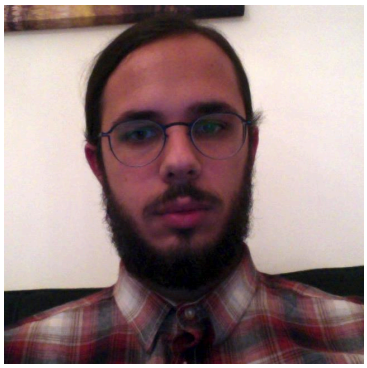
4 de dezembro de 2015

Shuttle Reservation System with User Reputation

Segurança Informática em Redes e Sistemas
Grupo 4 – Alameda

Daniel Sil
75522
daniel.sil@tecnico.pt

Miguel Pasadinhas
75714
miguel.pasadinhas@tecnico.pt



Carlos Carvalho
76012
carlosacarvalho@tecnico.pt

Problema

O objetivo deste projeto foi desenvolver um sistema de reservas num shuttle, usando um sistema de reputação. Este sistema permitirá dar prioridade a utilizadores com um maior *karma* (reputação associada a uma pessoa).

Neste sistema, a segurança é um aspeto de grande relevância, pois é necessário que a integridade do sistema seja mantida. Ou seja, temos de autenticar os intervenientes na comunicação, bem como usar canais seguros, que não permitam ataques de *replay* ou de *impersonation.* Dado que será desenvolvido como uma aplicação *web*, existem também ataques comuns a estas aplicações, como *Cross Site Scripting*, *Cross Site Request Forgery* e *SQL Injection*, os quais têm de ser tidos em conta.

Tipo de ameaça	Ações possíveis
Spoofing	Um atacante envia mensagens para o servidor, fazendo passar-se por motorista de um autocarro.
	Um utilizador efetua ações (registo/cancelamento de viagens) no sistema em nome de outro utilizador.
Tampering	Modificação da base de dados.
	Modificação de dados na comunicação entre o servidor e os clientes.
Repudiation	Um utilizador faz modificações ao sistema alegando posteriormente não ter sido ele o autor dessas modificações.
Information Disclosure	Leitura das mensagens e obtenção de informação (como, por exemplo, passwords).
Denial of Service	Envio de muitos pedidos ao servidor *web* para prejudicar o seu funcionamento.
Elevation of Previlege	Um atacante consegue um *role* que não lhe compete.

Tabela 1 – Análise STRIDE

Desenho da Solução

Overview

A nossa solução é uma aplicação *web*, pois esta tira partido dos canais de comunicação públicos da internet e é facilmente acessível por qualquer utilizador, sendo necessário apenas um *browser*. Assim, as reservas no *shuttle* são feitas através de um *browser* que comunica com um servidor central (daqui em diante referido como *Main Server*). Para registar as presenças no *shuttle* existe uma aplicação *web* residente num computador em cada *shuttle* (daqui em diante referida por *Shuttle Client*). O *Shuttle Client* permite que um condutor registe as presenças de quem viaja no *shuttle*. Por sua vez, esta informação é enviada para o *Main Server*, para gerir o *karma* de cada utilizador de acordo com as presenças. O sistema usa comunicação *https* para garantir a integridade e confidencialidade dos dados. Contudo, por motivos de demonstração de conhecimento, foi criado um canal de comunicação seguro, feito assumindo o uso de http entre o *Shuttle Client* e o *Main Server*. O *Main Server* corre numa máquina protegida por uma *firewall* e usa uma base de dados residente na mesma máquina. Esta descrição pode ser vista de uma forma geral na figura seguinte.

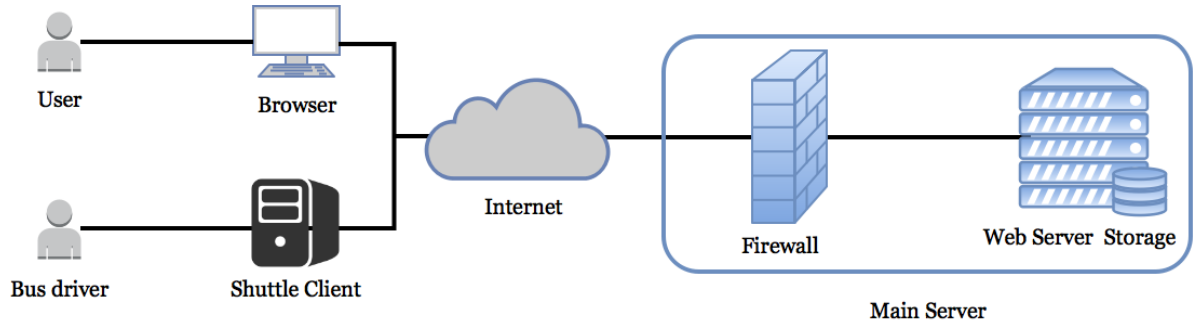


Ilustração 1 - Vista geral

Funcionalidade

Cada utilizador do sistema possui um *username* e uma palavra-passe única, bem como um documento de identificação associado, para eliminar o problema de múltiplas contas para o mesmo utilizador. Todos os utilizadores do sistema podem reservar viagens e desmarcá-las. O sistema possui ainda dois papéis de utilizador, cada um com privilégios e responsabilidades diferentes, podendo um utilizador acumular diversos. Estes papéis são:

- **Motorista** – tem a responsabilidade de conduzir o *shuttle* e de registar as presenças;

- **Gestor** – tem a responsabilidade de registar novos *shuttles* no sistema, marcar e desmarcar viagens e atribuir-lhes um motorista, bem como gerir os utilizadores, atribuindo-lhes papéis, ou resolvendo problemas como o roubo de uma conta, ou um registo com um documento de outra pessoa.

O registo de uma presença no autocarro pode ser efetuado apenas pelo Motorista que está atribuído a essa viagem. Os registos são feitos através da leitura RFID do cartão de identificação do viajante, ou em caso de falha deste sistema, é feito manualmente pelo Motorista, após verificação do respetivo documento.

O sistema de *karma* beneficia os utilizadores com maior karma permitindo-lhes reservar o seu lugar no *shuttle* com uma antecedência maior. Ou seja, um utilizador com *karma* 0 ou negativo apenas poderá reservar viagens com 12h de antecedência, enquanto um utilizador com o *karma* acima de 4320 poderá reservar uma viagem com 15 dias de antecedência. Cada ponto de *karma* permite reservar a viagem 5 minutos mais cedo que alguém com *karma* inferior em um ponto, ex. 1 ponto de karma permite reservar a viagem com uma antecedência de 12h05min.

Um utilizador começa com 144 pontos de karma, correspondente a 24h de antecedência. Por cada viagem reservada a que compareça recebe 12 pontos (1h) de *karma*. Por cada viagem a que não compareça perde 60 (5h) pontos de *karma*, sendo assim tido como uma reputação neutra o não comparecimento 1/6 das vezes. Se desmarcar a viagem perde *karma* de acordo com a seguinte função:

$$f(t_{now}, t_{book}, t_{departure}) = 1 + \left(\sqrt[3]{47} \times \frac{t_{now} - t_{book}}{t_{departure} - t_{book}}\right)^3$$

Esta função penaliza entre 1 e 48 pontos de *karma*, ou seja, de 5min a 4h, fazendo com que os utilizadores sejam desincentivados a marcar viagens não tendo a certeza se vão estar presentes, contudo, é menos penalizante desmarcar do que não comparecer, principalmente se a antecedência da desmarcação for grande.

Medidas de segurança

Para o desenvolvimento da nossa solução assumimos que existe um KEK (*Key Encrypting Key*) partilhado *offline* entre cada *Bus Server* e o *Main Server*. Esta chave é usada para autenticar os intervenientes no canal de comunicação seguro, bem como trocar chaves de sessão. O protocolo de comunicação é ilustrado na figura seguinte, em que ID corresponde à identificação do *shuttle*, para que o *Main Server* saiba qual KEK usar:

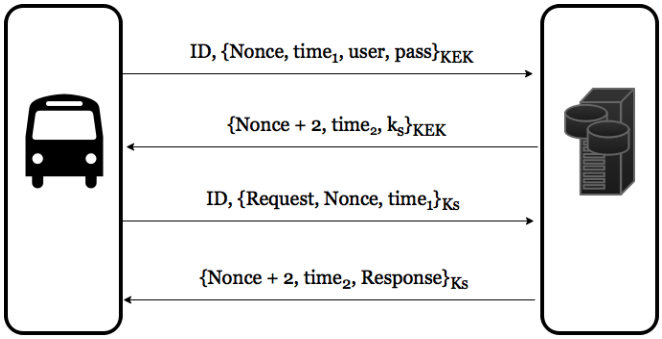


Ilustração 2 – Protocolo de comunicação

Para evitar ataques de XSS, todo o *output* proveniente de um *input* do utilizador é escapado. Para evitar ataques de CSRF, a cada pedido de uma página é incluído um *token* secreto para os formulários, e apenas são aceites inputs de formulários com um *token* correto.

Para minimizar o risco de ataques feitos a partir do interior é mantido um *log* de todas as modificações feitas na base de dados, com a identificação do utilizador que o fez, o endereço IP que vem no pacote IP, bem como a respetiva data.

De modo a minimizar o impacto de ataques de *Denial of Service*, foi configurado um módulo do nginx que atua como uma *firewall*, recusando pedidos de um IP quando este excede um número médio de pedidos por segundo definido, no nosso caso permitimos 5 pedidos. Em conjunto, é utilizado o fail2ban, que monitoriza os *logs* gerados pelo nginx e, quando observa que um dado endereço IP fez demasiados pedidos, coloca-o numa lista negra, não permitindo qualquer pedido http durante um tempo definido, no nosso caso, duas horas. Isto mitiga, em parte, a vulnerabilidade a *DDoS.* O fail2ban monitoriza também as ligações feitas por ssh à máquina onde se encontra o *Main Server* e a base de dados, recusando a ligação, durante duas horas, a endereços que falhem a autenticação três vezes de seguida.

Como dito anteriormente, todos as comunicações usam https. A possibilidade de ter múltiplas contas é mitigada, dado o uso obrigatório de um documento de identificação.

Para evitar ataques de *bruteforce* ao *login* e a automatização de criação de utilizadores foram adicionados *captchas* para impedir o uso de *bots.*

As *passwords* estão encriptadas, com hash, na base de dados, para assegurar que não são lidas por terceiros, nem mesmo por pessoas com acesso legítimo à base de dados.

Implementação da Solução

Para a implementação optámos por usar Laravel, uma *Framework* de Model-View-Controller, em php, para desenvolvimento de aplicações *web*. Esta *Framework* foi escolhida para facilitar, principalmente, o desenvolvimento das interfaces com o utilizador, bem como o *routing*, e para uma mais fácil tradução dos dados presentes na base de dados em classes php. Laravel permite também mitigar *SQL injection*, utilizando *prepared statements*. Também não permite atribuição de valores em massa a uma entrada de uma tabela na base de dados (preencher vários valores numa só *query*), exceto se permitido explicitamente pelo programador quais o podem ser.

Utilizamos *sqlite* como sistema de gestão de base de dados, por simplicidade, contudo seria fácil migrar o esquema para MySQL ou outro SGBD, dado que existem scripts que automatizam o processo.

Instalámos o *Main Server* numa máquina virtual, na qual configurámos um servidor nginx, respetiva *firewall* e fail2ban.

Foi implementado tudo o que foi descrito na descrição da solução acima, exceto a funcionalidade de leitura automática do documento de identificação, por falta de tempo e de *hardware*.

O sistema, como implementado, permite o registo dos utilizadores numa viagem mesmo quando esta viagem só tem partida marcada para daí a algum tempo. Isto permite que um driver possa cometer um erro e registar a viagem errada, ou que, com más intenções, registe uma viagem que não se vai realizar num tempo próximo. Isto devia ser controlado, deixando que uma viagem apenas pudesse enviar um registo dos utilizadores presentes na viagem a partir do momento da partida da mesma.

Para toda a encriptação, foi utilizado o protocolo AES, com chaves de 256 bits em modo CBC, pois esta pareceu-nos um bom compromisso entre eficiência da computação e a segurança oferecida. Para o protocolo HTTPS foi utilizado o protocolo *Transport Layer Security* 1.2, com AES com chaves de 256 bits em modo CBC, com HMAC para *message authentication* e ECDHE_RSA como mecanismo de troca de chaves.

A nossa solução não apresenta proteção contra a criação de uma conta com um Documento de Identificação alheio. Contudo, quem o faça não poderá viajar sem ter furtado o respetivo documento. Se um utilizador se quiser registar e verificar que o seu documento de identificação já foi utilizado, pode contactar um Gestor para regularizar a situação.

O certificado SSL é *self signed* devido à falta de verba para adquirir um de uma certificadora reconhecida. Num sistema real isto não deveria acontecer, pois exigiria uma confiança grande por parte dos utilizadores no fornecedor do serviço.

A proteção contra *DDoS* não é total, se o poder de computação do atacante for muito superior ao poder de computação do servidor, então o atacante terá sucesso.

A base de dados não deveria estar na mesma máquina que o servidor web, pois este é de acesso público, logo mais provável de ser atacado. No nosso caso está assim, por uma questão de gestão de tempo, pois não implementámos uma DMZ e uma rede privada.

No entanto, as ameaças comuns a aplicações *web*, como XSS, CSRF e SQL *injection* estão mitigadas.

O canal seguro implementado autentica o emissor da mensagem devido à chave secreta partilhada e desafios, impede *replay attacks,* com *timestamps*, e *information disclosure*. A integridade do mesmo é garantida, uma vez que todas as comunicações são JSON e é extremamente difícil alterar seletivamente uma mensagem e continuar a produzir JSON válido, depois de desencriptado.

Conclusão

Este projeto revelou-se muito mais intensivo na funcionalidade do que o esperado. Contudo, apresentou alguns desafios de segurança, embora a sua maioria sejam desafios recorrentes na *web*, não abrindo grande oportunidade a soluções inovadoras. Posto isto, implementámos um canal seguro, que não era necessário usando HTTPS, para demostrar conhecimento.

Conseguimos aplicar os conhecimentos adquiridos na disciplina, tanto na identificação de ameaças como XSS e CSRF, bem como no desenvolvimento de um canal de comunicação seguro, através da *web*.

Referências

Foram usadas as seguintes ferramentas:

- Laravel – esta *Framework* MVC escrita em PHP oferece mecanismos elegantes de tratar a persistência, bem como ferramentas de MVC tradicionais.
- fail2ban – esta ferramenta lê os *logs* do sistema (e.g. *logs* do *web server* ou *logs* de acesso ssh) e permite banir IPs com comportamento suspeito;
- nginx – *web server* para correr a aplicação.