



Universidade do Minho
Escola de Engenharia

ENGENHARIA DE SEGURANÇA

Practices for Secure Development of Cloud Applications

Mariana Fernandes A81728
João Costeira A78073
Paulo Mendes A78203

Conteúdo

1	Introdução	2
2	Tipos de serviços <i>cloud</i>	2
3	Ameaças à computação na <i>Cloud</i>	3
3.1	Perda, Vazamento e Violão de dados	3
3.2	Interfaces e APIs Inseguras	4
3.3	Negação de Serviço (DoS)	4
4	Problemas de Design	5
4.1	<i>Multitenancy</i>	5
4.2	<i>Tokenization</i> de Dados Sensíveis	5
4.3	Encriptação de Dados e Gestão de Chaves	6
4.4	Autenticação e Gestão de Identidade	9
5	Problemas de Implementação	9
5.1	Problemas de Partilha de Domínio	9
5.2	Garantir a Segurança das APIs	10
6	Conclusão	11
7	Bibliografia	11

1 Introdução

Atualmente, verifica-se um crescente uso das tecnologias *cloud*, para como plataforma/infraestrutura para uma aplicação, para armazenamento digital, ou até para acesso a poder de computação mais elevado.

Neste trabalho, iremos apresentar boas práticas de desenvolvimento de *software* seguro para a *cloud*. Iremos focar-nos apenas nas medidas suplementares a tomar para *aplicações* na *cloud*, todavia, as boas práticas de desenvolvimento seguro para aplicações "tradicionais" aplicam-se igualmente aqui.

2 Tipos de serviços cloud

Para a compreensão deste trabalho, é importante saber distinguir entre os vários tipos de serviços *cloud*:

- **IaaS** - Infrastructure as a Service (Infraestrutura como Serviço)
- **PaaS** - Platform as a Service (Plataforma como Serviço)
- **SaaS** - Software as a Service (Software como Serviço)

A principal diferença é que partes são geridas por quem está a desenvolver o *software* e que partes são geridas por outros.

O SaaS, é gerido de uma localização central, *hosted* num servidor remoto, acedível na rede e os utilizadores não são responsáveis por atualizações de *software* ou *hardware*. Alguns exemplos são *Dropbox*, *Salesforce*, *Cisco WebEx*, e a suite de aplicações da *Google* (Docs, Mail, Calendar...).

O modelo de PaaS é semelhante ao anterior, mas, ao invés de "entregar" *software* pela *internet*, providencia uma plataforma para a criação do mesmo. A plataforma é disponibilizada *online*, dando liberdade aos programadores para se concentrarem na construção do *software* sem precisarem de se preocupar com sistemas operativos, atualizações, armazenamento ou infraestrutura. Plataformas como *Windows Azure*, *Heroku* e *AWS Elastic Beanstalk* são exemplos de PaaS.

Por último, temos o modelo IaaS, que disponibiliza recursos de uma forma self-service onde as empresas podem aquirir mais recursos ao invés de comprarem *hardware*. Temos exemplos como *Amazon Web Services (AWS)*, *Cisco Metacloud* e *Google Compute Engine (GCE)*. Neste trabalho, iremos ter por base *software* assente em PaaS.



Figura 1: IaaS vs PaaS vs SaaS

3 Ameaças à computação na *Cloud*

3.1 Perda, Vazamento e Violação de dados

Existem já várias técnicas de ataque conhecidas, como *SQL Injections*, que podem permitir a um atacante obter dados confidenciais, e até alterá-los ou apagá-los. O impacto deste género de ataques, se o mesmo for bem sucedido, é bastante considerável, podendo até ser devastador para uma empresa. Se pensarmos que, na *cloud*, várias aplicações pode ter as suas bases de dados alojadas no mesmo servidor, então a gravidade deste ataque é muito maior, pelas consequências que pode trazer para os restantes clientes alojados e para o próprio serviço *cloud*.

A plataforma tem de se encarregar de proteger os dados guardados e em trânsito. Os dados estão em trânsito quando são enviados entre sistemas físicos (por exemplo, do servidor para o cliente), processos, máquinas virtuais ou placas de rede virtuais. No caso das máquinas virtuais, existe a possibilidade de um atacante conseguir ter acesso à infraestrutura onde a mesma assenta, conseguir "saltar" de uma máquina virtual para outra, ganhando acesso à mesma, ou conseguir forçar uma máquina virtual a migrar para um *host* que os atacantes podem controlar.

Quanto aos dados guardados na *cloud*, é importante salvaguardar a possibilidade de fazer *rollback* caso os mesmos tenham sido alterados ou apagados, quer através de *backups*, quer através de *logs*. Nestes casos, a integridade dos *backups* e *logs* tem de

ser igualmente garantida.

Porém, nem todas as soluções são diretas, por exemplo, a implementação de cifras criptográficas traz consigo outros desafios como a gestão de chaves, a correta implementação das cifras, as políticas em caso de perda de chave e em que camadas cifrar os dados.

3.2 Interfaces e APIs Inseguras

As APIs (*Application Programming Interfaces*) e as UIs(*User Interfaces*) são frequentemente as partes mais expostas de um sistema. Sendo que permitem ao clientes dos serviços *cloud* efetuar várias operações, muita da segurança do sistema depende da implementação correta de acesso de controlo e autenticação. É crucial que estas interfaces estejam protegidas de tentativas accidentais ou maliciosas de contornar a segurança das mesmas.

3.3 Negação de Serviço (DoS)

Um ambiente *cloud* é composto por várias camadas, que facilita o seu desenvolvimento, tornando vários componentes "estanques", que simplesmente assentam nou-tros, mas expõe uma maior superfície para ataques de negação de serviço. Vários sistemas, por exemplo, máquinas virtuais, memória, espaço em disco, rede, bases de dados, podem ser atacados individualmente ou em simultâneo, fazendo com que toda a plataforma fique indisponível. Enquanto que um sistema com mais camadas pode ser mais resistente contra certos ataques, como elevação de privilégios ou roubo de dados, é mais vulnerável a *DoS*, inutilizando uma camada, o que por sua vez faz com que todas as outras fiquem indisponíveis por dependerem dessa.

4 Problemas de Design

4.1 *Multitenancy*

Um dos principais desafios de desenvolver *software* para a *cloud* é lidar com os desafios de *multitenancy*. A *multitenancy* é uma referência ao modo de operação de software em que várias instâncias independentes de uma ou várias aplicações operam num ambiente partilhado. As instâncias (inquilinos) são logicamente isoladas, mas fisicamente integradas. O grau de isolamento lógico deve estar completo, mas o grau de integração física varia. Quanto mais integração física, mais difícil é preservar o isolamento lógico.

Para suportar vários inquilinos pode usar-se uma de duas abordagens: ter uma única base de dados que suporta todos os inquilinos; ou ter uma base de dados por inquilino. Esta última opção é a mais segura, pois é mais fácil implementar medidas que visem impedir o acesso ou modificação, acidental ou propositada, de dados não pertencentes ao inquilino.

O *design* do serviço *cloud* deve enforçar as seguintes políticas:

- Nenhum inquilino deve poder identificar ou determinar a existência de outros inquilinos
- Nenhum inquilino deve conseguir aceder aos dados de outro.
- Um inquilino não deve poder fazer operações que interfiram ou impeçam o serviço de outro inquilino.
- As configurações de um inquilino devem ser independentes das dos demais.
- Devem ser feitas auditorias e rastreios por inquilino.

4.2 *Tokenization* de Dados Sensíveis

Apesar da prevenção de ataques ser importante, é de igual importância tomar medidas que impeçam os atacantes de efetivamente ler os dados sensíveis que estão a ser alojados. Por dados sensíveis entende-se dados que não devem ser do conhecimento público geral, e só devem ser dados a conhecer em contextos específicos, por exemplo, cada utilizador ter acesso unicamente aos seus dados. Podem ser dos seguintes tipos

- Propriedade intelectual e outros dados de negócio como listas de clientes.
- Dados de pagamento (PCI - *Payment Card Industry*) como números de cartões.
- Informação de saúde protegida como registos médicos.

Na próxima subsecção, é explicado como usar a criptografia para proteger estes dados. No entanto, existem outras abordagens como a *tokenization* e mascarar dados.

Tokenization consiste em remover os dados sensíveis e guardar um *token* no seu lugar. Um exemplo prático disto é pagamentos com cartão, onde o número real do cartão é substituído por um *token*. Assim, o *token* é usado para efetuar transações e apenas uma entidade responsável por manter um dicionário de *tokens* sabe o número original do cartão. A criação do *token* é feita através de funções de *hash* ou outras funções computacionalmente impossíveis/difíceis de resolver. Podemos ver na figura seguinte o processamento de um pagamento utilizando *tokens*.



Figura 2: Processo de compra com *tokens*

Em certos casos, é preferível mascarar dados, ou seja, desassociar os mesmos da identidade que os torna sensíveis. O objetivo é ofuscar a parte que os identifica, para que os sistemas não tenham que lidar com dados sensíveis, querendo dizer que não há a necessidade de cumprir com normas de proteção de dados sensíveis ou implementar segurança mais forte. Isto é particularmente útil quando os sistemas ainda se encontram em testes, e precisam de ser alimentados com dados, mas ainda não dispõem das proteções corretas para poderem lidar com informação sensível.

4.3 Encriptação de Dados e Gestão de Chaves

Num sistema como uma *cloud*, o armazenamento dos dados como a sua partilha entre diferentes utilizadores/aplicações/servidores ocorrem a todo o momento. Desta forma, a adição de medidas de segurança no sentido de proteger os dados partilhados e armazenados é fundamental para garantir o bom funcionamento do sistema.

Uma das recursos clássicas que tem como intuito a protecção dos dados é a criptografia, desta forma é impeditido o acesso indevido aos dados neste sistema crítico e complexo, composto por múltiplos servidores e clientes/aplicações.

No ponto de vista da criptografia, os dados do sistema podem ser classificados em dois grandes grupos: por um lado o *data-in-motion*, também referenciado como *data-in-transit* ou *data-in-flight*, ou seja os dados que se encontram em transmissão entre diferentes componentes do sistema, por exemplo de servidor-servidor ou servidor-cliente.

Por outro lado, temos *data-at-rest* utilizado para caracterizar dados dentro do sistema, por exemplo dados armazenados em ficheiros ou base de dados.

De acordo com os dados, diferentes algoritmos/sistemas criptográficos são utilizados de forma a proteger a informação. No caso de *data-in-transit*, a segurança será garantida de acordo com protocolos como TLS/SSL ou IPSec.

Sobre os dados em armazenamento(*data-at-rest*), por padrão utiliza-se o algoritmo mais adequado para a formatação dos dados, por exemplo utilização de algoritmos criptográficos orientados a base dados ou ficheiros.

De notar que os algoritmos criptográficos por vezes são logo aplicados à medida em que os dados são escritos em memória.

No contexto de *clouds*, como o sistema é composto por múltiplos servidores/aplicações, o sistema implementa medidas criptográficas sempre que seja detectada a transferência dos dados de um ponto do sistema para o outro. Por exemplo antes de dados serem transmitidos pela rede, em *switches* existentes no sistema.

Gestão de Chaves

O recurso a sistemas criptográficos na protecção de dados, causa um novo problema no sistema: como implementar o sistema de manutenção/armazenamento das chaves.

Implementações inseguras do sistema de chaves podem causar libertação ou toda a informação nunca ser decifrada no caso de perda das chaves e consequentemente os dados são perdidos.

Sobre os dados em transmissão, os algoritmos referidos anteriormente (TLS/SSL ou IPSec), já efectuam a gestão de chaves por si mesmos. Mas de notar de como estamos perante algoritmos assimétricos, a autenticação da chave pública é de extrema importância, por isso recorre-se a certificados como o x509.

A manutenção da chave pública pode ser gerida pelo próprio sistema(*in-house*) ou

por outra empresa (*third-party*).

Sobre os dados em armazenamento (*data-at-rest*), normalmente é utilizada criptográfica simétrica, ou seja, mesma chave utilizada no processo de encriptação e decifração. Desta maneira, é adicionada uma medida extra para proteger a chave: a chave que é utilizada na criptografia (*data encryption key (DEK)*) é protegida por uma segunda chave (*key encryption key (KEK)*), ou seja, a chave principal é encriptada.

Em relação ao lugar de armazenamento das chaves, elas podem ser armazenadas em múltiplos servidores ou centralizadas entre múltiplos servidores. Cada uma das implementações tem as suas vantagens/desvantagens, mas por padrão em sistemas de *cloud*, recorre-se à centralização do armazenamento das chaves, devido ao controlo no acesso e implementação de redundância que permite a recuperação dos dados.

Notas sobre Encriptação e Chaves

- **Respeitar o regulamento:** Se o sistema implementado criptográfico gerência de chaves está de acordo com as leis dos países associados, desde acesso a backups/chaves de acordo com a lei.
- **Algoritmos criptográficos:** Não só utilizar algoritmos criptográficos considerados seguros, como também ter atenção aos modos do algoritmo, tamanho das chaves entre outros pormenores associados à implementação correta do algoritmo.
- **Uso do FIPS 140-2 (Federal Information Processing Standard) ou Módulos criptográficos validados:** O Sistema estar de acordo com standards ou sistemas implementados certificados como seguros.
- **Protocolos/Standards sobre chaves:** Recurso sempre que possível a protocolos como (X.509, RFC5280, OASIS KMIP) de forma a tornar o armazenamento de chaves seguro.
- **Integração entre o Sistema criptográfico:** Orientar o sistema à criptográfica, por exemplo se os dados na base de dados estão encriptados, pode causar problemas sobre pesquisa.
- **Escolher algoritmo adequado:** Escolher o algoritmo criptográfico adequado pro Sistema, por exemplo questões como cifra por blocos ou fluxo (*stream*).

4.4 Autenticação e Gestão de Identidade

Numa aplicação é fundamental identificar quem tem acesso a quais funcionalidades/dados. Desta forma a autenticação funciona como um filtro/barreira em relação a acesso.

Uma aplicação na magnitude de *cloud*, necessita implementações múltiplas de métodos de autenticação.

Desta forma, em relação a autenticação, a *cloud* é caracterizada por três ambientes: público, privado ou híbrido. Esta classificação permite depois identificar diferentes níveis de segurança e quem é que possui privilégios para aceder a cada um dos diferentes níveis.

Ao nível aplicacional privado, a autenticação pode recorrer a sistemas já implementados como *Microsoft Windows domain* e *LDAP*. Ao nível publico, recorre-se a mecanismos como *OATH* and *OpenID*.

Desde o processo de desenvolvimento da aplicação, é necessário ponderar em que locais da aplicação e que tipos de autenticação são utilizados/suportados.

No ponto de vista do utilizador, é interessante que o sistema seja visto como algo integrado em que só necessita autenticar uma vez (*single sign-on (SSO)*). Após o processo de autenticação seja efectuado com sucesso, o utilizador pode aceder a todas as funcionalidades da aplicação, em vez de estar constantemente a autenticar-se sempre que muda de regiões consideradas como pública/privada ou híbrida.

Desta forma, a utilização de ferramentas como *Security Assertion Markup Language (SAML)* podem ser utilizadas de forma a implementar um sistema *SSO*.

Bibliografia

5 Problemas de Implementação

5.1 Problemas de Partilha de Domínio

Apesar da conveniência proporcionada pelas plataformas de cloud, que permitem o armazenamento de conteúdo ou teste de aplicações de cloud, estas mesmas plataformas podem também conter os seus problemas de segurança, devido a pormenores da política da mesma origem. Esta política é um mecanismo existente nos browsers que impede que uma página que esteja a executar scripts cliente-side (ex.: JavaScript) consiga ler dados que se encontram fora do seu domínio de origem. Este mecanismo, apesar de ser benéfico do ponto de vista da segurança, vai ser um obstáculo caso seja desejado que os scripts cliente-side de dois serviços web com origens diferentes possam aceder a dados um do outro.

Esta restrição pode ser contornada se ambos os serviços estiverem alocados no

mesmo sistema de cloud: basta definirmos a propriedade do ficheiro JavaScript document.main das aplicações para a mesma origem, que é a fornecida pelo sistema de cloud. O problema desta decisão é que qualquer outro serviço que esteja aloocado na mesma cloud pode fazer o mesmo e passa a ter acesso aos dados das duas aplicações. Uma forma de evitar que isto aconteça é garantir que o sistema de cloud permite definir domínios específicos para as aplicações em questão.

5.2 Garantir a Segurança das APIs

De modo a garantir a Segurança das APIs utilizadas pela aplicação, é essencial que se tomem determinadas precauções durante o processo de desenvolvimento da mesma.

Várias APIs de serviços web não foram desenhadas para ser expostas ao utilizador final, e para estas, o seu acesso deve ser restringido o mais possível, de modo a que apenas um número reduzido de utilizadores aprovados a consigam aceder.

Isto pode ser atingido através da implementação de autenticação SSL mútua, ou de regras de firewall que limitem o acesso apenas a IPs conhecidos.

Para aplicações que lidem com o upload de ficheiros é aconselhável que sejam realizadas determinadas verificações, como análise assistida por um software antivírus.

Ao desenvolver a aplicação devem ser aplicadas ferramentas de teste apropriadas, de modo a que sejam abrangidas todas as vulnerabilidades tradicionais de um serviço web. A análise cuidadosa dos registos produzidos pelas ferramenta e aplicação no momento da tentativa de ataque, vai revelar se os ataques à aplicação estão a ser detectados, contidos e registados pela mesma.

6 Conclusão

O aparecimento e amadurecimento da computação *cloud* já proporcionou vantagens significativas para utilizadores de tecnologia de todos os tipos, e muitos acreditam que apenas começamos a explorar as possibilidades. Mas com novos avanços, surgem novos desafios de segurança. O caminho para a computação *cloud* segura não pode ser visto como uma simples lista de verificação de práticas, mas sim um processo que deve evoluir ao lado de novos casos de uso e avanços em nossa compreensão da segurança de TI e *software*.

Não devemos, portanto, descurar as medidas para desenvolvimento seguro das aplicações "tradicionalis" pois oferecem uma boa base de segurança. Compreender as diferenças entre arquiteturas tradicionais e serviços *cloud* ajuda também a perceber quais os pontos mais frágeis de um *software*.

7 Bibliografia

- <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>
- https://safecode.org/publication/SAFECode_CSA_Cloud_Final1213.pdf
- <https://www.techrepublic.com/article/how-to-prevent-the-top-11-threats-in-cloud-computing/>
- <https://www.gartner.com/en/information-technology/glossary/multitenancy>

Isto não é um formato usual para bibliografia. Sugiro que utilizem o próprio LaTeX que tem a vantagem de permitir referenciar facilmente no resto do texto.

Nota final: O ponto que é "criado" tem que ser escrito de forma mais