System Network (ASN). The definition of a ASN is that all an enterprise's networks are administratively controlled under a single domain (Cloud Flare, 2021). Think of an ASN as a big flee market that requires a customer to only know the name and place of the market, not having to know the specific vendors. To get to the ASN, or flee market of interest, the customer needs to know how to reach it. Each ASN is given a numeric name, ranging from 16-bits to 32-bits, from the Internet Assigned Numbers Authority (IANA) so that it can be identified; each ASN is registered locally their specific Regional Internet Registries (RIRs) (Cloud Flare, 2021). The bit length determined the length of the ASN's name used when the Internet was a smaller village. As the village became a metropolis, the name needed to become longer for the large influx of destinations. To service all the customers around the world, the IANA, on behalf of the ICANN (Internet Corporation for Assigned

Names and Numbers), set up five RIRs to hold and verify the different ASNs (Durand, 2020). For example, if you have a car auction in North America then most buyers will likely be from North America. If instead the auction is about Formula 1 race stocks, then the auction will likely be in Europe. It makes sense for the region associated with the car auction to hold the information associated with that event. That is the same idea ICANN holds for servicing ASN names, or numeric numbers; register the ASN with the appropriate geographic RIR. The problem the IT industry has been having is not where to register and retrieve ASN names, but instead how to verify that the name received is the correct one and has not been spoofed. The only practical, industry accepted CIA principle that could help here was integrity. Using Public Key Infrastructure (PKI) helped adhere to the CIA's principle of integrity by cryptographically signing BGP route advertisements with the correct, registered ASN name (Levy, 2018). Before this year, a wide acceptance of BGP cryptographic means were not well established. The DHS, NIST, and the industry have put their heads together to establish a robust methodology to mitigate BGP hijackings or spoofing. Leveraging RPKI, Amazon alone noticed last summer that a BGP spoofing event transiting across their networks decreased by 30 percent of the usually affected targets (Korsback, 2021). As more and more company's transition to RPKI, BGP will ultimately get safer.

Cryptographically Validating BGP

Securing the future of BGP requires cryptographically validated routing updates, by use of X.509 public key certificates. There have been numerous reversions over the past 30 plus years to contribute to the large number of attributes associated with the X.509 standard; however, one field that is important to check in the X.509 version 3 (X.509v3) is the subject PublicKeyInfo field (Yackel, 2020). Soliciting this X.509v3 field attribute provides the algorithm that is used for the public keys, and since not all algorithms are equivalent, understanding the correct bit-level encryption strength and inherit mathematical vulnerabilities within the algorithm is crucial. The idea behind encryption strength is the effort that the attacker is willing to expend, measured by the length in time, given they have unlimited computing power (Oracle, 2010). If an attacker can compromise the implementation, through side-channel attacks or social engineering techniques, or even compromise the algorithm itself, then there needs to be a way to switch out the public key certification quickly because the encryption strength has decreased. Identifying these changes and establishing the correct certificate revocation list, or A-list like feature, can be accomplished based on the algorithm. Most people would say that this cannot be done readily, but it was not long ago that the Merkle-Hellman Public Key Cryptosystem was broken; with Cisco's prediction that a quantum supremacy cryptanalysis revolution is coming near, it is important to look towards the past to be agile enough for the future (Shamir, 1984). X.509 public key certificates ensure that the Regional Internet Registries (RIRs) are able to verify any requests, and act as an Online Certificate Status Protocol (OCSP) point of presence. This attestation and verification of other certificate identity is provided by how the host Operating System (OS) maintains a list of hierarchical X.509 certificates, or trusted Root CA certificates (Durand, 2020 & Yackel, 2020). Within the RPKI architecture these trusted Root CA certificates are identified as a Trust Anchor (TA), with the location of the TA being called the Trust Anchor Location (TAL), to provide the ability to navigate through a CA repository (Durand, 2020). "A TAL contains both a uniform resource identifier (URI) that points to a trust anchor self-signed root certificate and the base64 encoded public

portion of the key that is used to sign that repository” (Durand, 2020). These TAL enable the robust, security conscious Route Origin Authorization (ROA) and Route Origin Validation (ROV) functions described later. While base64 is easily broken, the encryption strength of the underlying public key is not. This provides a scalable, operationally relevant cryptographic design that permits the necessary encryption strength desired by the industry.

ROV and ROA makes RPKI operationally relevant through simple and effective security schemes. To understand ROV, we need to explain ROA first. “Route Origin Authorization (ROA) is a cryptographically signed object that pairs an originating AS with a specific prefix, the length (size) of the prefix, and an expiration date” (Korsback, 2021). For anyone that has worked with BGP before, the ROA should be easily recognizable. ROA essentially signs the route so that enterprise validators can testify that the route being advertised is really the route that is being taken. Without the validation of routing updates, any criminal or nation-state can hijack the routing infrastructure to substitute whatever path they desire. Therefore, the rules of BGP should still apply here, accept the longest prefix per source to destination pairings, filter prefixes that would cause unwanted routing ASN transits, and much more. ROA instills the real equity needed to permit the functionality of ROV, with which RPKI benefits are seen. ROV’s purpose is simple, take in the ROA state, apply it to the enterprise’s TAL, if the information is not present go further up the chain to one of the RIRs service area’s (Korsback, 2021). Once independently verified and independently signed, the signed object is registered in each RIR repository so that every ISP for a given region can verify the integrity of routes in real-time (Korsback, 2021). After the information is posted to a RIR repository, cryptographic validators, usually separate from the BGP router itself, can obtain three states: valid, unknown, and invalid (Korsback, 2021). What an organization does with these three states are dependent on the corporation’s security posture, or risk acceptance. For example, the CISSP recommends that the C-suite level executives set forth a risk appetite that the company can follow. If there is a greater risk appetite, than the organization may decide to accept valid and unknown ROVs. However, if the company is risk adverse, then they are likely to only accept valid ROV. To summarize the ROA, ROV, and RIR registry process can be seen through a college plagiarism database analogy, perhaps just like the one we submit our paper to. Susie is taking an extremely hard English course for her Master’s in cybersecurity online degree program. Since she is an expert in web server exploitation, she decides program. Since she is an expert in web server exploitation, she decides to write a ten-page paper on the subject. Over a 12-week course, she wrote an elegant paper that consisted of properly cited sources and all the requirements set forth by the professor. After turning in her paper, the professor gave it to Plagiarism.com to check for integrity of original work. Plagiarism has stored papers in the database that was check against her writing, the absence of common writing signatures allowed the professor to pass Susie. Furthermore, once the paper was submitted by the professor, Plagiarism.com, that website stored the writing sample in its database and shared the sample across the plagiarism validator industry. This college writing example is very similar to how ROA and ROV work in the real world with BGP. For example, the databases where industry organizations store their information can either be private or publicly accessible, such as the Routing Arbiter Database (RADb) (Durand, 2020). The purpose of RPKI is to provide a large-scale, publicly available RPKI architecture to ensure secure BGP routing, but such does not stop private implementations of RPKI. Private extensions of RPKI will be discussed later in a newly thought up deployment scenario.

Poor implementation and maintenance practices will leave security holes in RPKI’s cryptographic integrity. Last year, President Trump and the Department of Justice’s Attorney General William P. Barr unexpectedly agreed with banning the China Telecom Corporation due to vast BGP hijackings across the Internet (Vaughan-Nichols, 2020). To make matters worse, last year also saw Russia’s Rostelecom hijack BGP traffic across hundreds of content delivery networks, deliberately siphoning data from companies like Amazon, Facebook, and Google (Doffman, 2020). Putting faulty contact information down, or frivolous origin ASN naming, can confuse legitimate customers and can tie up other organization’s resources (Durand, 2020). The difficulty with managing legitimate credentials is that the rest of the organization’s security needs to be accounted for. For example, owning, deploying, and maintaining a public key architecture is not cheap, and may not be economically feasible for all organizations. In the operational environment, there may only be one, or a few departments, that has a public key architecture in place. If that department has high employee turnover, or a large influx of new cyber professionals onboarding sessions, it is easy to forget to update the contact information associated with the ROA, or more specifically the X.509. Having differing maximum prefix lengths could filter out legitimate routing updates due to BGP choosing the longest prefix match as the standard preference (Durand, 2020). As stated earlier, if all attributes are the same up until the prefix checking step within the BGP best path selection process, then a variance in standard maximum prefix length advertisements may have drastic downstream effects. An example can easily be seen by way of an upstream BGP router truncating a 24-bit prefix mask to an 18-bit prefix mask. If any downstream routers are dependent on the 24-bit prefix mask selection criteria, because of perhaps an implemented downstream inbound prefix filtering list, then the information would not be delivered to the downstream router’s router information base (RIB) or existing IP routing table. Stale data can poison data caches and could be run by malicious domain users if the information is never reported through the proper channels (Durand, 2020). Poor cyber hygiene with respect to inventory management could lead to attackers gaining control of organization owned RPKI credentials.

Two Years in the Making

Current RPKI implementations are in the stage of proofs of concepts and industry buy-ins, with full-scale deployments being scheduled for the near-future: perhaps this summer. There are people hard at work designing and testing various implementation schemes for their organization’s RPKI infrastructure. One logical extension to their work would be to generate private, crypto accelerators by way of cached ROAs. Instead of having to solicit the publicly available RADbs every time, it would make sense for large companies to cache frequently used ROAs and run a validator to derive a ROV certainty level after a defined period. Such is similar to how Wide Area Network (WAN) accelerators work, implementations of various degrees of certainty for RPKI validation could accelerate customer traffic through shortcuts made by the Internet Service Providers (ISP). There are security issues with this concept, but just as vendors bypass firewalls when the information is passed through a verified, trusted encrypted tunnel, such could be the same with RPKI accelerators. Moving from private implementations of RPKI to cryptographic considerations, there is a pressing need to address the inherit vulnerability in stale public key encryption algorithms associated with the X.509 certificate. Hence, with Cisco’s Post-Quantum Cryptography timeline perhaps coming true seven years early, this year could roughly be the year for quantum supremacy for cryptanalysts. If that is in fact true, then that leads to questioning of the integrity of the security revolving around SHA256 (Cisco). Therefore, it is important for the enterprise’s network infrastructure and certificate capabilities to be as adaptable as possible. A security team must always be prepared for new, time-sensitive cybersecurity guidance requiring an immediate remediation of compromised, classically equivalent encryption strength level public key algorithms.

Conclusion and Updates

Leveraging cryptographically robust BGP validators and BGP secured infrastructure, through RPKI, will ensure the CIA triad. After going through the various security topics covered in this paper, it should now be evident that securing BGP will take an industry-wide effort throughout the lifetime of the RPKI implementation. Although efforts will be needed to maintain certificate and certificate validation integrity, keeping X.509 certificates up to date with the most secure cryptographic algorithms will insure Information Assurance (IA) across the community. As reviewed with Cisco's Post-Quantum Security brief in July of 2020, the community does not exactly know what is in store for future cryptanalyst quantum supremacy breakthroughs. No one knows what is in stored for quantum cryptanalysis accelerators. Over the last few years there has been much excitement around any possibility of parallelizing Grover's algorithm against symmetric key encryption, a cryptanalyst quantum supremacy breakthrough may in fact see just that. Moving away from future research endeavors, and focusing on current architecture considerations, managing security expectations has proved to be important with RPKI. Therefore, maintaining a security conscious mindset will mitigate pitfalls seen with the RPKIs, to include frivolous origin ASN naming, differing maximum prefix lengths, stale data, and improver inventory management. Understanding and applying mitigations to all of these pitfalls are crucial for keeping IA in the eyes of the customers. That means that the industry, should they choose to accept RPKI buy-in, needs to be vigilant and proactive throughout the entire lifecycle of their deployed infrastructure. Finally, proprietary RPKI accelerators will enable operationally acceptable performance standards for customer transit times. Looking into speeding up cryptographic methods is always a challenge, but the recommended objectives stated earlier in the paper can help navigate how to optimize the customer-provider and provider-RIR sides. Third party audits of an organization's RPKI accelerator can placate industry-wide CIA concerns of bypassing the security mechanisms put in place. Overall, RPKI has a bright future in hardening the Internet backbone infrastructure of tomorrow.

References

1. Abedin, B. (2021). Managing the tension between opposing effects of explainability of artificial intelligence: a contingency theory perspective. https://www.emerald.com/insight/content /doi/10.1108/INTR-05-2020-0300/full/pdf

2. American Chronicle. (2021). Nakasone says US works to stay ahead of cybersecurity curve. https://americanchronicle.com/?p=345

3. Anderson, S. (2019). NAVFAC works to cyber-secure short IT infrastructure with FRCS standardization. https://www.doncio.navy.mil /(a22r1y55pfqb21mqr0uphe55)/CHIPS /ArticleDetails.aspx?ID=12846

4. Arana, P. (2006). Benefits and vulnerabilities of Wi-Fi Protected Access 2 (WPA2). https://cs.gmu.edu/~yhwang1/INFS612 /Sample_Projects/Fall_06_GPN_6_Final_Report.pdf

5. Arista. (2021). Security advisory 0063. https://www.arista.com/en/support/advisories-notices/security-advisories/12602-security-advisory-63

6. Avast. (2021). What is an attack surface? https://www.avast.com/en-in/business/resources /what-is-attack-surface

7. Ball, M. (2020). Manned-unmanned teaming demonstrated with Navy EW aircraft. https://www.unmannedsystemstechnology.com/2020/02/manned-unmanned-teaming-demonstrated-with-navy-ew-aircraft/

8. Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R. (2018). Recommendation for Pair-WiseKey-Establishment Schemes Using Discrete Logarithm Cryptography. https://www.govinfo.gov/content/pkg/GOVPUB-C13-c4177b1be885e6692e676525fd6d0642/pdf/GOVPUB-C13-c4177b1be885e6692e676525fd6d0642.pdf

9. Barter, A. (2018). Cryptanalysis. https://alexbarter.com/cryptanalysis/

10. Bell, D. (2004). The UML 2 class diagram. https://developer.ibm.com/articles/the-class-diagram/

11. Bergamaschi, F., Daniel, R., Levy, R. (2020). The future of crypto: IBM makes a new leap with Fully Homomorphic Encryption. https://www.ibm.com/blogs/research/2020/12/fhe-progress-milestones/

12. Bigelow, S. (2016). Blue/green deployment. https://searchitoperations.techtarget.com/definition

/blue-green-deployment

13. Buni, J. (2019). How cloud computing will power smart cities. https://techhq.com/2019/06/how-cloud- computing-will-power-smart-cities/ (Links to an external site.)

14. CA DMV. (2021). Driving test criteria. https://www.dmv.ca.gov/portal/handbook/driving-test-criteria/pre-drive-checklist-safety-criteria/

15. CERN Computer Security. (2021). Common vulnerabilities guide for C programmers.

16. CVE. (2020). CVE-2020-26142. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-26142

17. CVE. (2020). CVE-2020-26147. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-26147

18. Chapple, M., Michael, J., Gibson, D. (2018). Certified Information Systems Security Professional. ISC2.

19. Chapple, M., Stewart, J., Gibson, D. (2018). Certified Information Systems Security Professional. ISC2.

20. Chen, H., Qiu, W., Ou, H., An, B., Tambe, M. (2021). Contingency-aware influence maximization: a reinforcement learning approach. https://arxiv.org/pdf/2106.07039.pdf

21. Choi, J., Guo, Y., Moczulski, M., Oh, J., Wu, N., Norouzi, M., Lee, H. (2019). Contingency-aware exploration in reinforcement learning. https://arxiv.org/pdf/1811.01483.pdf

22. Cimpanu, C. (2019). NASA hacked because of unauthorized Raspberry Pi connected to its network. https://www.zdnet.com/article/nasa-hacked-because-of-unauthorized-raspberry-pi-connected-to-its-network/

23. Cimpanu, C. (2019). Yubico to replace vulnerable YubiKey FIPS security keys. https://www.zdnet.com/article/yubico-to-replace-vulnerable-yubikey-fips-security-keys/

24. Cisco. (2020). 350-401 ENCOR Exam: Implementing Cisco Enterprise Network Core Technologies. https://learningnetwork.cisco.com/s/encor-exam-topics

25. Cisco. (2020). CCNP and CCIE Enterprise Core: ENCOR 350-401. Cisco Press.

26. Cisco. (2021). Cisco Secure Malware Analytics (formerly Threat Grid) Partners and Integrations. https://www.cisco.com/c/en/us/products/security/threat-grid/integrations.html

27. Cisco. (2021). FMC Ansible Modules. https://github.com/CiscoDevNet/FMCAnsible

28. Cisco. (2021). Overlay Transport Virtualization. https://www.cisco.com/c/en/us/solutions/data-center-virtualization/overlay-transport-virtualization-otv/index.html

29. Cloud Flare. (2021). What Is BGP? | BGP Routing Explained. https://www.cloudflare.com/learning/security/glossary/what-is-bgp/

30. Code Bless U. (2021). Size of Python data types. https://codeblessu.com/python/size-of-data-types.html

31. Comodo. (2021). Is Splunk a SIEM? https://www.comodo.com/is-splunk-a-siem.php

32. Cowan, C., Wagle, P., Pu, C., Beattie, S., Walpole, J. (2000). Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. https://www.cs.utexas.edu/~shmat/courses/cs380s_fall09/cowan.pdf

33. Crabtree, J. (2021). BGP Attacks Pose A Substantial Operation Risk -- Are Enterprises Paying Attention? https://www.forbes.com/sites /forbestechcouncil/2021/01/11/bgp-attacks-pose-a-substantial-operation-riskare-enterprises-paying-attention/?sh=4591535a45ba

34. Cyber Hedge. (2020). Vast majority of CEOs will be personally liable for breaches by 2024: Gartner. https://cyberhedge.com/insights/daily/2020/09 /07/vast-majority-of-ceos-will-be-personally-liable-for-breaches-by-2024-gartner/#:~:text=Summary,) %E2%80 %9D%20attacks%20anticipated%20by%202024.& text=Gartner%20also%20projects%20that%20financial,over %20%2450%20billion%20by%202023.

35. Dano, M. (2019). 5G May Drive Carriers to Share Networks – Like It or Not. https://www.lightreading.com/mobile/5g/5g-may-drive-carriers-to-share-networks-andndash-like-it-or-not-/d/d-id/752820

36. DeCarlo, A., Ferrell, R. (2021). The 5 different types of firewalls explained. https://searchsecurity.techtarget.com/feature/The-five-different-types-of-firewalls

37. Delua, J. (2021). Supervised versus unsupervised learning: what’s the difference? https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

38. Dhillon, G. (2016). What to do before and after a cybersecurity breach? https://www.american.edu/kogod/research/cybergov/upload/what-to-do.pdf

39. Doffman, Z. (2020). Russia And China ‘Hijack’ Your Internet Traffic: Here’s What You Do. https://www.forbes.com/sites/zakdoffman/2020/04/18/russia-and-china-behind-internet-hijack-risk-heres-how-to-check-youre-now-secure/?sh=6443c2ed5b16

40. Durand, A. (2020). Resource Public Key Infrastructure (RPKI) Technical Analysis. https://www.icann.org/en/system/files/files/octo-014-02sep20-en.pdf

41. Eastwood, B. (2020). The 10 most popular programming languages to learn in 2021. https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/

42. Ehsan, U., Wintersberger, P., Liao, Q., Mara, M., Streit, M., Wachter, S., Riener, A., Riedl, M. (2021). Operationalizing human-centered perspectives in explainable AI. https://www.researchgate.net/profile/Philipp-Wintersberger/publication/351426399_Operationalizing_Human-Centered_Perspectives_in_Explainable_AI/links/6097d958a6fdccaebd19ff2f/Operationalizing-Human-Centered-Perspectives-in-Explainable-AI.pdf

43. Gatlan, S. (2019). WPA3 Wi-Fi Standard Affected by New Dragonblood Vulnerabilities. https://www.bleepingcomputer.com/news/security/wpa3-wi-fi-standard-affected-by-new-dragonblood-vulnerabilities/

44. Geeks for Geeks. (2020). Python traceback. https://www.geeksforgeeks.org/python-traceback/?ref=lbp

45. Gite, V. (2022). Understanding /etc/shadow file format on Linux. https://www.cyberciti.biz/faq/understanding-etcshadow-file/

46. Gloukhovtsev, M. (2020). HOW 5G TRANSFORMS CLOUD COMPUTING. https://education.dellemc.com/content/dam/dell-emc/documents/en-us/2020KS_Gloukhovtsev_How_5G_Transforms_Cloud_Computing.pdf

47. Good Reads. (2021). Rocky Balboa quotes. https://www.goodreads.com/work/quotes /10446115-rocky-balboa

48. Goodin, D. (2020). NSA says Russian state hackers are using a VMware flaw to ransack networks. https://arstechnica.com/information-technology/2020/12/nsa-says-russian-state-hackers-are-using-a-vmware-flaw-to-ransack-networks/

49. Goodison, D. (2020). 10 Future Cloud Computing Trends To Watch In 2021. https://www.crn.com /news/cloud/10-future-cloud-computing-trends-to-watch-in-2021

50. Google American Fuzzy Lop. https://github.com /google/AFL

51. Google. (2021). Google Cloud, Nokia Partner to Accelerate Cloud-Native 5G Readiness for Communications Providers. https://cloud.google.com/press-releases/2021/0114 /google-cloud-nokia-telecommunications (Links to

an external site.)

52. Haldorai, A., Ramu, A. (2020). Security and channel noise management in cognitive radio networks. https://www.sciencedirect.com/science/article/abs/pii/S004579062030639X

53. Harris, S., Maymi, F. (2019). All-in-one CISSP exam guide. Mc-Graw Hill Education.

54. Harris, S., Maymi, F. (2019). All-in-one CISSP exam guide. McGraw Hill Education.

55. Harris, S., Maymi, F. (2019). All-in-one CISSP exam guide. McGraw Hill Education.

56. He, S., He, P., Chen, Z., Yang, T., Su, Y., Lyu, M. (2021). A Survey on Automated Log Analysis for Reliability Engineering. https://arxiv.org/pdf/2009.07237.pdf

57. Hong Kong CERT. (2017). WiFi Protected Access II (WPA2) Multiple Vulnerabilities (KRACK). https://www.hkcert.org/security-bulletin/wifi-protected-access-ii-wpa2-multiple-vulnerabilities-krack

58. Housley, R. (2021). Enhanced JSON Web Token (JWT) claim constraints for secure telephone identity revisited (STIR) certificates. https://www.rfc-editor.org/rfc/rfc9118.pdf

59. Howe, D. (2019). Exceeding authorized access. https://www.venafi.com/blog/deciphering-how-edward-snowden-breached-the-nsa (Links to an external site.)

60. Ingals, S. (2020) Firewalls as a Service. (FWaaS): The Future of Network Firewalls? https://www.esecurityplane.com/cloud/firewalls-as-a-service-fwass/

61. Ivanov, R. (2021). Cisco secure firewall: 7.0 release preview. https://www.cisco.com/c/dam/global/ru_ru/training-events/on-demand-webinars/may-20-2021-version-7-0-of-the-firewall-software.pdf

62. Jones, A. (2021). China launches hyperspectral Earth observation satellite. https://www.space.com/china-long-march-4c-launches-gaofen-5-02

63. Jones, M., Bradley, J., Sakimura, N. (2015). JSON Web Token (JWT). https://www.rfc-editor.org/rfc/pdfrfc/rfc7519.txt.pdf

64. Korsback, F. (2021). How AWS is helping to secure internet routing. https://aws.amazon.com/blogs/networking-and-content-delivery/how-aws-is-helping-to-secure-internet-routing/

65. Langston, J. (2019). With a “hello,” Microsoft and UW demonstrate first fully automated DNA data storage. https://news.microsoft.com/innovation-stories/hello-data-dna-storage/ (Links to an external site.)

66. Leetcode. (2021). Weekly Contest 237. https://leetcode.com/contest/weekly-contest-237/problems/single-threaded-cpu/

67. Levy, M. (2018). RPKI - The required cryptographic upgrade to BGP routing. https://blog.cloudflare.com/rpki/

68. Lockheed Martin. (2021). The Cyber Kill Chain. https://www.lockheedmartin.com/en-us/capabilities /cyber/cyber-kill-chain.html

69. Lori. (2019). NGFW logs. https://www.kaggle.com /lorisaysrawr/ngfw-logs

70. MITRE. (2021). CVE-2021-0249. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-0

71. MITRE. (2021). CVE-2021-3482. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3482

72. MSN Money. (2021). https://www.bing.com/search?q=palo+alto+stock&cvid=fe1099872b0641d8b9529da660c70303&aqs=edge.0.0l9.8071j0j4&FORM=ANAB01&PC=HCTS

73. McDaniel, T. Smith J., Schuchard, M. (2021). Flexsealing BGP Against Route Leaks: Peerlock. Active Measurement and Analysis. https://arxiv.org/pdf/2006.06576.pdf

74. Miller, R. (2020). Microsoft: Your Cloud Data May Soon Be Stored in DNA and Holograms. https://datacenterfrontier.com/microsoft-your-cloud-data-may-soon-be-stored-in-dna-and-holograms/

75. Mojidra, N. (2020). Stateful versus stateless firewalls. https://www.cybrary.it/blog/0p3n/stateful-vs-stateless-firewalls/

76. Monty Python’s Official Website. (2021). Monty Python and the Programming Language. https://pythonspamclub.com/monty-python-and-programming-language/

77. Muncaster, P. (2021). Forrester predicts mass cybersecurity brain drain. https://www.infosecurity-magazine.com/news/forrester-predicts-mass/

78. NIST. (2011). Final version of NIST cloud computing definition published. https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published (Links to an external site.)

79. NIST. (2013). Security and Privacy Controls for Federal Information Systems and Organizations. https://csrc.nist.gov/publications/detail/sp/800-53/rev-4/final

80. NIST. (2016). BGP Secure Routing Extension (BGP-SRx) Prototype.

81. NIST. (2016). Submission requirements and evaluation criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

82. NIST. (2016). Work with us: we collaborate with experts from industry, government, and academia. https://www.nccoe.nist.gov/sites/default/files/library/fact-sheets/work-with-us-fact-sheet.pdf

83. NIST. (2021). Cloud Computing. https://csrc.nist.gov/projects/cloud-computing

84. NIST. (2021). Information assurance. https://csrc.nist.gov/glossary /term/information_assurance

85. NIST. (2021). NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/about-rmf

86. NIST. (2021). New online tool to improve stakeholder engagement with SP 800-53. https://csrc.nist.gov/News/2021/new-online-tool-for-engaging-with-sp-800-53

87. NSA. (2005). NSA Information Assurance Director Speaking at the 14th Annual RSA Conference, 14-18 February 2005 – NSA/CSS. https://www.nsa.gov/news-features/press-room/Article/1633812/nsa-information-assurance-director-speaking-at-the-14th-annual-rsa-conference-1/

88. NSA. (2011). United States Signals Intelligence Directive. https://www.dni.gov/files/documents /1118/CLEANEDFinal%20USSID%20SP0018.pdf

89. NSA. (2018). Cloud Security Basics. https://apps.nsa.gov/iaarchive/customcf /openAttachment.cfm?FilePath=/iad/library/ia-guidance/security-tips/assets/public/upload/Cloud-Security-Basics.pdf& WpKes=aF6woL7fQp3dJihzxcGMSVxaQxM46fptuHkqRF

90. NSA. (2021). FAQs about Signals Intelligence. https://www.nsa.gov/about/faqs/sigint-faqs/

91. NSA. (2021). German cipher machines of World War II. https://www.nsa.gov/portals/75/documents /about/cryptologic-heritage/historical-figures-publications/publications/wwii/german_cipher.pdf

92. Nair, R. (2021). WPA3 – the new WiFi security protocol. https://techsphinx.com/hacking/wpa3-security-protocol/

93. Norton. (2018). What you need to do about the WPA2 Wi-Fi network vulnerability. https://us.norton.com/internetsecurity-emerging-threats-what-to-do-about-krack-vulnerability.html

94. OWASP. (2021). Buffer Overflow. https://owasp.org/www-community/vulnerabilities /Buffer_Overflow 249

95. OWASP. (2021). Fuzzing. https://owasp.org/www-

community/Fuzzing

96. Oracle. (2010). Key Length and Encryption Strength. https://docs.oracle.com/cd/E19424-01/820-4811/aakfw/index.html

97. Palo Alto Networks. (2020). Palo Alto Networks launches world's first ML-powered NGFW. https://www.paloaltonetworks.com/company/press/2020/palo-alto-networks-launches-worlds-first-ml-powered-ngfw

98. Pearson IT Certification. (2011). Understanding the three factors of authentication. https://www.pearsonitcertification.com/articles/article.aspx?p=1718488

99. Pike, J. (2000). Measurement and Signature Intelligence (MASINT). https://fas.org/irp/program/masint.htm

100. Practical Cryptography. (2012). Affine cipher. http://practicalcryptography.com/ciphers/classical-era/affine/

101. Practical Cryptography. (2012). Chi-squared statistic. http://practicalcryptography.com/cryptanalysis/text-characterisation/chi-squared-statistic/

102. Practical Cryptography. (2012). Index of coincidence. http://practicalcryptography.com /cryptanalysis/text-characterisation/index-coincidence/

103. Practical Cryptography. (2012). Quadgram Statistics as a Fitness Measure. http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/

104. Prado, K. (2019). Understanding time complexity with Python examples. https://towardsdatascience.com/understanding-time-complexity-with-python-examples-2bda6e8158a7

105. Python Geeks. (2021). Inheritance in Python with types and examples. https://pythongeeks.org /inheritance-in-python/

106. Quackit. (2021). Python 3 exception hierarchy. https://www.quackit.com/python/reference /python_3_exception_hierarchy.cfm

107. Rakhmetova, E., Combi, C., Fruggi, A. (2021). Conceptual modelling of log files: from a UML-based design to JSON files. http://ceur-ws.org /Vol-2958/paper3.pdf

108. Ranjith. (2020). Wacker: a WPA3 dictionary cracker. https://kalilinuxtutorials.com/wacker/

109. Regalado, P. (2020). Splunk Named a Leader in the Gartner SIEM Magic Quadrant For Seven Years Running! https://www.splunk.com/en_us /blog/security/splunk-named-a-leader-in-the-gartner-siem-magic-quadrant-for-seven-years-running.html

110. Robertson, J., Riley, M. (2018). The big hack: how China used a tiny chip to infiltrate US companies. https://www.bloomberg.com/news/features /2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies

111. Rodriguez, I. (2021). Inheritance and composition: a Python OOP guide. https://realpython.com/inheritance-composition-python/#whats-composition

112. Rojas, J. (2018). IP network traffic flows labeled with 75 apps. https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps/metadata

113. SSL. (2020). Key generation and attestation with Yubikey. https://www.ssl.com/how-to/key-generation-and-attestation-with-yubikey/

114. Sbeglia, C. (2021). Wi-Fi 7: what is it and when should you expect it? https://www.rcrwireless.com/20210127/network-infrastructure/wi-fi/wi-fi-7-what-is-it-and-when-should-you-expect-it

115. Security Trails. (2021). Intrusion detection systems: types, detection methods, and challenges. https://securitytrails.com/blog/intrusion-detection-systems

116. Sentinel One. (2019). Threat actor basics: the five main threat types. https://www.sentinelone.com/blog/threat-actor-basics-understanding-5-main-threat-types/

117. Shepherd, A. (2020). Why WPA3 may be no safer than WPA2. https://www.itpro.com/security/30848/why-wpa3-may-be-no-safer-from-attack-than-wpa2

118. Sickcodes. (2021). CVE-2021-29921.

119. Silberman, P., Johnson, R. (2004). A comparison of buffer overflow prevention implementations and weaknesses. https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-silberman/bh-us-04-silberman-paper.pdf

120. Smith-Tone, D. (2019). Practical Cryptanalysis of k-ary C*. https://eprint.iacr.org/2019/841.pdf

121. Snyder, D., Mayer, L., Weichenberg, G., Tarraf, D., Fox, B., Hura, M., Genc, S., Welburn, J. (2020). Measuring cybersecurity and cyber resiliency. https://www.rand.org/content/dam/rand/pubs/research_reports/RR2700/RR2703/RAND_RR2703.pdf

122. Sound, S. (2017). 1G, 2G, & 5G: The evolution of the G’s. https://mse238blog.stanford.edu/2017/07/ssound/1g-2g-5g-the-evolution-of-the-gs/

123. Stine, K., Kissle, R., Barker, W., Fahlsing, J., Gulick, J. (2008). Volume 1: Guide for Mapping Types of Information and Information Systems to Secure Categories. https://www.govinfo.gov/content/pkg/GOVPUB-C13-9af73460e5ff9ede8820abdf3a50041d/pdf/GOVPUB-C13-

124. Stirling, K. (2019). Hack the Machine 2019 sees big participation by Warfare Centers. https://www.navsea.navy.mil/Media/News/Article/1959310/hack-the-machine-2019-sees-big-participation-by-warfare-centers/

125. Suskovics, P. (2020). Creating the next-generation edge-cloud ecosystem. https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/next-generation-cloud-edge-ecosystems

126. Syed, O. (1999). The story of how chess was invented. http://arimaa.com/arimaa/links/chessStory.html

127. Thomson, I. (2016). Microsoft researchers smash homomorphic encryption speed barrier. https://www.theregister.com/2016/02/09/researchers_break_homomorphic_encryption/

128. Unick, C. (2020). Mobile threat visibility in FireEye Helix with Zimperium. https://www.fireeye.com/blog/products-and-services/2020/02/mobile-threat-visibility-in-fireeye-helix-with-zimperium.html

129. Vaughan-Nichols, S. (2020). DOJ urges FCC to revoke China Telecom's license. https://www.zdnet.com/article/doj-urges-fcc-to-revoke-china-telecoms-license/

130. Venafi. (2021). The future of security lies in Machine Identity Management. https://techcrunch.com/sponsor/venafi/the-future-of-security-lies-in-machine-identity-management/

131. Virginia. (2021). Virginia: department of elections. https://www.elections.virginia.gov/

132. Wang, S., Li, G., Yao, X., Zeng, Y., Pang, L., Zhang, L. (2019). A distributed storage and access approach for massive remote sensing data in MongoDB. International Journal of Geo-Information.

133. Wang, S., Li, G., Yao, X., Zeng, Y., Pang, L., Zhang, L. (2019). A distributed storage and access approach for massive remote sensing data in MongoDB. International Journal of Geo-Information.

134. Westover, B. (2019). Wi-Fi 7: The Next Big Thing in Wireless Internet. https://www.tomsguide.com /special-report/wi-fi-7-explained

135. Wi-Fi Alliance. (2020). Discover Wi-Fi: security. https://www.wi-fi.org/discover-wi-fi/security

136. Wi-Fi Alliance. (2021). Wi-Fi CERTIFIED WPA3™ December 2020 update brings new protections against active attacks: Operating Channel Validation and Beacon Protection. https://www.wi-fi.org/beacon/thomas-derham-nehru-bhandaru/wi-fi-certified-wpa3-december-2020-update-brings-new-protections

137. Wikipedia. (2021). Milky Way. https://en.wikipedia.org/wiki/Milky_Way

138. Williams, L. (2021). Navy looks beyond RMF to build cyber resilience. https://defensesystems.com /articles/2021/08/17/navy-cyber-rmf-software.aspx

139. Yackel, R. (2020). What is an X.509 Certificate? Understanding this Vital Safeguard. https://blog.keyfactor.com/what-is-x509-certificate

140. Zhang, Y., Ruan, C., Li, C., Yang, X., Cao, W., Li, F., Wang, B., Fang, J., Wang, Y., Huo, J., Bi, C. (2021). Towards cost-effective and elastic cloud database deployment via memory disaggregation. https://www.vldb.org/pvldb/vol14/p1900-zhang.pdf

141. Zimperium. (2021). Mobile threat defense and cyber security for mobile devices. https://www.zimperium.com/why-zimperium

142. eRacks. (2021). RAID Levels. https://eracks.com /raid-levels/ (Links to an external site.)