**SLIDE 1 - Francisco**

Boa tarde, o tema do nosso trabalho é sobre autenticação de dois factores, algo que o utilizador conhece, uma password, e algo que o utilizador tem, o smartphone neste caso, e também sobre proximidade.

**SLIDE 2**

Aqui podem ver a arquitectura do nosso sistema, temos a aplicação cliente que é feita em HTML e JavaScript. Esta aplicação interage com o servidor através de um canal seguro feito por nós. E depois para a autenticação de dois factores utilizamos o Duo que requer um smartphone. Além disto ainda garantimos proximidade entre o cliente e o smartphone como veremos mais à frente. A comunicação entre o smartphone e cliente é feita através de WPA2.

**SLIDE 3**

Aqui temos a nossa arquitectura de sistema com máquinas virtuais. Não implementámos uma aplicação para Android e temos uma VM que simula a funcionalidade do smartphone. O cliente e smartphone estão na mesma rede e o servidor noutra rede. Não usamos WPA2 com VMs na prática mas a teoria é usar.

**SLIDE 4**

Quanto à segurança da password, XSS e SQLI...

**SLIDE 5**

Não é o foco do nosso projecto, mas ainda assim usamos a função `password\_hash` do PHP que usa bcrypt e é considerada bastante robusta. Exigimos 10 caracteres com minúscula, maiúscula, número e símbolo. Fazemos escape de caracteres HTML especiais e usamos instruções de SQL preparadas.

**SLIDE 6**

Chegando agora a um dos focos do nosso projecto, temos autenticação de 2 factores com os serviços do Duo.

**SLIDE 7**

Aqui podem ver a ordem das mensagens do desafio e resposta. Primeiro o servidor, tendo chaves secretas do Duo, formula um desafio para o cliente. O cliente envia esse desafio para os serviços do Duo que por sua vez enviam uma notificação push ou código no SMS para o smartphone. O utilizador clica ok na notificação ou insere o código no cliente, o Duo envia ao cliente a resposta ao desafio que o cliente envia para o servidor que finalmente verifica se a resposta está certa.

**SLIDE 8 - Leonardo**

Quanto ao canal seguro entre cliente e servidor, teremos Diffie-Hellman efémero com RSA e AES em modo Galois/Counter.

**SLIDE 9**

Para relembrar o Diffie-Hellman temos aqui a Alice e o Bob que concordam num valor P e G, onde P é grande e G a raíz primitiva módulo P. Depois cada um gera um valor privado grande e aleatório com o qual calcula um valor público usando também o P e G, e após a troca dos valores públicos chegam a um valor partilhado entre eles.

**SLIDE 10**

Para termos segurança futura perfeita, que é o nosso objectivo ao usarmos Diffie-Hellman, em cada nova sessão geramos novos valores privados e públicos. Assim se alguma chave de sessão for descoberta apenas essa sessão é comprometida. E se chaves privadas de longa duração forem descobertas, sessões anteriores não são comprometidas.

**SLIDE 11**

Diffie-Hellman também é vulnerável a man-in-the-middle e por isso assinamos os valores públicos com chaves privadas RSA de longa duração. Aqui podem ver como o Diffie-Hellman é feito na nossa aplicação quando iniciamos o Registo ou Login.

Antes de mais, o cliente não faz operações criptográficas algumas. Isto porque achámos que não iria ser fácil, e talvez nem seja possível, implementá-las em JavaScript. Mas outra razão é que assim precisamos do Smartphone por perto para todas as comunicações com o servidor.

Portanto, primeiro o cliente faz login no smartphone para poder usar os seus serviços criptográficos. Depois pede ao servidor que inicie o Diffie-Hellman exchange. O servidor retorna o P, o G, o valor público do Diffie-Hellman em plaintext e assinado, e a sua chave pública RSA, que passa pelo cliente que envia para o smartphone.

O smartphone em teoria verificaria se o certificado do servidor está assinado por uma CA mas por uma questão prática não fazemos isso. Depois verifica a assinatura do valor público do Diffie-Hellman do servidor, e depois gera o seu valor público e calcula o segredo partilhado.

Depois o smartphone envia o seu valor público do Diffie-Hellman em plaintext e assinado para o servidor, e caso o cliente já esteja registado o servidor tem a sua chave pública RSA na base de dados e verifica este valor. Caso não esteja registado não verifica e não há problema porque não se perde confidencialidade de nenhuma das partes. Tendo o valor público do smartphone o servidor chega também ao mesmo valor partilhado.

**SLIDE 12 - David**

O Diffie-Hellman serve apenas para chegar a um valor partilhado e portanto para a cifra em si usamos AES, e como queremos garantir autenticidade barra integridade usamos o modo Galois/Counter que usa o modo Counter para cifrar, e um GMAC para a integridade.

**SLIDE 13**

Aqui podem ver o funcionamento deste modo. Recebe um IV, gera uma keystream com um contador e a chave. Cifra os blocos em paralelo, recebe ainda dados em plaintext opcionais, e com base nisto tudo gera o GMAC.

**SLIDE 14**

Por fim a proximidade...

**SLIDE 15**

Como vimos num slide anterior o cliente apenas consegue comunicar com o servidor tendo o smartphone por perto já que este contem a chave privada RSA e os métodos criptográficos.

Assim que a ligação com o smartphone se perde a nossa aplicação faz log out imediatamente.

E para a cifra e decifra de ficheiros que infelizmente não chegámos a implementar, era suposto os ficheiros de texto serem decifrados apenas em memória volátil e os ficheiros binários decifrados em disco e cifrados novamente assim que se perdesse ligação ao smartphone, este último que não consideramos uma solução robusta.

**SLIDE 16**

Para concluir, tirando o facto de não verificarmos o certificado do servidor, não termos implementado cifra e decifra de ficheiros, não termos associado a conta Duo com o smartphone do utilizador no registo mas sim no login, e também não termos cifrado os headers to HTTP, consideramos que a nossa lógica e implementação estão num bom caminho.