Algoritmos de resolución de problemas aplicados a ataques de canal cruzado y ataques de tiempo.

Rodrigo Castillo Camargo, Juan Camilo Hoyos

August 22, 2022

1 Introducción

En el mundo de la seguridad informática los ataques de canal cruzado (Side Channel Attacks) son los ataques que se aprovechan del funcionamiento de las máquinas, esto, porque al ser las máquinas sistemas dependientes de la física y no únicamente de la lógica, dependen de limitaciones físicas que pueden ser explotadas por un atacante.
Este artículo se enfoca principalmente en los ataques de tiempo, que son los ataques que vulneran la seguridad de las máquinas por el tiempo que toma el procesador de éstas en resolver operaciones matemáticas y/o lógicas.
Existen muchos tipos de ataques de tiempo, uno de los mas famosos es un ataque a las llaves criptográficas vulnerando el tiempo que se demora la operación del exponente modular: Timing Attacks - Paul C. Kotcher.
Este documento se enfocará únicamente en el ataque trivial de un ataque de tiempo aplicado a una función de comparación de strings usual implementada en Python.

```python
def simple_comp(str1, str2):
    """Intuitive comparison function."""
    if(len(str1) != len(str2)):
        return False
    else:
        for i in range(len(str1)):
            if(str1[i] != str2[i]):
                return False
        return True
```

Aunque esta función lógicamente funciona, su implementación en funciones sensibles como lo puede ser una comparación con un password, como la del siguiente ejemplo es vulnerable puesto que la máquina puede tomar distintos comportamientos dependiendo de los inputs que se proporcionen; Por ejemplo, la cadena "AAAAAAAA" tomará más tiempo de procesamiento que la cadena "AAA" puesto que la comparación de la longitud es verdadera, a su vez, la cadena "PAAAAAAA" tomará más tiempo que la cadena "AAAAAAAA" puesto que la comparación del primer carácter es verdadero y el programa tendrá que comparar el siguiente. Siguiendo esta lógica, un atacante puede medir los tiempos que toman cadenas en específico y reconstruir la contraseña a partir de inferir los tiempos de procesamiento.

```python
def super_secret_password(user_input):
    """Secret password comparisson against the given string."""
    secret = "PASSWORD"
    if(simple_comp(user_input, secret)):
        return True
    return False
```

Sin embargo, al ser los procesadores modernos tan rápidos y complejos, este proceso de reconstrucción puede fallar, esto se debe a que los procesadores pueden tomar acciones del sistema operativo que

retrasen alguna operación aleatoria. Esto obligará al atacante a tener que apoyarse en el teorema del límite central para poder reconstruir correctacmente la contraseña

Aún recurriendo al teorema del límite central, existen procesos muy largos que pueden afectar significativamente los tiempos de ejecución del programa en específico, esto puede inducir al atacante a errores en el proceso. Para esto, el atacante puede verificar si indujo un error previamente puesto que, al haber inducido un error, los resultados de tiempo que obtendrá en la faces siguientes del juego serán prácticamente aleatorios, de esto pasar, el atacante tendrá que eliminar los últimos caracteres que haya añadido y seguir atacando desde estados previos..

En este documento se busca explorar diferentes tipos de algoritmos para evaluar su eficacia al momento de romper una contraseña por medio de ataques de tiempo.

2 Formalización del problema

El problema de reconstruir una contraseña a partir de ataques de tiempo puede ser visto como un juego en el cuál el atacante dispone de una cantidad k de casillas en las cuales puede ubicar cualquiera de las 26 letras del alfabeto. de esta forma, tenemos un entorno que:
1. No es completamente observable, puesto que el agente no conoce todo el entorno.
2. Agente único, puesto que solamente el atacante decide sobre las letras que dispone en las k casillas.
3. Secuencial, puesto que cada letra que se ubica en la casilla c_k afecta el tiempo de ejecución de las otras casillas.
4. Discreto puesto que existe una cantidad finitade casillas y letras.
5. Conocido, puesto que el atacante conoce las reglas del juego.

El atacante está buscando un algoritmo que le proporcione restaurar la contraseña de la víctima con la menor cantidad de pasos posibles, es decir, con la menor cantidad de transiciones entre palabras.

2.1 Estado inicial

El atacante no tiene previo conocimiento de la contraseña de la víctima, por lo que el estado inicial será el string vacío.

2.2 Posibles acciones

Dado un string, el atacante podrá concatenarle a la izquierda cualquiera de las 26 existentes o eliminar la última.

2.3 Función de transición

dado un string s y una letra del abecedario k el atacante podrá concatenar $s + k_i$

2.4 Prueba de satisfacción del objetivo

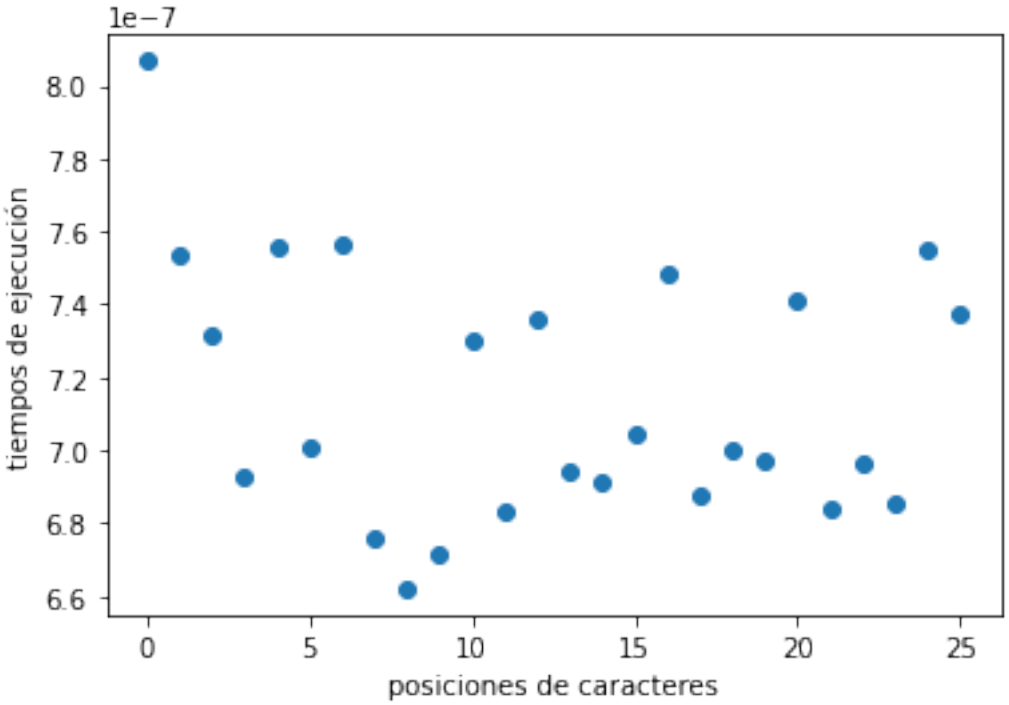
El atacante sabrá que reconstruyó la contraseña en el momento de que la función de password retorne un valor verdadero.

3 Métodos

3.1 Greedy algorithm

Greedy search o busqueda avara es el algoritmo que consiste en tomar la mejor desición posible en cada estado con el fin de llegar al objetivo de la forma más óptima posible, para esto, es necesario definir una métrica llamara "Heuristica" la cuál defina una medida cuantificable de que tan buena puede ser una acción. En este caso, una Heuristica puede ser el tiempo en el que se demora una máquina en rechazar una contraseña, pues, sabemos que si se demora mas, entonces es probable que la contraseña sea correcta. Sin embargo, se tiene el problema que se las máquinas tienden a tomar distintos tiempos

de ejecución al momento de ejecutar la misma acción, por lo que fue necesario definir la heurísitca como la media del tiempo resultado de ejecutar la misma acción en el mismo estado n iteraciones. Esta Heuristica es contraria a la heuristica en la cuál se toma como referencia al espacio, pues, entre mayor es el tiempo que toma un posible camino, es mejor.
Grafica que muestra el comportamiento de los diferentes tiempos de los diferentes inputs el cuál evidencia que la mejor opción es añadir la letra "A":



Teniendo un estado "A" , las posibles transiciones de este estado serían las cadenas de caracteres "AA" , "AB", "AC", ... "AZ". De las cuales, al medir los tiempos de ejecución en la función de comparación, se obtendría una lísta de la forma "AA:0,5", "AB:0.6" , "AC:0.5" , "A_n:0,5" , "AZ:0.5". de lo anterior, podríamos concluir que la mejor transición posible es la de tomar "AB" pues es la que mas tiempo toma.

Observe que esta "heuristica" nos ayuda a determinar la contraseña digito por digito, siguiendo la idea del algoritmo de busqueda avara. Note que se está dividiendo y encontrando una solución a cada sub-problema para al final tener una solución general.

3.2 Limited Greedy Algorithm

Similar al caso anterior con la particularidad de que se define un límite correspondiente a la cantidad máxima de caracteres que pueda tener el password. Así pues, de haber fallado en algún intento, el método será capaz de saber si hubo algún error. De no ser así, es posible que al haber un error el algoritmo de busqueda avara sin estar restringido llegue a la conclusión de que la contraseña es infinitamente larga aún sabiendo que tiene n caracteres.

3.3 Backtracking

Principalmente representado usando arboles o grafos, este método busca encontrar la solución a un problema buscando a partir de todos loshttps resultados posibles. Este algoritmo se detiene una vez prueba todas las posibilidades, para este caso empieza la búsqueda del primer digito de la clave, se hace una prueba de testeo para determinar si corresponde o no, en caso de que no sea así prueba con otro caracter hasta encontrar el que pasa el testeo. Podemos ver su representación en un árbol partiendo de la siguiente forma:

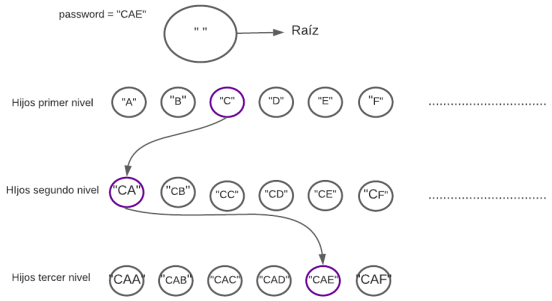


Figure 1: Representación del backtraking

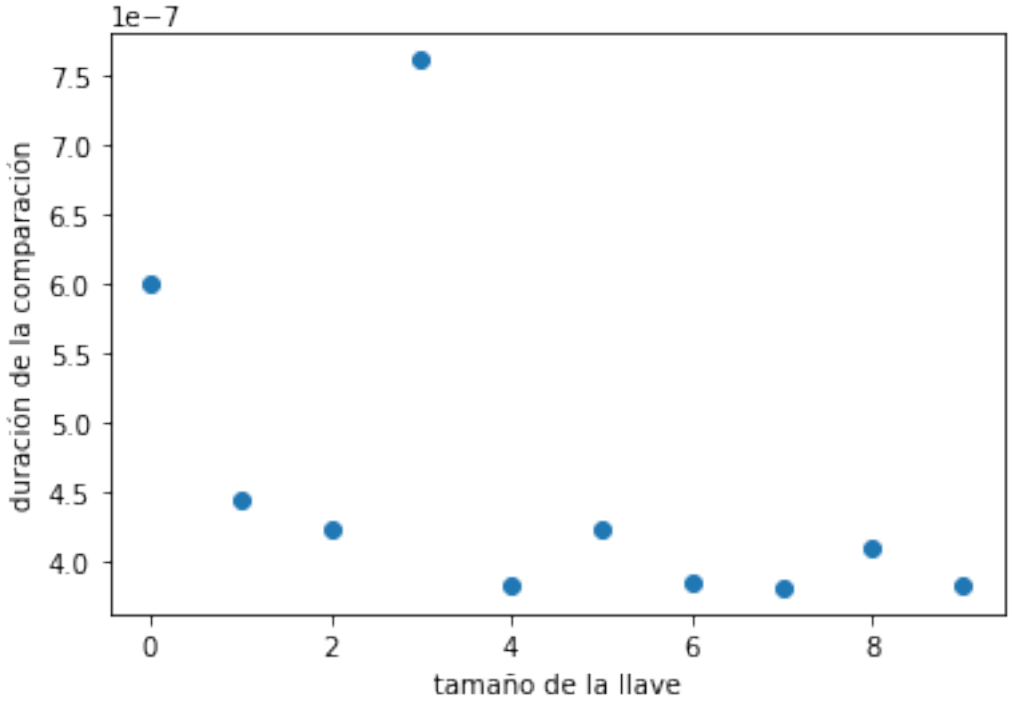
Notese que no se está recorriendo el arbol completo, más bien se va creando a medida se validan los digitos de la contraseña.
Para implementar el backtracking es necesario definir algún método que permita reconocer si la transición anterior estuvo errada. Para esto se usó el siguiente criterio:
Se tomó la misma heurística que se implementó en la busqueda avara.
Sabemos que un estado es erroneo sí en promedio todas las transiciones que se deriven de él tomarán el mismo tiempo en promedio, es decir, que si tomamos como referencia a la mejor transición de este estado, será casi lo mismo que tomar una transición aleatoria.
De esta forma, lo que se hizo fue tomar 3 veces la mejor transición y de esta manera, si las 3 son todas iguales, se sabe que el estado se encuentra en un camino correcto, sin embargo, de no ser todas iguales, se sabe que el estado está en un camino incorrecto.

3.4 brute-force

Este es un método trivial para decifrar contraseñas, el cual se basa en probar todas las combinaciones posibles de letras hasta dar con la contraseña. Sabemos que este método tiene una complejidad de $(O) = n^k$ donde n es la cantidad de caracteres que puede llevar la contraseña y $k$es el tamaño de la clave. Este método solo fue implementado con el fin de ser comparado con los otros métodos.

4 Resultados y discusión

Para todas las siguientes consideraciones se tomó como contraseña a la contraseña "ABC" de longitud 3. Para todos los métodos se usó la misma función para calcular la longitud de la contraseña, la cuál con 100.000 iteraciones por caracter retornó una la respuesta con una exactitud bastante precisa.
Grafica que muestra que la llave es de longitud 3:



4.1 Problemas del ambiente

Aún cuando es posible descifar la contraseña mediante ataques de tiempo, las varianzas de los tiempos en las máquinas son muy grandes, esto hace que para que los métodos funcionen sea necesario samplear una heurística de al menos 5.000.000 de iteraciones por cacacter. Esto hace que la implementación de los métodos anteriores, aunque no esté errada, sea muy demorada para poder tomar una muestra representativa de estas.

4.2 solución mediante Backtracking

La solución mediante backtracking es una solución posible, sin embargo, al ser el método responsable de evaluar si un camino es correcto o no tan demorado, hace que el algoritmo de backtracking pueda tomar varios días en su ejecución y eso hace que no se puedan tener métricas para este proyecto.

4.3 Solución mediante Bruteforcing

Este es un método que funciona para atacar contraseñas con una longitud corta, pues con un alfabeto de 26 caracteres la complejidad del algoritmo es de $O(n^{26})$ Sin embargo, se puede notar que para descifrar una contraseña de 3 caracteres es el método más eficaz.

4.4 Busqueda Avara - Greedy Search

El mejor algoritmo para descifrar una contraseña en este problema definiendo los ataques de tiempo fue el algoritmo de la busqueda avara, pues, aunque para un password de longitud 3 se demora aproximadamente 10 minutos en llegar a la solución optima, la complejidad de este algoritmo es de $O(n)$, lo que hace que sea mas eficiente que el algoritmo de fuerza bruta para contraseñas con mas caracteres.

4.4.1 Conclusiones

1. El mejor algoritmo para descifrar contraseñas mediante Timing Attacks es el algoritmo de busqueda avara. 2. El problema de crackear contraseñas mediante ataques de tiempo es un problema fascinante, sin embargo, al ser los procesadores tan variantes, es un problema que necesita de varias semanas para una correcta evaluación.