Hacia la detección de impostores mediante un modelo basado en tareas

Jorge Rodríguez-Ruiz, Raúl Monroy y J. Benito Camiña-Prado

Tecnologías de Información y Computación
Tecnológico de Monterrey, Campus Estado de México
Atizapán, Estado de México
{a00965439, raulm, a00965049}@itesm.mx

Resumen La detección de impostores es un problema abierto; resolverlo tendrá un fuerte impacto, debido al riesgo inherente de si alguien tomara poder de nuestro computador. Tradicionalmente, este problema se ha estructurado como la detección de una anomalía en las acciones (comandos) realizadas por un usuario; sin embargo, los resultados de este enfoque no han sido satisfactorios. Recientemente, se ha propuesto detectar impostores identificando anomalías en el uso de los objetos (del sistema de archivos); es decir, no es la acción, sino sobre qué se realiza ésta lo que caracteriza fielmente a un usuario. En este artículo, Se contribuye explorando esta idea un poco más; en particular, se considera que cada objeto puede asociarse con una tarea específica del usuario (por ejemplo, para un estudiante universitario, las tareas podrían ser: estudio, familia, etc.) El propósito final es establecer la siguiente hipótesis: mediante una secuencia relativamente corta de accesos a objetos podremos detectar la presencia de un impostor, pues traducida ésta en una secuencia de realizaciones de tareas se corrobore lo inusual que es con relación al perfil del usuario. En este artículo se detallan las bases del clasificador a construir, el tratamiento de las bitácoras de la base de impostores WUIL para extraer la información pertinente al modelo, y los experimentos que se realizarán para validar positiva o negativamente esta hipótesis.

Palabras clave: detección de intrusiones, conjunto de datos, grafos de navegación, seguridad computacional.

1. Introducción

Los ataques informáticos pueden originarse en el exterior o al interior de una organización (el hogar, una empresa, un cuerpo de gobierno, etc.) Los ataques internos son perpetrados dentro del perímetro seguro, por personas que disfrutan de nuestra confianza. Por ello, no es inusual que los atacantes internos tengan acceso directo a los recursos informáticos, mediante credenciales de acceso legítimas, y que, por lo tanto, sus acciones sean de hecho válidas con relación en la política implementada en equipos de seguridad. De lo anterior, se sigue, que es más difícil proteger a la organización de ataques internos. Dentro de la clase de ataques internos, existen los llamados de impostor, donde el perpetrador intenta

suplantar la identidad de un usuario legítimo. Estos ataques fácilmente pasan desapercibidos, pues el impostor tiene acceso a la máquina de la víctima objetivo, ya sea porque ésta abandonó una sesión sin cerrarla adecuadamente, o porque mediante la ejecución de algún código se obtuvieron las credenciales de acceso correspondientes.

El problema de detección de impostores se ha estructurado como uno de detección de anomalías. Ha sido estudiado activamente a partir del trabajo seminal de Schonlau *et al.* [18], que sugiere que para perfilar a un usuario es suficiente considerar el historial de comandos (sin parámetros) ejecutados por él durante una sesión UNIX. Para tal propósito, Schonlau *et al.* desarrollaron una base de impostores, comúnmente referida como *SEA* [17], y que ha sido aplicada como el estándar *de facto* para construir, validar y comparar una gran cantidad de métodos de detección (véanse, por ejemplo, [9,7]). Sin embargo, el desempeño de los detectores de impostores basados en SEA está lejos de ser satisfactorio [14]. Con el propósito de mejorar el poder de detección de impostores, se han considerado desde extensiones a SEA, incorporando, *e.g.*, argumentos de comandos [10], hasta el uso de información de actividad alternativa. Ejemplos de fuentes alternativas lo son el uso del teclado [6], u otros dispositivos E/S, como el ratón [4], o el uso de aplicaciones, *e.g.* un entorno gráfico para el manejo de documentos [16], o una abstracción que agrupa aplicaciones comunes con un tipo de actividad de bajo nivel (*e.g.*, la ejecución de cualquiera de los comandos `edit`, `vi`, o `xemacs` podría categorizarse como *edición*) [1].

En este artículo se adopta un enfoque alterno y novedoso [13,12], el cual construye el perfil de un usuario considerando los objetos del sistema de archivos que éste accede y cómo lo hace durante una sesión de trabajo cualquiera en su computador. Contrario a enfoques previos, este enfoque asevera que no son las acciones que realiza un usuario, sino el sobre qué las realiza el distingo mayor que le caracteriza. Los usuarios provienen de los datos obtenidos en la base de datos WUIL [12], la cual se explica en una sección posterior.

Aquí, se considera que un usuario organiza y estructura su sistema de archivos de modo que es fácil distinguir las tareas que realiza en cada jornada. Así, hipotetizamos, el sistema de archivos de un usuario comprende un fólder por cada una de las múltiples tareas del mismo, y que comprenden todos los archivos con relación en dicha tarea. Note que en el modelo se abstrae el comportamiento de un usuario en términos de las tareas que realiza, pero que dichas tareas se elevan a un nivel superior, *e.g.* padre de familia, colaborador, finanzas, etc., y no como actividades de bajo nivel, *e.g.* edición, compilación, búsqueda, etc.

El clasificador es un proceso markoviano, adoptado de uno propuesto en [5]. Para cada usuario en WUIL, se construye su modelo, considerando una secuencia de tareas (obtenida tras asociar el objeto de cada registro en la bitácora del usuario a una tarea o rol de este último). El modelo perfila al usuario mediante un conjunto de rutas probabilísticas en la ejecución de tareas, y nos permite obtener, dada una secuencia de tareas en las que se alega la participación del usuario, una calificación sobre la disparidad de ésta conforme a su perfil. Entre

menos similitud haya entre la secuencia y el modelo mayor será la calificación de desemejanza.

Perspectiva general del artículo El resto de este artículo está organizado de la siguiente forma. Primero, en la sección 2, se muestran las limitaciones fundamentales de los métodos de detección de impostores, basados en SEA. Segundo, en la sección 3, se bosqueja la base de impostores WUIL, la cual se divide en bitácoras en condiciones normales del usuario y bitácoras que aleatoriamente incluyen ataques simulados. Tercero, en 4, se describirá cómo construir el proceso markoviano para la clasificación y posterior detección de impostores. Cuarto, en sección 5, se propone una serie de experimentos a realizarse a partir del conjunto de datos con el propósito de validar la hipótesis de que una secuencia relativamente corta de ejecuciones de tareas es suficiente para distinguir la presencia o no de un impostor. Finalmente, en seción 6, se reportan las conclusiones que hemos obtenido hasta el momento, y delineamos trabajo futuro.

2. Un Recuento del estado del arte en bases de impostores

El estudio de la detección de impostores ha resultado en un gran número de publicaciones, por lo que remitimos al lector a [2], o incluso uno menos reciente [15], si se desea tener un recuento de los trabajos más prominentes del área. Aquí nos enfocaremos sólo en discutir los conjuntos de datos que han sido colectados y desarrollados con el propósito de experimentar y comparar diferentes métodos de detección de impostores, así como en describir la hipótesis que motivó el desarrollo de los mismos.

2.1. SEA

SEA contiene bitácoras de 70 usuarios UNIX, cada uno con un registro total de 150,000 comandos, sin argumentos, colectados a través de la herramienta de auditoría `aact`. Schonlau *et al.* han dividido el registro de cada usuario en bloques, cada uno de tamaño 100 comandos, y llamado *sesión*. Así, cada usuario tiene asociadas 150 sesiones.

Los usuarios registrados en SEA son separados en dos clases: legítimos, 50 usuarios, e impostores, los 20 restantes. De cada registro de usuario legítimo, las primeras 50 sesiones no sufren modificacion; éstas deben usarse en la construcción (entrenamiento) del método de detección de impostores; las últimas 100, sin embargo, son consideradas en la etapa de validación del método, e incluyen sesiones legítimas del usuario correspodiente, así como sesiones de uno o más de los usuarios impostores. SEA viene con una matriz que indica cuáles sesiones son legítimas, y cuáles impostor.

Si bien SEA hace posible realizar una comparación justa entre el desempeño de dos o más detectores de impostores, se ha mostrado que bajo condiciones más realistas de un ataque el desempeño de todos los detectores es verdaderamente pobre [14]. Adicionalmente, Schonlau *et al.* tomaron decisiones que afectan

dramáticamente la validez de los resultados. Por ejemplo, la selección de cuáles usuarios asumirían el rol de impostor no fue aleatoria, sino más bien arbitraria; más aún, algunos de estos supuestos impostores tienen un comportamiento inusualmente repetitivo, que puede detectarse incluso sin apoyo mecánico. Adicionalmente, todo impostor es, en principio, un usuario cualquiera interactuando ordinariamente con su computador, sin ninguna intención de usurpar la personalidad de un tercero o, peor aún, de lograr un objetivo (es decir, perpetrar un ataque). La decisión de si hay un impostor o nó se toma al final de cada sesión, que contiene 100 comandos: una cantidad significativamente grande para que tenga sentido práctico.

2.2. Extensiones de SEA

De lo anterior, se desprende la necesidad por remediar dos problemas principalmente: uno, mejorar los desempeños de los detectores, y dos, mejorar las condiciones de validación de la detección, incluyendo condiciones de ataque más realistas. Curiosamente, la comunidad se ha enfocado principalmente en resolver el primer problema; trabajos como [8,10,9] sugieren mejoras metodológicas en el uso de SEA, así como la incorporación de opciones y argumentos en los comandos ejecutados.

Con relación en el segundo problema, hasta donde conocemos, existen sólo dos trabajos: uno, *RACOON* [3], un mecanismo que permite sintetizar sesiones tipo SEA con el propósito de mejorar el modelo de detección de cada usuario (pero que no ha sido apropiadamente validado), y otro, *SEA-I* [14], que incluye sesiones impostoras sintetizadas considerando una estrategia de ataque.

2.3. Otras bases de impostores

Recientemente, han habido algunos intentos por conformar bases de impostores que permitan primero realizar comparaciones justas en el desempeño de dos o más detectores, y, segundo, que consideren ambientes de ataque más realistas. El más prominente de todos es el trabajo de [6], quienes desarrollaron una base que considera la detección de impostores en la captura de una contraseña. Sólo en este enfoque, contrario a todos los predecesores y muchos de sus sucesores, tiene sentido el enfoque uno contra el resto, pues se considera que el impostor conoce la contraseña de la víctima objetivo. Este trabajo reporta resultados de comparar una gran cantidad de métodos previamente propuestos, usando una base común de información y uniformidad en el manejo de datos. Concluye que aún estamos lejos de lograr el estándar en la detección de impostores o autenticación de usuarios.

Un trabajo de interés en este tenor es el que reporta [1], aunque tiene sus bemoles irreparables. Presenta una base de impostores cuya hipótesis es que puedes separar a un usuario mediante su perfil de acciones (de bajo nivel), como edición, búsqueda, etc. Sin embargo, al momento de presentar resultados traicionan la hipótesis al considerar además número de archivos no tocados recientemente, etc. Otro defecto en este ejercicio es que es necesario categorizar manualmente cada

comando en una de las acciones arriba mencionadas. Pero el más devastador es que no incluye ataques realistas a las máquinas en las que fueron obtenidos los perfiles de usuarios, sino que se limitan a presentar una máquina similar a un gran número de usuarios y sobre ésta conducen un ejercicio de captura la bandera, dentro de un período de tiempo. El ejercicio del ataque es interesante, pero tendría que practicarse sobre la máquina de cada usuario, pues de otro modo carece de validez.

3. La base de impostores WUIL

Es necesario, para una correcta experimentación, contar con un conjunto de bitácoras confiable, tanto de usuarios, como de ataques. El conjunto de datos WUIL cuenta con un registro de datos de 20 usuarios y datos recopilados de tres ataques simulados directamente en las máquinas de dichos usuarios.

El perfil de los usuarios reclutados no es homogéneo, incluyendo estudiantes universitarios de pregrado y posgrado, personal administrativo y también de primera línea. El nivel de adecuación y experiencia en aspectos propios de sistemas operativos también varía de usuario a usuario, así como su capacidad de seguir buenas prácticas en el manejo de información. Afortunadamente, para todos ellos WUIL incluye información moderadamente detallada sobre todos estos y otros aspectos que pueden en un momento dado usarse para explicar forasteros (*outliers*) en los resultados de detección de impostores obtenidos.

Para el diseño y posterior simulación de ataques impostores, los creadores de WUIL recurrieron a una encuesta, la cual contestaron casi 50 personas. Consensando los resultados de dicho ejercicio, diseñaron tres tipos de ataques: básico, intermedio y avanzado, los cuales consideran, respectivamente, un atacante ocasional sin ningún objetivo claro por comprometer al usuario víctima, otro que, por el contrario, tiene esa intensión, pero que no posee herramientas para perpetrarlo, y otro atacante con másintensión, dotado de herramientas, *scripts* entre otros. Estos tres ataques fueron simulados de manera uniforme y controlada, directamente en la máquina de los usuarios participantes, recopilando bitácoras y destruyendo toda información que pudiera comprometerles.

4. Creación del modelo de representación del usuario basado en tareas

4.1. Definición de Tarea

Para clasificar el comportamiento de cada usuario se usará el concepto de tareas, el cual se define como una agrupación lógica de carpetas con sus respectivos archivos y subcarpetas, los cuales utiliza el usuario para un fin en específico.

4.2. Obtención de las tareas a partir del conjunto de datos

Una vez definido el concepto de tareas es necesario saber cómo obtenerlas a partir de la información contenida en WUIL. El mapeo de carpetas y archivos

a tareas tiene cierta complejidad, ya que depende del orden que mantenga el usuario, la correcta separación de las carpetas asociadas a cada tarea y la granularidad con la que el usuario consideraría qué es una tarea y qué no lo es. La granularidad es un problema ya que si es muy gruesa cada archivo pasa a ser una tarea, lo que eliminaría la necesidad de agrupamiento, mientras que si es muy fina se pierden ciertos elementos que podrían indicar un mejor patrón de comportamiento del usuario. Para resolver el problema de la granularidad y evitar depender de la correcta separación en tareas por parte del usuario se decidió que se realizaría una asignación automática de las tareas a partir de los datos de la profundidad de navegación en el sistema de archivos contenidos en los datos de cada usuario. Se calculará entonces un punto de corte P por cada usuario el cual servirá para indicar en qué profundidad de navegación se mapea una carpeta como a una tarea, mientras que todos los archivos y subcarpetas contenidos en ésta pertenecerían a la misma tarea. Dentro de nuestros análisis preliminares pudimos observar que existen diferencias marcadas entre los patrones de comportamiento con respecto a la profundidad de navegación de un usuario normal y un atacante. En la Figura 1 podemos observar cómo estas variaciones son claras.

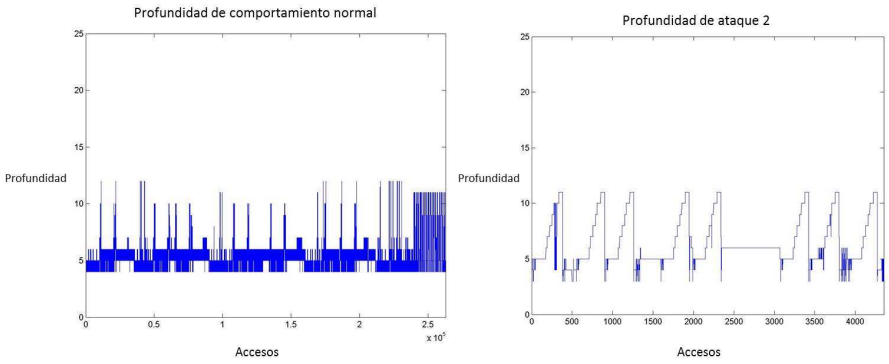

Figure 1. Comparación de comportamiento de profundidad de navegación del usuario contra profundidad de navegación de los ataques.

Al saber cuáles son las tareas realizadas por un usuario, se procede a realizar un mapeo donde a cada tarea se le asigna un valor numérico que la representará, teniendo así que el alfabeto a utilizar para la construcción del modelo y la utilización del clasificador contiene solamente elementos numéricos. La ventaja de tener solamente elementos numéricos es que permite un procesamiento más rápido al no tener que parsear por cada vez que se tenga una tarea todo el string que compone el nombre de la misma. El alfabeto a usar no es de tamaño fijo, ya que es del mismo tamaño que el número de tareas que un usuario genere y puede ser aumentado al momento de realizar una actualización del modelo para tomar en cuenta tareas que en un inicio existían. Para el caso de los archivos y carpetas

que no pertenezcan a ninguna tarea se les asignara un valor numérico temporal, con el cual serán identificadas al momento de realizar una clasificación. Este valor se puede hacer permanente al realizar una actualización del modelo.

4.3. Construcción del modelo

Una vez que se tienen las secuencias de tareas es necesario crear un modelo que nos permita obtener el comportamiento normal de un usuario. Para realizar este modelo se utilizó el método descrito en [5], en el cual se utiliza una cadena de Markov para crear la representación del comportamiento y posteriormente usarlo como un clasificador de una sola clase. La selección fue dada por la discusión en [11] dado que las cadenas de Markov permiten trabajar con datos no numéricos y tomar en cuenta un factor temporal, elemento que es importante para nosotros puesto que necesitamos observar cómo progresa a partir del tiempo y dar una predicción que nos permita decir esto. También se puede a partir de este modelo realizar la construcción de un clasificador de una sóla clase, el cual puede detectar anomalías a partir de solamente los datos del usuario sin tener que tomar en cuenta las bitácoras de ataques para lograr el entrenamiento.

Se necesitan definir algunos elementos de soporte que nos permitirán entender mejor cómo funciona el algoritmo que permite realizar la construcción del modelo, entre ellos una estructura que servirá para el cálculo de las probabilidades, los parámetros de control y algunas funciones que se realizan dentro del algoritmo. El primer elemento a definir es un estado e, el cual es una secuencia de tareas desde o hacia la cual se tienen transiciones. Para saber la cantidad de tareas que contiene cada estado se utiliza un parámetro de ventana w, el cual nos sirve para realizar un agrupamiento de una serie de tareas en un sólo estado, permitiendo representar de esta forma mejor las tareas que se realizan juntas en un traza. Se tiene también una estructura E que contiene todos los estados, así como sus transiciones y las veces que se ha observado dentro de los datos que se realizó esta transición. Por último se tiene una función corrimiento que nos permite dada una tarea y un estado, tomar la secuencia de tareas contenidas en dicho estado y crear un nuevo estado donde la tarea más vieja de esa secuencia, la cual se encuentra más a la izquierda dentro de la secuencia, se elimina en el nuevo estado y posteriormente se agrega la nueva tarea, lo que permite preservar el tamaño de la secuencia en concordancia con el tamaño de ventana w.

Antes de realizar los pasos del algoritmo se necesita tener una inicialización por cada traza que se incluya para crear el modelo, donde se tendrán dos estados llamados *actual* y *siguiente*. Cada estado esta etiquetado con una secuencia de tareas de tamaño w y el estado inicial por ejemplo tendría una etiqueta nulo,nulo si w=2. Este estado nulo servirá como estado ficticio de inicio, lo que nos permite modelar un usuario que no siempre inicie sus actividades con la misma tarea. Si es la primera traza que se va a usar se agrega el estado que contiene solamente tareas nulas a la estructura E. Después de esta inicialización se realiza una iteración de los siguientes pasos por cada tarea que exista en la traza.

1. Se realiza la función corrimiento con el estado *siguiente* y la nueva tarea leída de la traza.

2. Se verifica si existe una transición en E desde el estado *actual* hacia *siguiente*. Si la transición existe se suma 1 al contador asociado a las veces que esa transición se ha visto, en caso contrario se agrega la transición y el contador se inicializa en 1.
3. Se convierte el estado *siguiente* en el estado *actual*.
4. Si el estado *actual* no está contenido en E, se agrega.

Una vez que se han analizado todas las trazas se procede a realizar por cada estado en la estructura E el cálculo de probabilidad que a partir de un estado e se avance a un estado e' para todas las transiciones que existan desde e. Recordemos que si no existe una transición significa que ese comportamiento no se observó y por lo tanto no se incluye en el modelo. Para calcular la probabilidad de una transición entre e y e' se utiliza la siguiente formula:

$$\Pr(e, e') = \frac{\text{Número de trancisiones observadas}(e, e')}{\sum_{i=0}^{n} \text{Número de trancisiones observadas}(e, e'_i)}$$

Si tomamos que n es el número de estados a los cuales existe una transición desde el estado e, entonces tenemos que la probabilidad de que exista una transición es la proporción de la frecuencia de una transición entre e y e' y la cantidad de transiciones que parten desde e.

4.4. Actualización del modelo

Dentro de los sistemas de cómputo la estructura de los archivos no tiene un estado estable. Es por esto que para que el modelo pueda representar de una forma fiel el comportamiento de un usuario es necesario estarlo actualizando con las nuevas tareas que se vayan creando conforme el paso del tiempo. La estructura que se manejó anteriormente para la creación del modelo es la clave para realizar esto, ya que al estar guardada simplemente se tiene que actualizar tanto los contadores de transiciones ya existentes, los cuales refuerzan ciertos comportamientos en el modelo, así como agregar los nuevos estados y transiciones, para que estos no sean considerados anómalos en un futuro.

4.5. Utilización del modelo en el clasificador

Para poder clasificar una secuencia de tareas a partir del modelo creado se tiene que aplicar el mismo tratamiento que a los datos originales, es decir obtener las tareas a partir de una traza y realizar el mapeo numérico que represente cada tarea. A partir de aquí se utiliza el algoritmo detallado en [5] para lograr una clasificación a partir del modelo generado en el paso anterior. Para que el algoritmo pueda funcionar se toma en cuenta que existe por cada traza a analizar los valores X y Y que se irán actualizando conforme se vayan observando las tareas y comparándolas con el modelo, un valor Z entre 0 y 1 que servirá para penalizar el no encontrar una secuencia de tareas. También es necesario que

dentro del modelo se cumpla que para un estado e la suma de las probabilidades de las transiciones de e a todos los estados e' sucesores de e sea equivalente a 1, es decir que la función de probabilidad quede como sigue:

$$\sum_{e' \in \text{sucesores}(e)} \Pr(e, e') = 1$$

En [5] se puede observar que dependiendo del método de actualización de los valores X y Y es la efectividad del clasificador que se tiene, además de que para no tener diferentes algoritmos se manejan funciones vacías que sirven de comodín ya que después son reemplazadas por la función correcta de actualización. De los métodos propuestos seleccionamos el que nos da la probabilidad de falla, es decir el que nos indica la probabilidad de que una traza no exista o sea considerada anómala por tener una baja probabilidad de ocurrir. Al tener una selección no es necesario usar las funciones comodín que se manejan y se puede definir directamente el algoritmo de actualización de los valores X y Y. Para cada traza a analizar se considera que los valores de los coeficientes X y Y son inicializados en 0, además de que se tienen dos estados: *actual* y *siguiente*, los cuales son etiquetados con una serie de tamaño w de tareas nulas. Después de esta inicialización se realiza una iteración de los siguientes pasos por cada tarea que exista en la traza.

1. Se realiza la función corrimiento con el estado *siguiente* y la nueva tarea leída de la traza. En cuanto se han leído todas las tareas de una traza se termina de iterar.
2. Se actualizan los coeficientes de la siguiente manera.
   – Caso 1: Existe una transición en el modelo desde el estado *actual* hacia el estado *siguiente*. Si ocurre este caso se actualizan los coeficientes X y Y de la siguiente forma:

$$\begin{aligned} X &= X + 1 \\ Y &= Y + \sum_{e \in \text{sucesores}(actual) \wedge \text{siguiente} \neq e} P(actual, e) \end{aligned}$$

   – Caso 2: No existe una transición en el modelo desde el estado *actual* hacia el estado *siguiente*. Si ocurre este caso se actualizan los coeficientes X y Y de la siguiente forma:

$$\begin{aligned} X &= X + 1 \\ Y &= Y + Z \end{aligned}$$

   donde Z es la penalización que se da por no encontrar la transición (*actual*, *siguiente*) en el modelo. Es importante notar que el valor de Z sera estimado empiricamente durante los experimentos.
3. Se convierte el estado *siguiente* en el estado *actual*.

Una vez que se ha terminado de analizar la traza, se calcula el valor μ(traza)= Y/X, que indica qué tan bien predice la cadena de Markov la traza. Un μ(traza) bajo nos indica que la cadena tiene mayores probabilidades de ser predecida por el modelo. Para terminar, se tiene que dar una clasificación de si la traza es normal o anómala, por lo que se usa un valor de umbral r, donde:

$$\text{Clasificación(traza)} = \begin{cases} \text{Anómalo} : \mu(traza) \geq r \\ \text{Ordinario} : \text{En caso contrario} \end{cases}$$

5. Experimentos propuestos

Recordemos que la hipótesis es mostrar que con una secuencia relativamente corta de tareas podemos detectar un impostor, para lo cual se usará el modelo y clasificador descritos anteriormente. Para lograrlo, se tienen que resolver dos problemas principales: el encontrar el punto de corte P para determinar la profundidad desde la cual se tomarán las tareas y el ajuste de los parámetros del clasificador para que se pueda maximizar la detección de anomalías reduciendo la cantidad de falsos positivos. Para la determinación del punto de corte se necesita encontrar un método automático que pueda realizar esta acción. Se propone iniciar con un análisis estadístico a profundidad donde los datos de cada usuario se tomarán como una muestra y realizar un análisis para determinar cómo están distribuidas las profundidades de los usuarios. Creemos que a partir de estos análisis podemos encontrar un punto de corte que preserve la información necesaria para la correcta distinción entre un impostor y un usuario legítimo, permitiendo también la abstracción de los comportamientos en una serie de tareas. Una vez encontrado cómo se distribuyen los datos podemos determinar una combinación de estos análisis estadísticos que arroje un valor que se pueda usar para la determinación de las tareas.

El siguiente punto a tratar es cómo se realizarán los experimentos para observar el funcionamiento del clasificador, sin embargo es necesario determinar cómo se realizará el ajuste de parámetros, ya que actualmente no se tiene un valor establecido para el tamaño de ventana w, la penalización z y el umbral r. Estos parámetros se tendrán que ir probando conforme a los experimentos para determinar el comportamiento del clasificador con ligeros cambios en ellos. Proponemos fijar valores basándonos en [5], y a partir de ahí tomar un valor para realizar experimentos con él, para después aplicar análisis estadísticos a los resultados y determinar los mejores valores para los parámetros. Para validar que nuestro clasificador basado en un modelo a partir de las tareas funciona correctamente se tomará entre un 15 % y 25 % de las trazas, lo cual evitará que colaboren a la creación del modelo de comportamiento normal, para ser usadas en un esquema de validación cruzada. Al ser un clasificador de una sola clase no necesitamos entrenarlo con las trazas de ataques, ya que esto sería contraproducente puesto que sólo podría clasificar correctamente algunos ataques que ya se le hubieran mostrado anteriormente. Por lo que se usarán las trazas que se excluyeron de la creación del modelo y las trazas de los ataques para validar que el modelo construido para cada usuario es representativo de su comportamiento

y por lo tanto las trazas que pertenecen a comportamientos legítimos son clasificadas como legítimas, mientras que las trazas que pertenecen a un ataque son clasificadas con anómalas. Es importante recalcar que las trazas son de tamaños variables, por lo que si se logra obtener con trazas relativamente pequeñas una correcta clasificación sin incurrir en muchos falsos positivos o falsos negativos, entonces habremos validado nuestra hipótesis.

6. Conclusiones y trabajo futuro

En este trabajo se tomó un enfoque diferente al ya existente en la literatura para la detección de impostores, ya que estamos utilizando las acciones que hace el usuario sobre su sistema de archivos para obtener las tareas que realiza y así obtener un modelo de su comportamiento normal. Para lograr la clasificación se buscó un clasificador de una sola clase y basados en la discusión de [11], se optó por un clasificador basado en cadenas de Markov, método que se tomó de [5] y se adaptó para que funcionara en base a nuestro concepto de tareas. Actualmente se encuentran desarrollados los mecanismos que permiten la creación de un modelo a partir de trazas de tareas y el mecanismo que permite realizar una clasificación basada en el modelo. Con estos mecanismos desarrollados es posible empezar a realizar los experimentos propuestos. Análisis preliminares muestran que utilizando una clasificación basada en tareas se puede lograr esto, análisis como el de profundidades, que al mostrar gráficamente las profundidades de navegación podemos observar cómo emergen patrones los cuales son diferentes a los que se muestran en las trazas de los atacantes. Los resultados esperados a partir de los experimentos definidos, es llegar a demostrar que con nuestro enfoque se puede tener a partir de una serie relativamente corta de acciones, una clasificación que permita saber cuándo en el sistema está actuando un usuario legitimo o un impostor. A mediano plazo se planea la comparación contra otros clasificadores basándose en los mismos datos, puesto que esto nos puede mostrar deficiencias en algunos clasificadores, así como mejores formas de realizar la detección de impostores.

Referencias

1. Ben-Salem, M., S., S.: Modeling user search behavior for masquerade detection. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) Proceedings of the 2011 International Symposium Research Advances in Intrusion Detection, RAID'11. Lecture Notes in Computer Science, vol. 6961, pp. 181–200. Springer (2011)
2. Bertacchini, M., Fierens, P.: A survey on masquerader detection approaches. In: Proceedings of V Congreso Iberoamericano de Seguridad Informática. pp. 46–60. Universidad de la República de Uruguay (2008)
3. Chinchani, R., Muthukrishnan, A., Chandrasekaran, M., Upadhyaya, S.: RACOON: Rapidly generating user command data for anomaly detection from customizable templates. In: Proceedings of the 20th Annual Computer Security Applications Conference, ACSAC'04. pp. 189–204. IEEE Computer Society Press (2004)

4. Imsand, E.S., Hamilton, J.A.J.: Masquerade detection through GUIID. In: Proceedings of the Global Communications Conference, 2008. GLOBECOM'08. pp. 2104–2108. IEEE Computer Society Press (2008)
5. Jha, S., Tan, K.M.C., Maxion, R.A.: Markov chains, classifiers, and intrusion detection. In: Proceedings of the 14th IEEE Computer Security Foundations Workshop, CSFW'01. pp. 206–219. IEEE Computer Society Press (2001)
6. Killourhy, K.S., Maxion, R.A.: Why did my detector do *that*?! - predicting keystroke-dynamics error rates. In: Jha, S., Sommer, R., Kreibich, C. (eds.) Recent Advances in Intrusion Detection, 13th International Symposium, RAID 2010. Lecture Notes in Computer Science, vol. 6307, pp. 256–276. Springer (2010)
7. Latendresse, M.: Masquerade detection via customized grammars. In: Julish, K., Kruegel, C. (eds.) Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA 2005. Lecture Notes in Computer Science, vol. 3548, pp. 141–159. Springer (2005)
8. Maxion, R.A., Townsend, T.N.: Masquerade detection using truncated command lines. In: Proceedings of the International Conference on Dependable Systems & Networks. pp. 219–228. IEEE Computer Society Press, Washington, DC (June 2002)
9. Maxion, R.A., Townsend, T.N.: Masquerade detection augmented with error analysis. IEEE Transactions on Reliability 53(1), 124–147 (2004)
10. Maxion, R.A.: Masquerade detection using enriched command lines. In: Proceedings of the International Conference on Dependable Systems and Networks, DSN'03. pp. 5–14. IEEE Computer Society Press, San Francisco, CA, USA (June 2003)
11. Mazhelis, O.: One-class classifiers : a review and analysis of suitability in the context of mobile-masquerader detection. South African Computer Journal 36, 29–48 (2006)
12. Cami na Prado, J.B., Monroy, R., Trejo, L.A.: The windows-users and -intruder simulations logs dataset (wuil): An experimental framework for masquerade detection mechanisms. Expert Systems with Applications ?, ?–? (2013), submitted for publication
13. Camiña Prado, J.B., Monroy, R., Trejo, L., Sánchez, E.: Towards building a masquerade detection method based on user file system navigation. In: Batyrshin, I.Z., Sidorov, G. (eds.) Proceedings of the 10th Mexican International Conference on Artificial Intelligence, MICAI'11. Lecture Notes in Artificial Intelligence, vol. 7094, pp. 174–186. Springer (2011)
14. Razo-Zapata, I., Mex-Perera, C., Monroy, R.: Masquerade attacks based on user's profile. Journal of Systems and Software 85(11), 2640–2651 (2012)
15. Salem, M.B., Hershkop, S., Stolfo, S.J.: A survey of insider attack detection research. In: Stolfo, S.J., Bellovin, S.M., Hershkop, S., Keromytis, A., Sinclair, S., Smith, S.W. (eds.) Insider Attack and Cyber Security: Beyond the Hacker, pp. 69–90. Advances in Information Security, Springer (2008)
16. Sankaranarayanan, V., Pramanik, S., Upadhyaya, S.: Detecting masquerading users in a document management system. In: Proceedings of the IEEE International Conference on Communications, ICC'06. vol. 5, pp. 2296–2301. IEEE Computer Society Press (2006)
17. Schonlau, M.: Masquerading user data (2008), http://www.schonlau.net
18. Schonlau, M., DuMouchel, W., Ju, W., Karr, A., Theus, M., Vardi, Y.: Computer intrusion: Detecting masquerades. Statistical Science 16(1), 58–74 (2001)