TRABAJO DE FIN DE GRADO

Análisis y automatización de doble factor de autenticación en sistemas GNU/Linux

Sergio Roselló Morell

Tutor: Eduardo Ariols

Grado: Ingeniaría en Desarrollo de Contenidos Digitales

Convocatoria: 06.07.18

Agradecimientos

Debo agradecer a Eduardo Ariols, mi tutor del trabajo todo el apoyo y consejos dados. Estoy seguro de que sin su ayuda, este trabajo no hubiese llegado a su nivel actual. Durante el proceso de elección del trabajo, me ayudó a darme cuenta de lo que quería hacer exactamente y desde ese momento, no ha parado de inspirarme con distintas formas de ver las cosas. De eso, le estoy muy agradecido. No hubiese podido imaginar un tutor mejor para mi trabajo. También quiero agradecer a la universidad el buen trabajo a la hora de escoger al personal docente de mi grado, puesto que en todo momento han demostrado más que profesionalidad y compañerismo hacia mi y mis compañeros de carrera.

Esta oportunidad formativa no hubieses sido posible de no ser por el esfuerzo de mis padres y mis abuelos, que, sin ninguna duda y con plena confianza en mí, incitaron y asesoraron en los momentos más decisivos de mi vida. Vosotros sois la razón por la que soy quien soy. Os estoy y estaré eternamente agradecido por eso.

Por último, dar las gracias a toda la gente que ha dedicado su tiempo en ayudarme a sacar la mejor versión de mi mismo en este trabajo. A nivel personal y profesional. Muchas gracias a todos.

Resumen

Este documento explica al lector la experiencia que he tenido durante el periodo de realización del trabajo de final de grado. Este trabajo trata sobre la autenticación de un usuario a un sistema GNU/Linux, en concreto, mediante un dispositivo USB.

Durante la fase de investigación de las tecnologías existentes, encontré algunas que ofrecían usa solución elegante, mediante dbus pero acabando la fase de investigación encontré un proyecto llamado Pluggable Authentication Module (PAM) que redefinió la forma en la que planteaba el trabajo ya que es la forma por defecto de autenticar a los usuarios que tienen la mayoría de sistemas GNU/Linux.

Al iniciar el proyecto, al principio con enfoque mucho más práctico ha acabado teniendo un enfoque investigativo puesto que para implementar el módulo de autenticación, he tenido que construir una base fuerte sobre la que sentirme cómodo. Esta base es la que he tenido que esforzarme en entender puesto a que sin ella, el trabajo realizado, aunque funcionalmente completo, no me hubiese sido ni la mitad de estimulante e interesante.

Abstract

This document reports my experience as I work on creating a USB-centric authentication method for GNU/Linux.

during the research phase, i came across several elegant implementations, all of them worked with dbus. during the final stages of this period, i discovered PAM which changed my whole perspective on this project. most of systems use this module to enable authentication for their users.

at the start of this project i would've expected to code a lot more, but now i realise that without a solid foundation, i may have been able to do what i had proposed, but i would not have the understanding on how the PAM fits into the whole equation and the many benefits it provides. this, i think is the point of this work.

Índice

Índice de figuras

1. Estado del arte

Se investigará la evolución de los ordenadores, la implicación que estos tienen en nuestra sociedad actual y las precauciones de seguridad que debemos tener al usarlos.

1.1. Evolución de las tecnologías

Los ordenadores forman una gran parte de nuestra sociedad desde que fueron inventados. El primer ordenador que se creó fue el Mark I, en 1944. Este ordenador estaba hecho mayoritariamente para realizar cálculos, pesaba cinco toneladas y se sobrecalentaba. Desde Mark I, como podemos observar en la figura 1, los ordenadores han ido duplicando el número de transistores en circuitos integrados cada dos años [Moo98]. Hasta ahora, y seguramente unos años más en adelante, podamos ver que se sigue cumpliendo la ley de Moore.

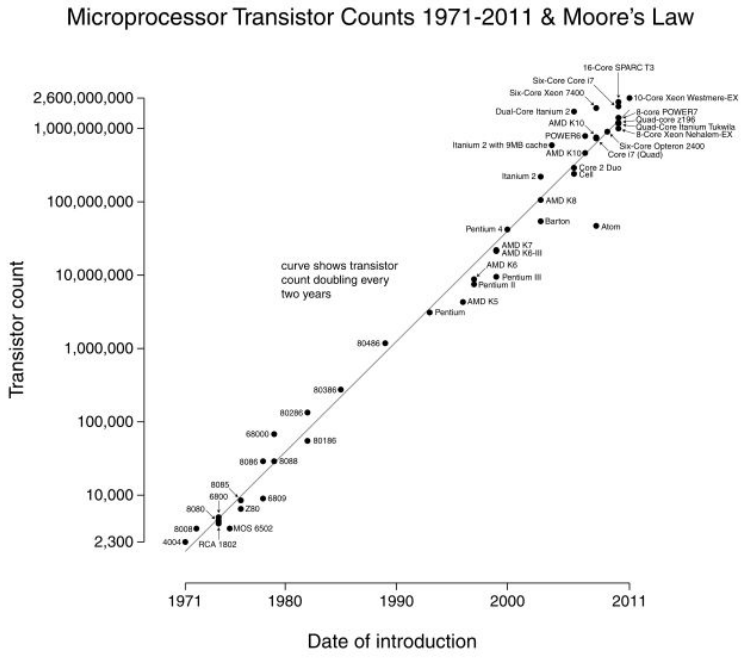

Figura 1: Ley de Moore

Durante el periodo de evolución de la informática, se pueden diferenciar cinco generaciones:

- **Primera generación:** Estos ordenadores no se parecían en nada a los ordenadores modernos, eran extremadamente grandes y poco sofisticados. Eran estructuras tan grandes como una habitación entera y usaban válvulas termodinámicas como switches y amplificadores. Las válvulas termodinámicas generaban mucho calor así que a pesar de contar con unidades de refrigeración enormes, se sobrecalentaban muy frecuentemente.

  Para interaccionar con estos ordenadores, los programadores usaban un lenguaje conocido como ”Lenguaje de máquina”que era directamente interpretable por el circuito.

  Esta generación tuvo lugar entre los años 1940 y 1956.

- **Segunda generación:** Estos ordenadores se distinguen de la primera generación porque consiguen sustituir las válvulas termodinámicas por el transistor. El cambio de válvulas a transistor implica más velocidad de cálculo y menos energía residual en forma de calor. Otra de las ventajas de los transistores es que, al ser más pequeños, reducen el tamaño del ordenador y lo hacen más económico.

  En esta generación, también se consiguió almacenar información en discos, siendo esta la primera forma de almacenamiento de información persistente.

  Esta generación tuvo lugar entre los años 1956 y 1963.

- **Tercera generación:** Se implementaron por primera vez los circuitos integrados, que contenían muchos transistores en chips semiconductores. Las ventajas de esta aportación fueron velocidad de cálculo, que aumentó mucho, el tamaño se redujo considerablemente y se continuaron abaratando los precios. En vez de ”Lenguaje máquina”, ahora los programadores usan monitores y teclados llamados TeleTYpewriter (TTY) para comunicarse con el ordenador. Hasta esta generación, los or-

denadores no estaban destinados para un uso personal. Solo las grandes empresas podían permitirse tener un ordenador.

Esta generación tuvo lugar entre los años 1964 y 1971.

- **Cuarta generación:** Es la generación con más impacto en la sociedad. La tecnología avanzó hasta tal punto que los fabricantes podían poner miles de transistores en un único circuito integrado. En esta época se puso a la venta el Intel 4004, el primer microprocesador en venderse de forma masiva. Este desarrollo inició la industria de los ordenadores personales.

  A mediados de los 70, salieron al mercado ordenadores como el Altair 8800 que venían a piezas y su usuario tenía que construir para usar. A finales de los 70, inicios de los 80, salieron al mercado ordenadores construidos de fábrica, como el Comodore PET, Apple II y el primer ordenador IBM. Al inicio de los 90, los ordenadores personales y la capacidad de crear una red de comunicación entre ellos dio paso a la creación de Internet. Se mejoró también la capacidad de almacenamiento de los discos además de la velocidad en general. Aparecieron los primeros ordenadores portátiles, que presentaban una Graphical User Interface (GUI) para facilitar la interacción entre el usuario y la máquina.

  Los usuarios de los ordenadores ya no tenían que ser usuarios técnicos ni debían realizar un curso de formación para saber usarlos. Esto avivó la tasa de adopción de esta tecnología enormemente.

  Incluso a principios de esta generación, al popularizarse tanto los ordenadores personales, ya urgía una forma de autenticación de los usuarios. A partir de esta época, se ha intentado concienciar a la población de que proteger los datos personales con una contraseña segura es la mejor forma de asegurar la privacidad de sus datos. Esta generación tuvo lugar entre los años 1971 y 2010.

- **Quinta generación:** Actualmente nos encontramos en esta generación. Algunos dicen que la adopción de los ordenadores cuánticos es el próximo gran paso, pero desde mi punto de vista, aún queda bastante para que eso ocurra.

Creo que la quinta generación se caracterizará por el cambio de mentalidad de los usuarios de ordenadores. Hoy en día, tener la información guardada en un ordenador ya no es un privilegio, ni una ventaja competitiva. La verdadera ventaja es la disponibilidad de la información en cualquier ordenador al que te conectes, proporcionando al usuario una forma de acceso a información revolucionaria. Nunca antes ha sido posible acceder desde cualquier sitio a la información de un usuario.

En la actualidad, es muy común almacenar todo tipo de información en nuestros ordenadores, incluso en nuestras cuentas on-line. Almacenamos desde imágenes hasta documentos importantes. La información almacenada en nuestros dispositivos es, por tanto, gran parte de nuestra vida. Por esta razón, es más importante que nunca proteger nuestras cuentas, tanto locales como on-line.

Esta generación empezó en 2010.

Ha sido a partir de la cuarta generación cuando los desarrolladores de aplicaciones han tenido que preocuparse de verificar que las credenciales que usan sus usuarios son correctas. Originariamente, GNU/Linux no tenía un método de autenticación definido. Esto significa que cada desarrollador gestionaba la autenticación del usuario de una forma distinta. Para los desarrolladores, esto significaba más tiempo perdido implementando medidas de seguridad que no guardan relación con el contenido de sus aplicaciones y para los administradores de sistemas significaba que tenían que mantener diversos métodos de autenticación (uno para cada aplicación) actualizados para seguir cumpliendo los estándares de seguridad deseados.

Existían dos prácticas para realizar la autenticación del usuario, la primera consiste en buscar los nombres y hashes de los usuarios en los archivos */etc/passwd* y */etc/shadow* y compararlo con lo que había introducido el usuario. La segunda mentalidad directamente gestionaban la autenticación de la forma que querían.

Como era de esperar, al no haber ni estándar ni protocolo, urgía una solución rápido. En 1995, SunSoft propuso un mecanismo llamado PAM. Este mecanismo proporcionaba

a los desarrolladores de aplicaciones una Application Programming Interface (API) que se ocupaba de manejar la lógica de autenticación y beneficiaba al administrador de sistemas porque unificaba el control de autenticación en un solo sitio, que además, permitía implementar distintos esquemas de autenticación.

1.2. Consecuencias de la evolución

El rápido desarrollo de esta tecnología, como era de esperar cuando las cosas se hacen rápido y a contra reloj, introdujo un factor de error en la ecuación. Desde los inicios, ya sea por fallo humano o casualidad, se habla de un fenómeno llamado *bug*. Este término forma parte de la jerga informática desde 1947, año en que, durante el ensamblaje del ordenador *Harvard Mark II*, tras el incorrecto funcionamiento del ordenador, los ingenieros revisaron las conexiones del ordenador y se encontraron con un insecto que adherido a dos cables, provocaba el fallo en el sistema. Desde entonces, se llama *bug* al fallo en un programa, ya sea lógico o sintáctico.

Debido a la rápido adopción de los ordenadores por la población, las compañías debían sacar los productos rápido y no tenían tiempo a arreglar algunos *bugs*. Un estudio realizado por el *National Institute of Standards and Technology (NIST)* concluyó en que los fallos en el software le cuestan a la economía estadounidense 59.5 billones de dólares anuales.[NIS02] Esto es un fenómeno imposible de evitar ya que en definitiva el software lo escriben seres humanos y nosotros, al ser seres imperfectos, no podemos producir software perfecto. Siempre hay algo que se nos escapa.

A medida que más usuarios usan programas, se van encontrando nuevos fallos, que dependiendo de la escala de gravedad podrían ser críticos, tanto para las empresas como para sus usuarios ya que si un usuario con malas intenciones encuentra un fallo de seguridad en la aplicación, dependiendo de su gravedad, podría, en teoría obtener información sensible de otros usuarios además de información interna de la compañía que ha hecho el software.

Para combatir este problema, algunas empresas han decidido recompensar a los usuarios que encuentran estos fallos. Al ofrecer una recompensa económica, la empresa incita al usuario a describir el fallo para que se pueda arreglar. Este tipo de programa se llama Bug Bounty Program (BBP). Existen varias páginas que se dedican a gestionar las ofertas de las empresas y los hallazgos de los usuarios para que ambos salgan ganando.

Hoy en día, tenemos todos nuestros datos *on-line*. Esto nos ofrece grandes ventajas como el acceso inmediato a nuestra información personal pero corremos un gran riesgo al confiar en las empresas que hacen que esto sea posible porque no existen programas sin *bugs*.

1.3. Necesidad de seguridad

A medida que ha ido evolucionando la tecnología, medidas de seguridad previamente válidas, han ido quedando deprecadas debido a fallos que se han encontrado en los protocolos o nuevas versiones de estas mismas. El campo de la seguridad en la informática ha ido evolucionando como si se tratara del juego del ratón y el gato en el que los desarrolladores arreglan e inventan nuevas formas de proteger la información de los usuarios y los hackers vulneran esas implementaciones.

Ahora mismo, los usuarios son los que más tienen que perder. Tenemos todos nuestros datos almacenados en servidores de grandes empresas como Google o Facebook y en el caso de que se filtre nuestra información al mundo, tanto fotos personales como documentos sensibles se verían expuestos a todos los usuarios de Internet.

Para evitar esto, estas empresas implementan métodos de seguridad cada vez más avanzados como el doble factor de autenticación para iniciar sesión en la cuenta.

Las medidas de seguridad que proporcionan las empresas como Google o Facebook son bastante buenas pero no son suficientes. Tampoco podemos pedir a estas empresas que implementen todo tipo de sistemas de autenticación de usuarios, ya que al final, tienen que

hacer el proceso fácil y sencillo para que le gente quiera y sepa usarlo.

Podemos aumentar la seguridad de nuestro sistema configurando nosotros mismos las distintas formas de autenticarnos en nuestros sistemas. Una de las formas en las que podemos aumentar la seguridad de nuestro sistema es mediante el DFA. Algunos ejemplos de esta implementación son:

- Contraseña + Llave USB
- Contraseña + App generadora de códigos (Google Authenticator)
- App generadora de códigos + Sensor de huella dactilar
- Contraseña + Sensor de retina

A cuanto más valor, ya sea económico o emocional, más medidas de seguridad serán implementadas en proteger dicha información.

En definitiva, es muy complicado tener un entorno seguro en el que poder confiar porque muchas de los vectores de ataque no dependen del usuario final, sino de terceros (tantos como distintos servicios use el usuario). Es importante tener un buen nivel de seguridad en todos los dispositivos, pero este no debe interponerse entre las tareas que un usuario tiene que hacer en su ordenador ya que de lo contrario, sería un inconveniente, no una ventaja. Cada usuario debe establecer el grado de seguridad de sus cuentas, tanto en Internet como de sistema.

Para la mayoría de usuarios, una contraseña, aunque menos segura que un DFA será más conveniente porque la naturaleza de la información que tiene que proteger no es tan importante como para asegurar su confidencialidad.

2. Introducción

En esta sección se presenta mi experiencia con la tecnología y como ésta ha derivado en este trabajo. Además, se expone la envergadura de este documento y las competencias adquiridas en la realización del proyecto.

2.1. Motivación

Desde pequeño, mi interés por los ordenadores era más que evidente. Recuerdo aún la forma en la que miraba el símbolo de sistema de Windows 7 pensando: *Ojalá saber como funciona esto*. Siempre ha existido ese interés por el software en mí. Entender como funciona, desde el programa más básico al más complicado, ha sido siempre la fuerza de atracción hacia este mundo. Al decidir dedicar mi vida profesional a ello, decidí iniciar mi formación con un grado universitario. Como todo alumno de primero de carrera, inicié el curso con ganas y Windows.

Durante el transcurso de la carrera, se hacía evidente que Windows no era el mejor sistema operativo para hacer las prácticas. Durante el primer curso usé una máquina virtual con Ubuntu para hacer las prácticas. Cuando vi que usaba más la máquina virtual que el sistema operativo sobre la que corría, supe que era hora de dejar atrás Windows. El primer sistema operativo que instalé fue Ubuntu y durante unos años dediqué mi tiempo libre a aprender a gestionar el sistema por comandos. Esto me llamaba la atención enormemente. Cuando me sentí lo suficientemente cómodo para cambiar nuevamente de sistema operativo, decidí cambiar a Arch linux aunque primero debía pasar por Manjaro, una distribución basada en Arch linux, más sencilla de utilizar, pero con el mismo sistema por debajo. Cuando finalmente me vi capaz de pasar a Arch linux y lo configuré correctamente, sentí que todos los años de aprendizaje habían llevado a buen puerto.

Todo el proceso de aprendizaje que he pasado durante estos años me ha servido para aprender y valorar el sistema operativo. Esta es la razón por la que supe desde el primer

momento en que nos dijeron que fuésemos pensando sobre que queríamos hacer el trabajo de fin de grado que quería hacerlo sobre GNU/Linux.

Tras toda esta experiencia, he querido ampliar mis conocimientos en GNU/Linux. Durante la reunión con mi tutor, vimos algunas opciones de investigación y finalmente, al estar interesados en la seguridad de las aplicaciones y sistemas, decidimos que era buena idea indagar por esa rama.

Durante el transcurso de la carrera, ya sea por requisito de los profesores o por mis proyectos personales, he sido un gran usuario del sistema de control de versiones, Git. Lo he usado desde prácticas en las que era necesario hasta para este mismo documento. Desde que empecé a usarlo, vi que no se trataba simplemente de una herramienta para gestionar mis proyectos, sino de un estilo de vida, una forma de enseñar y compartir los proyectos en los que estás involucrado.

La mayor parte de los programas que uso en mi sistema operativo se desarrollan de forma abierta en los que puedes ver el progreso y interaccionar con los desarrolladores, incluso ayudarles a detectar errores o solicitar un *pull request* para arreglar algún fallo.

Espero que el resultado de este proyecto consiga reducir el nivel técnico mínimo necesario para configurar un Doble Factor de Autenticación, bastante más seguro que cualquier sistema por defecto.

Cualquier usuario, técnico o no, debería tener fácil acceso a varias formas de verificar su identidad. Según estas, ya es éste, quien puede decidir cual usar, pero al menos, el usuario tiene la opción de elegir.

2.2. Descripción del proyecto

Este proyecto consta de dos partes coexistentes. Una parte de investigación sobre las distintas formas de autenticación en sistemas GNU/Linux y una parte práctica.

La meta inicial del proyecto era desarrollar un sistema de DFA para dichos sistemas pero al descubrir durante la fase de investigación que ya existía, se cambió de perspectiva ya que no tenía sentido plantear un mecanismo de autenticación superior al de los expertos desarrolladores que implementaron PAM en 1995. Como consecuencia de este descubrimiento se decide implementar un DFA para demostrar los pasos a seguir en su desarrollo pero se encontró un proyecto [1] en GitHub que hacía exactamente lo que tenía pensado implementar. Tras encontrar dicho proyecto se cambió de perspectiva. Se decidió desarrollar un script en BASH que facilitase al usuario no técnico la configuración del módulo Universal Serial Bus (USB) para PAM.

El script que he creado ayuda al usuario a seleccionar el USB que quiera usar, le explica las diferencias entre las distintas formas que hay de configurar el Módulo y le deja decidir donde quiere introducir la línea de configuración ya que al ejecutar las órdenes de forma secuencial, dependiendo de la posición en la que se coloque la línea, el módulo hará una cosa o otra.

Previo a el script, para configurar el módulo USB, el usuario debía leer la documentación oficial de PAM para averiguar que archivo es el que debe configurar y como lo debe hacer. Lo que consigo con mi script es reducir la curva de aprendizaje necesaria para configurar un método de autenticación más seguro que el de la contraseña por defecto. Esto hará que más usuarios tengan acceso a mejor autenticación en sus ordenadores sin que pierdan conveniencia y comodidad.

2.3. Competencias adquiridas

Al acabar cualquier tarea propuesta en la vida uno siente una sensación de realización fuertemente vinculada con esta. Además de realización, si uno se para a analizar, se da cuenta de las distintas aptitudes y actitudes desarrolladas, de forma involuntaria tras la realización del proyecto. En el caso de este proyecto, siento que he reforzado y

[1] `https://github.com/ColumPaget/pam_usbkey`

desarrollado las siguientes.

2.3.1. Aptitudes adquiridas o reforzadas

Las aptitudes que he desarrollado y su explicación son:

- **BASH**

  Decidí usar este lenguaje de programación para hacer el script porque sabía que no me iba a encontrar con complicaciones a la hora de asegurar que funcione en cualquier equipo ya que todos los programas usados en el script vienen instalados de forma predeterminada en la gran mayoría de sistemas.

- **C**

  El PAM está escrito en C y he tenido que leer y entender como funciona el submódulo USB para ver como se comunican ambos módulos.

- **Python**

  En la primera implementación que hice, usé este lenguaje de programación para bloquear la pantalla del ordenador mediante dbus.

- **Linux**

  Durante el transcurso del proyecto he estado investigando el funcionamiento de Linux aunque más detalladamente en los mecanismos de autenticación de usuarios que tiene.

Las siguientes dos, aunque de forma indirectamente relacionadas con la realización del proyecto, directamente relacionadas con la realización de este documento.

- **Vim**

Este documento ha sido escrito completamente en Vim. La extensión de este documento ha servido para aprender tanto funciones básicas como funciones avanzadas de Vim

- **LaTeX**

  Al igual que en el caso de Vim, la complejidad de este documento ha servido para desarrollar un mejor entendimiento de este sistema de composición tipográfica.

2.3.2. Actitudes adquiridas o reforzadas

La mayor parte de estas actitudes las he desarrollado en otros proyectos y ya sabía que era capaz de demostrarlas. La última, sin embargo me ha sorprendido reconocer que la poseo.

- **Resolutivo**

  Durante la primera iteración de investigación/implementación, encontré una forma de desbloquear el equipo mediante dbus. Esta forma, aunque funcional, no es elegante. Es más parecido a un "hack"que a una solución elegante y segura. Si no hubiese "dado un paso atrás", no hubiese continuado investigando/implementando otras soluciones.

  Cuando encontré PAM_USB, me obcequé en hacer que funcionase, ya que si no funcionaba, no podía avanzar en el trabajo. De nuevo, tuve que dar un paso atrás para buscar otras implementaciones para el módulo PAM con USB.

- **Persistencia**

  Guarda una relación directa con la forma en la que he enfocado el trabajo.

- **Adaptabilidad**

  Durante toda mi vida de estudiante, mi forma de retener la información y por consiguiente aprender, ha sido escuchando al profesor. Leer los apuntes a mi me sirve

como refuerzo a la explicación del profesor. Durante este proyecto, no he seguido esa misma dinámica, ya que no he tenido un profesor a quien escuchar, he tenido que investigar y formarme a base de la lectura de las páginas de documentación y los comentarios informativos en el código.

2.4. Estructura del documento

Este documento está compuesto por varias secciones.

- **Estado del arte**

  En este apartado se describe la evolución de los ordenadores, poniendo en contexto de esta forma la importancia de los mecanismos de autenticación en relación a los usuarios de los mismos y los tiempos modernos.

- **Introducción**

  Se redactan las fuerzas mayores que han desencadenado en la realización de este proyecto además de una breve descripción del proyecto y las aptitudes y actitudes desarrolladas a consecuencia de él.

- **Objetivos**

  Análisis de los objetivos generales y específicos propuestos durante el desempeño del proyecto.

- **Métodos de trabajo**

  Observación de la forma en la que se ha trabajado. Breve explicación de la misma, ventajas e inconvenientes y esquema general de los pasos realizados durante el proyecto.

- **Investigación y resultados**

Esta sección consta de dos partes principales. Una de investigación y otra de Desarrollo. En la primera, se hace una investigación general de las distintas formas de autenticación disponibles tanto a usuarios como a administradores de sistemas de grandes redes de ordenadores. Tras el análisis, se explica por qué se decidió optar por PAM, sus ventajas e inconvenientes relacionadas con las formas previamente analizadas y se procede a explicar en detalle dicho módulo. Una vez analizado el módulo, se explica como sería el proceso de creación de un sistema PAM, se explican las distintas formas existentes de usar un módulo PAM para demostrar su adaptabilidad. El segundo apartado contiene el análisis de las distintas pruebas realizadas con sistemas con la misma finalidad y finalmente el análisis, estructura y desarrollo del script que se ha creado para ayudar a normalizar este mecanismo de autenticación.

- **Conclusiones**

  Una vez expuesto todo el contenido del proyecto, se llega a una conclusión del proyecto además de posibles formas de expandir el trabajo hacia el futuro.

3. Objetivos

Lista de objetivos, de generales a específicos a medida que se avanza el trabajo.

- **Expandir mi conocimiento sobre GNU/Linux**

  Además de ser un sistema operativo versátil y fehaciente, es una prueba viviente de que se puede hacer código libre en comunidad y de calidad.

- **Contribuir a la comunidad de desarrolladores con alguna aportación**

  Durante la investigación y desarrollo del trabajo de fin de grado, he aprendido sobre las distintas tecnologías gracias a los manuales oficiales de PAM. Esto, como debería ser, es software y documentación disponible a cualquier persona, con una licencia GNU General Public License que: garantiza tu libertad de distribuir y cambiar todas las versiones de un programa para asegurar que permanece libre para todos sus usuarios. [Sta12]

  Creo que se debería contribuir a la comunidad de desarrolladores en general, además, si puedo beneficiar a alguien con el documento o código generado durante este trabajo, quedo más que satisfecho.

- **Investigar ámbito de seguridad**

  Se decidió que la investigación debía ser en el ámbito de seguridad porque tanto a mi tutor como a mi nos interesaba la idea.

- **Investigación y análisis del DFA en GNU/Linux**

  Se decide ahondar en el ámbito tras ver un vídeo en el que se presentaba un mecanismo para evitar la fase de autenticación en un sistema operativo basado en GNU/Linux. La clave estaba en el uso de una herramienta pre-instalada para enviar mensajes a las aplicaciones encargadas de autenticar la identidad de los usuarios informándolas de que el usuario ya estaba autenticado. Estas herramientas no esta-

ban monitorizadas por los administradores de sistemas debido al gran volumen de información que pasa a través de ellas.

- **Crear un sistema DFA**

  El objetivo inicial en este trabajo era crear un sistema que permitiese al usuario bloquear y desbloquear el ordenador con un USB. Cuando se empezó a investigar las distintas formas de hacer esto, el primer paso lógico era centrarse en dbus ya que existían proyectos que conseguían este objetivo. Tras el primer ciclo de investigación/implementación conseguí bloquear el sistema al enviar un comando mediante dbus al programa que se encarga de bloquear el sistema pero debido a varios factores que se explicarán más adelante, no era una solución buena.

- **Desarrollar un módulo PAM para la autenticación mediante USB**

  Tras una investigación inicial, se descubrió que esta librería es la forma por defecto de autenticación de usuarios en la mayoría de sistemas GNU/Linux. Su diseño contemplaba la posibilidad de utilizar múltiples factores de autenticación. Para avanzar, ahora se debía entender como funcionaba PAM y como crear un módulo de autenticación por USB que lo implementara.

- **Desarrollo de script para facilitar adopción**

  El cauce de este proyecto se centró en facilitar la adopción de este módulo a los usuarios, de forma que más gente no técnica pueda acceder a mejor autenticación. Esto se consigue creando un script que al ejecutarlo se encargue de la parte más complicada del proceso de instalación de un Doble Factor de Autenticación: la configuración, ya que si se configura erróneamente, existe la posibilidad de que el usuario se queda bloqueado fuera de su ordenador.

- **Elección del lenguaje para el script**

  El siguiente objetivo fue la elección del lenguaje. Al elegir lenguaje de programación para el script tuve en cuenta varios factores. Conocimiento actual del lenguaje,

complejidad de implementación en dicho lenguaje, disponibilidad de dicho lenguaje en los posibles sistemas donde se use PAM y en caso de no haber usado el lenguaje previamente, curva de aprendizaje del mismo. Finalmente, el lenguaje de programación usado ha sido BASH debido a que su integración con GNU/Linux es excelente, la curva de aprendizaje no era exagerada y que es compatible con todos los sistemas en los que funciona el módulo.

- **Enumeración de dispositivos USB conectados al sistema**

  Una vez escogí BASH. Debía averiguar como usarlo para encontrar los dispositivos USB en el sistema.

4. Métodos de trabajo

Durante el desarrollo del proyecto, he seguido una metodología de desarrollo ágil, ya que a mi parecer, es la forma más eficiente de trabajar en un proyecto. Comparada con el desarrollo en cascada, aporta muchas ventajas al flujo de trabajo.

En esencia, las metodologías ágiles son una evolución de la conocida implementación en cascada ya que generalmente se consiguen mejores proyectos, mayor satisfacción del cliente y mayor unión entre los miembros del equipo de desarrollo y el cliente.

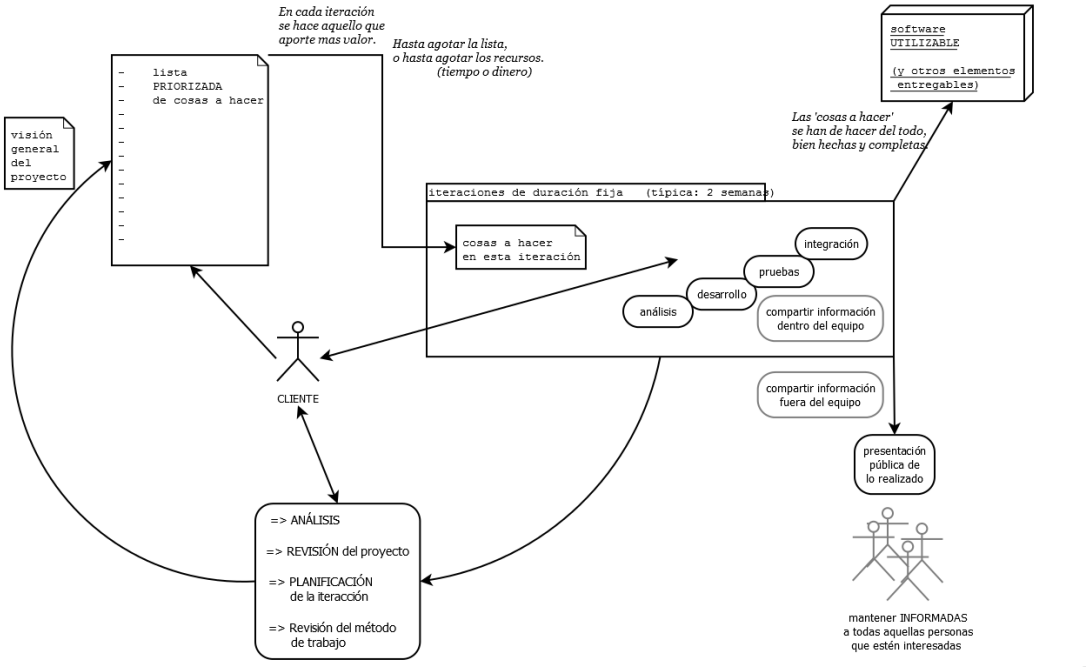

Figura 2: Diagrama explicativo de una Metodología ágil

Esta metodología se basa en una serie de ideas que quedan bien plasmadas en su manifiesto. Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.[Mik02]

- **Individuos e interacciones** sobre procesos y herramientas

- **Software funcionando** sobre documentación extensiva
- **Colaboración con el cliente** sobre negociación contractual
- **Respuesta ante el cambio** sobre seguir un plan

Las ventajas de este estilo de desarrollo frente a estilos más conservadores como el de cascada son, entre otras, que el cliente trabaja con el equipo de desarrollo constantemente para adaptar el producto a las características deseadas, el flujo de trabajo se basa en la repetición de actividades como análisis de requisitos, diseño, implementación, testeo y producción de forma iterativa hasta conseguir un producto acabado.

Diría que he seguido bastante de cerca el estilo de desarrollo propuesto por esta metodología. Durante el periodo de desarrollo del trabajo, he mantenido una relación cercana con mi tutor, manteniéndole siempre al día sobre los progresos del proyecto, cada semana aproximadamente nos reuníamos para ver el estado del proyecto, en la que le describía los progresos prácticos conseguidos. Cuando decidimos que dbus no era una buena solución, supe responder al cambio y adaptar el proyecto a nuevas y mejores soluciones.

A medida que avanzaba el proyecto, el área de investigación se volvía más pequeña y enfocada. Durante las primeras semanas, me centré en investigar el transcurso de los métodos de autenticación. Una vez analizadas las distintas formas de autenticación, profundicé en dbus, implementé un programa que enviaba un mensaje mediante este bus al programa encargado de bloquear y desbloquear el sistema y conseguí bloquear el sistema. Esto me servía como prueba de concepto pues, si había logrado bloquear, solo faltaba añadir una capa de abstracción más que enviase el comando de bloqueo si las acciones del usuario cumplían con el comportamiento esperado.

Durante el periodo de investigación de dispositivos con los que podía realizar un DFA llegué a la conclusión de que en realidad cualquier dispositivo hardware es válido. Cuando alguien piensa en DFA, lo más común es pensar en las aplicaciones como Google Authenticator. Lo siguiente que viene a la mente es usar una llave USB como segundo

factor de autenticación, pero la realidad es que se pueden usar todo tipo de dispositivos, algunos ejemplos son: auriculares (comprobando que se complete un patrón como por ejemplo insertar y extraer el jack de audio dos veces seguidas), una tarjeta de red *Wi-Fi* (comprobando el *fingerprint* de los dispositivos que emiten una señal *Wi-Fi* con una lista de *fingerprints* aceptados) o cualquier módulo Bluetooth (comprobando el *fingerprint* contra una lista de *fingerprints* aceptados). Solamente es necesario que el sistema operativo lo reconozca.

Finalmente, tras investigar las ventajas e inconvenientes de varios dispositivos hardware, me decanté por el USB como segundo método de autenticación. Aunque otros dispositivos como auriculares pasan desapercibidos, el sistema de verificación no puede interactuar con ellos y aunque no sea un requisito, el hecho de poder interactuar con el USB, me proporciona más formas de verificar al usuario.

5. Investigación y resultados

Durante el desarrollo de esta sección se han investigado todas las técnicas de autenticación de usuarios encontradas para GNU/Linux. Sin estos conocimientos no hubiese quedado contento ya que aunque se hubiese podido hacer un mecanismo para iniciar sesión, pero no se hubiese entendido ni la filosofía ni la comunidad detrás de GNU/Linux.

En esta sección, se describen las distintas tecnologías de autenticación existentes, los problemas actuales de esas tecnologías, se defenderá la necesidad de una seguridad más agresiva, tanto para los usuarios normales como para las grandes empresas y se desglosarán las distintas formas de conseguir esta seguridad actualmente. Además, se describirá en detalle como funciona PAM, desde el punto de vista del desarrollador de aplicaciones, del administrador de sistemas y del desarrollador de módulos. Se describirán las distintas formas de autenticación probadas, desde el uso de dbus hasta el uso de PAM con sus ventajas e inconvenientes.

5.1. Comunidad

Al saber que quiero publicar el script y las partes del trabajo que puedan resultar útiles a la comunidad, se ha hecho una investigación para averiguar donde y como se debería subir el contenido. El principal sitio donde ***Falta completar***

Durante el desarrollo del proyecto, durante la búsqueda de información relacionada con PAM me he encontrado con escasa información de calidad en español, por eso, y siguiendo la filosofía de libertad de documentación y software, me gustaría publicar partes de este documento para facilitar el entendimiento de PAM a compañeros de habla española.

5.2. Análisis de sistemas de autenticación en sistemas GNU/Linux

La parte de análisis es fundamental para conocer los métodos existentes actualmente para autenticar a usuarios. Sin este análisis inicial, hubiese sido mucho más complicado realizar el proyecto debido a que el desconocimiento de las distintas formas de autenticación no me hubiese llevado a poner a PAM en perspectiva.

La forma en la que se autentica la identidad de los usuarios de sistemas ha ido evolucionando desde que se vio que era necesaria. A continuación se exponen las formas más populares de gestión de usuarios y credenciales, tanto en el ámbito empresarial, como en el ámbito personal.

5.2.1. Inicios de la gestión de permisos

Cada objeto tiene asociada una tabla de 9 bits, los tres primeros indican los privilegios de lectura, escritura y ejecución del usuario que posee el objeto. Los tres siguientes son para la lectura, escritura y ejecución de los usuarios pertenecientes al grupo que posee el objeto y los tres últimos son de lectura, escritura y ejecución de los usuarios que no pertenecen a ninguno de las dos primeras categorías. Esta categoría se llama *others*. Además de estos 9 bits, también pueden incluir el SetUid, SetGid y el StickyBit. A pesar de ser un sistema muy simple de gestionar privilegios, cumple la mayoría de escenarios posibles en sistemas UNIX e incluso a día de hoy, se sigue usando en todos los sistemas GNU/Linux ya que proporciona una forma sencilla y eficiente de visualizar los privilegios de los objetos y modificarlos. Esta forma de gestionar los privilegios de los objetos se puede denominar Access Control List (ACL).

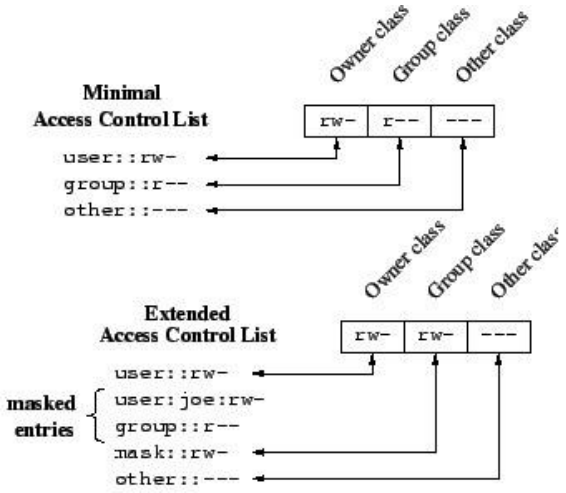

Figura 3: Access Control List

5.2.2. Role Based Access Control

Este esquema de seguridad está diseñado para organizaciones o sistemas en los que van a interactuar distintos usuarios con una gran cantidad de datos. El sistema defiende de que, en lugar de tener una tabla por cada objeto, definiendo la forma que tienen los usuarios de interactuar con él, se deberían establecer una serie de transacciones, que dependiendo del rol serán distintas. Estas transacciones, una vez definidas cambian poco porque un usuario específico va a usar unos documentos específicos, dependiendo de la responsabilidad que tenga en la organización. En la figura 4 se puede ver claramente como dependiendo del rol se va a poder acceder a ciertos objetos.

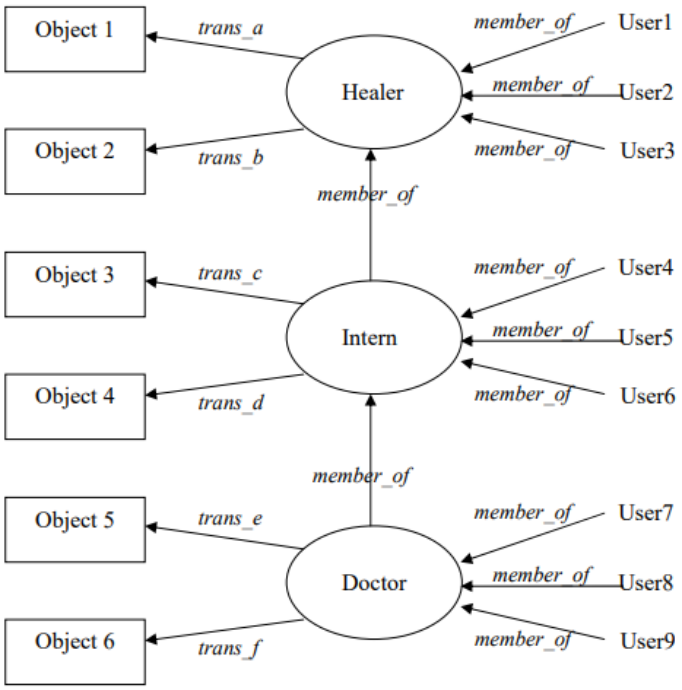

Figura 4: RBAC

Dos de las ventajas de este sistema son que cumplir el principio de menor privilegio es relativamente sencillo, ya que se puede conseguir no proporcionándole al usuario más transacciones de las que debe tener. La otra ventaja innata de Role Based Access Control es la separación de deberes. Esto es: En el caso de tener que realizar una transferencia bancaria, nunca se debería de poder proporcionar al mismo individuo el control de todo el flujo, ya que se le está dando la oportunidad de cometer algún tipo de irregularidad. Con RBAC puedes asignar dos transacciones, una que permita a un usuario solicitar una transferencia y otra que permita a un usuario validar la transferencia.

5.2.3. Lightweight Directory Access Protocol

Este protocolo está diseñado para permitir acceso a directorios complacientes con el estándar X.500 *Directory Access Protocol* a sus usuarios. Uno de varios protocolos que tiene una aplicación para autenticarse con el fin de acceder al directorio X.500[M

W97]. La forma de acceder a los datos cambia según el protocolo de acceso a X.500. Por ejemplo, en esta implementación, la forma de acceder a los datos sería:

```
cn=Rosanna Lee, ou=People, o=Sun, c=us
```

mientras que la implementación de Microsoft sería:

```
/c=us/o=Sun/ou=People/cn=Rosanna Lee
```

Cada uno de estos protocolos define una forma de buscar en X.500 información. Podemos ver que la implementación de *Lightweight Directory Access Protocol (LDAP)* está ordenada de derecha a izquierda, separada por el carácter (,) mientras que la implementación de Microsoft, Microsoft Active Directory (MAD) está ordenada de izquierda a derecha y separada con el carácter (/). Aunque no sea un método de autenticación que sucede en el mismo sistema, me parece que es lo suficientemente interesante como para incluirlo ya que es un ejemplo de autenticación en red.

5.2.4. SASL

Es un protocolo que proporciona métodos de añadir autenticación a protocolos de red mediante identificación y autenticación de los usuarios conectados al servidor. Además, gestiona el nivel de seguridad que se desea establecer para las futuras interacciones entre el servidor y el usuario conectado. Si se llega a la conclusión de que si que se requiere una capa de seguridad, ésta se añade entre el propio protocolo y la conexión.[Mye97] Las distintas formas de autenticarse a un servidor con este protocolo son:

- **Anonymous:** Usado para autenticar a clientes a servicios anónimos. El cliente envía un token (Correo electrónico) para permanecer identificado con el servidor. Es una forma sencilla y rápida de implementar, pero no es segura.
- **CRAM-MD5:** Usa el nombre de usuario y una contraseña para autenticar a los usuarios, pero solamente se transfiere la contraseña hasheada. Esto implica que no

se pueden usar métodos de autenticación normal como PAM, que no soporta extracción de contraseñas. Es una forma simple y segura de autenticarse con el servidor.

- **KERBEROS_V4:** Forma de autenticación fiable. Rápida, pero complicada de implementar. Muy segura.
- **por defecto (autenticación y autorización):** Usa el nombre de usuario y la contraseña para autenticar a los usuarios. La forma más rápida y sencilla pero poco segura.
- **SCRAM-MD5:** Deprecada.
- **DIGEST-MD5:** Basada en CRAM-MD5 pero da soporte a más características. Solo se transfieren las contraseñas hasheadas, por tanto no se puede usar PAM como backend. Es simple y seguro.
- **LOGIN:** Usa nombre de usuario y contraseña para autenticar a los usuarios. Rápida, simple de implementar pero nada segura.
- **OTP:** One Time Password
- **SECURID:** Usa una clave de un dispositivo hardware para autenticar a los usuarios. Buena velocidad, difícil de implementar pero buena seguridad.

Este protocolo ofrece una ventaja muy significativa. Proporciona a los desarrolladores la posibilidad de implementar su propio mecanismo para que utilice SASL.

5.2.5. Kerberos

Kerberos es un protocolo de autenticación de red diseñado para proporcionar un alto nivel de seguridad en la forma en la que el cliente y el servidor se autentican. Para conseguir esto, usa criptografía de clave secreta.[MIT98]

El protocolo parte con la base de que Internet no es un sitio seguro. Muchos protocolos ni siquiera usan algún tipo de seguridad. Existen herramientas con intención maliciosa para capturar las contraseñas de la red, por tanto, transferir credenciales descifradas a través de la red, es una práctica muy insegura. Algunas páginas usan Firewalls para solucionar sus problemas de seguridad, pero los Firewalls suponen que el problema está en el exterior cuando deberían asumir que se pueden vulnerar desde dentro también. Los Firewalls son inaceptables porque restringen la forma que tienen los usuarios de acceder a internet.

Kerberos es la solución para este tipo de problemas de seguridad. El protocolo de kerberos usa una criptografía muy fuerte para que tanto el servidor como el cliente puedan comunicarse a través de una red insegura. Tras haberse autenticado mediante kerberos pueden cifrar la comunicación entre ellos.

5.2.6. NIS

Proporciona información a los sistemas que tiene que tener para que cualquier usuario pueda autenticarse en cualquier sistema de esa misma red. Por tanto, debe mantener sincronizados y actualizados datos como:

- Nombres de usuario, contraseñas y directorios principales de cada usuario (/etc/-passwd)
- Información de grupos (/etc/group)
- Nombres de sistemas y direcciones IP (/etc/hosts)

Esta información la administra el servidor central. Dependiendo del tamaño de la red, el administrador de sistemas puede decidir replicarla en más ordenadores (esclavos) que se mantienen actualizados siempre con el servidor maestro (cada vez que este se actualiza) Una de las ventajas de esto es que al estar los datos distribuidos, si se cae el servidor

maestro, los usuarios de la red no sufren ningún percance, ya que la información está reflejada en los esclavos. Otra de las ventajas de mantener la información distribuida es que los esclavos también pueden responder a peticiones de clientes, por tanto, si un esclavo tarda menos en responder que el servidor, este puede facilitar la información al cliente.

La diferencia entre NIS y NIS+ es que el último implementa muchas mejoras, incluida entre ellas, la posibilidad de tener dominios jerárquicos.

Sin embargo, la mayor parte de administradores de sistemas recomendarían usar NIS, ya que es bastante más sencillo de administrar.

5.2.7. SSH

Una de las formas que han tenido los usuarios de conectarse de forma remota con un sistema ha sido el Telecomunications Network (Telnet). Esta opción, aunque válida, es poco segura porque la comunicación entre la máquina y el usuario se envía sin cifrar a través de la red.

Se desarrolló SSH para evitar esto.

5.3. Análisis de PAM

Tras la investigación previa, se concluye que PAM es la vía por la que se tiene que seguir. El hecho de que sea compatible con varias de las distintas formas descritas previamente nos muestra su universalidad. Es una herramienta que tanto usuarios noveles como administradores de sistemas con experiencia pueden configurar. Además, es tan válido en un ambiente empresarial como en un sistema personal. Eso demuestra su amplia compatibilidad. Por estas razones, creo que la mejor opción para implementar un DFA es PAM.

En esta sección se describirá en detalle PAM. Se describirá lo necesario para poder integrarlo en una aplicación, lo necesario para codificar un módulo que lo implemente y el papel del administrador de sistemas a la hora de unificar ambas partes.

5.3.1. PAM, una visión de conjunto

En 1995 SunSoft introdujo PAM. Dado que en ese momento, no existía un protocolo o diseño específico para realizar la autenticación, su uso se implementó en la mayoría de sistemas. Estos incluyen: RedHat 5.0, SUSE 6.2, Debian 2.2, Mandrake 5.2, Caldera 1.3, TurboLinux 3.6, Solaris, AIX, HP-UX y Mac OS X. Finalmente se volvió un estándar y ahora la mayoría de distribuciones GNU/Linux lo implementan [IBM09].

Hay cuatro módulos distintos definidos por el estándar PAM. Cada uno cumple una función específica pero indispensable para gestionar las cuentas de usuario en sistemas[Hat97]. Por lo general, cada programa que usa PAM define su propio *service* de forma que el programa que gestiona el login, define un *service* llamado *login* aunque se puede dar el caso de que dos aplicaciones usen el mismo *service*. Un ejemplo sería que dos aplicaciones que gestionan el bloqueo de pantalla usan el mismo *service*, *lock*. Por defecto, la configuración de las aplicaciones PAM tiene lugar en el directorio /etc/pam.d. Cada *service* tiene su propio archivo de configuración.

Para que un usuario consiga autenticarse por PAM, deben existir tres partes fundamentales. Se debe usar una aplicación *PAM-aware*, debe existir un módulo para gestionar la o las interfaces implementadas por dicha aplicación y el administrador de sistemas debe incluir el módulo en el archivo de configuración del *service* al que pertenece esa aplicación.

Su diseño permite que se verifiquen varios módulos secuencialmente creando así el primer sistema de múltiple factor de autenticación conocido.

5.3.2. Arquitectura de PAM

Los componentes clave del framework PAM son la librería de autenticación, en forma de API (Front-end) y los módulos de autenticación en forma de Service Provider Interface (SPI) (Back-end). Las aplicaciones usan la API mientras que los sistemas de autenticación usan la SPI.

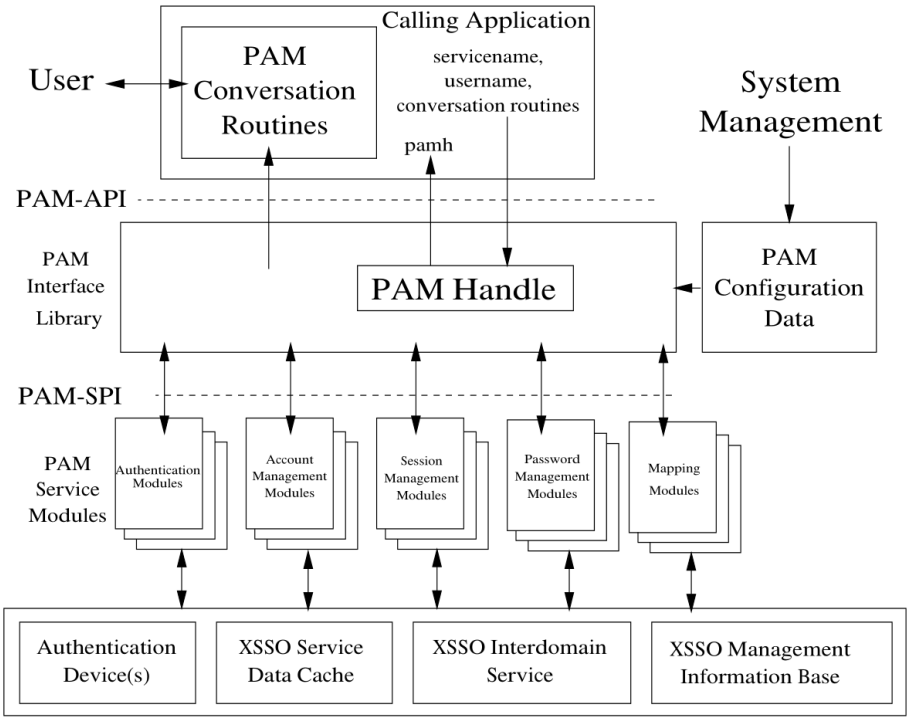

Figura 5: Arquitectura PAM

La figura 5 ilustra la relación entre las aplicaciones, la librería PAM y los modulos de autenticación. Cuando una aplicación llama a la API de , ésta carga el módulo de autenticación correspondiente (definido en el archivo de configuración del servicio). La petición inicial se envía al módulo cargado para que realice la operación deseada. Una vez realizada la operación, la capa PAM devuelve la respuesta del módulo de autenticación a la aplicación[V S95].

5.3.3. Archivos de configuración

Por defecto, el framework lee los archivos de configuración de /etc/pam.d/. Si no se encuentra el directorio, se usará el archivo /etc/pam.conf aunque este método está deprecado. Cada servicio tiene su propio archivo. Aunque varias aplicaciones puedan usar el mismo servicio, por ejemplo varias aplicaciones de bloqueo de pantalla pueden usar el servicio *screenLock*, lo estándar es que cada aplicación cree un servicio propio. Estas aplicaciones se deben encargar de crear el servicio y colocarlo en el directorio. Estos archivos siguen la siguiente estructura:

```
interfaz_modulo  control  nombre_modulo    argumentos_modulo
```

El campo *interfaz_módulo* puede tener cuatro tipos, cada uno de estos pertenece a un tipo de proceso de autorización distinto.

- *auth:* Este módulo autentica a los usuarios. Por ejemplo: Pide y verifica la contraseña del usuario.
- *account:* Verifica que se permita el acceso. Por ejemplo: Comprueba si se permite iniciar sesión a un usuario a una hora del día.
- *password:* Cambia la contraseña del usuario.
- *session:* Configura y gestiona las sesiones de los usuarios. Por ejemplo: Monta la partición del usuario en el directorio &home.

El siguiente campo, *control* indica a PAM que hacer con el resultado recibido. Hay varios tipos de control:

- *required:* Para que el usuario consiga autenticarse, el módulo debe devolver una respuesta positiva. Si el módulo devuelve una respuesta negativa, no se le notifica al usuario hasta que se acaben de pasar todos los módulos.

- *requisite:* Para que el usuario consiga autenticarse, el módulo debe devolver una respuesta positiva. Si el test falla, se avisa al usuario inmediatamente después de la ejecución del módulo.

- *sufficient:* Si el módulo devuelve un resultado negativo, éste se ignora. Si ningún módulo previo *required* ha fallado, no son necesarios más módulos para autenticar correctamente al usuario.

- *optional:* Se ignora su resultado. Solo se necesita para autenticar al usuario si ningún otro módulo referencia la interfaz.

- *include:* Referencia todas las líneas del archivo especificado.

El tercer campo del archivo, *nombre_módulo*, proporciona a PAM el nombre del módulo que contiene la interfaz del módulo. No es necesario poner una ruta absoluta al archivo si éste está en el directorio por defecto.

El último argumento, *argumentos_módulo* proporciona al módulo de autenticación los argumentos necesarios para su correcto funcionamiento.

Se muestra un ejemplo de este documento en la figura 6.

```
#%PAM-1.0
auth      required  /lib/security/pam_securetty.so
auth      required  /lib/security/pam_pwdb.so shadow nullok
auth      required  /lib/security/pam_nologin.so
account   required  /lib/security/pam_pwdb.so
password  required  /lib/security/pam_cracklib.so
password  required  /lib/security/pam_pwdb.so shadow nullok use_authtok
session   required  /lib/security/pam_pwdb.so
```

Figura 6: Archivo de configuración de un servicio de ejemplo

La primera línea es un comentario. Cualquier línea que empiece por (#) es un comentario. Los módulos de autenticación son las líneas 2, 3 y 4. El módulo

```
auth required /lib/security/pam_securetty.so
```

asegura que, si un usuario está intentando autenticarse como root, el tty desde el cual se está autenticando está en el archivo /etc/securetty. Si no está el tty ahí, cualquier intento de inicio de sesión como usuario root fallará con un mensaje de login incorrecto.

El módulo

```
account required /lib/security/pam_pwdb.so
```

comprueba la expiración de un usuario. Si la cuenta ha expirado o la contraseña ha expirado y el usuario no introduce la nueva, el módulo fallará.

El módulo

```
password required /lib/security/pam_pwdb.so shadow nullok use_authtok
```

Llama otra vez a *pam_pwdb* para cambiar la contraseña del usuario en caso de que la haya cambiado. Los argumentos le indican a módulo que escriba las contraseñas en *shadow*, que acepte una contraseña en blanco como válida y que use la nueva contraseña, en caso de que se haya cambiado.

El módulo

```
session required /lib/security/pam_pwdb.so
```

se encarga de guardar en el log del sistema el inicio de sesión. Si la operación no se consigue realizar, el usuario no podrá iniciar sesión en el sistema.

5.3.4. PAM-API

El proceso de autenticación lo gestiona la librería PAM via una llamada a la función *pam_authenticate()*. El valor devuelto por esta función determinará si el usuario ha sido autenticado. Si la librería PAM debe preguntarle al usuario por la contraseña o su nombre de usuario, lo hará. Si la librería PAM autentica mediante el uso de un protocolo silencioso, también lo hará. (Un dispositivo hardware por ejemplo)[And13]

Las funciones más importantes para nuestra interfaz de módulo (auth) de la API PAM son:

```
#include <security/pam_appl.h>
const char* nombre_servicio;
const char* usuario;
const struct pam_conv* conversacion_pam;
pam_handle_t** gestor_pam;

int pam_start(nombre_servicio, usuario, conversation_pam, gestor_pam);
```

La primera función a la que se tiene que llamar es a *pam_start()*. Esta crea el contexto PAM e inicializa la transacción PAM.

- *nombre_servicio*: Especifica el nombre del servicio que usar. Se leerá el contenido del servicio de /etc/pam.d/*nombre_servicio*
- *usuario*: Especifica el nombre de usuario. Si es *null*, el módulo se encarga de solicitarlo si es necesario
- *conversacion_pam*: Apunta a una estructura de tipo *pam_conv* que describe la función de conversación a usar.
- *gestor_pam*: Dada una respuesta correcta (PAM_SUCCESS) *gestor_pam* contiene el contexto necesario para próximas peticiones a funciones PAM.

Esta función devuelve:

- PAM_ABORT: fallo generál
- PAM_BUFF_ERR: Error en el buffer de memoria
- PAM_SUCCESS: Transacción creada correctamente
- PAM_SYSTEM_ERR: Error del sistema

La última función a la que se tiene que llamar es a *pam_end*. Invalida *gestor_pam*. A partir de este momento, ya no se podrá usar para interaccionar con la API PAM.

```
#include <security/pam_appl.h>
pam_handle_t* gestor_pam;
int estado_pam;

int pam_end(gestor_pam, estado_pam);
```

- *gestor_pam*: Estructura que identifica la operación con la API PAM.
- *estado_pam*: El estado devuelto por el último método PAM. Se le pasa a la función *cleanup()* para que ésta gestione debidamente la liberación de los datos.

Esta función devuelve:

- PAM_SUCCESS: Transacción creada correctamente
- PAM_SYSTEM_ERR: Error del sistema

PAM también proporciona funciones para acceder y actualizar información PAM. Para esto, hace uso de funciones getter y setter. Estas dos son *pam_set_item()* y *pam_get_item()*. La función setter:

```
#include <security/pam_modules.h>
pam_handle_t* gestor_pam;
int tipo_de_dato;
const void* dato;

int pam_set_item(gestor_pam, tipo_de_dato, dato);
```

La función getter:

```
#include <security/pam_modules.h>
pam_handle_t* gestor_pam;
```

```
int tipo_de_dato;
const void** dato;

int pam_get_item(gestor_pam, tipo_de_dato, dato);
```

A estas dos funciones se les pasan:

- *gestor_pam*: Estructura que identifica la operación con la API PAM.
- *tipo_de_dato*: Indica el tipo de dato que va a leer a continuación. Pueden ser: PAM_SERVICE, PAM_USER, PAM_TTY entre otros
- *dato*: Dato a cambiar o obtener de PAM

pam_set_item() devuelve:

- PAM_BAD_ITEM: Se ha intentado asignar un valor inaccesible.
- PAM_BUFF_ERR: Error en el buffer de memoria
- PAM_SUCCESS: Transacción creada correctamente
- PAM_SYSTEM_ERR: Error del sistema

pam_get_item() devuelve:

- PAM_BAD_ITEM: Se ha intentado obtener un valor inaccesible.
- PAM_BUFF_ERR: Error en el buffer de memoria
- PAM_PERM_DENIED: El valor del dato era *null*.
- PAM_SUCCESS: Transacción creada correctamente
- PAM_SYSTEM_ERR: Error del sistema

pam_authenticate() es la función más importante de la API , ya que todo gira en torno a esta. Al haber preparado todo anteriormente con la función *pam_start()* esta función es bastante sencilla de entender.

```
#include <security/pam_appl.h>
pam_handle_t* gestor_pam;
int flags;

int pam_authenticate(gestor_pam, flags);
```

A esta función se le pasan los argumentos:

- *gestor_pam*: Estructura que identifica la operación con la API PAM.
- *flags:* Pueden ser dos:
  - PAM_SILENT: no registra ningún mensaje.
  - PAM_DISALLOW_NULL_AUTHTOK: Si el usuario no tiene un token de autenticación registrado, hace que el módulo devuelva PAM_AUTH_ERR

pam_authenticate() devuelve:

- PAM_ABORT: Aplicación deberá salir inmediatamente
- PAM_AUTH_ERR: Usuario no autenticado
- PAM_CRED_INSUFFICIENT: Aplicación no tiene suficiente información para autenticar al usuario
- PAM_AUTHINFO_UNVALID: Módulos han sido incapaces de acceder a la información de autenticación
- PAM_MAXTRIES: Uno o más módulos han llegado al límite de intentos. No volver a intentar

- PAM_SUCCESS: Usuario satisfactoriamente autenticado
- PAM_USER_UNKNOWN: Usuario no reconocido

Existen varias funciones más en la API, que son esenciales para la gestión de las otras interfaces de módulo pero para entender PAM y el módulo *auth* no son necesarias.

pam_conv no es una función, sino una estructura que permite al módulo y a la aplicación que le ha llamado comunicarse directamente.

```
struct pam_conv{
    int (*conversacion) (int, struct pam_mensaje **,
          struct pam_respuesta **, void *appdata_ptr);
    void *appdata_ptr;
};
```

donde *conversacion* es:

```
int conversacion(int numero_mensajes, const struct pam_message **mensaje,
                 struct pam_response **respuesta, void *appdata_ptr);
```

Generalmente se usa para mostrar y recibir información del usuario. Se especifica cuando se llama a *pam_start()* al iniciar el procedimiento. Esta estructura debe tener dos elementos para ser válida:

- Función de conversación pasada por referencia

  Un puntero a la función de conversación. Esta función es la que gestiona la comunicación entre el módulo y la aplicación.

- *void* appdata_ptr*

  Al pasarse mediante la función de conversación al módulo y ser devuelta por el mismo, la aplicación puede usar este campo para pasar información específica de la aplicación al módulo.

5.3.5. PAM-SPI

Igual que la API, la SPI proporciona funciones para poder gestionar los cuatro pilares del módulo PAM (*authentication*, *account*, *session* y *password*) pero nosotros nos vamos a centrar en las funciones necesarias para la parte de *authentication*. El proceso de verificación se gestiona con la función *pam_sm_authenticate()*.

```
#define PAM_SM_AUTH
#include <security/pam_modules.h>
pam_handle_t* gestor_pam;
int flags;
int argc;
const char** argv;

PAM_EXTERN int pam_sm_authenticate(gestor_pam, flags, argc, argv);
```

Se usa *PAM_SM_AUTH* para asegurar que los prototipos de los módulos estáticos estén bien declarados. Los parámetros que se le pasan a *pam_sm_authenticate()* son:

- *gestor_pam*: Estructura que identifica la operación con la API PAM.
- *flags:* Pueden ser dos:
  - PAM_SILENT: no registra ningún mensaje.
  - PAM_DISALLOW_NULL_AUTHTOK: Si el usuario no tiene un token de autenticación registrado, hace que el módulo devuelva PAM_AUTH_ERR
- *argc* y *argv*: Funcionan como el paso de parámetros a una función estándar en C.

pam_sm_authenticate() devuelve:

- PAM_AUTH_ERR: Usuario no autenticado
- PAM_CRED_INSUFFICIENT: Aplicación no tiene suficiente información para autenticar al usuario

- PAM_AUTHINFO_UNVALID: Módulos han sido incapaces de acceder a la información de autenticación
- PAM_MAXTRIES: Uno o más módulos han llegado al límite de intentos. No volver a intentar
- PAM_SUCCESS: Usuario satisfactoriamente autenticado
- PAM_USER_UNKNOWN: Usuario no reconocido

Aunque esta sea la única función necesaria para implementar el módulo *authentication*, existen varias funciones que ayudan a la implementación de esta función como *pam_get_user()* o *pam_get_item()*. La segunda está descrita en la sección superior.

```
#include <security/pam_modules.h>
pam_handle_t** gestor_pam;
const char** usuario;
const char* avisoUsuario;

int pam_get_user(gestor_pam, user, prompt);
```

pam_get_uer() se usa para obtener el usuario que está solicitando la autenticación. Devuelve el nombre del usuario especificado por *pam_start()*. Si no se especificó ningún usuario, se devuelve el valor de *pam_get_item(gestor_pam, PAM_USER, ...)*. Si este valor es *null*, averigua el usuario mediante *pam_conv()*.

- *gestor_pam*: Estructura que identifica la operación con la API PAM.
- *usuario*: Se devuelve un puntero a *user*. El valor del usuario no se debería liberar ni modificar.
- *prompt*: Diálogo que se le presenta al usuario al pedirle la contraseña.

Los valores de retorno de esta función son:

- PAM_SUCCESS: Nombre de usuario devuelto satisfactoriamente.

- PAM_SYSTEM_ERROR: Puntero a *null* pasado por parámetros

- PAM_CONV_ERROR: La función de conversación proporcionada por la aplicación no obtuvo un nombre de usuario.

5.3.6. Módulos DFA para PAM

Existen varios módulos que utilizan PAM para realizar el proceso de autorización del usuario. Cada uno proporciona al usuario distintas cualidades.

- Google Authenticator

  Este [2] módulo usa One Time Password (OTP) como sistema para verificar la autenticidad del usuario.

- pam_geoip

  Este [3] módulo autentica al usuario según la zona geográfica en la que se encuentre el sistema.

- pam-face-authentication

  Este [4] módulo autentica al usuario mediante un escaneo de sus rasgos faciales.

- poldi

  Este [5] módulo autentica al usuario mediante una tarjeta inteligente.

[2] `https://github.com/google/google-authenticator-libpam`
[3] `http://ankh-morp.org/code/pam_geoip/`
[4] `https://code.google.com/archive/p/pam-face-authentication/`
[5] `http://www.g10code.com/p-poldi.html`

5.3.7. Un entorno PAM, uniéndolo todo

A pesar de ser un sistema más convencional de autenticación, se decidió crear un módulo USB porque la disponibilidad de tarjetas inteligentes o aplicaciones de autenticación es inferior a la de USBs. Además, cualquier persona posee un USB hoy en día. Esto acerca más el DFA a los usuarios.

Al ya existir un módulo [6] DFA por USB, se decidió no reinventar la rueda y ayudar de otra forma a la comunidad. Aún así, se explicarán los pasos necesarios para la creación de dicho módulo.

Antes de nada, debemos establecer el papel del USB en el módulo. El hecho de que se use un USB nos da pié a multitud de opciones. Al ser un dispositivo que el ordenador reconoce sin problemas, podemos interaccionar con el, sin necesidad de descargar e instalar programas que harían la adopción del módulo mucho más nicho.

Antes de empezar con el módulo en sí, se debe explicar cómo se detectan los USBs en el sistema. Se debe entender que antes de que nosotros podamos usar el dispositivo, el sistema debe haberlo reconocido e iniciado un proceso de comunicación entre él mismo y el USB. Una vez el USB ya está detectado por el sistema, nuestro programa debería funcionar sin problemas.

Los USB, como cualquier dispositivo hardware tienen grabados unos datos desde fábrica, para identificarlos de forma única. Estos datos son los que usa el módulo para diferenciar entre dispositivos y solo devolver *PAM_SUCCESS* cuando detecta el indicado.

Para crear este módulo, nos aprovecharemos de la SPI de PAM. Como ya hemos visto anteriormente, únicamente necesitamos implementar un método, *pam_sm_authenticate()*. Este comprueba que el dispositivo USB sea el indicado y si lo es, devuelve *PAM_SUCCESS*. Para comprobar el identificador único del USB antes debemos proporcionarle al módulo la lista de identificadores que están permitidos. Estos parámetros, además de algunos

[6] `https://github.com/ColumPaget/pam_usbkey`

otros como el usuario, la TTY y los hosts remotos. Estos parámetros los introduce el administrador de sistema en el campo *argumentos_modulo* dentro del *service* de la siguiente manera:

```
auth required pam_usbkey.so user=* rhosts=* tty=* key=12345678910
```

El módulo los recibe mediante los parámetros *argc* y *argv* de la función *pam_sm_authenticate()* y los trata para poder usarlos. Una vez tiene los parámetros, debe comparar de forma no excluyente los datos. En el fragmento superior, los parámetros indican que cualquier usuario, representado por el carácter (*) en cualquier TTY con cualquier dirección IP puede autenticarse en el módulo si detecta que el identificador USB es 12345678910. Tras comparar los parámetros, se ejecuta el siguiente pseudocódigo:

```
if (UsuarioCoincide && (TTYCoincide || HostCoincide)){
        if (! IdentificadorCoincide) return(PAM_PERM_DENIED);
        return(PAM_SUCCESS);
}
return(PAM_IGNORE);
```

Si el usuario coincide y coincide el TTY o la dirección IP desde la que se conecta el usuario, si no coincide el identificador del USB devuelve *PAM_PERM_DENIED*. Si el programa aún no ha devuelto nada, significa que si que coincide el identificador así que devuelve *PAM_SUCCESS*. Si no coincidía el usuario y el TTY o la dirección IP, devuelve *PAM_IGNORE*.

Para crear una aplicación *pam-aware* debemos usar los métodos de la API. Lo primero que debemos hacer es definir nuestra conversación. Esto permite una comunicación directa entre un módulo y una aplicación. En el código de demostración inferior, vemos como se inicializa la estructura.

```
struct pam_conv conv = {funcion_conversacion, NULL};
```

Una vez tenemos eso, debemos inicializar el gestor, este se encarga de identificar el proceso. Esto se hace con el código a continuación.

```
if ((ret = pam_start("bloqueo", usuario, &conv, &gestor_pam) != PAM_SUCCESS)
        errx(EXIT_FAILURE, "PAM: %s", pam_strerror(gestor_pam, ret));
```

Llama a *pam_start()* con los argumentos necesarios descritos anteriormente y controla el valor devuelto englobando la llamada en la condición del *if*. De esta forma, nos aseguramos que si algo sale mal en el proceso, lo tenemos controlado y podemos gestionar el error.

Una vez tenemos todo el entorno preparado, podemos hacer uso de la función *pam_authenticate()*. El resultado de llamar a esta función le dirá a la aplicación si podemos o no iniciar sesión en el sistema. Un ejemplo de implementación:

```
if (pam_authenticate(gestor_pam, 0) == PAM_SUCCESS) {
    pam_setcred(gestor_pam, PAM_REFRESH_CRED);
    pam_end(gestor_pam, PAM_SUCCESS);
    ev_break(EV_DEFAULT, EVBREAK_ALL);
    return;
}
```

Una vez entra dentro de la condición el usuario ya ha sido autenticado por PAM. Lo que queda hacer es preparar todo para que el programa termine correctamente. Estos pasos son: actualizar las credenciales, terminar la comunicación con PAM y salir del programa.

Estas dos implementaciones no van a funcionar entre sí a no ser que el administrador de sistemas las vincule en el archivo de configuración (*service*) del módulo. En este archivo, el administrador de sistemas introducirá la sentencia que une ambas cosas. Como hemos visto anteriormente, un posible ejemplo es:

```
auth required pam_usbkey.so user=* rhosts=* tty=* key=12345678910
```

En este caso, se indica que la interfaz del módulo es de autenticación, que el módulo pam_usbkey.so debe devolver una respuesta positiva. Los parámetros que se le proporciona al módulo son los cuatro de la regla.

5.4. Pruebas Doble Factor de Autenticación desarrolladas según tecnología

Durante el desarrollo del proyecto he implementado y probado una serie de mecanismos DFA. Este es el resumen de estos distintos proyectos, sus problemas y ventajas.

5.4.1. dbus

El primer proyecto que intenté, dbus, esencialmente es algo muy parecido a una autopista para el sistema. A través de él se pueden comunicar todos los programas entre ellos. Esto significa que dependiendo de los métodos disponibles en cada programa, se podría enviar (mediante un programa que esté a la escucha de una señal) una señal al programa de bloqueo diciéndole que se apague/pare, que el usuario ya está autenticado. Bastantes programas de bloqueo en distribuciones como Ubuntu tienen un sistema dbus implementado así que merecía la pena probar.

Las ventajas de este tipo de sistemas son:

- **Ligero**

  El programa que yo debía implementar simplemente estaba a la escucha de una señal/patrón constantemente. Cuando lo recibía enviaba un comando al programa de bloqueo, que bloqueaba o desbloqueaba la pantalla del sistema.

- **Sencillo de implementar**

  En Python, estamos hablando de aproximadamente 40 líneas de programa.

- **Silencioso**

  Es perfecto para no dejar rastro en el sistema. Comprobar los logs de dbus es como buscar una aguja en un pajar, por ese demonio pasan tantos datos que sería muy

poco práctico monitorizarlo. Es perfecto para una puerta trasera en un sistema, pero ese no es mi objetivo.

Los inconvenientes de una implementación de este estilo son:

- **Específico**

  Cada distribución usa un programa de bloqueo de pantalla, esto significa cambiar la lógica del programa cada vez que quiera usarlo en un ordenador distinto al mío.

- **Ejecución constante**

  Aunque sea ligero, el programa debe estar en constante ejecución. Esto, por poco que sea, consume tiempo de procesamiento del procesador.

- **No hace uso de las herramientas proporcionadas por GNU/Linux**

  Este enfoque hacia la autenticación de usuarios no hace uso de las utilidades proporcionadas por GNU/Linux para autenticar un usuario.

- **Difícil de configurar**

  Aunque su implementación es relativamente sencilla, al querer reducir la barrera de entrada del usuario medio a sistemas DFA, el proceso de configuración del programa era demasiado complicado. Se tiene que configurar para que se ejecute siempre que se inicia el sistema.

Durante una serie de iteraciones de investigación/implementación se continuaron probando programas basados en el mismo principio. Todos ellos se encontraron en GitHub.

En la primera fase de implementación se creó un script que bloqueaba el ordenador al ejecutarlo.

La siguiente fase se dedicó a buscar ejemplos de implementaciones parecidas, para ver como conseguían desbloquear la pantalla mediante dbus. Se encontraron varios pro-

yectos, entre ellos usblock. Este proyecto gestionaba el bloqueo/desbloqueo de la pantalla mediante dbus, un buen ejemplo de implementación.

Al descargar y ejecutar el programa, me encontré con que estaba usando una librería obsoleta. Hardware Abstraction Layer (HAL). Esta librería fue reemplazada por udev. Al ver esto, solamente podía leer el programa, pero no podía probarlo en mi sistema.

Haciendo balanza de las ventajas e inconvenientes, se decidió buscar una solución distinta.

5.4.2. PAM

La siguiente tecnología que encontré fue PAM. Esta vez, aseguré que era exactamente lo que necesitaba ya que no quería perder más tiempo probando e intentando arreglar cosas que no tenían sentido. Al investigar más a fondo los distintos tipos de autenticación existentes en GNU/Linux y ver que muchos de ellos eran compatibles con PAM, supe que debía seguir investigando esta tecnología.

El primer proyecto que encontré, fue PAM_USB[7]. Las siguientes fases de investigación/implementación se dedicaron a investigar y entender este módulo para PAM. Cabe mencionar que en este momento el desarrollo de un módulo propio para PAM ya no tenía sentido, ya que un usuario de GitHub llamado aluzzardi, el creador del programa había creado un módulo muy bien documentado y programado. Cuando lo descargué para probar y configurar en el sistema, el módulo no detectaba los dispositivos USB que conectaba. Tras una investigación, se averiguó que este módulo también usaba HAL para detectar los dispositivos conectados.

Al ser un proyecto popular y usado por la comunidad, se pensó que seguramente existiesen pull requests por miembros de la comunidad para portar el proyecto a udev.

[7] `https://github.com/aluzzardi/pam_usb`

Efectivamente, si existían. Al descargar y probar este [8] me encontré con que también habían complicaciones. Se proporcionaban una serie de scripts para ayudar con la configuración del módulo. Ambos parecían estar escritos en la versión 2 de Python pero ninguno parecía funcionar en mi sistema.

Al experimentar esto, cambié de proyecto a uno menos ambicioso. Empecé a usar pam_usbkey[9]. Este módulo, esencialmente hace lo mismo que PAM_USB pero no tiene tanto nivel de seguridad, ya que comprueba el ID del USB pero no una llave firmada dentro del USB.

Las ventajas de usar módulos para PAM son:

- **Integración absoluta con cualquier sistema que use PAM**

  Al ser un módulo tan extendido, esto significa que prácticamente cualquier usuario de GNU/Linux e incluso MacOS puede disfrutar de als ventajas del DFA

- **Ejecución bajo petición**

  El módulo se carga cuando una aplicación solicita que se use dicho módulo para proceder a verificar la identidad del usuario

- **Versatilidad**

  El programador del módulo puede usar cualquier método imaginable para autenticar al usuario, PAM ofrece la posibilidad de gestionar todos los módulos por igual.

- **Configurabilidad**

  El administrador de sistemas puede intercambiar los módulos que va a usar una aplicación de forma muy sencilla.

- **Configuración centralizada**

[8] `https://github.com/aluzzardi/pam_usb/pull/34`
[9] `https://github.com/ColumPaget/pam_usbkey`

El administrador de sistemas tiene todos los archivos de configuración en un directorio, ordenados según *service*

- **Facilidad de uso para usuarios no avanzados**

  Un usuario medio puede configurar un módulo PAM sin mucho problema.

Los inconvenientes de PAM son:

- **Curva de aprendizaje**

  Aunque sencilla, el programador debe aprender como funciona PAM para crear un módulo.

5.5. Creación del script

Este es la parte práctica, derivada de la parte teórica de este trabajo. El script se he escrito en BASH. Un lenguaje intérprete de comandos. Esto significa que une distintos programas disponibles, como: *ls, mv, cd, sed* o *cat* para crear un único programa, que cumple una función específica.

5.5.1. Selección del lenguaje

Se decidió usar BASH porque está disponible en todos los equipos que usan PAM, me interesaba aprenderlo inicialmente y la curva de aprendizaje inicial no parecía muy complicada.

5.5.2. Aprendizaje del lenguaje

Antes de desarrollar el script, debía aprender como funcionaba el lenguaje así que busqué un curso on-line para aprender. Este[10] curso, casualmente también código abierto a la comunidad, explica los conceptos básicos y las buenas prácticas asociadas al desarrollo de scripts en BASH.

5.5.3. Estilo de código

Durante el desarrollo del script he usado la guía de estilos de Google[11]. Pienso que ceñirse a un estilo y ser consistente a la hora de programar ayuda enormemente a la comprensión del código.

5.5.4. Qué va a solventar

El script está diseñado para ser de utilidad una vez se consigue instalar el módulo. Una vez hecho eso, se ejecuta el script para gestionar la generación de la regla a insertar en el archivo indicado. Este proceso es el más complicado de toda la gestión porque al existir multitud de opciones, el usuario menos experimentado se puede ver inundado entre ellas. El script informa en todo momento los pasos que está haciendo y explica que posibles acciones puede tomar a continuación.

La finalidad del script es facilitar la configuración de un módulo PAM.

5.5.5. Funcionalidad

El script, solo se inicia, informa al usuario que debe tener precaución, ya que una mala configuración puede resultar en el bloqueo de sesión permanente. Tras aceptar las

[10] `https://guide.bash.academy/`
[11] `https://google.github.io/styleguide/shell.xml`

cláusulas, se le pide que introduzca el USB que quiere usar como DFA. Tras haber introducido el USB se le presentan los dispositivos encontrados en el sistema y se le pide que seleccione uno. Al seleccionar uno, se pasa al siguiente paso. En este paso se informa al usuario de las distintas opciones que puede elegir y se le pide que seleccione una. Una vez tenemos la sentencia de configuración, se analiza el archivo en el que se va a insertar y se le muestran al usuario solamente las líneas necesarias con su número de línea. Se le pregunta en que línea quiere poner la regla y verificamos con el usuario. Si todo va bien, se le pregunta una última vez si está seguro de que quiere proceder. Si dice que si, escribimos la línea en el archivo de configuración. El script permite usarse con parámetros para usuarios más avanzados, si no, se usa la configuración por defecto.

5.5.6. Problemas encontrados durante el desarrollo

Durante el desarrollo, se han encontrado algunos problemas leves debidos a falta de práctica con el lenguaje. Esto es normal, ya que todos los lenguajes de programación tienen su notación específica.

5.5.7. Ventajas de usar BASH

Durante el desarrollo del script ha sido más que evidente el valor que aporta BASH. El hecho de poder integrar programas específicos en un programa global, permite al programador centrarse en la lógica de la aplicación deseada y delegar la gestión de partes específicas a los programas para completar la tarea.

6. Conclusiones

En esta sección, se analizará el desarrollo del proyecto y describirán posibles formas de continuar con él.

6.1. Conclusiones del proyecto

Durante el desarrollo del proyecto ha habido veces en las que he sentido que me superaba, que era imposible hacer lo que tenía que hacer y que la mejor estrategia era abandonar y hacer otro trabajo el año que viene. Todas estas, son las fases naturales de un proyecto tan exigente como estimulante. Es importante identificar la parte en la que te encuentras, proceder según el plan y entender que cuando las cosas se vuelven complicadas es que se está explorando más allá de la zona de confort. Me ha gustado mucho investigar profundamente el funcionamiento de PAM porque ha hecho que me dé cuenta de que puedo entender y construir software directamente relacionado con algo que uso día a día. Durante la carrera, hemos aprendido a diseñar y construir software, ahora he aprendido a aplicar lo aprendido a un caso de uso real.

Como consecuencia del trabajo, al tener que plasmar todo lo aprendido en un lienzo, he tenido que entender perfectamente como funciona la tecnología a explicar. La mejor manera de aprender es explicando.

Nunca se nos ha planteado un trabajo tan grande y este está diseñado para afianzar los conocimientos y actitudes desarrolladas durante el periodo de la carrera. En retrospectiva, entiendo y valoro la razón de un trabajo de final de grado ya que enseña a llevar un proyecto de investigación.

Finalizando el proyecto, siento que me he ampliado mis conocimientos sobre las distintas formas de autenticación y convertido en un experto en PAM. Esta sensación no la hubiese desarrollado de no ser por este trabajo.

En relación a el análisis de los sistemas DFA puedo concluir en que PAM al fin y al cabo, no es más que una herramienta para facilitar el desarrollo de buenos sistemas de autenticación. El uso que se le pueda o quiera dar más en adelante depende de cada desarrollador.

Está claro que a media que avanzan los tiempos, vamos a tener que inventar nuevas formas de verificar a los usuarios. La belleza de PAM es que estas nuevas formas se pueden acoplar directamente, como una pieza de lego, en el sistema.

Es imposible desarrollar una forma de verificar al usuario ya que a medida que se desarrollan nuevas, se descubren formas de vulnerarlas. A pesar de las implicaciones negativas que eso conlleva, también incita a la comunidad a desarrollar nuevas y mejores técnicas día a día.

6.2. Trabajos futuros

Tras finalizar este trabajo me siento completamente capaz de contribuir a la comunidad haciendo módulos de autenticación para PAM nuevos. Algunos ejemplos son:

- Autenticación mediante huella dactilar
- Autenticación mediante escaneo de retina

Creo que estos ejemplos serían muy útiles a gente que necesite una mayor seguridad de datos. También puede ser útil a empresas, que quieran centralizar el acceso a áreas restringidas mediante sistemas de autenticación como PAM. Debo publicar las partes informativas de este proyecto a mi repositorio de GitHub para asegurar que la gente se pueda beneficiar del trabajo de investigación. Debo subir también el script creado a GitHub y explicar detalladamente como utilizarlo para acercar más a los usuarios no técnicos esta posibilidad.

Siglas

ACL Access Control List.

API Application Programming Interface.

BASH Bourne Again SHell.

BBP Bug Bounty Program.

DFA Doble Factor de Autenticación.

GUI Graphical User Interface.

HAL Hardware Abstraction Layer.

LDAP Lightweight Directory Access Protocol.

MAD Microsoft Active Directory.

NIS Network Information Service.

NIST National Institute of Standards and Technology.

OTP One Time Password.

PAM Pluggable Authentication Module.

RBAC Role Based Access Control.

SASL Simple Authentication and Security Layer.

SPI Service Provider Interface.

SSH Secure SHell.

Telnet Telecomunications Network.

TTY TeleTYpewriter.

USB Universal Serial Bus.

Glosario

Access Control List Modelo de permisos POSIX-compliant simple pero potente. Uno de las primeras formas de implementaciones de privilegios.

Arch linux Sistema basado en GNU/Linux que sigue una filosofía muy enfocada a la simplicidad.

Back-end Parte que implementa la lógica del programa. Generalmente, el usuario no interacciona con ella.

Bluetooth Estandar para la comunicación inalámbrica de corta distancia entre dispositivos.

bug Un bug, es un fallo en la línea de ejecución de un programa. ya sea lógico o sintáctico..

C C es un lenguaje de programación de bajo nivél con un paradigma imperativo.

dbus Un sistema para habilitar la comunicación entre programas.

Doble Factor de Autenticación Proporciona una capa más de seguridad. Es la combinación de dos de los siguientes factores: Conocimiento (Algo que sabe el usuario) Posesión (Algo que tiene el usuario) y herencia (algo que es el usuario).

fingerprint En el campo de la seguridad informática, se le aplica este nombre al resultado del escaneo de las características, tanto hardware (número de serie) como software (versión del software, identificador) de un dispositivo o programa.

Firewalls Parte de un sistema o una red que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaicones no autorizadas.

Front-end Parte de un programa o aplicación que interactúa con el usuario.

Git Software de control de versiones, creado para facilitar el desarrollo en el kernel de Linux. Ha sido adoptado por toda la comunidad de desarrolladores.

GitHub Plataforma que implementa el cliente Git.

GNU/Linux Combinacón del kernel creado por Linus Torvalds y de los programas creados por el proyecto GNU.

kerberos Protocolo de autenticación de red.

Lightweight Directory Access Protocol Protocolo de acceso y búsqueda de datos a la base de datos distribuida X.500.

Linux Kernel creado por Linus Torvalds.

Manjaro Distribución basada en Arch linux, que simplifica el proceso de instalación y ofrece varios programas instalados de forma predeterminada.

Python Python es un lenguaje de programación de alto nivel con paradigma funcional y orientado a objetos.

Role Based Access Control Sistema que trata de gestionar la seguridad de una forma basada en roles y no tan granular como ACL.

root La cuenta con mayor nivel de privilegios en GNU/Linux.

script Programa informático que sigue una sintaxis específica creado para cumplir una funcion específica.

SetGid Set group ID on execution: Si un archivo ejecutable contiene este bit, permite a usuarios ejecutar el archivo con los mismos privilegios que el grupo que posee el archivo.

SetUid Set user ID on execution: Si un archivo ejecutable contiene este bit, permite a usuarios ejecutar el archivo con los mismos privilegios que el usuario que posee el archivo.

StickyBit Solo permite modificar el archivo/directorio por el usuario que lo ha posee.

SunSoft Empresa tecnológica que propuso la implementación de PAM.

tty Los terminales antiguos, se conectaban al ordenador mediante teletipos. tty viene de la palabra TeleTYpewriter en ingés.

Ubuntu Distribución basada en GNU/Linux comercial gratuita.

udev Proporciona un directorio dinámico de dispositivos en el que elimina o añade nodos según los dispositivos que estén conectados. Generalmente se encuentran en /dev.

USB Interfaz que permite la conexión de periféricos a diversos dispositivos.

Vim Editor de textos modal.

Wi-Fi Red inalámbrica que permite la comunicación entre dispositivos, simulando una conexión por cable.

X.500 Es una base de datos distribuida que ofrece la posibilidad de buscar información por nombre (páginas blancas) y buscar información (Páginas amarillas).

Bibliografía

[V S95] R. Schemers V. Samar. *UNIFIED LOGIN WITH PLUGGABLE AUTHENTICATION MODULES (PAM)*. Oct. de 1995. URL: http://www.linux-pam.org/pre/doc/rfc86.0.txt.gz.

[Hat97] Red Hat. *User Authentication with PAM*. Abr. de 1997. URL: http://archive.download.redhat.com/pub/redhat/linux/4.2/en/os/sparc/doc/rhmanual/doc073.html.

[M W97] S. Kille M. Wahl T. Howes. *Lightweight Directoyr Accesss Protocol (v3)*. Dic. de 1997. URL: https://tools.ietf.org/html/rfc2251.

[Mye97] J. Myers. *Simple Authentication and Security Layer (SASL)*. Oct. de 1997. URL: https://tools.ietf.org/html/rfc2222.

[MIT98] MIT. *Kerberos, the network authentication protocol*. Jul. de 1998. URL: https://web.mit.edu/kerberos/.

[Moo98] Gordon Moore. *Cramming More Components onto Integrated Circuits*. Ene. de 1998. URL: http://www.cs.utexas.edu/~fussell/courses/cs352h/papers/moore.pdf.

[Mik02] Arie van Bennekum Mike Beedle. *Manifesto for Agile Software Development*. Feb. de 2002. URL: http://agilemanifesto.org/.

[NIS02] NIST. *Software Errors Cost U.S. Economy 59.5 Billion Annually*. Oct. de 2002. URL: https://web.archive.org/web/20090610052743/http://www.nist.gov/public_affairs/releases/n02-10.htm.

[IBM09] IBM. *Understanding and configuring PAM*. Mar. de 2009. URL: https://www.ibm.com/developerworks/linux/library/l-pam/index.html.

[Sta12] Richard Stallman. *GNU General Public License*. Ene. de 2012. URL: https://www.gnu.org/licenses/gpl-3.0.en.html.

[And13] Thorsten Kukuk Andrew G. Morgans. *Linux-PAM, The Application Developer's Guide*. Sep. de 2013. URL: http://www.linux-pam.org/Linux-PAM-html/adg-overview.html.