Tecnologias de Segurança

TP3 - Modelo Formal de Controlo de Acesso

Diogo Marques	**Pedro Sousa**	**Rui Lopes**
PG55931	PG55994	PG56009

Índice

Introdução

Este relatório tem por objetivo apresentar o modelo de segurança desenvolvido no âmbito da unidade curricular de Tecnologias de Segurança, sendo o mesmo fortemente baseado no *Bell–LaPadula* para garantir propriedades de confidencialidade, bem como outros modelos semelhantes para dotar o sistema dum maior dinamismo e consistência.

Enquanto implementação do modelo em questão, o grupo optou por desenvolver uma *Web API* que disponibiliza todas as funcionalidades comuns a um sistema de ficheiros, sendo permitido criar compartimentos aos quais os utilizadores estão associados e daí escrever/ler ficheiros.

Por fim, numa tentativa de tornar a experiência de utilização mais rica, elaborámos uma interface minimalista que facilita a autenticação de utilizadores e a partir daí concede uma visão mais coesa de todo o sistema, mais concretamente os compartimentos e ficheiros acessíveis ao utilizador em questão.

Modelo de Controlo de Acesso

Dependendo do tipo de ambiente onde o sistema será aplicado, o modelo formal sustenta regras de controlo de acesso para um fim específico, que normalmente corresponde à obtenção de garantias como confidencialidade e segurança.

Embora o enunciado sugira somente o *Bell-LaPadula* como fonte de inspiração, julgamos que tal não é suficiente à construção de um modelo genérico e aplicável em diversos ambientes. Nesse sentido, surgiu a necessidade de combinar vários modelos, respeitando, sempre, as propriedades de cada um.

Obviamente, o modelo de controlo de acesso perfeito não existe. No fundo, tudo se trata de *trade-offs* entre confidencialidade, integridade e disponibilidade.

Bell–LaPadula

Este modelo está especialmente voltado para garantir a confidencialidade do sistema, para isso o espaço de endereçamento é dividido em níveis de segurança hierárquicos sobre os quais os utilizadores operam e os ficheiros estão posicionados. Assim, as operações de escrita/leitura são efetuadas com base na compatibilidade entre níveis.

Propriedade Simples

Um utilizador não pode ler ficheiros cujo nível de confidencialidade seja superior ao seu. Deste modo, fugas de informação tornam-se impossíveis a partir de utilizadores com baixa patente de confidencialidade.

Propriedade Estrela

Um utilizador não pode escrever em níveis cuja confidencialidade seja inferior à sua. Por conseguinte, alguém *top-secret* é incapacitado de escrever informação sensível onde não deve.

Isto é especialmente relevante para evitar fugas por descuido, visto que, à partida, um sujeito *top--secret* é de confiança.

Funcionamento

Para melhor compreender o funcionamento deste modelo, apresenta-se um breve exemplo no qual recorremos aos dados apresentados nas seguintes tabelas.

O utilizador *Rui* é classificado como *secret*, como tal, é capaz de ler os ficheiros *main.py*, *text.txt* e *object.jar*, visto terem associados uma confidencialidade mais relaxada.

Utilizador	Confidencialidade
Diogo	Unclassified
Pedro	Classified
Rui	Secret
Tiago	Top-Secret

Tabela 1: Confidencialidade dos utilizadores

Ficheiro	Confidencialidade
main.py	Unclassified
text.txt	Classified
object.jar	Secret
file.c	Top-Secret

Tabela 2: Confidencialidade dos ficheiros

Por outro lado, o *Pedro* é *classified*, daí que não tenha escrito o ficheiro *main.py*. Posto isso, conseguimos concluir que esse ficheiro foi obrigatoriamente escrito por *Diogo*, dado ser o único utilizador com nível inferior ou igual a *unclassified*.

Embora a confidencialidade seja plenamente salvaguardada, o mesmo não acontece com a integridade, pois um utilizador de baixa patente (*Diogo*) tem capacidade para danificar ficheiros com elevado secretismo (*file.c*).

Biba

Este modelo formal é basicamente o dual do *Bell–LaPadula*, daí que a propriedade de integridade seja preservada em detrimento da confidencialidade, assim sendo o espaço de endereçamento passa a ser dividido em níveis de integridade hierárquicos onde as escritas/leituras invertem de sentido.

Propriedade Simples

Um utilizador não pode ler ficheiros cujo nível de integridade é inferior ao seu. Deste modo, será impossível induzir em erro qualquer sujeito, visto que esse sujeito apenas confia noutros que são tão ou mais íntegros que ele próprio.

Propriedade Estrela

Um utilizador não pode escrever em níveis cuja integridade é superior à sua. Quer isto dizer que alguém de baixa patente não é capaz de influenciar as decisões tomadas por alguém mais integro.

Numa breve analogia, o Presidente da República não confia em mim como informador (não sou da confiança dele), no entanto eu acredito nas declarações dele (o presidente tem a confiança dos cidadãos).

Funcionamento

Repetindo o exemplo anteriormente apresentado, o *Diogo* pode ler os ficheiros *main.py*, *text.txt* e *object.jar*, visto terem sido escritos por sujeitos de patente superior ou igual à sua. No entanto, o *Rui* só tem acesso de leitura a *objet.jar*, e sendo ambos *strong*, foi obrigatoriamente o *Rui* que escreveu esse ficheiro. No fundo, o *Diogo* confia em todos os utilizadores, e o *Rui* confia somente nele próprio.

Utilizador	Integridade
Diogo	Weak
Pedro	Medium
Rui	Strong

Tabela 3: Integridade dos utilizadores

Ficheiro	Integridade
main.py	Weak
text.txt	Medium
object.jar	Strong

Tabela 4: Integridade dos ficheiros

Muralha da China

Uma combinação dos modelos anteriores oferece propriedades de confidencialidade e integridade. No entanto, ambiente reais, nomeadamente o empresarial, requerem outros tipos de garantias. Se uma pessoa trabalha e tem acesso aos dados da empresa *X*, então jamais poderá conhecer informações associadas à firma concorrente *Y*.

Posto isto, e no sentido de enriquecer o modelo final, as classes de conflito traduzem a concorrência entre compartimentos. Mesmo que um utilizar possua níveis de integridade e confidencialidade compatíveis com o ficheiro, o acesso será negado caso exista um conflito entre compartimentos.

Funcionamento

Embora um utilizador possa estar associado a vários compartimentos, os ficheiros estão localizados apenas num. Posto isto, a verificação de conflitos é bastante facilitada, porque basta verificar a condição $\text{compartment.of.file} \subseteq \text{compartments.of.user}$.

Utilizador	Compartimento
Diogo	Braga
Diogo	Porto
Rui	Lisboa

Tabela 5: Compartimentos dos utilizadores

Ficheiro	Compartimento
main.py	Braga
text.txt	Porto
object.jar	Lisboa

Tabela 6: Compartimentos dos ficheiros

Compartimento	Compartimento conflituoso
Braga	Lisboa
Porto	Lisboa

Tabela 7: Classes de conflito entre compartimentos

Neste exemplo os compartimentos *Braga* e *Porto* são conflituosos com *Lisboa*, sendo por isso definidas duas classes de conflito, perante tal evidencia nenhum utilizador que esteja associado a *Braga* ou *Porto* poderá ter acesso ao ficheiro *object.jar*, visto que este pertence a *Lisboa*.

Como as classes de conflito são recíprocas, esta lógica é válida ao contrário, daí que *Rui* não conheça os recursos *main.py* e *text.txt*, tendo a sua visão limitada ao compartimento *Lisboa* enquanto a ele estiver associado.

Modelo Final

Uma combinação entre o *Bell–LaPadula* e *Biba* resulta numa formalização trivial do *Lipner*, no entanto julgamos que a resolução de conflitos é relevante em cenários mais realistas, daí que a nossa proposta de soluções seja uma combinação entre os três modelos previamente descritos.

Ao extrairmos os pontos fortes de cada modelo, conseguimos obter uma solução que assegura as propriedades de confidencialidade, integridade e respeito pelo concorrência.

Propriedades

Num cenário baseado em um modelo, ao verificarmos se um utilizador pode escrever/ler determinado recurso, costumamos verificar apenas uma condição, seja ela o nível de integridade ou confidencia-

lidade. No entanto, a nossa proposta de solução requer três verificações por cada acesso, o que obviamente penaliza a performance.

- **Confidencialidade**
  1. Não ler ficheiros cujo nível de confidencialidade é superior ao meu;
  2. Não escrever em níveis cuja confidencialidade é inferior à minha.
- **Integridade**
  1. Não ler ficheiros cujo nível de integridade é inferior ao meu;
  2. Não escrever em níveis cuja integridade é superior à minha.
- **Resolução de Conflitos**
  1. Não escrever/ler ficheiros cujo compartimento é conflituoso com algum dos meus.

No fundo, cada conjunto de regras pode ser traduzido numa *lattice* parcialmente ordenada que descreve de forma mais sucinta e concreta o comportamento do modelo face a leituras e escritas.

- $l \in L$: nível de confidencialidade
- $i \in I$: nível de integridade
- $d \subseteq D$: conjunto dos compartimentos

Tendo isto em mente, os atributos $p_A = (l_A, i_A, d_A)$ pertencentes ao utilizador *A* podem ser comparados com os do ficheiro *B*, $p_B = (l_B, i_B, d_B)$, a fim de autorizar o acesso solicitado pelas operações.

$$p_A \leqslant p_B \iff l_A \leqslant l_B \land i_A \geqslant i_B \land d_B \subseteq d_A$$

Com base nesta definição de ordenação, operações de leitura e escrita requerem a seguinte verificação:

$$A \text{ pode ler } B \iff p_A \geqslant p_B$$

$$A \text{ pode escrever sobre } B \iff p_A \leqslant p_B$$

Tal como nos dois primeiros modelos, a operação de leitura é o dual da escrita, e tendo em conta que o nosso modelo resulta, em parte, duma combinações entre esses dois, faz sentido que tal propriedade continue a verificar-se.

Funcionamento

Para melhor compreender o funcionamento do modelo, convém apresentar um caso prático onde todas as propriedades são aplicadas.

Neste exemplo, o *Diogo* tem acesso aos compartimentos *Braga* e *Porto*, tendo diferentes atributos associados em cada um. Nesse sentido, é-lhe permitida a leitura de *object.jar*, dado que a sua confidencialidade e integridade são respetivamente superior e inferior à do ficheiros.

Por outro lado, o *Diogo* não é capaz de ler *text.txt*, porque o seu nível de confidencialidade não lhe confere acesso. No entanto, a escrita em *main.py* e *text.txt* é viabilizada pelo parâmetro de integridade.

Utilizador	Confidencialidade	Integridade	Compartimento
Diogo	Secret	Strong	Braga
Diogo	Classified	Weak	Porto
Pedro	Top-Secret	Medium	Braga
Rui	Classified	Weak	Lisboa

Tabela 8: Atributos associados aos utilizadores

Ficheiro	Confidencialidade	Integridade	Compartimento
main.py	Top-Secret	Weak	Braga
text.txt	Secret	Weak	Porto
object.jar	Unclassified	Medium	Porto
file.c	Classified	Strong	Lisboa

Tabela 9: Atributos associados aos ficheiros

Compartimento	Compartimento conflituoso
Braga	Lisboa
Porto	Lisboa

Tabela 10: Classes de conflito entre compartimentos

Por fim, o facto de *Braga* e *Porto* serem conflituosos com *Lisboa* impede que o *Rui* entre nesses compartimentos, ficando limitado à leitura de *file.c*. Em suma, a formulação matemática das permissões permite calcular computacionalmente os acessos concedidos a cada utilizador.

Utilizador	Acesso de Leitura	Acesso de Escrita
Diogo	object.jar	main.py, text.txt
Pedro	object.jar	Sem ficheiros
Rui	file.c	Sem ficheiros

Tabela 11: Acessos concedidos aos utilizadores

Melhorias ao Modelo

Apesar do modelo garantir diversas propriedades e ter um âmbito de aplicação diversificado, de pouco serve em cenários reais se não puder ser atualizado em tempo real. Assim sendo, a implementação desenvolvida permite modificar os atributos de confidencialidade e integridade associados a ficheiros, sendo essa operação de gestão dirigida exclusivamente por sujeitos confiáveis.

Princípio da Tranquilidade

Uma vez que os atributos associados aos recursos são dinâmicos, basta garantir que a modificação dos mesmos não ocorre durante acessos de leitura/escrita de outros utilizadores, daí que o modelo assuma o **Princípio da Tranquilidade Fraca**.

Posto isto, os recursos do sistema são dotados de um ciclo de vida que corresponde às transições dos seus atributos, ou seja, o ficheiro *A* permanece *top-secret* durante alguns anos e depois é realizada uma expurgação que o torna *secret*, algo bastante comum em documentos de estado.

2020-10-02 09:28:07 $\longrightarrow$ *top-secret*, *strong*

2022-05-22 08:07:40 $\longrightarrow$ *classified*, *strong*

2026-10-25 22:48:30 $\longrightarrow$ *unclassified*, *weak*

Neste exemplo de transição de estados, o ciclo de vida do ficheiro pode ser dividido em duas categorias. Uma respeitante à confidencialidade e outra à integridade. Deste modo, a gestão de atributos torna-se mais dinâmica e diminui ao máximo as dependências.

Sujeitos Confiáveis

Tal como referido anteriormente, somente utilizadores confiáveis são capazes de alterar os atributos de recursos, no entanto surge a dúvida de decidir quem é confiável ou não. Perante tal problema, o

grupo optou por atribuir essa responsabilidade a uma entidade superior, neste caso um administrador de sistema.

Para além de identificar entidades confiáveis, o administrador também cria os compartimentos e procede à associação dos mesmos com os utilizadores. De realçar que um sujeito ser confiável num compartimento não implica que também o seja nos restantes.

Registo Dinâmico de Atributos

De modo a proceder à atribuição dos níveis de confidencialidade e integridade, primeiro é preciso conhecer a gama de valores disponível e a relação de ordem entre eles, ou seja, apesar de *top-secret* ser evidentemente superior a *secret*, isso deve estar escrito duma forma mais genérica.

Confidencialidade	Nível
Unclassified	1
Classified	2
Secret	3
Top-Secret	4

Tabela 12: Níveis de confidencialidade

Integridade	Nível
Weak	1
Medium	2
Strong	3

Tabela 13: Níveis de integridade

Quando um administrador regista um nível do confidencialidade, o mesmo atribui um nome que serve de identificador e um inteiro que o permite comparar com os demais parâmetros. Além disso, para inserir um nível entre *classified* e *secret* basta incrementar em uma unidade *secret* e *top-secret*, a fim de deixar livre a terceira posição.

Numa outra perspetiva, o registo de novos níveis não compromete a segurança do sistema, visto que a ordenação previamente vigente continua a ser respeitada, apenas foi adicionado algo novo que torna o sistema mais rico e dinâmico em termos de classificação dos atributos associados a recursos.

Regras de Gestão de Ficheiros

Ao serem disponibilizadas várias combinações entre atributos, torna-se complicado identificar a mais adequada a cada tipo de ficheiro. Dado isso, o grupo decidiu estabelecer um conjunto de regras sobre as ações que afetam diretamente o ciclo de vida.

Criação de Ficheiro

Quando um utilizador cria um ficheiro, por padrão, este herda a confidencialidade e integridade atribuídos ao utilizador no compartimento em questão. Precisamente por isso nenhuma das regras anteriormente definidas é violada, pois o utilizador tem permissão de escrita e leitura sobre recursos do mesmo nível.

Eliminação de Ficheiro

A operação de eliminação é um caso particular da escrita. No entanto, é algo mais sensível, visto negar futuros acessos ao recurso. Assim, apenas sujeitos confiáveis são capazes de executar tal ação, respeitando a condição de terem confidencialidade e integridade superior ou igual à do próprio ficheiro.

$$p_A \trianglerighteq p_b \iff l_A \geqslant l_B \land i_A \geqslant i_B \land d_B \subseteq d_A$$

Esta condição é significativamente diferente das apresentadas anteriormente, dado que os atributos não se aplicam da mesma forma, ou seja, apesar de alguém *top-secret* não conseguir escrever em *secret*, a leitura é permitida e portanto existe autoridade para remover o ficheiro.

Numa outra análise, a integridade *weak* não pode influenciar sujeitos *strong*, mas o contrário acontece, daí que a eliminação deva preservar essa relação. No fundo, a condição acima apresentada assegura que o utilizador *A* é tão ou mais confiável/íntegro que o ficheiro *B* em todos os aspetos.

Reclassificação de Atributos

Para cumprir efetivamente com o princípio de tranquilidade fraca, os sujeitos confiáveis são capazes que reclassificar os atributos dos ficheiros, para isso seguem um conjunto de regras e visam salvaguardar as propriedades do modelo.

$$l_A \geqslant l_{\text{B old}} \wedge l_A \geqslant l_{\text{B new}}$$

$$i_A \geqslant i_{\text{B old}} \wedge i_A \geqslant i_{\text{B new}}$$

Para que o utilizador *A* reclassifique a integridade de *B*, este deve possuir um nível superior ou igual. Além disso, a nova classificação do ficheiro não pode exceder o atributo do utilizador, caso contrário alguém pouco íntegro poderia induzir em erro os restantes participantes, violando assim a **Propriedade Estrela** do modelo *Biba*.

Seguindo a mesma lógica, a desclassificação de ficheiros ocorre quando o próprio utilizador já tem conhecimento acerca dos mesmos, portanto alguém de baixa patente estará impossibilitado de despromover ficheiros aos quais não tem acesso de leitura.

Este processo de reclassificação para níveis inferiores costuma ser acompanhado da expurgação de informação, no entanto, devido à falta de tempo, a implementação desenvolvida não disponibiliza essa funcionalidade, permitindo somente alterar a classificação sem modificar o conteúdo.

Reclassificação Temporária

Ainda como medida adicional, o grupo pensou em implementar um mecanismo de reclassificação temporária para dotar o sistema duma maior flexibilidade e limitar os impactos provocados por uma desclassificação indevida.

Para tal, bastaria aos utilizadores de confiança indicar a nova classificação e o intervalo de tempo sobre o qual as alterações seriam efetivas, após esse limite o ciclo de vida seria automaticamente modificado, fazendo com que o ficheiro voltasse às permissões anteriores.

Seja como for, e novamente por questões de tempo, não tivemos a oportunidade de implementar tal funcionalidade, algo que definitivamente enriqueceria o sistema em termo de utilização.

Servidor

Para a devida concretização do modelo formalizado, o grupo optou pela implementação de uma *Web API* como servidor de informação. Este servidor foi desenvolvido com recurso à linguagem *Elixir* e à *framework web Phoenix*.

Sistema de Dados

No entanto, antes da implementação propriamente dita, começamos por delinear o sistema de dados: tabelas, atributos e relacionamentos. Tal pode ser visto na Figura 1.

Pode-se dizer que a entidade central do sistema de dados será a tabela *UsersCompartments*, que define se existe uma relação entre um dado utilizador, ficheiro e compartimento, bem como o nível de integridade e confidencialidade associados.

Será contra esta tabela que serão verificadas a maioria das *queries* ao sistema. Excetuando as que se tratam de conflitos entre compartimentos e, portanto, serão verificadas através da tabela *CompartmentConflicts*. Essencialmente, estamos perante uma matriz de autorização não esparsa.

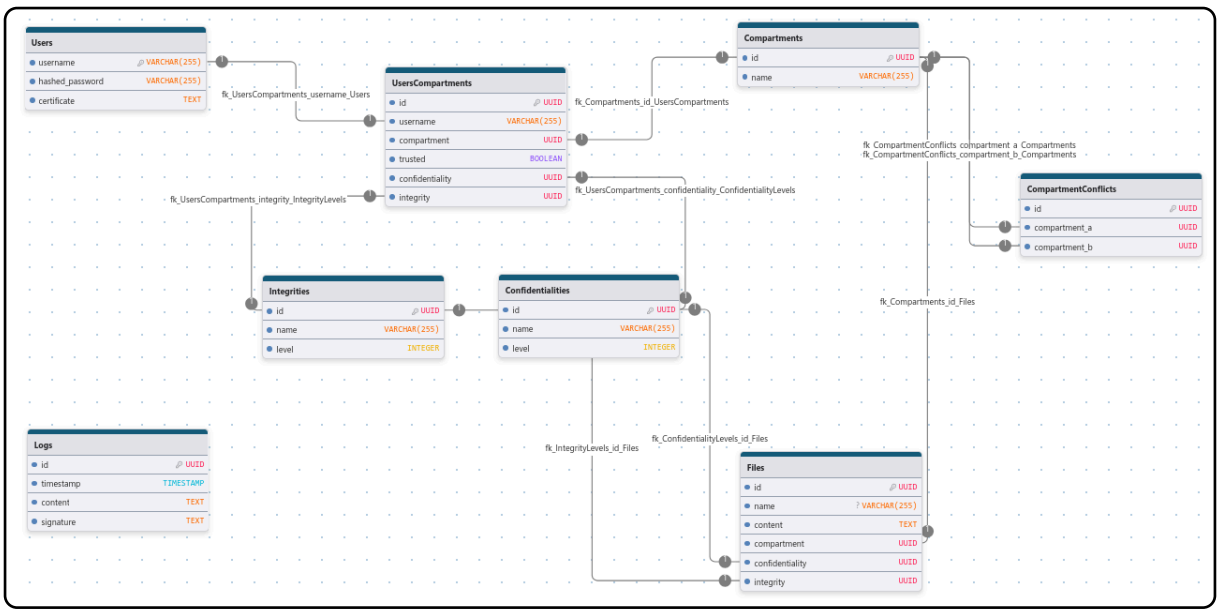

Figura 1: Modelo lógico do sistema de dados

Endpoints

Esta secção pretende listar, exaustivamente, os *endpoints HTTP* disponibilizados pelo servidor implementado.

Autenticação

- Registo: POST `/register`
- Login: POST `/login`
- Refrescar token: POST `/refresh`
- Logout: POST `/logout`

Utilizadores

- Ver o próprio utilizador: GET `/users/me`
- Pedir certificado de um dado utilizador: GET `/users/:username/certificate`

Ficheiros

- Listar os ficheiros a que um utilizador tem acesso: GET `/files`
- Adicionar ficheiro: POST `/files`
- Ler ficheiro: GET `/files/:id`
- Editar ficheiro (conteúdo): PUT `/files/:id`

Sujeitos Confiáveis

- Editar confidencialidade: PUT `/files/:id/confidentiality`
- Editar integridade: PUT `/files/:id/integrity`
- Eliminar ficheiro: DELETE `/files/:id`

Compartimentos

- Listar compartimentos a que temos acesso: GET `/compartments`

Administradores

- Criar compartimento: POST `/compartments`
- Criar conflito entre compartimentos: POST `/compartments/conflict`
- Adicionar utilizador a compartimento: PUT `/compartments/:id/:username`
- Remover utilizador de compartimento: DELETE `/compartments/:id/:username`
- Listar todos os níveis: GET `/levels`
- Criar nível: POST `/levels`

Autorização

O servidor desenvolvido (em forma de *API*) utiliza um sistema de autorização baseado em *tokens JWT*, criadas e assinadas pelo próprio servidor. Estas *tokens* existem, sempre, em pares: *access* e *refresh*.

A *token* de *access* possui um tempo de vida de apenas 15 minutos, enquanto que a *token* de *refresh* possui um tempo de vida muito maior, de 30 dias. Este sistema híbrido permite que apenas a *token* com menor tempo de vida seja frequentemente exposta, sendo que a *token* de *refresh* apenas é exposta para refrescar a *token* de *access* e é, imediatamente, revogada a seguir, dando lugar a um par completamente novo.

Autenticação (2FA)

Com vista em aumentar a segurança do sistema de autenticação, para além do método tradicional baseado em *email* e *password*, foi implementada também a autenticação por duplo fator (*2FA Authentication*).

É sabido que este tipo de autenticação mais tradicional (baseada em *email* e *password*) possui diversas falhas de segurança inerentes à sua forma de funcionamento. Por exemplo, é comum que os utilizadores façam uso de *passwords* com pouca segurança ou, até, que utilizem a mesma *password* em diferentes serviços. Dada a existência de sujeitos confiáveis no nosso sistema torna-se, ainda mais, imperativa a implementação de uma autenticação baseada em *2FA*.

Quando um utilizador se regista, este recebe um *link* para usar numa aplicação *TOTP* (*Time Based One Time Password*), associando o segredo *TOTP* (acordado com o servidor) no seu dispositivo pessoal. Este segredo irá permitir, posteriormente, a geração de um código aleatório para a realização de *login*.

Assim sendo, o servidor, para além de verificar as duas primeiras credenciais, verifica se o código *TOTP* é válido, com base no segredo previamente acordado.

Logs

Numa perspetiva de tornar o sistema auditável e associar os utilizadores às suas ações (assegurar a propriedade de não-repúdio), o grupo definiu um protocolo de comunicação no qual os pedidos realizados pelo cliente são acompanhados duma assinatura de `method + path + body`, algo que vai presente no campo `X-Signature` do *header HTTP*.

Uma vez que o servidor possui o certificado do cliente, consequentemente também conhece a chave pública, nesse sentido a validade da assinatura pode ser confirmada, garantido portanto que o pedido foi efetivamente enviado pelo utilizador correto.

Por fim, caso a assinatura seja verificada com sucesso, o servidor devolve o conteúdo requisitado pelo cliente, sendo o *log* e respetiva assinatura armazenados numa tabela da base de dados. Caso contrário, é devolvido um código de erro `401` que reencaminha o cliente de imediato para a página de *login*, ou, então, `400` quando o cliente nem sequer preenche o campo `X-Signature`.

De realçar que, em qualquer um dos casos, o *log* de acesso ficou armazenado. E, portanto, tentativas de *Denial of Service* ficam, também, registadas.

Cliente

Como o objetivo de fornecer uma experiência de utilização agradável, o grupo optou por desenvolver uma *GUI* baseada na ferramenta *Textual*, visto esta permitir o desenvolvimento rápido de interfaces gráficas com o mínimo de esforço. Além disso, acresce a possibilidade de executar o programa tanto na *terminal* como num *browser*.

GUI

A fim de concretizar a entrada e registo de utilizadores no sistema, foram elaborados dois menus para as respetivas funcionalidades. Na página de *login* basta ao utilizador indicar o *email* e *password*, enquanto na página de registo a lógica é semelhante, mas sendo necessário indicar o certificado.

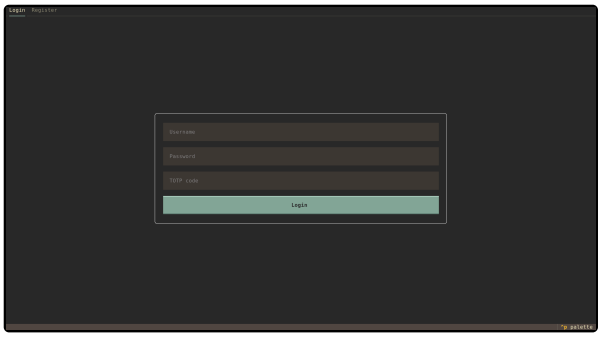

Figura 2: Página de *login*

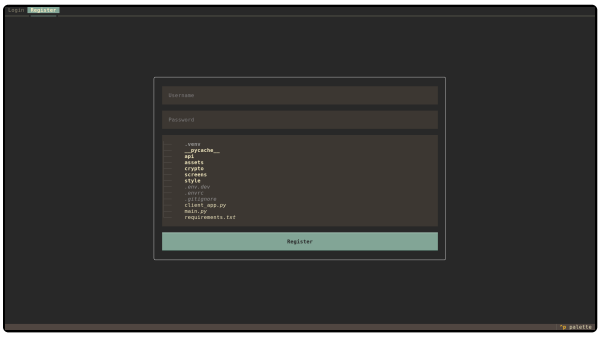

Figura 3: Página de registo

Em seguida apresenta-se a página para visualização dos ficheiros contidos nos compartimentos associados ao utilizador, bem como um *modal* para leitura e escrita dos próprio ficheiros. Além disso reparamos que na primeira página nem todos os botões são selecionáveis, algo que é definido pelas políticas de controlo de acesso apresentadas e discutidas anteriormente.

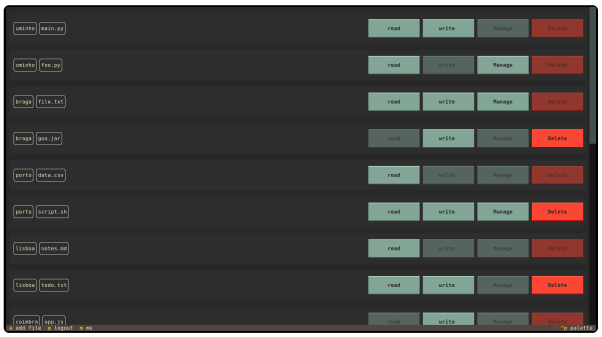

Figura 4: Página de visualização de ficheiros

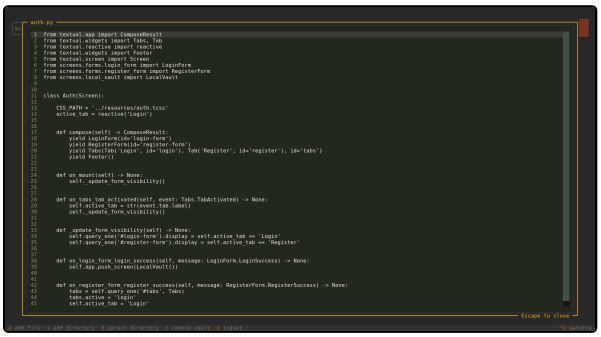

Figura 5: Página de edição de ficheiro

Por fim, desenvolvemos uma página para suportar a adição de ficheiros, sendo que esta permite modificar à partida os atributos associados ao recurso, a fim de não herdar automaticamente as propriedades que o utilizador possui sobre o compartimento em questão.

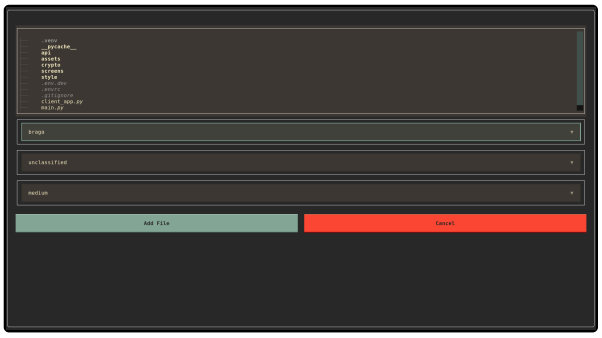

Figura 6: Página de criação de ficheiro

Figura 7: Página de gestão de atributos

Por outro lado, ao clicar no botão *Manage*, o utilizador tem a possibilidade de reclassificar os ficheiros, algo permitido por um conjunto de *selectors* que identificam os níveis para os quais é permitido mover os atributos do recurso.

Conclusão

Ao longo deste trabalho, o grupo desenvolveu e implementou um modelo formal de controlo de acesso que combina as propriedades do *Bell–LaPadula*, *Biba* e Muralha da China, visando assegurar simultaneamente confidencialidade, integridade e gestão de conflitos de interesse entre compartimentos.

Ao integrar os três modelos, definiu-se uma estrutura matemática clara e rigorosa para a verificação de permissões de leitura/escrita, tendo por base os atributos dos utilizadores e dos ficheiros. Adicionalmente, foram introduzidas funcionalidades práticas como a reclassificação de atributos, gestão dinâmica de níveis e distinção entre utilizadores confiáveis e não confiáveis, elementos esses que reforçam a flexibilidade e segurança do sistema.

Do ponto de vista técnico, foi implementada uma *API* que respeita as políticas de segurança delineadas, complementada por uma interface gráfica intuitiva que facilita a interação do utilizador com o sistema. Além disso, a utilização de *logs* com assinatura digital garante o princípio de não-repúdio e permite auditar todas as ações realizadas.

Apesar das limitações de tempo que impediram a implementação de certas funcionalidades, o trabalho realizado oferece uma base sólida, extensível e aplicável a ambientes com necessidades diversas de controlo de acesso.