UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais - iNCE

BRUNO SILVA FERREIRA

AUTENTICAÇÃO SEGURA: autenticação alternativa usando biblioteca PAM.

Rio de Janeiro
2014

BRUNO SILVA FERREIRA

AUTENTICAÇÃO SEGURA: autenticação alternativa usando biblioteca PAM.

Monografia apresentada ao Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à conclusão do curso de especialização em Gerência de Segurança da Informação (MSI).

Orientador: Oswaldo Vernet, D.Sc.

Rio de Janeiro

2014

AUTORIZAÇÃO

BRUNO SILVA FERREIRA, autorizo o Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais (iNCE) da UFRJ a divulgar total ou parcialmente a presente monografia através de meio eletrônico e em consonância com a orientação geral do S*i*BI.

Rio de Janeiro, 15/12/2014.

_____

Assinatura

À minha família, por sua capacidade de acreditar e investir em mim.

AGRADECIMENTOS

A Deus por ter me dado saúde e força para superar as dificuldades.

A esta universidade, a qual também faço parte, seu corpo docente, direção e administração que oportunizaram a janela que hoje vislumbro.

Ao meu orientador professor Oswaldo Vernett, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos.

Aos meus pais, Gelson e Sandra, pelo amor, incentivo e apoio incondicional.

A minha avó Lizete pelos lanches surpresa nas horas de estudo.

A minha excepcional namorada pela compreensão e incentivo.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu sincero muito obrigado.

RESUMO

FERREIRA, Bruno. **AUTENTICAÇÃO SEGURA: autenticação alternativa usando biblioteca PAM**. Rio de Janeiro, 2014. Monografia (Especialização Gerência de Segurança da Informação) - Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais (iNCE), Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2014.

A autenticação baseada em senhas digitadas pelo teclado ainda é a forma mais comum de autenticação em computadores. Este trabalho tem por objetivo propor uma forma alternativa de utilizar senhas complexas, seguindo um padrão de complexidade baseado na personalização que o usuário deverá montar.

Além da senha complexa o usuário não pode repetir as que já foram digitadas, evitando assim outros tipos de ataque como K*eylogging* e *ShoulderSurfing.* Com o término do experimento o programa será disponibilizado para que a comunidade de software livre utilize e melhore o projeto, visto que não existe sistema totalmente seguro. A alternativa sugerida neste trabalho deve ser usada como uma autenticação secundária, incrementando a política de segurança.

ABSTRACT

The keyboard typed passwords still the most common way until today of athentication on computers. The key goal of this work is propose an alternative way of using complex passwords, following a pattern of complexity created by the user. For example, can be configured a sequency pattern of 5 uppercase letters, 5 numbers, and 5 symbols, by the way, the user just have to input any combination of character inside this parameters and will be authenticated. For this work will be developed an experiment based on the free authentication unified libray PAM. Basically, following the last example, the user can be accepted using passwords such as: "**ASDJO09877*&*!@**", "**RRTVZ45688&%%%@**", "**CZCQP33240)((!!**", and so on. Technically talking, thoses are pretty strongs passwords to break with attack like brute force, and should take an eternity to be cracked by an ordinary computer. But, this is an example of a strong password to memorize and its a real struggle, whats turns down a lot of users to use such combinations.

In addition to the complex password, the user cant repeat those already entered, thus avoiding other attacks as Keylogging and ShoulderSurfing. In the end of the experiment the program / code will be shared with the free software community to use and improve the project, since there is no a totally secure system. The alternative given in this paper should only be used as a secondary authentication method, increasing the security policy.

LISTA DE ILUSTRAÇÕES

FIGURA 1: TODAS AS SENHAS SERÃO ACEITAS PELO MÓDULO POIS SEGUEM O PADRÃO...14
FIGURA 2: 92% DAS SENHAS FORAM REUTILIZADAS EM OUTROS SERVIÇOS.....16
FIGURA 3: ESQUEMA PAM DETALHADO. 25
FIGURA 4: CADA PILHA É RESPONSÁVEL POR UMA ETAPA. 29
FIGURA 5: CONFIGURAÇÃO DO ARQUIVO /ETC/PAM.D.....31

LISTA DE QUADROS

Quadro1 – Tipos de Ataque..... 19
Quadro2 – Formas de obtenção de senhas 20
Quadro3 – Grupos de gerência 26
Quadro4 – Sinalizadores de controle 28

LISTA DE ABREVIATURAS E SIGLAS

API	Aplication Programming Interface
AUTH	Authentication
BSD	Berkeley Software Distribution
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
PAM	Pluggable Authentication Modules
PIN	Personal Identification Number
PSN	PlayStation Network
NCE	Núcleo de Computação Eletrônica
RFC	Request For Comments
UFRJ	Universidade Federal do Rio de Janeiro
USB	Universal Serial Bus

SUMÁRIO

1 INTRODUÇÃO 12
1.1 MOTIVAÇÃO 12
1.2 OBJETIVOS 13
1.3 RELEVÂNCIA 15
2 METODOLOGIA DE PESQUISA 17
2.1 TIPO DE PESQUISA 17
2.2 TIPOS DE ATAQUE MAIS COMUNS 18
3 DESENVOLVIMENTO 21
3.1 AUTENTICAÇÃO NO UNIX 21
3.2 AUTENTICAÇÃO USANDO PAM 23
3.3 ENTENDENDO A CONSTRUÇÃO DE UM MÓDULO 26
3.3.1 Compilando o módulo 29
4 CONCLUSÃO 33

INTRODUÇÃO

Segurança da informação certamente é um dos assuntos mais importantes quando falamos sobre internet e sistemas da informação, afinal ninguém quer ter seus dados e informações vasculhadas ou destruídas por um estranho. Atualmente existe uma infinidade de golpes na internet, principalmente para roubar as senhas de acesso dos usuários, e como um ciclo vicioso a cada golpe descoberto outros surgem em seu lugar.

1.1 MOTIVAÇÃO

Uma recomendação comum que é passada para os usuários nos sistemas operacionais, serviços online, ou aplicativos, é que se use senhas complexas. Porém esta tarefa é cada vez mais complicada. Um dos motivos alegados é a sobrecarga de tarefas que as pessoas têm na sua rotina diária, logo decorar uma senha como "**123456"** se torna mais atraente que "**ASDBBUJO#09877153$*&*!@")(**", por exemplo. Apenas para efeitos de comparação, na primeira senha numérica um computador comum a quebraria rapidamente utilizando o método de força bruta. Enquanto que a segunda senha levaria 1 octilhão de anos para ser quebrada, de acordo com o *site howsecureismypassword.net*.

No mundo Linux/Unix/BSD (Berkeley Software Distribution) existe uma biblioteca que auxilia o processo de autenticação chamada PAM (Pluggable Authentication Modules). O Linux -PAM é uma suíte de bibliotecas distribuídas que viabilizam ao administrador do sistema especificar a forma que as aplicações autenticam os seus usuários. Basicamente, é um mecanismo bastante flexível para autenticação de usuários.

Como a biblioteca já tem anos de utilização e é bem consolidada na comunidade *opensource* ela se tornou uma opção notável para a implementação da parte prática deste trabalho. Já que não será preciso se preocupar com as excentricidades de cada sistema operacional visto que o PAM faz o papel de intermediador entre a aplicação e a autenticação nos serviços do sistema. Assim naturalmente é possível focar na ideia central do trabalho que é propor uma autenticação alternativa.

Por fim, a autenticação é um dos assuntos mais pertinentes na segurança da informação e iniciativas para sua proteção pode ser vista em várias camadas desde os protocolos de comunicação, trocas de chaves seguras, *hardware* e etc. Porém as iniciativas voltadas para auxiliar o usuário são mais escassas quando comparadas as outras soluções.

1.2 OBJETIVOS

O objeto principal deste trabalho é demonstrar e propor uma nova forma de autenticação baseada na biblioteca PAM, usando senhas complexas, porém, personalizada pelo administrador/usuário do sistema. Assim ao invés de decorar uma senha absoluta, o usuário terá que memorizar o padrão criado, e o sistema irá comparar se o que foi digitado confere com o padrão criado. Suponhamos que o administrador tenha definido que a senha deverá ter 15 caracteres e será divididos em 3 blocos. Logo, se o usuário criar o seguinte padrão: 5 letras minúsculas; 5 dígitos numéricos; 5 letras maiúsculas, qualquer sequência de caracteres digitada dentro desse padrão pré estabelecido pelo administrador/usuário será aceita para autenticação.

Por exemplo, pode ser configurado um padrão de 5 letras maiúsculas, 5 números, e 5 símbolos, dessa forma, bastaria o usuário inserir qualquer combinação dentro desses parâmetros que ele seria autenticado. Para isso será desenvolvido um experimento baseado na biblioteca de autenticação unificada *PAM. B*asicamente, seguindo o exemplo anterior, o

usuário poderá ser admitido utilizando senhas como: "**ASDJO09877*&*!@**", "**RRTVZ45688&%%%@**", "**CZCQP33240)((!!**", e assim por diante. Computacionalmente, estas são senhas complexa para ataques de força bruta, e que exigiriam longo tempo para ser quebrada por um computador comum. Porém, este é um exemplo de senha complexa de se decorar, o que acaba desestimulando a maioria dos usuários a utilizar combinações desse nível. Exemplos:

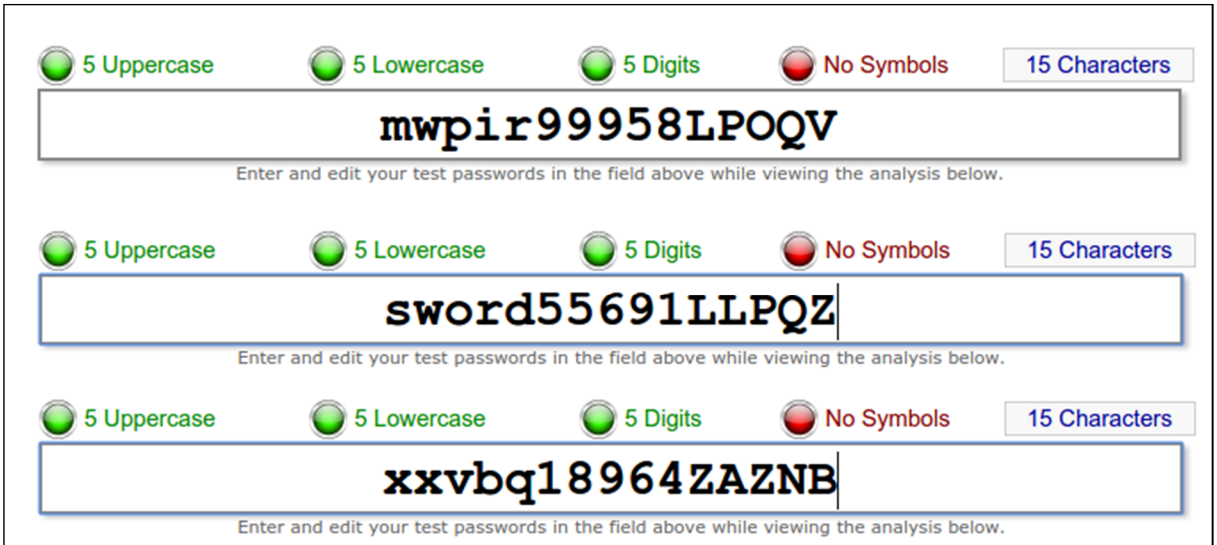

Figura 1: Todas as senhas serão aceitas pelo módulo pois seguem o padrão.

Dessa forma, de acordo com o estudo do pesquisador Steve Gibson do site Gibson Research's, "[...] aumentamos o tamanho do *haystack*, ou seja, o tamanho do palheiro, pois sabemos que a senha (agulha) será encontrada (quebrada) pelo atacante em algum momento." (GIBSON, 2012). O tempo para quebrar a senha por força bruta depende totalmente do tamanho do "*search space*", ou espaço de busca. A primeira senha do exemplo tem um tamanho de espaço de busca de 7.82×10^{26}, o que levaria 2.48 mil séculos em um cenário de conjunto de máquinas tentando ao todo 100 trilhões de tentativas por segundo. É claro que esse número pode variar bastante com o passar dos anos devido ao avanço das técnicas de força bruta e potência do hardware em geral. Porém, computacionalmente é um belo exemplo de senha complexa e ao mesmo tempo simples para o usuário, que precisará lembrar apenas a ordem do tipo de caractere e a quantidade.

Este trabalho visa abordar a questão da autenticação segura em ambientes Linux/Unix/BSD utilizando senhas complexas sem sobrecarregar o usuário. O objetivo secundário deste trabalho é disponibilizar para comunidade de software livre um módulo PAM com uma nova proposta e aberto para melhorias.

1.3 RELEVÂNCIA

A autenticação é uma das camadas mais importantes da segurança de um sistema. É através da autenticação que uma pessoa (ou programa) devidamente autorizada tem a permissão para acessar um sistema, a autenticação confirma se a identidade é confiável. É preciso distinguir autorização de autenticação. "A autenticação lida com a questão de determinar se o usuário está ou não se comunicando com um processo específico. A autorização se preocupa com o que esse processo tem permissão para fazer." (TANENBAUM, 2003, p.835).

De nada adiantam firewalls corporativos ou sistemas de detecção de intrusão de última geração se a senha de um sistema for a mais óbvia possível, estiver à vista de todos ou for compartilhada entre um grupo de pessoas.

Como constatado no estudo "*A large-scale study of web password habits*" (HUNT, 2011), para a maioria dos usuários, o problema do crescente aumento de senhas para contas é resolvido com uma pequena coleção de senhas que ele reutiliza. Para um usuário com 30 contas/senhas, o problema não é lembrar as 30 senhas, mas sim lembrar qual das 5 ou 6 senhas foi usada. Isto aparece ser feito usando uma combinação de memória, papéis de recado, tentativa e erro, e *resets* de senha.

No breve estudo sobre o caso das 77 milhões de senhas roubadas da PSN (PlayStation Network), rede social para jogos da Sony, vários lições foram aprendidas, como é mostrado

no site do pesquisador e analista de segurança da informação Troy Hunt. Uma delas é reutilização das senhas entre outros sistemas da empresa, isto mostra como as pessoas acabam usando a mesma senha por conveniência aumentando as chances de perder de uma só vez suas credencias em diversos serviços. No estudo foi apresentado que 92% das senhas eram reutilizadas em outro serviço.

"Nada disso é realmente uma surpresa, porém continua sendo alarmante. Nós sabemos que as senhas geralmente são muito curtas, muito simples, muito previsíveis e muitas vezes

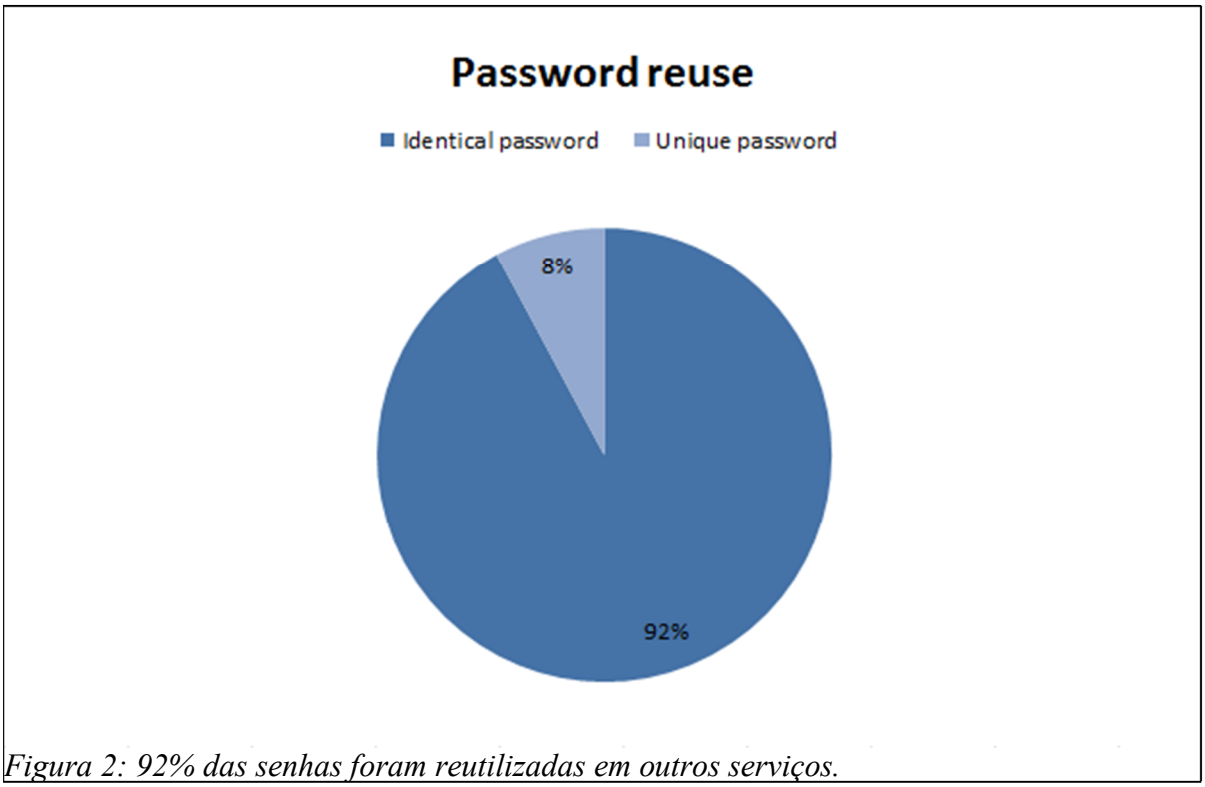

Figura 2: 92% das senhas foram reutilizadas em outros serviços.

repetidas pelos usuários em outros serviços." (HUNT, 2011).

Existem várias alternativas interessantes como os gerenciadores de senhas *KeepassX* e o *1Password*. Esses programas armazenam todas as suas senhas em uma base única, que é criptografada. A ideia é que o usuário apenas lembre a senha mestra para abrir a base e consultar todas as suas senhas, podendo utilizar senhas mais complexas sem o desgaste de decorá-las. A ideia é muito boa, porém existe um alto risco envolvido caso a senha mestra seja descoberta, pois o usuário perderá todas as suas senhas de uma só vez.

Com a alternativa que será apresentada é possível que se abra um campo de pesquisa novo, pois o método utilizado ainda é pouco difundido. O tema autenticação segura é de extrema relevância para a segurança da informação como mostram diversos estudos.

METODOLOGIA DE PESQUISA

> Tomada num sentido amplo, pesquisa é toda atividade voltada para a solução de problemas; como atividade de busca, indagação, investigação, inquirição da realidade, é a atividade que vai permitir, no âmbito da ciência, elaborar um conhecimento, ou um conjunto de conhecimentos, que nos auxilie na compreensão desta realidade e nos oriente em nossas ações.(ELISABETE MATALLO MARCHESINI DE PÁDUA, 2002, p. 31)

2.1 TIPO DE PESQUISA

A metodologia de pesquisa utilizada foi a exploratória, visto que o objeto deste trabalho será propor uma metodologia diferenciada de autenticação. É também explicativa, porque objetiva tornar o assunto inteligível, verificando os fatores que contribuem para ocorrência do fenômeno, no caso a utilização de senhas frágeis pelos usuários e a biblioteca de autenticação PAM.

Foram realizados experimentos em laboratório com o módulo criado resultando em um estudo de caso com finalidade prática.

2.2 TIPOS DE ATAQUE MAIS COMUNS

Um atacante pode ser definido como alguém que tenta violar ou uso não autorizado de um serviço, computador ou rede, de forma bem ou mal sucedida. Este conceito é bem amplo e pode abranger tanto algo grave como roubo de dados confidencias como um simples mau uso de sistema de e-mail para envio de spam, mas normalmente a primeira barreira encontrada pelos atacantes é a senha.

Para que um sistema tenha certeza que o usuário é quem diz ser, torna-se necessário que haja uma autenticação. Para isso existem os Fatores de Autenticação, divididos em 3 categorias:

• Algo que só o usuário sabe: senha, frase de segurança, PIN (Personal Identification Number).

• Algo que só o usuário possui: cartão de identificação, security token.

• Alguma característica que só o usuário tem: impressão digital, padrão de retina, padrão de voz e outras características biométricas.

A senha é o método mais utilizado por ser mais barato e simples de utilizar embora apresente alguns inconvenientes. Se for muito simples é fácil de ser quebrada; se for muito complexa é mais fácil de esquecer. Mas, como (teoricamente) trata-se de um segredo do tipo "só você sabe", é suficiente para comprovar a autenticidade desse usuário.

As senhas possivelmente são o elo mais fraco na segurança da informação. Qualquer sistema que precisa estar seguro deveria ser protegido por algum esquema de autenticação em multi fator, como biometria, *smart cards*, *tokens*, ou certificados digitais. As senhas podem ser facilmente comprometidas para serem confiáveis em autenticação de uma etapa. Entretanto, implementar autenticação multi fator pode ser difícil, caro, e em alguns sistemas pode não ser complemente suportado. Por essa razão, continua sendo importante entender os diferentes métodos de quebra de senha ou adivinhação:

Quadro 1 – Tipos de Ataque

Ataque	**Descrição**
Advinhação	Normalmente é primeiro tipo de ataque empregado por ser simples e algumas vezes eficaz. Senhas como "*password*", "*administrator*", nome dos filhos, próprio nome, nome do cônjuge, datas de nascimento, ou número de telefone ainda são usados pelos usuários.
Wordlist/Dicionário	O atacante usará um programa que tentará todas as possibilidades de uma lista de palavras ou dicionário. Ataques de dicionário podem ser feitos repetidamente tentando se logar no sistema, ou pela coleta de senhas criptografadas e tentando encontrar uma correspondência ao se criptografar as senhas da lista de palavras.
Bruteforce/Força bruta	Este ataque normalmente é feito por último, ou depois de um ataque de dicionário, pois pode demorar muito. Usando uma única CPU moderna, uma senha de quatro letras minúsculas pode ser quebrada em poucos minutos. Dependendo se a senha contiver caracteres especiais, combinação de letras maiúsculo-minúsculas, números, ela será praticamente impossível de ser descoberta. Porém, este tempo pode ser significativamente diminuído com o uso de técnicas de processamento distribuído, onde várias máquinas trabalham simultaneamente na quebra da senha.
RainbowTables/Senhas pré-computadas	É essencialmente um ataque de força bruta onde o trabalho já foi feito antes. Tabelas com os *hashes* das palavras são geradas usando computação distribuída. Uma vez que a tabela tenha sido gerada, o tempo para encontrar a senha de qualquer tamanho é insignificante, mesmo senhas complexas podem ser encontradas em minutos. Basta ter a tabela adequada, já existem sites que vendem tabelas pré-computadas para quebra de senha de sistemas Microsoft, por exemplo.

É importante ressaltar que com o aumento exponencial do poder computacional das máquinas fica cada vez mais fácil e rápido a quebra de senhas mais simplórias. O que antes levava dias e até meses para um computador comum, hoje pode ser feito em minutos até por um *smartphone*. Portanto também é importante conhecer e prevenir-se contra os métodos de aquisição de senhas. Antes de atacar a senha propriamente dita, o atacante deverá coletá-la de alguma forma, existem diversos métodos para isso, porém os mais comuns são:

Quadro 2 – Formas de obtenção de senhas

Método	**Descrição**
Keylogger/Monitoramento de toques do teclado	Feito via software malicioso que se instala na máquina através de *malwares*, engenharia social, invasão, é muito usado por atacantes que roubam dados bancários conhecidos como *bankers*, e também para o roubo de senhas.
ShoulderSurfing/Observação	Ataque simples que não requer software ou hardware, apenas olhos treinados e se posicionar estrategicamente atrás da vítima. Também pode ser feito via câmera de vigilância.
Engenharia Social	Utilizam-se técnicas de persuasão e convencimento. Uma técnica bem sucedida e comum é simplesmente ligar para o *Helpdesk* e dizer: "Olá, sou Bob diretor sênior de TI em São Paulo. Tenho uma apresentação para dar ao meu chefe, o CIO, e eu não posso entrar no servidor XYZ para obter minhas anotações. Você poderia por favor "resetar" a minha senha agora? Eu tenho que estar na reunião em 5 minutos." Muitos operadores desavisados fariam o *reset* da senha.
Roubo da base de senhas	A base de dados completa das senhas dos usuários é normalmente armazena em um único arquivo no disco. No UNIX, este arquivo é o /etc/passwd ou /etc/shadow, e no Windows, este arquivo é o SAM (C:\Windows\System32\config\SAM) ou o arquivo base do Active Directory, ntds.dit. Seja qual for o caso, o atacante de posse desses arquivos pode rodar programas de quebra de senha a fim de encontrar senhas fracas dentro do arquivo.
Replay Attack	Em alguns casos o atacante não precisa quebrar a senha. Ele pode usá-la mesmo de forma criptografada para logar no sistema. Isto usualmente requer a reprogramação do software cliente para fazer o uso da senha criptografada.
Sniffing /Interceptação de tráfego	Este ataque acontece de forma passiva sem transmitir nada pela rede, apenas capturando o tráfego. Vários protocolos (Telnet, HTTP Basic, FTP) usam senhas em texto claro, o que significa que por padrão o protocolo não criptografa nada entre o cliente e o servidor. Um atacante com um analisador de protocolos pode observar e capturar o tráfego a procura de senhas. Nenhum esforço a mais é necessário, basta o atacante usar a senhas capturadas e se autenticar nos sistemas. Entretanto, a maioria dos protocolos usa algum tipo de criptografia nas senhas. Nestes casos, o atacante terá que realizar um ataque de dicionário ou força bruta à senha, a fim de tentar descriptografá-la.
KeyStroke	Feito via hardware, normalmente usa-se um *stick* USB ou PS/2 plugado disfarçadamente na máquina alvo. Uma vez que o usuário pluga um desses drives, um programa cavalo de Tróia é instalado, gravando as teclas digitadas e enviando os dados de volta para o atacante.

DESENVOLVIMENTO

Toda e qualquer reflexão sobre o porvir da Sociedade da Informação deve apoiar-se numa análise da mutação contemporânea da relação com o saber, em que a velocidade do surgimento e da renovação dos saberes e do *know-how* é avassaladora. Neste capítulo, serão apresentadas as etapas de autenticação básica de sistemas *Unix*, autenticação usam a biblioteca PAM, e a construção de um módulo PAM.

4.1 AUTENTICAÇÃO NO UNIX

A segurança da informação é um assunto que sempre foi respeitado no mundo Linux/Unix/BSD. Existem diversas iniciativas de segurança da informação que surgiram nessas plataformas. Historicamente a autenticação de usuários no UNIX era baseada na entrada de login e senha que era checada com o conteúdo em texto puro do aquivo /etc/passwd. Conforme as redes evoluíram, ficou claro que não era seguro armazenar senhas em texto claro em um arquivo legível para todos os usuários, como de costume um incidente de segurança mostrou isso. Posteriormente em 1987, Jule Haugh criou a suíte *Shadow Password* e em 1992 foi portada para o Linux, apenas um ano depois do anúncio oficial do projeto Linux. Até hoje a suíte Shadow Password é usada na maioria das distribuições Linux, basta verificar a presença do arquivo /etc/shadow ou /etc/master.passwd que armazena os "hashes" das senhas com acesso apenas pelo super usuário.

Um *hash*, também conhecido como "*digest*", é uma espécie de "assinatura" ou "impressão digital" que representa o conteúdo de um fluxo de dados. Os *hashes* não são "cifragens", apenas transformam os dados do texto (claro ou cifrado) num pequeno "*digest*",

de tamanho fixo, numa operação de via única, não importando o tamanho da entrada. As entradas devem possuir qualquer comprimento, porém as saídas devem sempre possuir o mesmo comprimento. A função *hash* **h=H(m)** deve ser fácil de calcular para qualquer **m** dado. A função inversa $\mathbf{m=H^{-1}(h)}$ deve ser impossível de ser obtida, **H(x)** deve ser injetiva, ou seja; não podem existir **x** e **y** onde $\mathbf{x \neq y}$ tal que **H(x) = H(y)**. Pequenas variações na senha também geram *hashes* totalmente diferentes impossibilitando a dedução pela sequência.

Senha	Hash (MD5 – digest de 128bits)
abc	900150983cd24fb0d6963f7d28e17f72
abb	ea01e5fd8e4d8832825acdd20eac5104
aba	79af87723dc295f95bdb277a61189a2a

Virtualmente todos os sistemas *Unix-like* usam senhas com sistema *shadow*. O conceito básico do sistema *shadow* é fácil de entender. Kurt Seifried, autor de "Linux Administrator's Security Guide" explicou o surgimento do *shadow.* Por muitos anos a solução foi bastante simples e eficaz, bastando fazer um *hash* das senhas e armazenar esse *hash*; quando um usuário se autentica, faz-se um *hash* da senha digitada e, se os *hashes* coincidirem, a senha é, obviamente, a mesma. (SEIFRIED, 2001).

Com o passar do tempo, o poder computacional cresceu e ficou mais fácil quebrar até mesmos as senhas com *hash*; com isso o Linux e outros sistemas UNIX migraram para sistemas de criptografia mais fortes (geralmente o *BlowFish*). Além do nome de usuário e da senha com *hash*, o arquivo /etc/shadow também contém informações sobre o envelhecimento da senha.

4.2 AUTENTICAÇÃO USANDO PAM

PAM (Pluggable Authentication Modules) foi proposto primeiro pela Sun Microsystems em 1995 por Vipin Samar e Charlie Lai, em Outubro do mesmo ano foi escrita a RFC 86.0 (Request For Comments) por V. Samar e R. Schemers, ambos da SunSoft. No ano seguinte foi integrado para o RedHat Linux, subsequentemente, PAM se tornou o padrão de esquema de autenticação para o Linux e outras variantes do UNIX. PAM tem se padronizado como um componente do processo de padronização X/Open Unix. Isto resultou no padrão X/Open Single Sign-on(XSSO).

O PAM é uma API (Aplication Programming Interface) que cuida da autenticação de um usuário para um serviço. Antes do PAM, aplicativos como login (e rlogin, telnet, rsh) procuravam pelo nome do usuário em /etc/passwd e depois comparavam os dois e autenticavam o nome digitado pelo usuário. Todos os aplicativos usavam esses serviços compartilhados, embora os detalhes da implementação e a autoridade para configurá-los não fossem compartilhados.

Em seguida, os desenvolvedores do aplicativo tentavam codificar seus próprios processos. Com isso veio a necessidade de separar o aplicativo e o módulo de segurança (um módulo de segurança comum pode ser compartilhado por aplicativos e ser configurado conforme necessário).

O mecanismo PAM integra vários esquemas de autenticação de nível inferior em uma API de nível superior que permite que os programas que dependem da autenticação sejam gravados independentemente do esquema de autenticação subjacente. O principal recurso do PAM e a configuração dinâmica da autenticação através de um arquivo /etc/pam.d ou /etc/pam.conf.

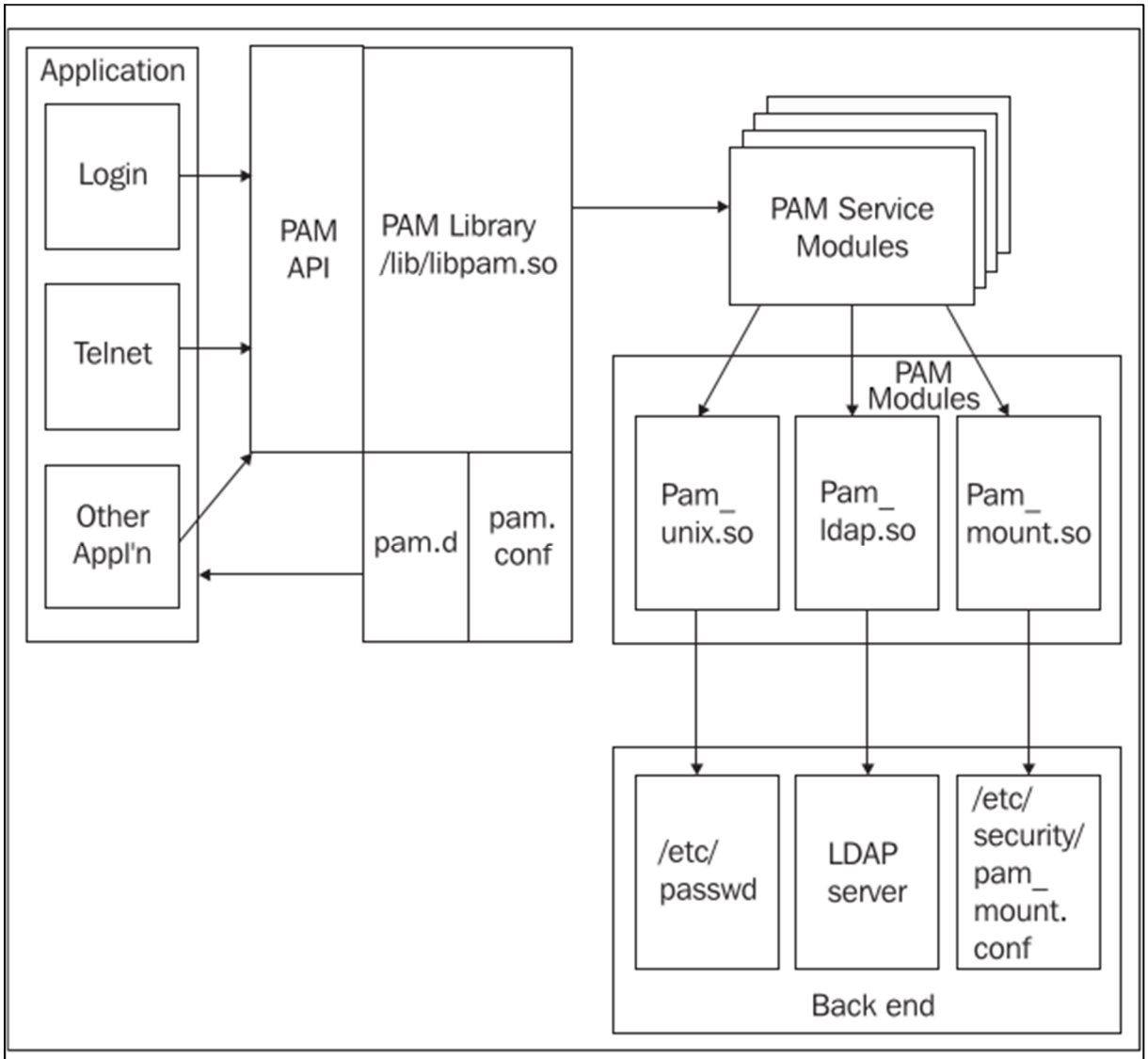

Figura 3: Esquema PAM detalhado.

O PAM pode ser configurado para impedir que determinados programas autentiquem os usuários e para avisar quando determinados programas tentam fazer a autenticação. Os programas do PAM usam os módulos do PAM (módulos de autenticação): eles são anexados aos aplicativos no tempo de execução para funcionar. Um módulo pode oferecer métodos para autenticação de usuários com um determinado *back end* ou preparar um ambiente de trabalho para os usuários. Uma pilha de módulos habilita o uso de várias técnicas de validação durante uma tentativa de *login*. O administrador pode até mesmo exigir que todos os módulos em uma pilha devem aceitar a tentativa de *login* para autenticar o usuário, ou pode escolher que um módulo é suficiente, ou misturar e combinar conforme necessário, sem ter que recompilar os programas ou reiniciar o computador.

PAM permite *single sign-on* através de múltiplos servidores. Adicionalmente, existe um grande número de *plugins* que variam seu potencial. É importante avaliar o nível de segurança provido pelos *plugins,* pois o sistema é tão seguro quanto seu elo mais fraco.

A grande vantagem para os administradores de sistemas é que o conhecimento em implementações PAM em um sistema UNIX pode facilmente ser levado para outro sistema da família UNIX. Logo aprender PAM fará dele um melhor administrador de sistemas UNIX, distribuições Linux, família BSD e Solaris.

Além do mais, o arquivo de senha não é escalável. Ele pode funcionar bem com 100 usuários, mas trabalhando com 5000 usuários é uma história completamente diferente. PAM pode escalar facilmente para 10000 usuários dependendo do *back end* escolhido.

4.3 ENTENDENDO A CONSTRUÇÃO DE UM MÓDULO

É possível achar diversos módulos para quase qualquer situação, ou talvez uma combinação de módulos que pode resolver seu problema. Mas ainda assim é possível terminar em uma situação que não possa encontrar um módulo adequado e precise construir seu próprio módulo PAM.

A biblioteca PAM *runtime* tem uma estrutura API (Application Programming Interface) bem definida. A PAM API é bem extensa e parecida em todos os sistemas operacionais UNIX e Linux. Apenas um pequeno número de diferenças existe, mas qualquer programador consegue contornar. As diferenças são primariamente relacionadas a função de conversação, mostrada mais abaixo. A Linux-PAM fornece um como uma função da biblioteca enquanto outras implementações PAM exigem o programador desenvolva uma função de conversação.

Existe um utilitário chamado pamtester desenvolvido por Moriyoshi Koizumi, a fim de ajudar os desenvolvedores de módulos, mas também pode ajudar os administradores de sistemas para testar novas configurações PAM. .

A biblioteca separa as tarefas de autenticação em 4 grupos de gerência independente, são eles:

Quadro 3 – Grupos de gerência

Grupo de gerência	**Descrição**
Auth	O módulo *auth* adquire da aplicação (*prompt*) do suário dados para realizar a identificação, assim como uma senha.
Account	O módulo *account* verifica os vários aspectos da conta do usuário, como limite de acesso em um período de tempo particular, ou de localização particular. Pode ser usado

	para limitar o acesso ao sistema baseado no sistema de recurso.
Session	O módulo *session* é usado para prover funções antes e depois de estabelecer a sessão. Este contém um conjunto do ambiente, como logando etc.
Password	O módulo *password* é normalmente empilhado no módulo *auth*. Ele é responsável pela atualização do token de autenticação do usuário, muita das vezes uma senha.

Dessa forma, as bibliotecas PAM incluem funções para iniciar o módulo PAM e checar as credenciais de autenticação. Como pode ser observado, cada etapa é atribuída a um módulo ou pilha do inglês *stack*. Este é um dos recursos mais úteis e é chamado de *stacking*, ou empilhamento de módulos. Para cada grupo de gerência pode ser definido um conjunto ou uma pilha de módulos, os quais serão usados um de cada vez. Assim nota-se que os detalhes da autenticação são completamente escondidos da aplicação. Mas a beleza do PAM não para por aqui, mesmo que o programa já tenha sido compilado, você pode mudar o comportamento da autenticação editando o arquivo de configuração em /etc/pam.d.

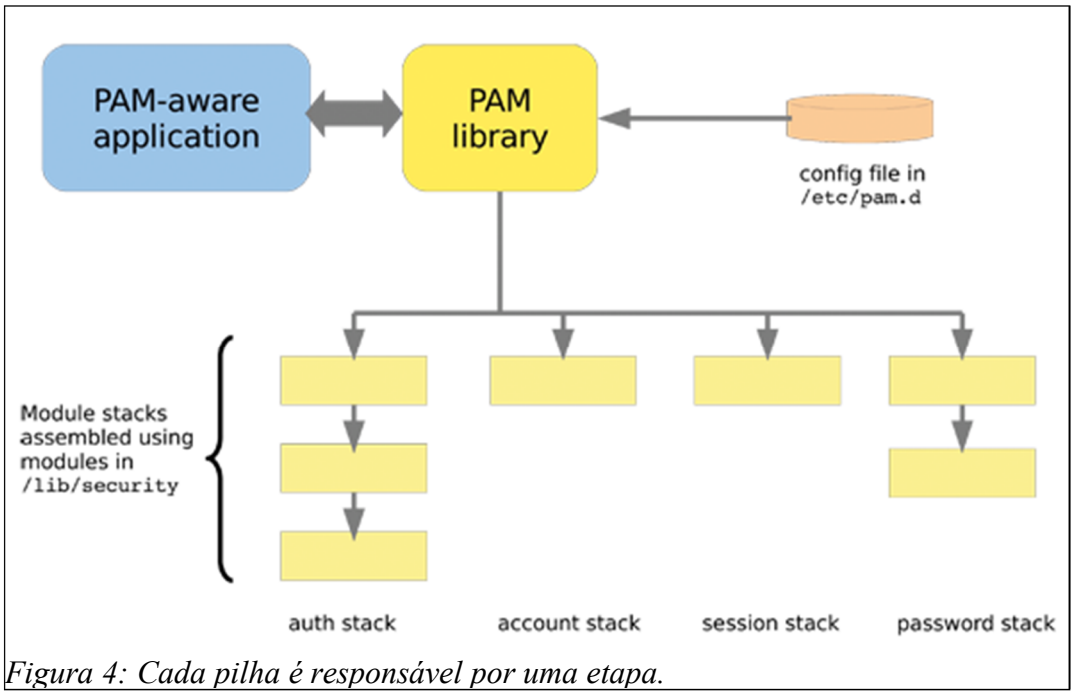

Figura 4: Cada pilha é responsável por uma etapa.

Outro campo importante e presente nos arquivos de configuração é o *control-flag,* este especifica a ação a ser tomada dependendo do resultado do módulo PAM. Mais de um módulo PAM pode ser especificado para uma dada aplicação (isto é chamado de empilhamento stacking). Agora, a ideia básica é que quando uma aplicação compatível com PAM quer executar uma atividade de autenticação, ela pede a biblioteca PAM para fazer o trabalho. A biblioteca PAM em seguida, chama cada um dos módulos na pilha de atividade em ordem.

Cada um desses módulos faz a sua parte, e retorna uma resposta de sucesso ou falha para a biblioteca. A biblioteca PAM combina estas respostas de sucesso/falha em um resultado para a pilha como um todo. Este resultado é então retornado para o aplicativo. A maneira pela qual os resultados de sucesso/falha dos módulos individuais são combinados é determinado pelo sinalizador de controle associado com o módulo, ou *control flag.* O *control-flag* é o segundo campo das entradas no arquivo de configuração PAM, também determina a relativa importância dos módulos na pilha. Os quatro valores possíveis para este campo são: *required, requisite, optional* e *sufficient*.

Quadro 4 – Sinalizadores de controle

Control Flag	**Descrição**
required	Este módulo deve retornar sucesso para o serviço ser garantido. Se este módulo é um em uma série de uma pilha de módulos, todos os outros módulos ainda são executados. A aplicação não será informada sobre qual ou quais módulos falharam.
requisite	Tal como required, exceto que a falha aqui, termina a execução para todos os módulos e retorna o status da falha para a aplicação.
optional	Como o nome sugere este módulo não é obrigatório. Se ele for o único módulo, entretanto, este retorna um status para a aplicação podendo causar uma falha.
sufficient	Se este módulo obtiver sucesso, todos os módulos que ficaram na pilha são ignorados e o sucesso é retornado para a aplicação.

PAM é um poderoso *framework*, e pode ser difícil enxergar todas as possibilidades que podem estar erradas. Se PAM for configurado erroneamente, o ambiente pode ser facilmente comprometido por um cracker ou mesmo script kiddies. Ou mesmo deixar o sistema em um estado onde ninguém conseguirá logar e corrigir o erro, principalmente em sessões remotas. Mas o pior cenário é quando uma pessoa não autorizada consegue logar no sistema. Portanto mudar a configuração do PAM é um negócio sério.

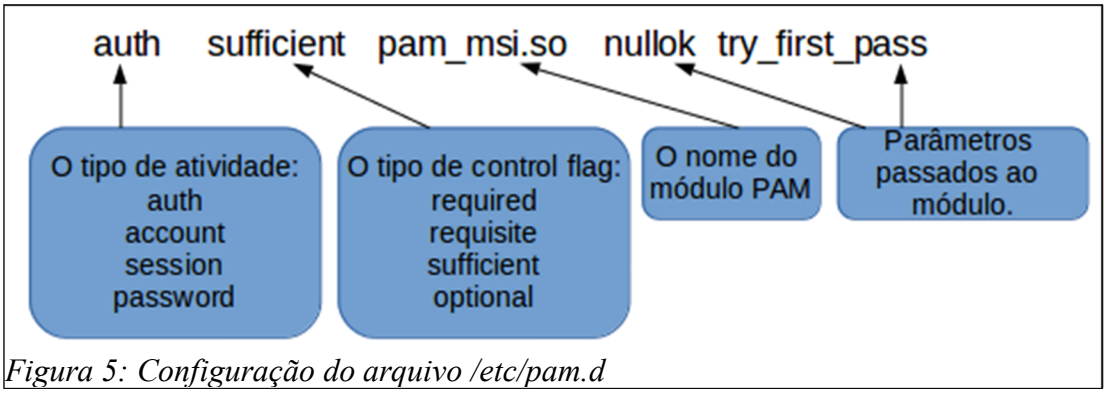

Figura 5: Configuração do arquivo /etc/pam.d

O módulo pam_deny deve ser considerado como um componente essencial em uma configuração PAM moderna. O módulo pode ser incluído como o último módulo em qualquer pilha para qualquer serviço como uma solução à prova de falhas no arquivo de configuração. Assim como uma diretiva *deny all* em um firewall, se nenhum outro módulo aprovou ou negou o acesso ao serviço é interessante saber que será sempre negado na última fase.

Os módulos PAM são objetos compartilhados (*shared objects*), ou seja, arquivos ".so". Um objeto compartilhado pode ser carregado sob demanda, assim o subsistema PAM não exige uma completa recompilação se um módulo é adicionado, removido, ou modificado.

4.3.1 Compilando o módulo

A linguagem de programação dominante do UNIX é o C, e é muito mais simples desenvolver novos módulos ou aplicações PAM-AWARE em C do que qualquer outra linguagem. Pode soar como algo muito complicado desenvolver um módulo próprio PAM, mas a grande maioria dos módulos é pequena, variando entre 100 até 1000 linhas de código em linguagem C.

Para construir um módulo PAM é necessário inserir em seu código fonte a seguinte biblioteca: #include <security/pam_modules.h> e compilar da seguinte forma:

```
$gcc -fPIC -DPIC -shared -rdynamic -o pam_msi.so pam_msi.c
```

Assim que for compilado basta copiar o arquivo resultante "pam_msi.so" para o diretório de objetos compartilhados tipicamente localizados em /lib/security (GNU/LINUX), ou /usr/lib (FreeBSD) e ter o prefixo pam como sufixo do nome do arquivo.

```
$sudo cp pam_msi.so /lib/security/
```

Digamos que o administrador deseja proteger o servidor samba com o módulo personalizado deste trabalho, pam_msi.so, a configuração do arquivo /etc/pam.d/samba ficaria:

```
@include common-auth

@include common-account

@include common-session-noninteractive

auth    requisite       /lib/security/pam_msi.so
```

Antes de colocar em produção ele pode realizar um teste com o utilitário pamtester no serviço. O pamtester é muito simples e funciona com basicamente três parâmetros: serviço, usuário e módulo de gerência (authenticate, session, account, password).

```
$pamtester -v samba brunof authenticate
```

Como devemos lidar com a parte de autenticação todo o código deverá estar contido na função pam_sm_authenticate, que cumpre a tarefa de autenticar um usuário.

```
int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc,
const char **argv) {

/*Código do módulo se resume dentro dessa função*/...

}
```

A função principal para este trabalho é a pam_get_item. Nela podemos guardar em ponteiros itens como usuário e senha digitados, informação fundamental para validar se a senha digitada se encontra dentro do padrão preestabelecido.

```
extern int pam_get_item(const pam_handle_t *pamh,

int item_type,

const void **item);
```

Esta função é usada para se obter um valor específico de um item_type. Normalmente se o módulo quer obter o nome do usuário não se usa esta função, mas sim uma chamada a função *pam_get_user*.

Em termos simplificados, basicamente o código resumido do módulo ficará:

```
#define PAM_SM_AUTH

#include <security/pam_modules.h>

int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc,
const char **argv) {

char *pass;

pam_get_item(pamh, PAM_AUTHTOK, (const void **)(const void*)&pass);

if (fnverifica(*ponteirosenha) != SUCESSO) /*Aqui ficará a função que
verifica se a senha está dentro do padrão*/

{

return (PAM_AUTH_ERR);

}

return(PAM_SUCCESS);

}
```

Não é o escopo deste trabalho ensinar passo a passo a construção de um módulo PAM, porém as partes mais relevantes do código foram abordadas para que fins de aprendizado. Não obstante, o autor disponibiliza todos os códigos e arquivos na íntegra no CD entregue junto com o trabalho impresso, e que ficará disponível na biblioteca desta Universidade. Uma versão online e atualizada também estará disponível em um repositório público GitHub no endereço eletrônico https://github.com/bruno-sf/ufrj-msi.

CONCLUSÃO

A partir do que foi descrito neste trabalho, observa-se a miríade de possibilidades que um módulo PAM pode oferecer ao administrador de sistemas *opensource*. Como foi visto existe diversos tipos de ataques contra senhas, e é apenas uma questão de tempo para que um atacante consiga quebrá-la. A ideia é que esse tempo seja o suficiente para desencorajar o atacante.

Apesar de o trabalho ter sido desenvolvido em ambiente Linux, nada impede que a solução seja usada ou portada para outros sistemas que usam o padrão PAM. Não obstante, a flexibilidade da biblioteca PAM é uma das suas maiores vantagens, além de desonerar o programador com as especificidades de autenticação de cada sistema. Sendo assim, este trabalho apenas apresenta uma forma alternativa de autenticação, se aproveitando das capacidades da biblioteca PAM para concretizar a ideia na prática.

A utilização do código aqui exposto e produzido neste trabalho pode ser usado e compartilhado por profissionais que desejam enriquecer sua política de segurança. Porém, é muito importante ressaltar que a solução aqui proposta não deve ser usada como método de autenticação principal, pois a mesma não foi largamente testada e pode apresentar *bugs* e vulnerabilidades. Além disso, a ideia foi apresentada, mas serve como base para outros modelos de autenticação que possam surgir, incentivando outros estudantes e profissionais da área de segurança da informação a desenvolverem seus próprios métodos.

Assim, o código da biblioteca criada neste trabalho se encontra disponível no CD, junto com um pequeno guia para configurar e instalar em sistemas Linux, também ficará disponível *online* para a comunidade em repositório público *GitHub* mencionado na seção anterior.

Contudo, deve ser reiterado que mesmo usando um bom padrão e construindo senhas computacionalmente fortes, não se pode esperar que apenas uma solução proteja os sistemas de acesso indevido. A segurança da informação é construída em camadas. Além da constante

busca por informação e capacitação do profissional de segurança, pois sem boas informações é impossível tomar boas decisões.

O presente trabalho não esgota todas as reflexões de estudos na área, ainda mais quando o objeto de pesquisa está em constante atualização, como acontece com os sistemas de autenticação. A todo o momento surgem novos recursos e formas de autenticação segura que podem ser aprimorados e aproveitados pelos profissionais de segurança. O importante é ocorrer o envolvimento dos profissionais com essas alternativas, superando a resistência e aprendendo a criar seus próprios mecanismos de defesa.

REFERÊNCIAS BIBLIOGRÁFICAS

MORGAN, Andrew G.; KUKUK, Thorsten. **The Linux-PAM Module Writer's Guide**. Version 1.1.2, 31., California, 2010.

SMITH, Roderick W. **LPIC-2 Linux Professional Institute Certification Study Guide: Exams 201 and 202**., John Wiley & Sons, mai.2011.

FLORÊNCIO, Dinei; HERLEY, Cormac. **A Large-Scale Study of Web Password Habits Microsoft Research.** In: WWW 2007 / TRACK: SECURITY, PRIVACY, REALIABILITY, AND ETHICS, 657, 2007, Canada, BANFF, 2007, p. 657-665.

SAMAR, V.; SCHEMERS, R.; **Open Software Foundation Request for Comments:86** , Outubro 1995.

GIBSON, Steve. **How Big is your Haystack?**, California, 2012. Disponível em: https://www.grc.com/haystack.htm . Acesso em 19 dez. 2014.

TANENBAUM, Andrew S. **Redes de Computadores**, Tradução da Quarta edição, Rio de Janeiro, Elsevier, 2003.

TORRES, Gabriel. **Redes de computadores**, Versão Revisada e Atualizada, Rio de Janeiro, Nova Terra, 2009.

DE PÁDUA, Elisabete Matallo Marchesini. **Metodologia Da Pesquisa**, Coleção Magistério: Formação e Trabalho Pedagógico, São Paulo, 2002

HUNT, Troy. **A brief Sony password analysis**, Sydney, 2011. Disponível em: http://www.troyhunt.com/2011/06/brief-sony-password-analysis.html. Acesso em: 19 de dez. 2014.

SEIFRIED, Kurt. **Linux Administrator's Security Guide**, Alberta, 2001. Disponível em: https://www.seifried.org/lasg/. Acesso em 18 de dez. 2014.

GLOSSÁRIO

API Interface de Programação de Aplicações.

BLOWFISH Criptografia simétrica desenvolvida em 1993 e usada até hoje.

DIGEST Usado para confirmar identidade antes de enviar dados sensíveis.

HARDWARE Dispositivos físicos

HASH Transformação de uma grande quantidade de dados em uma pequena quantidade de informações

JAVA Linguagem de programação de alto nível e orientada a objetos.

LINUX Kernel livre utilizado em sistemas GNU/Linux.

OPENSOURCE Código aberto, diferente de software livre.

PAM Bibliotecas de autenticação

PAMTESTER Ferramenta para testes de bibliotecas PAM.

PLUGINS Módulo de extensão também conhecido como add-in ou add-on.

REDHAT LINUX Sistema Linux largamente utilizado em grandes empresas.

SALT Dados aleatórios inseridos em funções hash.

SHADOW Usado para aumentar o nível de segurança das senhas armazenadas.

SINGLE SIGN-ON Propriedade que permite o usuário logar uma vez e ter acesso ao restante dos sistemas sem ter que se re-autenticar novamente.

SOFTWARE LIVRE Permite-se adaptações ou modificações em seu código de forma espontânea, ou seja, sem que haja a necessidade de solicitar ao seu proprietário.