Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Segurança Informática

Aula 8

Licenciatura em Engenharia Informática
Licenciatura em Informática Web

Computer Security

Lecture 8

Degree in Computer Science and Engineering
Degree in Web Informatics

Sumário

Definição de Segurança da Informação e discussão acerca da análise do risco associado a vulnerabilidades, bem como da problemática da sua quantificação. Evolução para o estudo de detetores de vulnerabilidades em sistemas distribuidos em redes de área local.

Summary

Definition of Information Security and discussion concerning the analysis of the risk associated with vulnerabilities, as well as of the problem of quantifying that risk. Evolution to the study of the detection of vulnerabilities in distributed systems over local area networks.

1 Securança da Informação

Information Security

1.1 Uma Definição de Segurança da Informação

Information Security Definition

A definição de Segurança da Informação ou da área da Segurança da Informação não é consensual na literatura ou em grupos da especialidade. A ter de se usar uma definição, pode optar-se pela que é descrita na norma internacional a ISO/IEC 27000 (ISO/IEC_JTC_12009), ou na legislação norte-americana, que condiciona as normas do *National Institute of Standards and Technology* (NIST), normalmente aceites no âmbito da segurança informática:

> Segundo esse conjunto de referenciais, a expressão ***Segurança da Informação***, por vezes abreviado por SegInfo, é sinónimo de **proteção da informação e dos sistemas de informação das ameaças às três propriedades fundamentais CIA** —*Confidentiality, Integrity and Availability* (**Confidencialidade**, **Integridade** e **Disponibilidade**).

Do processo da Segurança da Informação fazem parte diversos componentes, nomeadamente **a identificação das ameaças, dos recursos críticos e das vulnerabilidades**, **a quantificação do risco e a concretização dos objetivos de segurança para determinada organização ou entidade**, nomeadamente através de **um documento escrito**, assim como a discriminação dos mecanismos e tecnologias usados para atingir esses objetivos. Esse **documento designa-se por política de segurança**.

1.2 Quantificação do Risco

Risk Quantification

Numa organização, **a aposta em mecanismos de segurança é algo difícil de negociar**, sobretudo porque **o investimento em segurança pode não ser reavido nem a curto nem a médio prazo**, e porque esse investimento não parece poder ser comparado imediatamente com o valor do equipamento ou informação que se quer proteger. É, ainda assim, possível **definir uma fórmula simples para o cálculo do risco** (R) **associado a determinada vulnerabilidade** num sistema informático, que pode ajudar nessa comparação:

$$R = P \times V,$$

em que V **define o valor do sistema ou informação a proteger**, ou o valor implicado num incidente de segurança relacionado, e P **define a probabilidade de ocorrência de um ataque** que explore a vulnerabilidade respetiva.

Se o risco for reduzido a um valor monetário, poder-se-á melhor equacionar no âmbito da gestão financeira e de negócio da organização. Provavelmente, **aceitar-se-á que a quantia a investir na segurança de um sistema ou da informação terá de ser inferior ou, no máximo igual, ao risco (em termos de valor) associado a esse sistema ou à informação**[1].

Por **exemplo**, considere que a atualização de uma aplicação *Web*, suscetível a ataques de injeção de código, tem um custo $1000, e que a base de dados gerida por essa aplicação guarda informação com um valor de $5000. Estima-se que a probabilidade de ocorrer um ataque de injeção de código é de 10%, porque a aplicação é apenas acedida via Intranet, depois dos utilizadores serem devidamente autenticados e as suas atividades mo-

[1] Pfleeger, C. P. and S. L. Pfleeger (2007). Security in Computing - 4th ed. Upper Saddle River, NJ, Prentice Hall.

nitorizadas. Neste caso, o risco era de $500, pelo que a gestão podia concluir que o gasto na segurança era desproporcional. Caso essa aplicação pudesse ser acedida via Internet por qualquer utilizador, e a probabilidade subisse para 21%, o caso seria revisto.

Note-se que, apesar do discurso com índole aparentemente simples desta secção, **a fórmula é de difícil aplicação**, porque implica:

1. **Identificação e caracterização de vulnerabilidades**, bem como dos **ataques** a que está afeta;
2. **Dedução da probabilidade** de um ataque que explore as vulnerabilidades identificadas.

Como discutido em baixo, **a identificação e caraterização de vulnerabilidades é**, por isso, **importante para atacantes e profissionais da segurança da informação**.

1.3 O Caminho Percorrido
The Road so Far

A Criptografia preocupa-se com a proteção de dados confidenciais, autenticação de entidades, integridade dos dados, anonimato, não repúdio, etc., a montante dos problemas de segurança. Contudo, **a área segurança da informação não se resume à da criptografia**. Daqui em diante, e até próximo do final desta unidade curricular, trata-se **o tema das vulnerabilidades e dos ataques que as exploram**, bem como dos sistemas e outras tecnologias que se usam para as identificar, resolver ou atenuar os seus efeitos. Estas vulnerabilidades estão presentes em sistemas informáticos terminais (e.g., computadores pessoais) e de rede (e.g., *routers* ou *switches*) e devem-se, na sua maioria, a **(i) problemas de desenho** (ou projeto), **(ii) de realização** ou **(iii) de administração** de sistemas operativos e de serviços. Por vezes, as vulnerabilidades surgem de situações que o programador ou o arquiteto de sistema não é capaz de colocar à partida, simplesmente **porque é humano**.

2 Detetores de Vulnerabilidades
Vulnerabilities Detectors

O **sucesso de um ataque**, ou da **técnica usada para o prevenir**, **depende de um passo inicial importante** (pelo menos para os atacantes e profissionais experientes): **a identificação do Sistema Operativo** (SO) **e das vulnerabilidades da máquina** alvo.

Sabendo o SO, é possível obter imediatamente **um catálogo de todas as vulnerabilidades** que saíram com a primeira versão ou em versões posteriores do mesmo. **Sabendo vulnerabilidades, podem-se explorar ou mitigar.** A identificação prévia do SO e das vulnerabilidades poupa tempo ao responsável pelo sistema e ao atacante, resultando numa probabilidade de sucesso maior para as técnicas de defesa ou de ataque, conforme o interveniente. **As ferramentas e técnicas de identificação de SOs e de serviços, versões e vulnerabilidades** atingem, por isso, um grande relevo nesta parte do processo da segurança da informação.

2.1 Flâmulas
Banners

Uma das formas mais simples de identificar o SO consiste em observar o *banner* publicitado por servidores aquando do acesso à máquina. Muitos servidores enviam **uma mensagem de boas vindas aquando da primeira ligação**, onde **anunciam o seu nome, nome a versão do *software* servidor** e, frequentemente, **o SO** em que estão a executar.

Esta forma de identificação é muito genérica porque **permite identificar a família de SOs** (e.g., UNIX/Linux, MS Windows, MacOS, Cisco IOS, etc), **mas não todos os serviços ou vulnerabilidades** dos mesmos. Este método **não é útil para identificar o SO da maioria das máquinas cliente (porque não têm serviços a correr) e não funciona quando se suprimem ou alteram (de propósito) os *banners* dos servidores**.

Em baixo, mostra-se um exemplo de como se pode obter alguma informação acerca da máquina que serve o site da Microsoft: `$ telnet www.microsoft.com 80`

```
GET /index.php HTTP/1.1
host: www.microsoft.com
<cr>
```

O *output* na conclusão bem sucedida do procedimento anterior é algo parecido com:

```
(...)
HTTP/1.1 200 OK
(...)
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET

(...)
```

que mostra claramente que o SO da máquina destino é o Windows e o servidor de páginas *HyperText Markup Language* (HTML) é o *Internet Information Services* (IIS) versão 8.0.

2.2 Impressão Digital da Pilha IP
IP Fingerprinting

Quando falamos em identificação do SO ou das vulnerabilidades, estamos potencialmente a pensar fazê-lo remotamente via rede de computadores (a técnica anterior também funciona via rede). **Uma das melhores maneiras de fazer a identificação é através** da técnica a que

se chama de **impressão digital da pilha *Transmission Control Protocol*/*Internet Protocol*** (TCP/IP) (da designação inglesa IP *fingerprinting*). A verdade é que:

1. **Todos os sistemas** com ligação à Internet possuem esta pilha;
2. **A especificação** dos protocolos da suite TCP/IP **deixa espaço para alguma personalização**, que pode variar de SO para SO;
3. O **comportamento padrão permite expansões**, que são fontes de diferenças.

Desvios ao comportamento padrão da pilha são **indicadores do SO utilizado e da respetiva versão**. A **reação a situações absurdas não é** normalmente **coberta pela especificação** do protocolo, logo **é resultado de decisões dos programadores** do SO. A existência ou não de reações erradas é especialmente útil para **identificar diferentes versões do mesmo SO, remendos** (*patches*) ou *Service Packs*.

Algumas ferramentas que procuram obter o IP *fingerprinting* são a `nmap` (tratada a seguir e numa aula prática), a RING e a Cron-OS, embora estas últimas constituam mais protótipos utilizáveis para provas de conceito, do que ferramentas de qualidade. Enquanto que **a RING procura explorar o tempo de reação do SO a pacotes TCP** enviados pela rede, **a `nmap` foca-se nas respostas a esses estímulos**.

No caso da `nmap`, **os estímulos são**, são em grande parte, **segmentos** TCP ou **datagramas** *User Datagram Protocol* (UDP) **especialmente forjados**, conhecidos como **sondas**. Estas sondas são **mais eficazes**, mas também **mais arriscadas**, pois são pacotes forjados e que podem não se parecer com comunicações normais, revelando uma tentativa de intrusão. O **risco** pode ser **minimizado** através de **sondas furtivas** (*stealth probes*), i.e., aparentemente normais e não facilmente correlacionadas entre sí no tempo.

2.3 RING

RING

A especificação do protocolo **TCP** determina que as partes interatuantes de uma ligação **devem retransmitir os dados caso não haja resposta** depois de um estímulo, mas *não determina exatamente quantas vezes devem retransmitir ou quanto tempo* devem esperar entre retransmissões. Estes graus de liberdade são explorados pela ferramenta RING, que os usa da seguinte forma:

1. **a ferramenta envia um pedido de início de ligação** (um segmento SYN) para o sistema alvo **e espera pela resposta, à qual nunca responde**;
2. Depois **conta o número de vezes que tentam retransmitir** e o **tempo entre retransmissões**;

A ferramenta **só funciona para sistemas que tenham portos TCP abertos** (e.g., não funciona para serviços UDP) e a sua **identificação não é muito granular**.

2.4 *Network Mapper* (NMAP)

Network Mapper (NMAP)

A ferramenta `nmap`[2], **uma das mais conhecidas desta área**, agrupa diversas técnicas de identificação, mas a ideia básica é a de **enviar vários pacotes TCP ou UDP para o SO alvo e esperar pela resposta**. Depois **combina as respostas** aos vários estímulos de modo a **obter uma assinatura** que pode comparar com uma **base de dados de assinaturas**. Esta base de dados é **aberta e está em constante evolução**, já que os SOs também estão em constante mutação e saem remendos (*patches*) que alteram o comportamento quase diariamente. A `nmap` é de **uso livre**, mas deve ser manuseada com cuidado.

É interessante saber que, **quando a ferramenta não tem a certeza absoluta do SO que descobriu, dá uma sugestão estatística**, e que também pode ser usada para identificar serviços a correr em determinada máquina.

Algumas das técnicas que a `nmap` utiliza são as seguintes:

- **Identificação do algoritmo de geração de números iniciais de sequência** do TCP (incremental, por saltos, pseudo-aleatório, verdadeiramente aleatório, etc);
- **Identificação das opções TCP** suportadas e da ordem por que são indicadas;
- **Resposta a um segmento TCP sem *flags* e sem opções** para um porto aberto (isto revela (i) se o sistema responde, (ii) se o sistema usa a opção *Don't Fragment* na resposta e (iii) quais as *flags* e valores usados na resposta);
- Resposta a um datagrama **TCP com um conjunto absurdo de *flags*** para um porto aberto (alguns sistemas respondem, outros não);
- **Resposta a um datagrama TCP ACK sem opções** enviado para um porto aberto (a resposta correcta é ACK/RST, mas há sistemas que respondem de outra forma);
- Resposta a um segmento TCP SYN sem opções enviado para um porto fechado (a resposta deve ser como a de cima);
- Resposta a um segmento TCP ACK sem opções enviado para um porto fechado (a resposta deve ser como a de cima);

[2] `http://nmap.org`

- Resposta a um segmento TCP com conjunto absurdo de *flags* e enviado para um porto fechado (a resposta deve ser como a de cima).
- **Resposta a um datagrama UDP enviado para um porto fechado**.

2.5 Iludir o NMAP
Deceive NMAP

É possível iludir o `nmap`, e.g., **alterando o núcleo** (*Kernel*) do SO, de modo a que **responda a pedidos TCP ou UDP de uma forma não catalogada** (nem sempre é fácil alterar o núcleo de um SO). Também se pode tentar **evitar que os datagramas estímulo cheguem à máquina alvo** ou que as **repostas cheguem inalteradas à ferramenta**. Note-se que, **se os segmentos forem absurdos**, podem sempre **ser bloqueados** (*dropped*) por *firewalls* do tipo filtro de pacotes. Se os segmentos não forem absurdos, **o sistema *gateway* pode mudar as respostas com valores aleatórios** para minimizar a probabilidade de deteção. Esta possibilidade requer, contudo, tecnologia mais complexa e, portanto, mais cara.

2.6 Inventariação de Serviços
Service Scanning

Os ataques exploram normalmente vulnerabilidades de sistemas operativos ou de serviços. Portanto, é também **útil avaliar, mais detalhadamente, quais os serviços que correm em determinada máquina** antes de a proteger ou atacar, bem como a sua versão. A inventariação mais elementar **consiste em identificar quais os portos da camada de transporte acessíveis** numa máquina alvo. Se esses portos estiverem associados a serviços, então esses serviços estão provavelmente disponíveis. Contudo, o **facto dos portos estarem abertos não significa que o serviço esteja disponível** a qualquer cliente (e.g., pode haver regras de controlo que limitem o seu uso).

2.7 Rastreio de Portas TCP
Port Scanning TCP

A forma mais comum de verificar se existe um serviço a correr em determinada porta TCP é **tentar estabelecer uma ligação nessa porta**. Se a ligação for estabelecida, o serviço está disponível. E.g., fazer `telnet` para uma máquina numa determinada porta pode dar imediatamente a indicação se essa porta tem um serviço associado ou não.

Para um atacante, o problema reside no facto de que, **se a ligação for completamente estabelecida, este deixa um rasto** (e.g., o servidor pode registar o endereço IP de quem se ligou e depois não mandou dados). Assim, na maior parte das ocorrências de **atividades de rastreio maliciosas**, o atacante opta por **mandar apenas um segmento SYN** e esperar pela resposta, **sem estabelecer a ligação**. Se a resposta for um SYN/ACK, existe um serviço ali a correr; se for um RST (*Reset*), não existe serviço associado a esta porta.

Este **comportamento também é detetável**, mas não tão facilmente como para o descrito antes, porque o serviço (da camada acima, nunca chega a ver uma ligação iniciada).

Um atacante pode enveredar por **outras técnicas** para evitar ser detetado, nomeadamente:

- **Enviar um segmento FIN, em vez de um SYN**: se o porto estiver fechado e a pilha TCP/IP estiver bem implementada, deverá ser enviado um segmento RST; caso contrário, não é enviada nenhuma resposta. Esta solução é melhor que a anterior na medida em que os sistemas de deteção de intrusão teriam de guardar informação acerca das sessões para identificar que os segmentos FIN solitários eram maliciosos.
- **Sobre-fragmentar os pacotes**, enviando os pedidos SYN ou FIN divididos em vários pacotes IP, de modo a forçar que o sistema de deteção de intrusões tenha de os desfragmentar antes de os identificar. Claro que isto também causa alguma sobrecarga nos sistemas terminais do atacante e vitima.
- **Usar mediadores** (não discutido aqui), e.g., mediação com servidores *File Transfer Protocol* (FTP); ou Mediação com Oráculos (*idle scan*).

2.8 Rastreio de Portas UDP
Port Scanning UDP

O reconhecimento de serviços **em portos UDP é mais difícil** porque **não existe o conceito de ligação**. Nestes caso, envia-se também um datagrama UDP para o porto em questão, e espera-se a resposta: se **não houver serviços a correr** nesse porto, é **enviado um datagrama ICMP com um erro**. Se houver serviços a correr na porta, pode ou não haver resposta. **Neste caso, a identificação é feita pela negativa.**

2.9 Reconhecimento de Versões de Servidores
Identification of Server Versions

Saber as portas que estão abertas **pode dar uma indicação inicial de qual é o serviço** que as está a usar. E.g., a porta 80 tem normalmente um servidor *HyperText Markup Language* HTTP associado; um servidor telnet estará à escuta na porta 26; um servidor *Post Office Protocol* (POP) estará à escuta na porta 110; etc. Contudo, **este tipo de informação não deve ser entendida como terminante**, já que podem, e.g., existir servidores HTTP a operarem noutras portas; ou porque o número da porta não nos dá a versão do servidor.

Para aumentar a precisão da sua avaliação, a ferramenta `nmap` procura **questionar os serviços cujas portas estão abertas com sondas adicionais**. Uma base de dados de assinaturas permite-lhe obter uma informação mais precisa do serviço que corre nessa porta.

2.10 Inventariação de Deficiências de Administração

Identification of Administrative Vulnerabilities

A administração de sistemas computacionais é uma tarefa complexa e a correta configuração dos servidores é essencial para reduzir o número vulnerabilidades dos sistemas. Esta correção pode ser vista de dois ângulos:

1. **Correção dos serviços prestados** (i.e., testar se o servidor funciona);
2. **Segurança do sistema contra explorações ilegítimas** de falhas de configurações.

As **ferramentas de inventariação de deficiências** de administração **procuram vulnerabilidades conhecidas em máquinas**, facilitando o trabalho do humano administrador de sistemas que, humano que é, comete erros.

O ***OpenVAS*** (*Open Vulnerability Assessment System*[3]) é a versão gratuita e derivada do *Nessus*. É **uma ferramenta pública de inventariação remota de deficiências e possui uma arquitetura cliente/servidor**. Um cliente pode pedir ao servidor que faça uma auditoria ao seu sistema ou a outro sistema dentro da rede de área local (ou acessível a partir desta), e este último **devolve um relatório final após o trabalho concluído**. O relatório inclui **a lista de serviços encontrados e das vulnerabilidades detetadas**. Pode ainda usar servidores remotos instalados nas máquinas cliente (chamados *sensores* ou *agentes*) para **realização de auditorias internas** (e.g., palavras-chave fracas).

2.11 Comentário Intermédio

Intermediate Remark

Note que a discussão anterior (e a que se segue) **é feita do ponto de vista do profissional de segurança da informação e do atacante**. Deve refletir qual o lado com que se identifica e repensar a atitude se necessário. A explicação é feita desta forma porque se acredita que é preciso saber como um atacante pensa e atua para se ser um bom profissional na área.

3 Cenários Absurdos

Absurd Cases

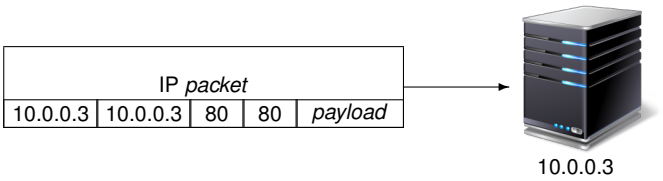
O **desenho, implementação ou administração de um sistema de informação é** normalmente **dominado por cenários normais de operação**. Torna-se **difícil a um ser humano abstrair-se do desenlace normal** de uma situação para desenvolvimento do sistema, pelo que se torna difícil discriminar os cenários anormais de execução. Pior que isso, **o conjunto de cenários anormais é aberto**, no sentido em que pode dilatar-se no futuro, sendo quase **impossível precaver-se contra todas as situações anómalas**.

Os cenários anormais podem **surgir a partir de uma conjugação de fatores não premeditados**, **ou a partir de intenções maliciosas**. No primeiro caso, essa conjugação **pode resultar na falha do sistema de modo imprevisível** (e.g., ecrã azul). No segundo caso, a conjugação de fatores é o **resultado da tentativa de exploração de vulnerabilidades derivadas de mau desenho** (e.g., especificação do protocolo) **ou de realização deficiente** (e.g., implementações mal feitas da pilha TCP/IP). A seguir, procura-se discutir este tema através de exemplos de ataques (alguns clássicos) que o ilustram.

3.1 Ataque LAND

LAND Attack

Um ataque bastante simples e anteriormente muito **eficaz no bloqueio de sistemas servidores** era designado por *LAND Attack*. O ataque **consistia em enviar um segmento TCP SYN com o mesmo endereço fonte e destino e o mesmo porto fonte e destino para a máquina vítima**. A **pilha TCP/IP (defeituosa) de alguns**



SOs respondiam a este pedido e entravam em ciclo logo de seguida, ficando **lentos ou incapazes de responder a pedidos legítimos**. Este ataque pode ser evitado usando *firewalls* de perímetro que bloqueiem pacotes IP vindos do exterior com IP fonte vindo do interior, ou usando *firewalls* pessoais que ignorem pacotes IP vindos do exterior com o mesmo IP fonte que a máquina onde estão instalados.

É possível bloquear ataques LAND usando *firewalls* pessoais, colocando uma regra que rejeite pacotes vindos da própria máquina protegida. E.g., as regras `iptables` seguintes protegem contra este ataque:

```
$ iptables -A INPUT -s own_ip/32 -j DROP
$ iptables -A INPUT -s 127.0.0.0/8 -j DROP
```

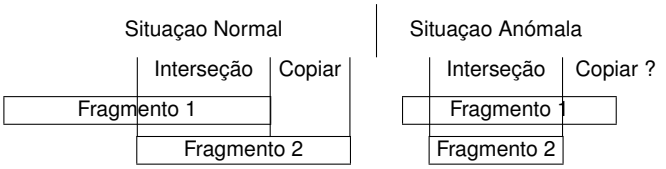
3.2 Ataque Teardrop

Teadrop Attack

O ataque *Teardrop* **explorava uma vulnerabilidade de implementação do algoritmo de desfragmentação IP**. Neste caso, o algoritmo problemático **verificava se dois**

[3] Ver `http://www.openvas.org/`.

fragmentos do mesmo pacote se sobrepunham, e calculava a parte que devia ser copiada em caso de sobreposição. **O ataque consistia em forjar dois fragmentos de modo a que um fragmento estivesse dentro do outro** (este é o cenário absurdo que o implementador do algoritmo não considerou). O algoritmo iria então calcular uma área a copiar negativa, e dado a função que implementava a cópia só aceitar valores sem sinal, **interpretava aquele valor como um valor de grande dimensão**, levando o procedimento a **procurar o fragmento numa posição errada de memória** (fora dos limites). O



impacto exato deste ataque é **imprevisível e pode variar**, mas era natural levar a uma falha mais ou menos grave do sistema vítima. Esta vulnerabilidade **resolve-se com uma melhor implementação do algoritmo** ou com um *firewalls* de perímetro que forcem a desfragmentação de datagramas à entrada da rede protegida.

3.3 Ataque Echo-Chargen
Echo-Chargen Attack

As máquinas UNIX possuem **dois serviços para teste** de problemas em rede:

1. o `echo` (porta 7), que reenvia para a fonte qualquer pacote UDP que receba;
2. e o `chargen` (porta 19), que gera uma cadeia de caracteres e a envia para a fonte, sempre que recebe um pacote UDP na sua porta.

O ataque *Echo-Chargen* consiste em **enviar um pacote UDP com o mesmo endereço IP fonte e destino para a máquina vítima**, e **com os portos UDP de um e de outro serviço na fonte e no destino**.

O `echo` reenvia para o `chargen`, e o `chargen` reenvia para o `echo`. Para evitar este ataque, teria de se negar a existência de pacotes com pedidos UDP para portos que são usados por aqueles serviços. As duas regras seguintes permitiam a qualquer *Gateway* com Linux bloquear estes ataques:

```
$ iptables -A INPUT --sport 19 --dport 7 -j
DROP
```

```
$ iptables -A INPUT --sport 7 --dport 19 -j
DROP
```

3.4 Ping of Death
Ping of Death

O ataque *Ping-of-Death* explora **outra falha da implementação do algoritmo de desfragmentação** (i.e., montagem) de um pacote *Internet Control Message Protocol* (ICMP). O algoritmo de desfragmentação **assume que o tamanho dos pacotes não vai para além do limite máximo estipulado** (daí o cenário absurdo), e a implementação errónea **construía os vários fragmentos antes de calcular o tamanho máximo** que o pacote tinha.

O **ataque consistia em enviar um pacote ICMP demasiado grande mas muito fragmentado**, e esperar que o fragmentador o montasse antes de verificar que **já tinha escrito em zonas da memória em que não devia**.

3.5 Ataque de Inundação de Início de Ligação (SYN)
SYN Flood Attack

Uma ligação TCP tem três fases: (i) **sincronização** (ou inicialização); (ii) **comunicação** e (iii) **finalização**. Durante a sincronização, **um cliente inicia o pedido de ligação e servidor responde**, passando a respetiva máquina de estados da pilha para o estado SYN-RECV, aguardando a confirmação de estabelecimento final do cliente. **Nesse estado**, **o servidor aloca alguns recursos de memória e processamento para guardar e processar informação relativa à ligação**.

O **ataque de inundação SYN consiste em gerar e enviar para o servidor um número inusitado e contínuo de pedidos de sincronização** (provavelmente com endereço fonte falsificado ou *spoofed*). O servidor é **obrigado a guardar o estado de um grande número de ligações meio estabelecidas** e a aguardar por respostas que não vão chegar. Tem também de **lidar com as possíveis retransmissões** que o TCP define para o caso em que não há respostas. Isto tem como consequência a **negação de serviço a clientes legítimos**, já que o servidor está entupido com pedidos falsos.

Este ataque **constitui, ainda hoje, um dos problemas mais difíceis de colmatar** na área da segurança da informação, e está **na base dos grandes ataques de Negação de Serviço Distribuidos** (*Distributed Denial of Service* (DDoSs)) cometidos diariamente. Parte do problema deriva do facto de serem usadas botnets para efetuar DDoSs, sendo impossível distinguir pedidos legítmos de ilegítimos. Algumas das **possíveis soluções** a estes problemas passam por **aumentar o poder de processamento e memória do servidor**, e.g., com redundância (*Clouds*), implementar uma solução chamada SYN *cookies*, *random drop*, *firewalls* de perímetro do tipo filtro de circuitos, e Sistemas de Deteção de Intrusões baseados na Rede.

4 Erros de Realização
Implementation Errors

Um dos maiores problemas de segurança em sistemas

de informação é a exploração de **erros de realização de código** ou **implementação deficiente**. A exploração destes erros pode induzir situações de **negação de prestação de serviço** ou **situações de penetração e compromisso** do sistema. Estes erros devem-se fundamentalmente à incapacidade do programador, como humano que é, de antecipar qualquer combinação complexa de cenários anormais e, sobretudo, **ao facto de poder não dominar a linguagem que está a usar na implementação**.

No âmbito dos problemas de realização, mencionam-se normalmente dois tipos de ataque ou vulnerabilidades[4]:

1. Vulnerabilidades de transbordamento da memória (*memory overflow*);
2. Vulnerabilidades associadas a cadeias de formato.

Estas duas vulnerabilidades/ataques são brevemente descritas nas secções seguintes.

4.1 Transbordamento da Memória
Buffer Overflow

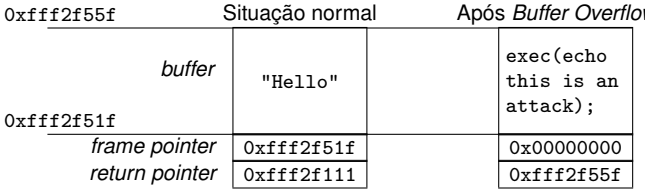
Um dos **ataques mais comuns** a código defeituoso consiste em provocar **transbordamentos de memória (*memory overflows*)**. Os problemas associados ao transbordamento da memória são **responsáveis por grande parte dos problemas de segurança em sistemas informáticos dos últimos 17 anos**. O conjunto de vulnerabilidades que proporcionam os ataques de *buffer overflow* foi, até há pouco tempo, o **mais importante e preocupante** de todas as vulnerabilidades, embora se esteja agora a **assistir a uma diminuição do risco que representa**.

Este ataque consiste em **provocar o transbordamento da memória para locais que podem alterar o fluxo do programa de modo lucrativo para o atacante**, nomeadamente para possibilitar a **execução de código malicioso**. A alteração do fluxo pode ser conseguido de duas formas básicas:

1. atribuição de **um valor errado** a determinada variável;
2. **modificação dos endereços de contexto local ou de retorno**, de modo a irem para zonas contendo código máquina malicioso.

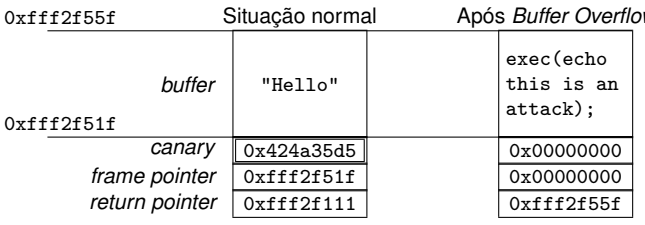
Este ataque pode acontecer em programas escritos numa **linguagem que não verifique nativamente que determinado valor pode estar a ocupar mais do que deve em memória**.

Na figura incluída em baixo, de uma forma simplista, ilustra-se o que aconteceria caso um atacante conseguisse transbordar a memória atribuída à cadeia de caracteres *buffer* na pilha do programa, que em condições normais deveria ocupar 64 bytes. No lado esquerdo da figura exemplifica-se o que poderia decorrer de uma situação normal de execução do programa, mostrando as posições de memória e o conteúdo de cada uma das variáveis na pilha. No lado direito da figura mostra-se o aspeto de um ataque bem sucedido, em que o transbordo provocou a reescrita do *frame* e do *return pointer*, sendo que este último passou a apontar para o início do *buffer*, **onde agora está código executável, injetado pelo atacante**.



Parte da solução para estes problemas consiste em **colocar canários a guardar os valores dos ponteiros**: estes canários não são mais do que valores que separam os ponteiros de fluxo das restantes variáveis do programa. **Devido às suas posições**, se alguém tentar **alterar os ponteiros**, o **canário é quase de certeza modificado**. Estes canários podem ter valores gerados por funções criptográficas de modo a serem difíceis de adivinhar por um atacante remoto sem acesso direto à memória. **Antes de se usar o ponteiro de retorno ou de contexto local, o(s) canário(s) é(são) verificado(s), e o código só é executado se os verificado canários não tiverem sido modificados**.

A figura incluída a seguir mostra a posição do canário relativamente a variáveis da pilha. Também se ilustra, do lado direito, o que aconteceria caso este tentasse fazer um transbordamento, como antes. Neste caso, o valor do canário era reescrito, situação que seria detetável visto o valor do canário ser verificado em tempo de execução.



4.2 Ataque com Cadeias de Formato
Format String Attack

O ataque mais popular que usa **dados de entrada com códigos de controlo** é conhecido por **ataque com cadeias de caracteres de formato**. De modo a perceber melhor o funcionamento e o problema que está na base deste ataque, fixemo-nos em programas escritos na **linguagem C**, que é **vulnerável a este tipo de ataques**. As funções de escrita de dados formatados em C (`printf`, `fprintf`, etc.) utilizam **um parâmetro de entrada que**

[4] Por vezes, a designação dada ao ataque é a que se usa para definir a vulnerabilidade ou vice-versa.

define o formato de saída dos dados que começa com o caracter %. Se for dada a hipótese a um **atacante de controlar a cadeia de caracteres de formato**, então este pode forçar o programa a revelar informação acerca de endereços de memória ou até a injetar código através do uso de %n.

Para perceber melhor o que aqui está escrito, considere o exemplo do diapositivo seguinte, em que um atacante consegue imprimir os endereços de memória de determinada *string*, recorrendo apenas e só à alteração da própria *string*.

```
#include <stdio.h>
#include <string.h>

int main (int argc, char **argv)
{
char buf[100];
snprintf(buf, sizeof(buf), argv[1] )
printf("Memory: %x\n", buf);
printf("Buffer size is: (%d) \nData input: %s \n",
    strlen(buf), buf);
return 0;
}
```

Considere que o código anterior é compilado e executado com um *input* **normal**:

```
$ cc teste.c
$ ./a.out Bob
```

Neste caso, obtém-se o seguinte *output*:

```
Buffer size is:  (3)
Data input:  Bob
```

Por outro lado, considere agora que o programa é executado com o *input* especialmente forjado incluido a seguir:

```
$  ./a.out "Bob %x %x"
```

Neste caso o resultado é como se mostra em baixo:

```
Buffer size is:  (14)
Data input:  Bob 0 25c86300
```

Nota: o conteúdo exposto na aula e aqui contido não é (nem deve ser considerado) suficiente para total entendimento do conteúdo programático desta unidade curricular e deve ser complementado com algum empenho e investigação pessoal.