Mestrado em Comércio Electrónico

Construindo *websites* seguros e confidenciais em ASP.NET

Trabalho apresentado para a obtenção do grau de Mestre em Comércio Electrónico

Autor
António Augusto Nunes Godinho

Orientador
Jorge Augusto Castro Neves Barbosa
Professor do Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra

Coimbra, Fevereiro, 2017

AGRADECIMENTOS

À minha mulher,
pela paciência pelo meu/nosso tempo despendido em aulas, em programação, a escrever esta dissertação, e por me incentivar e apoiar ao longo de todo o percurso.
Aos meus pais,
que espero que se sintam orgulhosos.
À minha família e amigos,
obrigado por aturarem o meu feitio.

RESUMO

O desenvolvimento de *websites* exige que aqueles que os desenvolvem, tenham de implementar medidas necessárias para proteção dos seus utilizadores. Esta proteção implica medidas de segurança que abrangem:

- *O conteúdo das páginas entregues ao cliente;*
- *A informação que transite entre as várias páginas;*
- *O conteúdo dos parâmetros enviados através de query string;*
- *A ocultação do histórico de navegação do utilizador.*

Assim, descreve-se a implementação de várias versões de um mesmo *website*, utilizando diferentes medidas de segurança, testando as suas implementações, descrevendo o modo como estas poderão ser implementadas na plataforma ASP.Net e quais as limitações da mesma para a sua implementação. Descrevem-se os detalhes técnicos desenvolvidos para cada uma destas implementações.

PALAVRAS CHAVE

ASP.NET, encriptação de *query string*, *URL ReWrite*, *Routing*, privacidade, ocultação da navegação do utilizador

ABSTRACT

The development of websites requires that those who develop, to have to implement the necessary measures for the protection of its users. This protection requires security measures to protect:

- The content of the pages delivered to the client;
- The information transiting between the various pages;
- The content of the parameters sent through query string;
- Concealment of the user's browsing history.

It describes the implementation of several versions of the same website, using various security measures, testing their implementations and describing how these will be implemented in ASP.Net platform and what limitations found. It describes the technical details developed for each of these implementations.

KEYWORDS

ASP.NET, query string encryption, URL ReWrite, Routing, privacy, concealment of user navigation

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ABREVIATURAS/ACRÓNIMOS

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
CSS	Cascading Style Sheets
DDOS	Distributed Denial Of Service attack
DES	Data Encryption Standard
DLL	Dynamic-link library
DOS	Denial Of Service attack
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IIS	Internet Information Services
ISAPI	Internet Server Application Programming Interface
JQUERY	JavaScript Query Library
MVC	Model–view–controller
NIST	National Institute of Standards and Technology
PC	Personal Computer
PHP	PHP: Hypertext Preprocessor
PKI	Public Key Infrastructure
POODLE	Padding Oracle On Downgraded Legacy Encryption
RC2/3/4/5/6	Rivest Cipher 2, 3, 4, 5 ou 6
RFC	Request for Comments
SEO	Search engine optimization
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3SVC	IIS World Wide Web Publishing Service
WAS	IIS Windows Process Activation Service
XML	eXtensible Markup Language

1. Introdução

Esta dissertação do Mestrado em Comércio Eletrónico, mestrado com uma forte componente assente na plataforma da Microsoft, a ASP.NET, tenta descrever um conjunto de orientações para a implementação de um *website* em ASP.NET. Várias versões do mesmo *website*, de uma empresa fictícia de material informático chamada "*BeGamer*", foram desenvolvidas para esta dissertação. Essas versões tentam demonstrar, a utilização de diferentes metodologias de proteção e ocultação da informação.

1.1. Contexto

Nos últimos anos o volume de transações de comércio eletrónico cresceu de uma forma exponencial. Este crescimento reflete-se quer em volume, quer em número de operações efetuadas. De facto, hoje em dia, quase todos os tipos de prestação de serviços implementam soluções que permitem efetuar as suas operações *online*. Este crescimento é apenas perturbado devido ao sentimento de insegurança por parte dos utilizadores em relação a efetuar operações de comércio eletrónico.

1.2. Definição do problema

Nos dias que correm, são recorrentes os ataques na *internet* e, em especial, a *websites* de *e-commerce*. Estes ataques têm permitido a obtenção ilícita de dados privados, sejam eles dados de cartões de crédito, dados de identificação dos clientes ou credenciais de acesso aos referidos *websites*, ou dados que violam a privacidade dos mesmos. Existem perdas monetárias diretas devido a estas fraudes e gera-se um forte sentimento de insegurança devido às notícias sobre este género de ataques.

De facto, existe um problema muito grave, em relação ao fato de muitos dos utilizadores usarem as mesmas credenciais em diversos *websites* ou serviços. Embora este tipo de conduta, de utilização das mesmas credenciais em múltiplos serviços, seja uma quebra de segurança (algo que poderá ser do senso comum), com a dispersão existente de serviços por múltiplas plataformas/*websites*, os diferentes sistemas de autenticação, a adoção deste procedimento facilita aos utilizadores recordar os seus dados de acesso. Porém, este tipo de comportamento permite que credenciais usurpadas através de um *website*, devido a falhas de segurança (seja por falhas de implementação ou por falhas tecnológicas), possam ser utilizadas com sucesso noutros *websites* seguros, apesar de estarem bem implementados.

Deste modo, é comprometida a confiança dos utilizadores de forma generalizada, em todo o tipo de serviços mesmo aqueles que supostamente são seguros.

É neste meio de permanente desconfiança, em que a segurança dos *websites* é constantemente colocada à prova, que aqueles que desenvolvem *websites* de *e-commerce* têm a obrigação e a responsabilidade de criarem *websites* seguros, fomentando deste modo a confiança dos utilizadores e contribuindo para o crescimento deste tipo de serviços.

Existem diversas metodologias que podem ser utilizadas de modo a criar *websites* seguros. Estas metodologias podem ser combinadas de modo a elevar a segurança destes *websites*. É necessário procurar formas de manter a informação dos seus clientes confidencial e assegurar a proteção e a confidencialidade dos seus dados.

1.3. Objetivos das implementações

Esta dissertação focou-se na implementação de várias técnicas de segurança, nomeadamente na proteção da informação que transita entre páginas, na navegação de acesso aos produtos disponíveis no *website* e no modo como os dados dos utilizadores são acedidos, podendo estas técnicas servir de guia, mesmo para programadores que transitem de outras linguagens *server side* diferentes de ASP.NET, cujas implementações diferem da plataforma da Microsoft. Com este fim, foram desenvolvidas sete versões do mesmo *website*, com diferentes implementações, de modo a testar a sua validade.

Cobrir a fundo os algoritmos matemáticos existentes, os diversos protocolos de encriptação utilizados na *internet*, especialmente para a *web* e suportados pela plataforma ASP.NET, seria demasiado extenso e muito para além do objetivo desta dissertação. Serão apenas abordados os detalhes da encriptação utilizados nas diversas implementações.

1.4. Estrutura do documento

Numa primeira parte deste documento (capítulos 2 a 4), são descritas as várias técnicas e mecanismos utilizados, bem como o seu funcionamento e integração na plataforma .NET.

Numa segunda parte (capítulo 6), descrevem-se as várias implementações exemplificativas, os seus objetivos, as dificuldades encontradas e, no caso de existirem, as soluções utilizadas para ultrapassar os problemas encontrados.

2. Funcionalidades/mecanismos utilizados

A construção e implementação de *websites* em ASP.NET, utilizadas para exemplificar as soluções para o desenvolvimento desta dissertação, pressupõem um conjunto de funcionalidades e de medidas de segurança. Este conjunto de medidas inclui instalação e configuração do próprio servidor *web* (*IIS*) e partes programáticas.

2.1. HTTPS – Hyper Text Transfer Protocol Secure

O *HTTPS* é uma implementação do protocolo *HTTP* sobre uma camada adicional de segurança que utiliza o protocolo *SSL/TLS*. Essa camada adicional permite que os dados sejam transmitidos por meio de uma ligação encriptada e em que é verificada a autenticidade do servidor e do cliente por meio de certificados digitais. O porto *TCP* utilizado para o protocolo *HTTPS* é o 443 (Wikipedia, 2015d).

No protocolo *HTTP*, que utiliza o porto *TCP* 80, a transmissão de informação é efetuada em texto simples. Com a utilização de ferramentas de análise de tráfego em redes informáticas, é possível obter os endereços acedidos por um utilizador, o conteúdo da informação transmitida pelo servidor e os dados submetidos em formulários nas respetivas páginas. Mais concretamente, no caso de páginas com formulários de autenticação, é possível desta forma serem obtidas as credenciais que são submetidas nas caixas de texto da página. Para além disto, é também possível obter o conteúdo *HTML*, *CSS*, imagens, entre outros.

Na utilização deste protocolo, toda a informação trocada entre cliente e servidor é transparente, podendo ser fácil violar a sua confidencialidade ao ser acedida por terceiros, através de simples ferramentas e sem serem necessários conhecimentos informáticos profundos.

Torna-se, pois, imperativo a exigência de utilização do protocolo *HTTPS* em detrimento do *HTTP*, caso contrário quaisquer soluções de segurança estão fragilizadas à partida.

2.1.1. Exigir todos os pedidos através do *HTTPS* no *IIS*

Para o suporte do protocolo *HTTPS* no *IIS* (*Internet Information Services*, o servidor *web* da Microsoft) é necessária a existência de um certificado válido. Este pormenor, tal como a instalação deste no *IIS*, os vários tipos de certificado disponíveis, entre outros, estão para além do tema desta dissertação. Para os efeitos desta dissertação, assumiu-se que existe um certificado válido, que o servidor está configurado para suportar *HTTPS* no porto 443 e que, como tal, todas as ligações entre cliente e servidor são seguras e encriptadas.

Existem duas formas distintas de se forçar a utilização do protocolo *HTTPS* no *IIS* e emASP.NET. Primeiro, através do mecanismo *URL ReWrite* e segundo, programaticamente.

2.1.2. Segurança do *HTTPS*

Nesta dissertação, alguns dos exemplos visam proteger a informação e a privacidade do utilizador, partindo do pressuposto de que o túnel *SSL/TLS* foi comprometido, sendo possível aceder à informação trocada.

O protocolo *HTTPS* suporta vários algoritmos de encriptação, tais como TLS v1, v1.1 e v1.2 e SSL v1, v2 e v3. É da responsabilidade dos administradores de sistemas verificar os avisos de segurança, desativando os algoritmos de encriptação em que tenham sido encontradas fragilidades. Um exemplo deste tipo de problema ocorreu durante o ano de 2014, quando foram descobertas vulnerabilidades no algoritmo de encriptação SSL3 (Team, 2014), que é comprometido através de uma técnica chamada *POODLE*.

2.2. *URL ReWrite*

O *URL ReWrite* é um mecanismo que permite intercetar antecipadamente um pedido de um cliente e o reencaminhar para um *URL* diferente (Mitchell, 2004b).

Este mecanismo é utilizado principalmente devido a dois motivos:

1. *Primeiro, aceder a páginas que mostram resultados de pedidos a base de dados, com utilização de valores de query string (Mitchell, 2004b). Isto é, um valor da query string é mascarado, sendo incorporado no endereço acedido. Um exemplo deste tipo de implementação acontece quando o utilizador acede ao endereço https://v1.mce-godinho.isec.pt/Categorias/Teclados, quando fisicamente o URL existente é a http://v1.mce-godinho.isec.pt/categorias.aspx?id=3.*
   Deste modo, torna-se também possível proteger a utilização de passagem de valores por *query string*, dificultando a poluição dos parâmetros e tentativas de ataques à base de dados do *website*.
2. *Segundo, para tornar o URL mais amigável, quer para os utilizadores como também para os motores de pesquisa, SEO (Search Engine Optimization). Com o endereço http://www.mybanksite.com/aboutus.html, podemos entregar ao utilizador a página page1.html, ou utilizar o endereço de acesso http://www.mybanksite.com/about-us/ para mostrar a mesma página page1.html (Tero, 2011). Para além de tornar o URL interpretável, por ser constituído por um conjunto de palavras que têm um determinado significado, este mecanismo, desde que seja bem utilizado, permite que os motores de pesquisa, através da utilização por palavras-chave, possam indexar com mais exatidão os conteúdos a que se refere cada uma das várias páginas de um website.*

Este mecanismo pode ser implementado através de um módulo do *IIS*, sendo esta a forma mais simples de o implementar. Nos servidores *web IIS* ou *Apache* esta funcionalidade é instalada e configurada de forma idêntica. No *Apache* é apenas necessário descomentar uma linha na configuração. Em *IIS* utilizando a ferramenta gratuita *Web Platform Installer (Web PI)*, que permite entre outras coisas instalar módulos no *IIS*, sendo apenas necessário procurar pelo módulo que se deseja instalar.

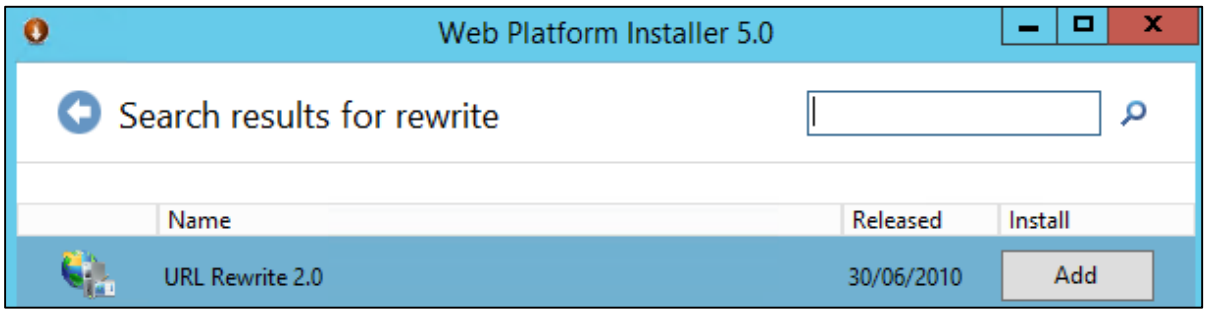

Figura 1 - Instalação do módulo URL Rewrite 2.0

2.2.1. *URL ReWrite* através do *IIS*

Como foi referido no ponto 2.2, é necessário que o servidor suporte o *URL ReWrite*; como tal este módulo deve estar instalado no *IIS*.

É possível adicionar reencaminhamentos de dois tipos: estáticos e dinâmicos.

- *O encaminhamento dinâmico permite a utilização de regras (através de expressões regulares) que funcionam como que um filtro, de modo a que uma única regra se possa aplicar a um grande conjunto de casos, tornando essas regras ideais para se aplicar neste tipo de redirecionamentos. A utilização de expressões regulares, permite identificar uma determinada string ou um padrão, sendo frequentemente utilizadas em ASP.NET, para validar o formato do conteúdo introduzido em caixas de texto, por exemplo na introdução de matrículas de carros.*
- *O encaminhamento estático é definido por dois endereços, um público a que corresponde, internamente, um outro endereço.*

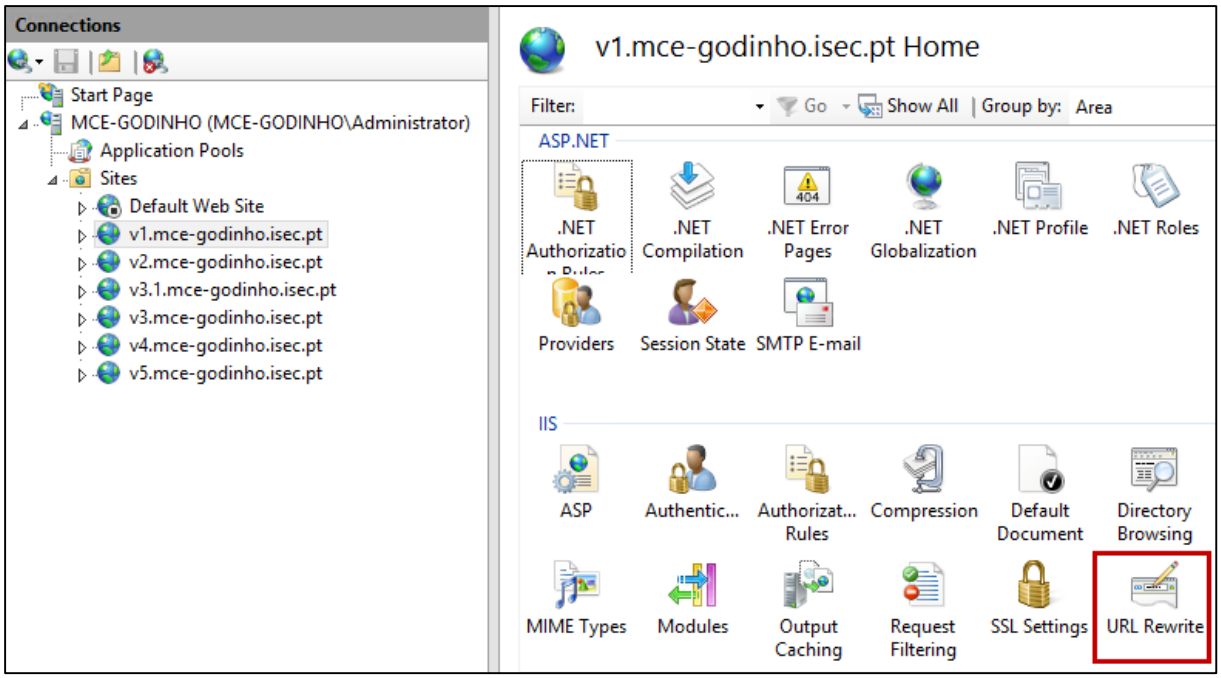

Figura 2 - Consola de administração do IIS

As regras são inseridas através da consola de gestão do *IIS*, acedendo à opção *URL Rewrite*, ou através do ficheiro *web.config* (ficheiro de configuração de um *website* ou plataforma *web*, referido em maior detalhe no ponto 2.4).

Por exemplo, de modo a forçar a utilização de *HTTPS* e redirecionar os pedidos *HTTP* para *HTTPS*, a seguinte regra pode ser utilizada:

```
<rewrite>
    <rules>
        <rule name="Redirect to HTTPs" enabled="true" stopProcessing="true">
            <match url="(.*)" />
            <conditions>
                <add input="{HTTPS}" pattern="OFF" />
            </conditions>
            <action type="Redirect" redirectType="Permanent" url="https://{HTTP_HOST}/{R:1}" />
        </rule>
    </rules>
</rewrite>
```

2.2.2. *URL ReWrite* em ASP.NET

É possível implementar o *URL ReWrite* programaticamente de duas formas distintas. De forma estática, através do ficheiro *Global.asax*. No seguinte exemplo, utilizado em testes nesta dissertação, é efetuado o *URL ReWrite* da página */Login.aspx*, mostrando o conteúdo da página */Autenticacao/Login.aspx*.

```
void Application_BeginRequest(object sender, EventArgs e)
    {
        if (!(Request.Url.AbsolutePath.EndsWith("/Login.aspx",
StringComparison.InvariantCultureIgnoreCase)))
        {
            System.Web.HttpContext.Current.RewritePath(("/Autenticacao/Login.aspx"));
        }
    }
```

O *URL ReWrite* não é possível (como se verá posteriormente) no ficheiro de código (*code behind*) de uma aplicação ASP.NET., tendo em conta o seu ciclo de vida. No início do processamento do pedido por parte do cliente, os cabeçalhos (*headers*) são enviados para o cliente com o respetivo código *HTTP* (200, 302, entre outros). O processamento da página começa quando já se ultrapassou o último evento que permitiria o *URL ReWrite*.

É necessário perceber o *pipeline* do *IIS* (modo como o *IIS* processa os pedidos dos utilizadores) e a sequência de eventos de uma aplicação em ASP.NET, para se entender porque, após um determinado evento, se tornará impossível o *URL ReWrite*.

Existe uma sequência no conjunto de eventos que são processados pela classe *HTTPApplication*. O número de eventos varia com a versão do *IIS*, existindo alguns que requerem uma versão específica da *Framework .NET*. No *IIS* nas versões 7 e 8, existem os

eventos a seguir apresentados e o *URL ReWrite* é processado do seguinte modo (Microsoft, 2015a); (Mishra, 2010); (Yakushev, 2008):

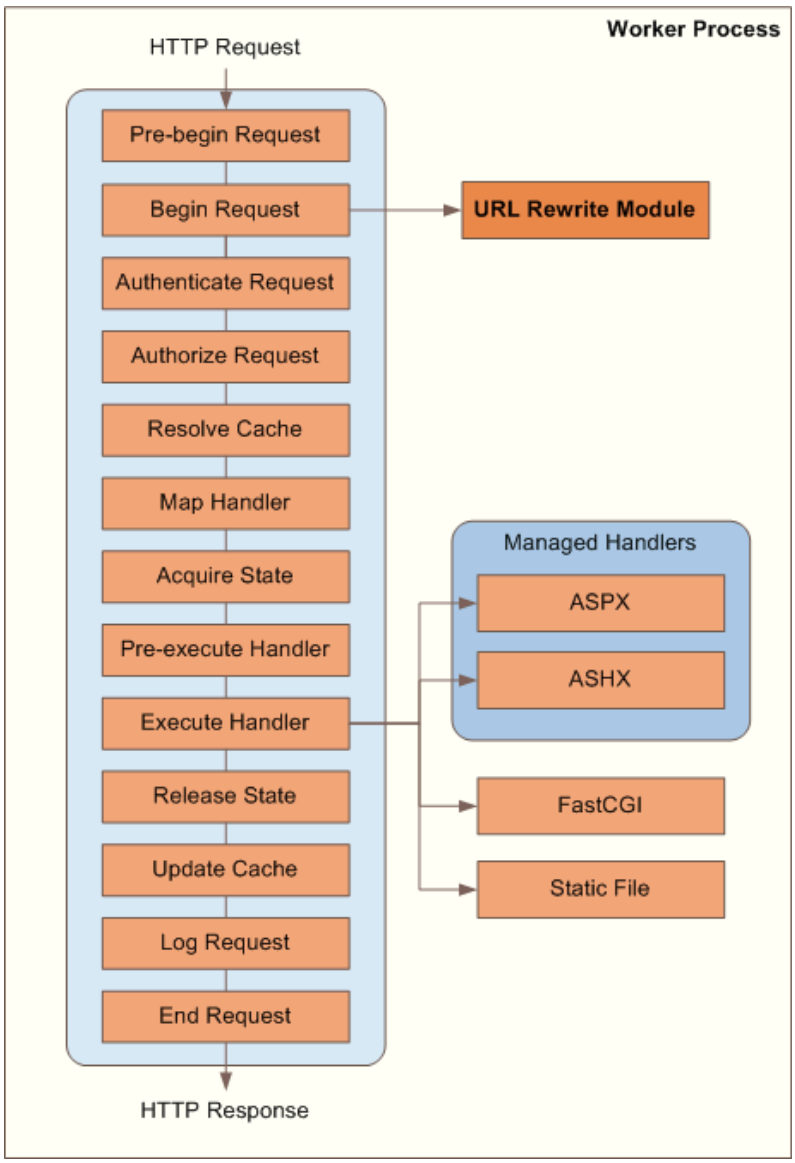

Figura 3 - URL ReWrite HTTP Application pipeline

2.3. Encriptação

Para efeito desta dissertação, utilizaremos encriptação através de chaves simétricas, isto é, uma chave que permite encriptar/desencriptar conteúdo. A utilização de um conjunto de chaves diferentes (assimétricas), por exemplo um par de chaves pública e privada, apenas tornaria a parte da encriptação mais complexa, não sendo este o foco desta dissertação.

A plataforma ASP.NET tem acompanhado a evolução dos algoritmos modernos de encriptação, incorporando o seu suporte na *framework*, e descontinuando aqueles em que foram descobertas fragilidades.

O algoritmo DES continua a ser utilizado; apesar de ser um algoritmo com mais de 30 anos (Wikipedia, 2015b). A pequena dimensão da sua chave em conjunto com o seu algoritmo, foi demonstrado que com ataques do tipo *brute-force* é fácil ultrapassar o seu processo de encriptação, sendo portanto, desaconselhada a sua utilização (Wikipedia, 2015c). O algoritmo 3DES utiliza uma implementação tripla do DES (daí o seu nome); no entanto, é um processo algo lento e que consome muitos recursos.

Por recomendação da Microsoft, o protocolo que deve ser utilizado na plataforma ASP.NET é o *AES* (*Advanced Encryption Standard*, também conhecido por Rijndael). O *AES* é o algoritmo sucessor do *DES*, que utiliza chaves até 256 bits, sendo bastante eficiente o seu suporte quer por *software* quer por *hardware*. Foi selecionado entre várias hipóteses, através de um concurso do NIST (*National Institute of Standards and Technology* dos EUA). Segundo o NIST, ele combina as características de segurança, desempenho, facilidade de implementação e, exigindo flexibilidade. Com alta resistência a ataques como *power attack* e *timing attack* e, exigindo pouca memória, é adequado para dispositivos móveis (Wikipedia, 2015a).

Para a implementação desta dissertação, foi desenvolvida uma classe com o nome MestradoEncriptacao, que possui métodos que permitem gerar chaves, encriptar e desencriptar *strings* de texto com essa mesma chave, utilizando o protocolo *AES*.

2.4. Encriptação de configurações

Em *ASP.NET*, as configurações de uma determinada aplicação *web*, são definidas no ficheiro *web.config*. Este ficheiro de configuração em formato *XML* é dividido em várias partes, desde referências das bibliotecas (*DLL's*) a serem carregadas, às credenciais de autenticação para acesso a bases de dados. Estes ficheiros, com extensão *.config*, não são servidos aos clientes pelo *IIS*, isto é, se um cliente tentar aceder a diretamente ao endereço de um destes ficheiros, o cliente receberá o erro 404 de ficheiro ou diretoria não encontrada. É necessário ter em conta que estes são ficheiros de texto e como tal legíveis, em caso de alguma falha de segurança do servidor podem serem expostos ao exterior e o seu conteúdo colocado a descoberto.

Como medida adicional de segurança é possível encriptar partes específicas do ficheiro *web.config*, neste caso específico a parte das credenciais de acesso às bases de dados. Nesta dissertação, a secção de *connectionStrings* é definida do seguinte modo, sendo possível obter o endereço do servidor de base de dados, nome da instância, e respetivo *login* e *password*:

```
<connectionStrings>
    <add name="BeGamerConnectionString" connectionString="Data Source=MCE-GODINHO\GODINHODB;Initial
Catalog=BeGamer;Persist Security Info=True;User ID=sa;Password=G0d1nho." providerName="System.Data.SqlClient" />
</connectionStrings>
```

A Microsoft disponibiliza a ferramenta de registo do *IIS* (Aspnet_regiis.exe), que permite entre outras funções, encriptar ou desencriptar secções de configuração da configuração. Esta ferramenta permite a utilização de vários parâmetros, que podem especificar

o tipo de provedor de encriptação, a *pool* da aplicação *web*, além de outros. (Microsoft, 2014c). Dependendo da configuração, do número de *websites* e da versão .NET que se encontra no servidor/*website*, pode-se listar as várias *pools* correndo o seguinte comando numa linha de comandos/*power Shell*:

```
PS C:\Users\Administrator> C:\Windows\System32\inetsrv\appcmd list apppool
APPPOOL "DefaultAppPool" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "Classic .NET AppPool" (MgdVersion:v2.0,MgdMode:Classic,state:Started)
APPPOOL ".NET v2.0 Classic" (MgdVersion:v2.0,MgdMode:Classic,state:Started)
APPPOOL ".NET v2.0" (MgdVersion:v2.0,MgdMode:Integrated,state:Started)
APPPOOL ".NET v4.5 Classic" (MgdVersion:v4.0,MgdMode:Classic,state:Started)
APPPOOL ".NET v4.5" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v1.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v2.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v3.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v4.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v3.1.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v5.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v6.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
APPPOOL "v7.mce-godinho.isec.pt" (MgdVersion:v4.0,MgdMode:Integrated,state:Started)
```

No exemplo acima exposto, é possível verificar a existência de várias *pools*, para cada uma das versões implementadas para o *website*. Se o objetivo for verificar a secção de *connectionStrings* do *website* da versão 3 (neste caso a v3.1), o comando a utilizar será o seguinte:

```
PS C:\Users\Administrator>
C:\Windows\system32>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_regiis -pef
"connectionStrings" C:\inetpub\mestrado\Dropbox\Tese.Mestrado\Mest.Site.V3.1
Microsoft (R) ASP.NET RegIIS version 4.0.30319.34209
Administration utility to install and uninstall ASP.NET on the local machine.
Copyright (C) Microsoft Corporation.  All rights reserved.
Encrypting configuration section...
Succeeded!
```

O resultado final desta operação de encriptação é o seguinte:

```
<connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>Rsa Key</KeyName>
          </KeyInfo>
          <CipherData>

<CipherValue>dJ7aKnTG/5FtxTNtLgAI8KfSXYQVRKwba/G3lS5Y5HHf8zWLMqCjCFSNetjyWz7n8IIsJsuXcWdZH7N9wKhS3pK
```

```
88+geaGaH4PZplEDIQVFi/hYPDclRpNJ79n9tZPQvPBXhUmg+4RhIe+D+I5pQXhwu1LaoKIUYUh9h7VJmvu1UOmTZ+S2o+3JcHE1
Pv5VfAFD+F+8HZF5XOxMAkOI04GEyvSeUHJaH9P1pF5MhQclDLesb9E0X1Iz1M82FZmhysZ5qu3ihaznQktMl3KCbziqMvLUh16c
vrw4i6I5ow5M7TbPcwez83RHWqtTFIAahAEvVP5iIFRl0C9PqnnpQ2A==</CipherValue>
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>

<CipherValue>qdFY4CBy/GcOrYMfC9421uXm7lPxv8a0i8FbzbmfJjCeoxgrVsT19/3IWj/YagdCe/j/7AeK9v2mhvpTC0pXJc4
w8zO05EjgPvd3T6Sdz1nH4SA2q4d0w9oDZcNrKeVr18KfwSlROWcuer9YQu+FXW3nkUQpwaS8mDJJaQIwFNmbQ/giNut5ufoJPKX
itWh4HmZwdBcrB4a435p1krCli71DwHZNrw0QloV7ptOMfHmAKAZg816j6shYKN59qVow87JkoMdHcxYI8yXUg75FhGDwE1EhNhu
9Pwb7JZzbCChouT7gB23jv4UcN2QuR6gke/rHjelPGGxqmjKPeEwBKQjnrvAsr3AD</CipherValue>
      </CipherData>
    </EncryptedData>
  </connectionStrings>
```

Este é um procedimento muito pouco utilizado, sendo, no entanto, relativamente fácil de implementar; deveria ser um procedimento obrigatório para servidores que se encontrem em produção.

É necessário ter em mente que a encriptação é efetuada utilizando uma chave gerada e existente no servidor. O que quer dizer que a utilização desta ferramenta para desencriptar a informação, necessita de ser efetuada no mesmo servidor onde foi efetuada a encriptação (de modo a ser utilizada a mesma chave), ou exportar a chave para permitir a sua utilização noutro servidor (Microsoft, 2014c).

3. Fluxo de informação e informação do utilizador

As aplicações *web* utilizam o protocolo *HTTP* na transmissão dos pedidos e respostas entre o *browser* e o servidor *web*. O *HTTP* é um protocolo *stateless*, ou seja, um protocolo sem estado, o que na prática significa que não é incluído o estado entre os pedidos e as respostas. Assim, cada pedido efetuado por parte de um cliente é tratado como se fosse um novo, um pedido independente, não sendo mantida nenhuma relação de histórico entre os vários pedidos que compõem a navegação pelo *website* (Oneda, 2010).

O comportamento tradicional das aplicações implica que, após um *PostBack*, os campos do formulário de uma página são limpos, apagando desta forma o seu conteúdo. A plataforma ASP.NET fornece várias soluções de modo a resolver este problema, através da gestão de estado (Bhimani, 2015).

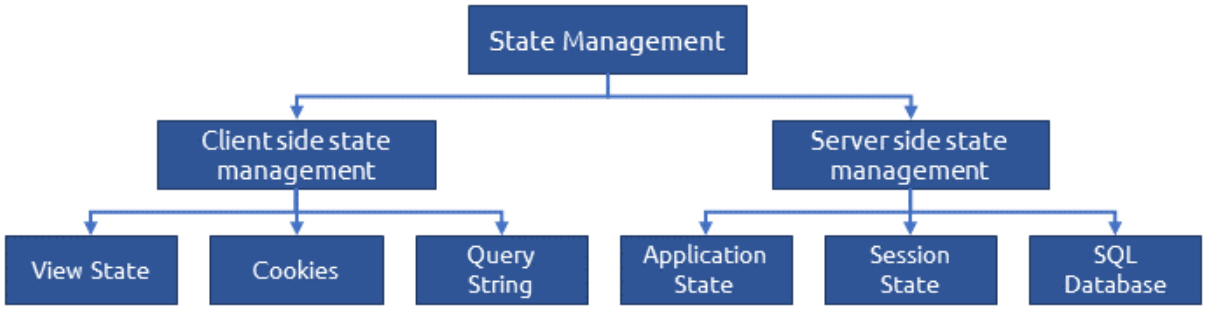

Figura 4 - State Management em ASP.NET

Do lado do cliente, a parte de *ViewState* é fácil de verificar, analisando o código *HTML* de uma página em ASP; podemos observar os seguintes campos:

```
<div class="aspNetHidden">
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="5chRA6w03gBLDGO7LbmHWWDjPoyif68rvt8VLWVsLm6goTX6giGlQ2emEMDjBrl3yvb6ock4KoyKI5j8W+mi1t9lHEkN8UtWif41UW0I9/7
g3w99Ior8M+SS4B1iV7ETwrV49ZmaiTzHqLIzQxNk4+gSRyGZPSj4phTOqiNftZZ5I7VBvw/9e0Tp6KJzOmTssSmgQFvi10xFfA5fZjIivQ1+vkL7L
GVu7RdQk+m+AnPe58OeZjxd3NHfUUxYmb3r3/UzAPM9NYk/uBYuqvHVmTsoZihUNfUBUCzkJmW8QOjuMQHyxYNh2DjAhX9JUKkdz8JRJcEQFjPuGed
7RIr/x/dVbrwPxc0hmBXTOlIF4tRpTKZidAgzPf/B58MJvwAFTn5iLl7fxNJ0ZOLDEuORZxjKnsXFOxLrW3h3uFxi52drjBUhtBhUELmgk+v7XJxeu
GOni8JVu3SWHQ5K4+qXinL3J22cFnr601Uojn3r5jxsHzKhQ0gV7adCIm+P9NN+celhfXXENUBrTCyVxhe5sONDemBIClpZzpvRmlHkgEacWQ4ejrq
my6w5GSdcnm2z1UGmtjr5OI6aQSiadqaueHJo9Mlm0D7fDUChz6bgb+QD5/z6Yr/YP1QSC9URsevz+TQak3nXSxeDi9wuZ7+fWL7sJTO6cVFinsHlh
631mtqqJ8j5idf4/3HzboIbFwHaY53bfW1BGnMDqiAag18FVv4+nF/XzEAXaGvoOi5CXon5EDs+4eYQi4b7YWKNXIhhdQHczYf1jpbpf3LRQb5OCcz
nPaLZxB4Lvc9FDeTQWc75DMv4cpTdFPJWVZkLTejQKnMhv65NqqC5dz7v5nifOZo12hZxQsSB6dwKfpLTTYPxLML2BNwcCR6AxSi7BG03o9LP3JENs
OEvubzC7+nDq6Pyim1mIpCGGeWNIuvu7YK2qFtT6OUpKyB3naDmxZz0yCdaUR+PBYO+VF8xBmY7oApv/pQYNAsNu9exGkcqkCgS59AKxCzUsbgmjmG
j/DocnZklfwkkUMYH4R8nWdOQv/ZsOMnpflDR5aLAstrcwnClkEEqDJm98NVJb3OiTqTX/z9L3+EerAox0w36Jz+zrjB1sB/r8LhtCEIH6MQuB6jsD
FkZStbpdr4s1GMa2xNbtHs8aZ3llQJLB/K+afa/XPZmeo54P5fSNsDLC/S4q24g3sbARqc9IQ9+r5TwfjLlq3ZbBLnSqcpX35YhBx2bmoXoCa3p7PO
naK0TQEcagzJRJxc+++ubY+ab96X8lDDXd9EIfE+LCCMGYqFBTC+Dh12lUb4DEj+iA+bDo8C0oAOaRMc3l01X8kwkC9FYtNu7VLA+UnX7TX7jzCovl
pXk2Dzvt4sFqIOw9ni8U6RtRe39x17KftXrG/BRviQF0BodLe1gU/crEyWhUS2gEDDysJPfJURgp7AO1v1wzu7Eupy8KGT6PzxaCQQsk39vo7UgNMd
aub2Xu3k+qxKR/9/Dje7r1BUWG1bAMUMz/d/Hx1kcroLpE+IxJs2TXiv5+rXuv9qndviEkoptRaMiIFJkpp2oMBaDWWR7fKGZsAWz3Xn/WFW9ejYUX
WYWSyVUjcMOr57a2Sz5GkUwt/Gd/Ee8DosAdNuDNNnsnidYfnB4XAZl6eYgIcXsKnXAwwL3Zj8k6TCtg/SGgO7XgLP3crq8F4LssuCIa6XO8lt7Hrc
kjyHmIdgfY0BPgq8ucUEiIN3BcM0hf3Got9sUbQLpwpdsDwh3VeWqI7hM8rP5E7NuUrgUA31DYso5Wb5VqL3jafNsFo1sNuZ5lSy1IKszpgPNhZELy
KSgHowLN+pVJ1UI0iSbTO7aOk9Ww3meWpkgmH33TjDQhGEdwbM+7o5Xo
```

Não foram utilizados *cookies* nas várias implementações desta dissertação. Os *cookies*, são pequenos ficheiros com dados que são guardados do lado do cliente. A aplicação *web* indica ao *browser* onde e como os guardar, sendo posteriormente lidos. Como serão guardados do lado

do cliente, torna-se muito fácil alterá-los, tornando então necessário encriptá-los e/ou utilizar funções de *hash* para verificar a sua integridade. Resumindo, seria introduzir potenciais falhas e vulnerabilidades, quando já existe um conjunto de potenciais problemas de segurança em que nos devemos focar. O *standard* para guardar dados de um utilizador, nesta dissertação, serão as sessões, que são guardadas do lado do servidor e evitam os problemas acima descritos.

Utilizaremos três formas de passagem de informação entre páginas ou ao longo do nosso *website* (Microsoft, 2008):

- *Cross-site POST.*
- *Querystring – a query string é um modelo clássico de manutenção de estado da página. É um conjunto de pares/valores que são anexados ao URL (Camargo, 2013);*
- *Sessões.*

3.1. *Cross-site POST*

Este mecanismo é uma das bases do funcionamento do protocolo *HTTP*, permitindo a submissão de formulários e envio da informação introduzida para o servidor, funcionando do mesmo modo em todas as linguagens *server side*. Permite indicar ao servidor que existem dados anexados no corpo da resposta que é enviada ao servidor, e que este a deve aceitar (por norma as mensagens enviadas ao servidor são de pedidos de páginas de um determinado endereço) (Wikipedia, 2015e).

Quando o *browser* envia um pedido POST de um elemento *web form*, o tipo de conteúdo é "*application/x-www-form-urlencoded*". O formato para codificação utiliza pares chave-valor, permitindo a utilização de chaves duplicadas. Cada par chave-valor é separado pelo caracter *'&'* e cada chave e o seu respetivo valor são separados pelo caracter *'='*. Pares de chaves e valores são ligados através do caracter *'+'* e finalmente é utilizada a codificação do *URL* para todos os caracteres não-alfanuméricos. Por exemplo, num formulário temos na esquerda (antes do caracter *':'* o nome do campo do formulário, algo do género *<input type="text" id="Nome">*) e os valores introduzido nos campos do formulário.

```
Nome: Jonathan Doe
Idade: 23
Fórmula: a + b == 13%!
```

Estes valores serão codificados no corpo da mensagem como:

```
Nome=Jonathan+Doe&Age=23&Formula=a+%2B+b+%3D%3D+13%25%21
```

Este mecanismo é bastante simples e não fornece a proteção necessária à informação, mas o funcionamento da *web* assenta neste mecanismo. É necessário perceber que o protocolo

HTTP 1.1, neste campo em particular, em nada mudou desde meados dos anos 90 (IETF, 2014). O protocolo *HTTPS* é idêntico ao *HTTP*, no que diz respeito à utilização e tratamento de dados submetidos através de formulários; no entanto, a camada de encriptação permite, para além de outras medidas, encriptar o corpo da mensagem e os valores submetidos por formulários, permitindo, deste modo, proteger os dados enviados.

3.2. *Query string*

A *query string* faz parte do endereço (*URL)* de uma página, não fazendo parte do caminho físico de acesso à página, isto é, sem esta secção do *URL* a página poderá a ser acedida com sucesso. Este conjunto de caracteres é constituído do mesmo modo que os *Posts*, descrito no ponto 3.1.

No exemplo do ponto 3.1 os conjuntos de pares chave-valor foram colocados no corpo da mensagem de resposta para o servidor. Utilizando a *query string*, ao *URL* base é adicionado o caracter *'?'* e complementamos o *URL* com os pares chave-valor, sendo cada um separado pelo caracter *'&'* e cada chave e o seu respetivo valor são separados pelo caracter *'='*.

```
http://v1.mce-godinho.isec.pt/SiteGestao/Produtos/EditarProduto.aspx?id=8&user=godinho
```

Como se pode verificar, utilizando *query string* é exposta no *URL* a informação que está a ser passada para o servidor. Para além da exposição da informação, a utilização deste tipo de funcionalidade requer cuidados especiais, sendo suscetível a ataques do tipo *DOS*, *HTTP Pollution* ou *SQL injection*.

3.3. Ataque DOS por *query string*

A exposição da *query string* pode ser utilizada para a introdução de valores incorretos, num ataque de *"DOS"* (*Denial Of Service*). Potenciais atacantes criam *scripts* que efetuam pedidos com o *URL* com valores na *query string*, que podem ser válidos ou inválidos, com o objetivo de efetuar um grande número de pedidos ao *website* e podendo, por exemplo, esgotar quer o limite máximo de ligações permitidas do servidor, quer os recursos dos servidores remotos de bases de dados. Este tipo de ataques costumam ser bem-sucedidos quando coordenados e direcionados de várias fontes para um mesmo servidor. O melhor exemplo de um ataque bem-sucedido deste tipo foi efetuado ao próprio Google (D'Mello, 2014). Quando se efetua uma pesquisa no Google, por exemplo "ataques DDOS google", o *URL* resultante é esta:

```
https://www.google.pt/?gws_rd=ssl#q=ataques+DDOS+google
```

Quando milhões de clientes efetuam múltiplas ligações simultâneas e com uma cadência com um intervalo de milissegundos, o serviço pode simplesmente deixar de funcionar (D'Mello, 2014).

3.3.1. Poluição da *query string* - *HTTP Pollution*

A técnica *HTTP Pollution* é utilizada por atacantes, fornecendo vários parâmetros na *query string* com o mesmo nome, causando a aplicação *web* que interpreta os seus valores de forma imprevista. Ao tentar explorar estes efeitos, um *hacker* tenta ignorar a validação dos parâmetros de entrada, provocando erros na aplicação ou modificando os valores de variáveis internas (OWASP, 2014).

Os vários *Request For Comments* (RFC – normas dos protocolos utilizados na *internet* entre outras) relativos ao protocolo *HTTP* não incluem o modo de interpretar vários parâmetros de entrada com o mesmo nome.

O facto de não existir uma norma que explique como lidar com estas situações não é, necessariamente, uma vulnerabilidade. No entanto, se o programador não estiver ciente do problema, a presença de parâmetros duplicados pode produzir um comportamento anómalo nos pedidos ao servidor, podendo potencialmente ser explorado de forma maligna. Uma análise mais aprofundada exigiria três pedidos *HTTP* para cada um dos parâmetros *HTTP*:

1. *Enviar um pedido contendo o nome do parâmetro padrão e valor e registar a resposta HTTP. Por exemplo, pagina? par1 = val1*
2. *Substituir o valor do parâmetro por um valor adulterado, apresentar e guardar a resposta HTTP. Por exemplo, pagina? par1 = HPP_TEST1*
3. *Enviar um novo pedido combinando os passos 1 e 2. Mais uma vez, guardar a resposta HTTP. Por exemplo, pagina? par1 = val1 & par1 = HPP_TEST1*

Dever-se-ia a seguir, comparar as respostas obtidas durante os pontos anteriores. Se a resposta a partir de 1. é diferente de 2. e a resposta a partir de 3. também diferente de 2., existe um problema de poluição da *query string*, que pode eventualmente ser explorado para um ataque (OWASP, 2014).

3.3.2. Injeção de SQL

A injeção de SQL (*SQL injection)* é uma técnica utilizada por *hackers*, em que é introduzido o código SQL na resposta enviada ao servidor. Este código SQL, pode ser introduzido através de campos de formulários, da utilização da *query string*, ou até da fabricação de uma resposta fictícia.

Utilizando o *URL* do ponto 3.2, existe um parâmetro que corresponde ao *id* de um produto que é enviado através da *query string*. Este é um procedimento bastante comum e que, quando mal implementado, pode ser utilizado para injetar código SQL.

```
http://v1.mce-godinho.isec.pt/SiteGestao/Produtos/EditarProduto.aspx?id=8
```

Com o valor do *id* passado por *query string* será efetuado um pedido à base de dados para aceder à informação deste item. O código *SQL* utilizado será algo do género:

```
SELECT * FROM [BeGamer].[dbo].[ProdutosDetails] WHERE id = 8;
```

Dependendo da API utilizada pela aplicação *web* e da DBMS (exemplo: PHP + PostgreSQL, ASP+SQL SERVER) é possível executar múltiplos *querys* com apenas uma chamada. Terminando o comando anterior com o caracter *';'* poderá ser possível executar novos comandos, após um comando anterior ser terminado. Utilizando novamente o endereço do exemplo acima descrito, uma tentativa de injeção de SQL poderia ser algo do género (OWASP, 2015):

```
http://v1.mce-godinho.isec.pt/SiteGestao/Produtos/EditarProduto.aspx?id=8; INSERT INTO users (…)
```

Em ASP.NET é necessário validar os parâmetros de entrada quando estes são enviados/obtidos por *query string*. Nesta dissertação, esta validação é efetuada em dois passos e do seguinte modo (o código apresentado pertence à página de edição de um produto):

```
if (Request.QueryString["id"] != null)
  {
  …
    using (SqlCommand cmd = new SqlCommand(query, con))
    {
       …
       cmd.Parameters.Add("@id", SqlDbType.Int).Value = Int32.Parse(Request.QueryString["id"]);
       …
```

Podemos verificar que o acesso a um campo da *query string* é efetuado através da propriedade *QueryString* do pedido (*Request*). Inicialmente é verificado se existe uma chave com o nome *'id'* na *query string*; caso exista uma chave com este nome, o valor da *query string* é passado ao comando enviado à base de dados através do método indicado, existindo ainda outros métodos do objeto *SqlCommand* que podem ser utilizados. Esta segunda parte é de extrema importância, porque a plataforma ASP.NET possui mecanismos que controlam os valores passados pela *query string*, filtrando-os e prevenindo tentativas de ataques de *SQL Injection*, sem a necessidade da verificação e análise do conteúdo por parte do utilizador.

Existem outros mecanismos que podem ser utilizados na verificação dos dados introduzidos. Para os formulários, na plataforma ASP.NET existe um conjunto de controlos de validação, que permitem validar se um campo é numérico, se tem um determinado formato, ou até se o seu valor se encontra dentro de um intervalo.

3.4. Sessões

Uma sessão é definida como o período de tempo que um utilizador interage com uma aplicação *web* (Howard, 2000). A utilização de sessões em ambientes *web* permite identificar e diferenciar cada um dos clientes que lhe estão a aceder a uma determinada aplicação, criando deste modo um espaço de memória exclusivo para cada um deles, do lado do servidor onde a aplicação é executada (Oneda, 2010).

Em termos práticos, uma sessão é definida por uma coleção do tipo dicionário de pares de valores-chave, que serão utilizados pelo utilizador durante a sua sessão.

Durante um pedido efetuado a uma página é verificado se existe uma sessão criada para o utilizador, caso contrário esta será criada e gerado um identificado (*ID*) de 24 caracteres pela classe *SessionIDManager*. Por exemplo, quando um utilizador seleciona opções num formulário de numa página, a aplicação *web* pode armazenar estes valores na instância de sessão ASP do utilizador (Howard, 2000):

```
Session("Stocks") = "MSFT; VRSN; GE"
```

Em termos da plataforma ASP.NET, é criada uma instância da classe *HttpSessionState*, que pode ser acedida pela propriedade *Session* existente no contexto da página acedida. O utilizador pode verificar o *ID* gerado que identifica a sua própria sessão (Oneda, 2010):

```
protected void Page_Load(object sender, EventArgs e)
{
     lblSessionID.Text = String.Format("Session ID: {0}", Session.SessionID);
}
```

Armazenando os dados dos utilizadores deste modo, é possível em qualquer página adicionar valores à sessão, que permitem posteriormente, na mesma ou em qualquer outra página, recuperar os valores existentes na sessão. É necessário ter em conta que as sessões dos utilizadores, por razões de segurança e razões de otimização de recursos, têm um tempo de vida máximo, após o qual a sessão expira e todos os dados são apagados de memória. Torna-se necessário verificar se a sessão é válida, testando se existe a chave da qual pretendemos obter o valor:

- *Na página em que é adicionado um campo à sessão:*

```
Session("ProdutoID") = 12345;
```

- *Quando se pretende recuperar o conteúdo de uma chave existente na sessão:*

```
```

```
if (Session("rodutoID") != null)
{

    cmd.Parameters.Add("@id", SqlDbType.Int).Value = Int32.Parse(Session("ProdutoID"));
}
else
{
    cmd.Parameters.Add("@id", SqlDbType.Int).Value = 0;
}
```

3.5. Conclusão

A utilização dos mecanismos de *CrossSite-Post*, Sessões e *query string* têm como grande vantagem a facilidade de implementação, quer na passagem de valores para uma outra página, quer a recuperá-los numa outra página. No entanto, a utilização de *query strings* tem como desvantagens:

- *serem facilmente legíveis;*
- *o URL ter um limite máximo de caracteres;*
- *poder ser adulterada, com utilização de valores inválidos ou injeção de código malicioso.*

Este último ponto tem implicação direta nesta dissertação. O objetivo de ocultar a navegação é dificultado a partir do momento em que o *URL* e respetiva *query string* possam ser intersetados através de *sniffers* de rede.

Se tivermos em conta as formas disponíveis de passagem de informação entre páginas, a utilização de sessões é o mecanismo que mais se enquadra no objetivo desta dissertação, de modo a permitir ocultar dados da navegação. Infelizmente, devido às sessões residirem em memória do lado do servidor, podendo conter uma elevada quantidade de dados, poderá tornar-se demasiado pesado em termos de utilização de recursos, especialmente se existir um elevado número de utilizadores a aceder ao *website*, podendo tornar a navegação demasiado lenta ou até mesmo impossível, esgotando os recursos do servidor.

4. ASP.NET

O ASP.NET é a plataforma da Microsoft para desenvolvimento de aplicações *web* em .NET, que permite gerar páginas que contêm HTML, CSS, JavaScript e que contém funcionalidades do lado do servidor (W3Schools, 2012). O *ASP.NET Web Forms* é parte da estrutura da plataforma ASP.NET, sendo a plataforma lecionada neste mestrado para desenvolvimento de aplicações *web*. Será utilizada para as várias implementações nesta dissertação.

Web Forms é uma metodologia de criação de páginas *web* na plataforma ASP.NET, em que cada página é dividida em dois ficheiros. Um primeiro documento com a extensão .aspx que é aquele a que o utilizador acede e servido pelo *IIS*; este ficheiro é uma combinação de HTML, *scripts* do lado do cliente e controlos da plataforma. O segundo ficheiro é o ficheiro chamado de *code-behind*, que é um ficheiro que pode ser em *C#* (ou *VB* nas versões mais antigas*)*, que é o código que será interpretado pelo servidor. Quando um utilizador efetua um pedido a uma página, esta é executada do lado do servidor pela *framework*, a qual gera o HTML que é enviado ao cliente e que permite ao *browser* interpretar a página (Microsoft, 2014e).

4.1. Arquitetura e componentes do *IIS*

Desde a versão 6.0 do *IIS* até a versão à versão atual 8.5 (versão utilizada na implementação prática desta dissertação, em Windows 2012 R2), existiu um conjunto significativo de alterações à sua constituição e funcionamento. Existem pequenas diferenças entre as versões 7 e 8, sendo necessário apenas perceber o seu funcionamento no global. Uma descrição detalhada de todos os elementos do servidor *web IIS* vai muito para além do que se pretende nesta dissertação. São descritos com maior detalhe os componentes que intervêm na implementação prática.

O *IIS* funciona a dois níveis, estando os componentes distribuídos entre os níveis Kernel e Utilizador. O *IIS* contém vários componentes que desempenham funções importantes para as funções das aplicações e do servidor *web* em sistemas Windows. Os seus componentes têm responsabilidades, tais como estar à escuta nos portos para receber as solicitações feitas ao servidor, ou gerir os processos de leitura de ficheiros de configuração. Esses componentes incluem os serviços de escuta dos protocolos de rede, *Hypertext Transfer Protocol Stack (HTTP.sys)* e serviços como o de publicação (serviço WWW) World Wide Web e Serviço de Ativação de Processos do Windows (WAS).

Ao nível do Kernel o *IIS* é composto pelo Hypertext Transfer Protocol Stack (HTTP.sys). É o primeiro ponto de passagem de um pedido enviado para o *IIS*, isto é, é responsável por receber e responder aos pedidos dos clientes. Existe também um conjunto de elementos responsáveis pela comunicação com a pilha protocolar de rede (podendo também receber pedidos de aplicações que não sejam *HTTP*). É a parte que a nível do Kernel do sistema, escuta os protocolos *HTTP* e *HTTPS* (Kenneth Schaefer, 2011).

Ao nível do Utilizador existem dois serviços, *World Wide Web Publishing Service* (W3SVC) e o *Windows Activation Service* (WAS) que funcionam internamente no processo

Svchost.exe, sendo responsáveis pela monitorização de *performance*, monitorização do processamento e gestão da configuração. O *IIS* contém um mecanismo chamado *Application Pool*, que permite isolar diferentes aplicações *web*. Deste modo, podem ser definidas diferentes configurações de segurança e de utilização de recursos, criando "estanquidade" entre estas (Abhijit, 2010). A organização dos níveis e distribuição dos componentes encontra-se descrita na figura 5.

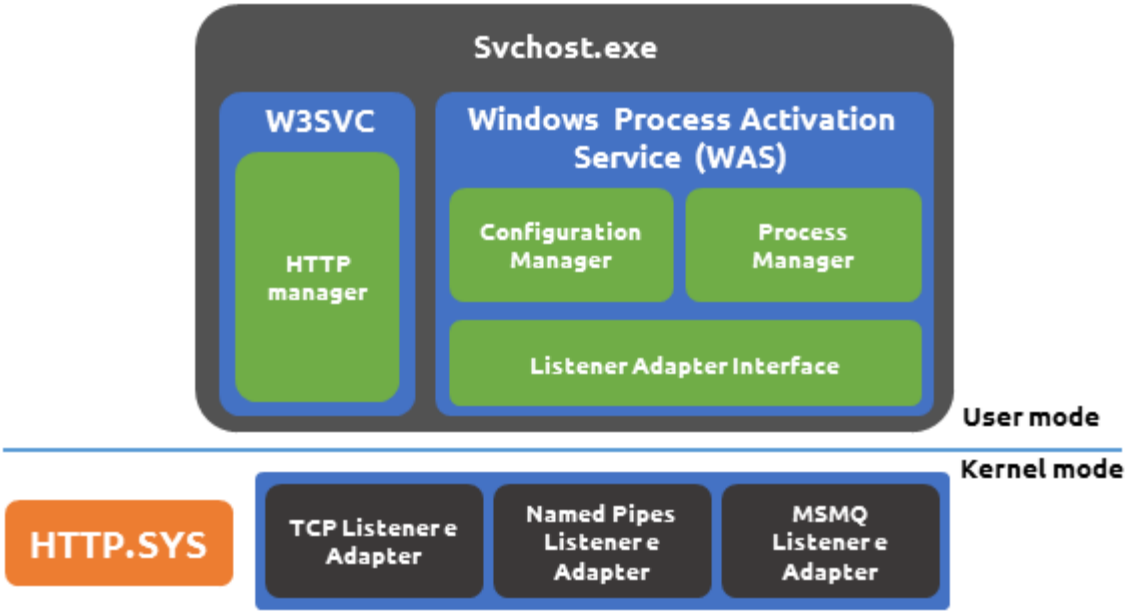

Figura 5 - Arquitetura do IIS 7 e 8

O papel do HTTP.sys funciona ao nível do subsistema de rede do sistema operativo Windows, recebendo pedidos da rede e transmitindo-os ao *IIS* para processamento. É também responsável pelo percurso inverso de envio das respostas para os clientes (Templin, 2007), como demonstrado na figura 6.

Nas últimas versões do *IIS*, o W3SVC é responsável pela configuração do HTTP.sys (ou reconfiguração), e por notificar o WAS quando um pedido é colocado na fila de espera, isto é, quando chega um novo pedido ao *IIS* (Templin, 2007), como demonstrado na figura 6.

O WAS é responsável por encaminhar os pedidos recebidos para a *Application Pool* pretendida. Cada *Application Pool* tem um ou mais processos (*Worker Process*), w3wp.exe, geridos pelo WAS e que estão encarregados de processar os pedidos destinados à *pool* (Abhijit, 2010), como demonstrado na figura 6.

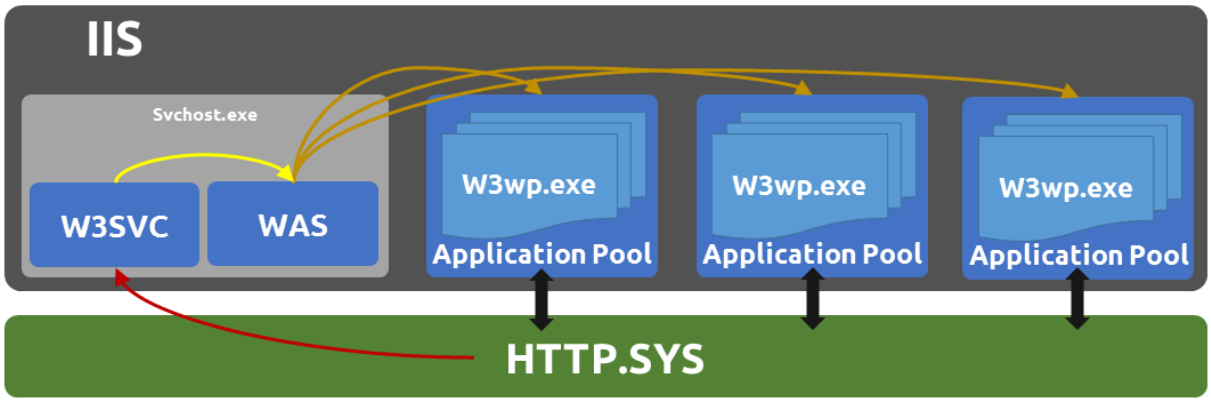

Figura 6 - Processamento de pedidos pelo IIS

O processo (*Worker Process*) ao receber um pedido, verifica o *URL* pedido de modo a verificar a extensão Internet Server API (ISAPI) correta. Estas extensões são o modo que permite ao *IIS* processar os pedidos para diferentes recursos. Quando o suporte à *framework* ASP.NET está instalada no *IIS*, é instalada uma extensão, a aspnet_isapi.dll, sendo adicionado um mapeamento para os documentos com extensão .aspx (Abhijit, 2010). Do mesmo modo, quando é instalado o *PHP5* com o *IIS*, é adicionada a extensão php5isapi.dll e respetivo mapeamento para os documentos com extensão *.php* e *.php5*. Quando a extensão aspnet_isapi.ddl é carregada, é chamado o *HTTPRuntime* e começa o processo de passagem *HTTP Pipeline* propriamente dita. Um *HTTP handler* em ASP.NET é um processo que é executado em resposta a um pedido feito a uma aplicação *web* em ASP.NET. O *handler* mais comum é o responsável por processar as páginas do tipo .aspx, existindo outros *handlers*, como por exemplo o que processa os pedidos .asmx, páginas de *web services* em ASP.NET (Microsoft, 2014d). Existem outros que poderão ser incluídos, externos à *Microsoft*, como o já referido para suporte do PHP5.

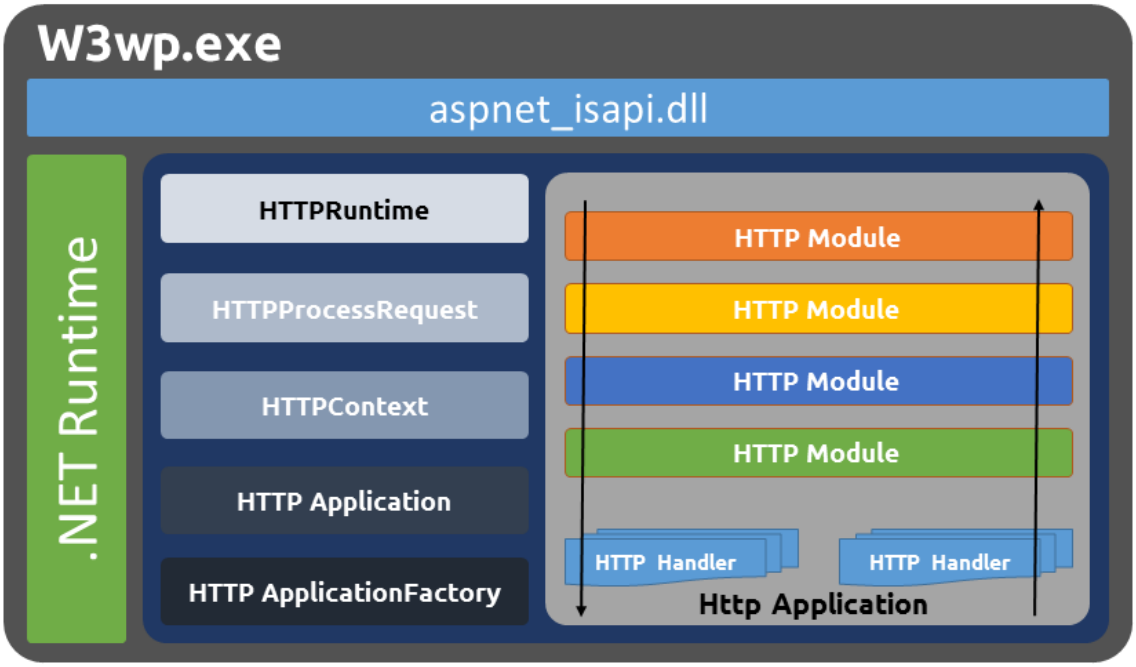

Figura 7 - Estrutura do w3wp.exe (Worker Process)

Um *HTTP Module* é um *assembly* que é chamado em cada pedido efetuado à aplicação *web*. Os *HTTP Modules* são executados como parte da *pipeline* e têm acesso aos eventos do ciclo de vida do pedido. Permitem portanto examinar o pedido e efetuar algum tipo de ação, do mesmo modo que permitem examinar a resposta e modificá-la (Microsoft, 2014d).

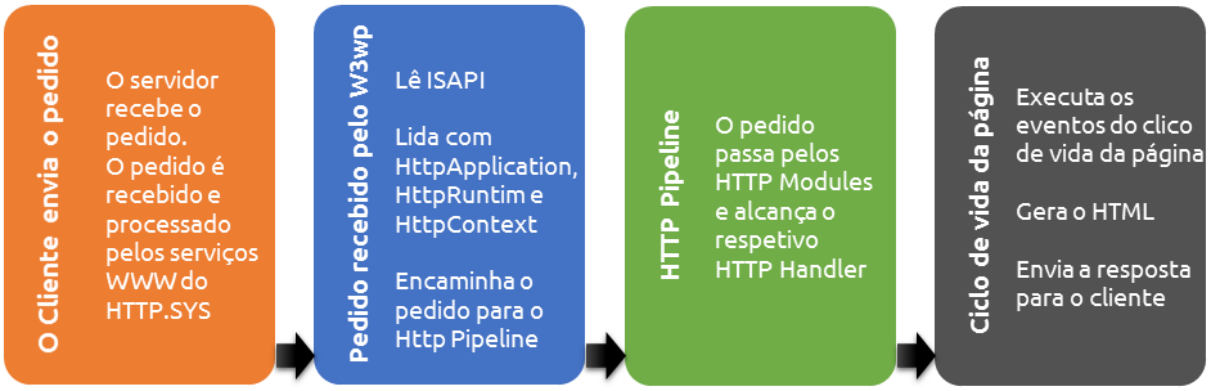

Figura 8 - Percurso interno de um pedido ao IIS

Torna-se importante compreender a estrutura e organização destes componentes internos, visto que duas das implementações passaram por criar um novo *HTTP Module* e, noutro caso, por criar um *HTTP Handler*, sendo as soluções encontradas para alterar tanto o pedido à entrada e a resposta à saída, como também o modo como uma página é processada.

4.2. HTTP Module e HTTP Handler

No ponto 4.1 estes dois componentes foram introduzidos e foi representado na figura 7 o modo como um pedido percorre o *HTTP pipeline.* Se for necessário alterar o modo como certas páginas são tratadas, ou executar instruções antes ou após o processamento da página, como por exemplo em mecanismos de geração de estatísticas, *URL ReWrite*, autenticação/autorização, entre outros, existem dois modos de o efetuar, através de *HTTP Handlers* e de *HTTP Modules* (Koirala, 2009).

Quando se cria um *HTTP module* e este é incorporado na *HTTP Pipeline*, o que efetivamente acontece é permitir o acesso aos eventos do ciclo de vida do pedido efetuado pelos clientes; por esse motivo, é muitas das vezes referido como processador de eventos do *IIS*. Entre outras funcionalidades, os módulos podem examinar e modificar o conteúdo enviado de volta para o cliente. É possível registar um *HTTP Module* num dos eventos, sendo executado como demonstrado na figura 9.

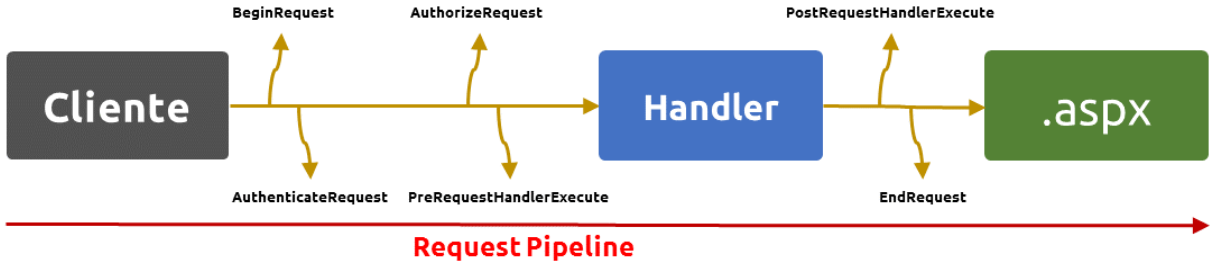

Figura 9 - Percurso do pedido ao longo dos vários eventos

É possível verificar que o último componente a processar um pedido é um *HTTP Handler*. Como referido anteriormente, cada *HTTP Handler* processa um pedido com base na extensão do documento pretendido, sendo também possível mapear um *HTTP Handler* para múltiplas extensões (Koirala, 2009). Por exemplo na Figura 10, um pedido para um documento com a extensão .xpto, é executado o *HTTP Handler* 3.

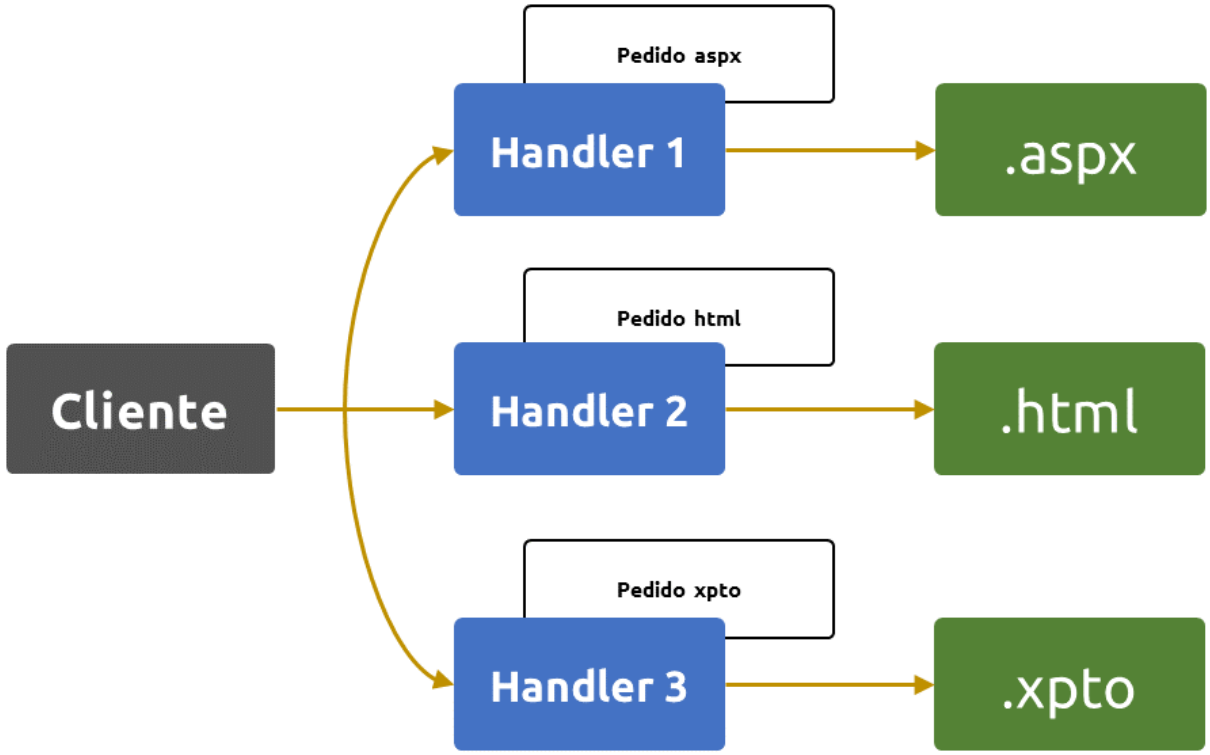

Figura 10 - Mapeamentos dos Handlers por extensão

4.3. Ciclo de vida de uma página em ASP.NET

Cada vez que um pedido para uma página ASP.NET chega ao servidor *web*, este é processado através de vários estágios (*pipeline*). A classe que corresponde ao pedido ASP.NET é instanciada e o método *HTTP ProcessRequest* é invocado (Mitchell, 2004a):

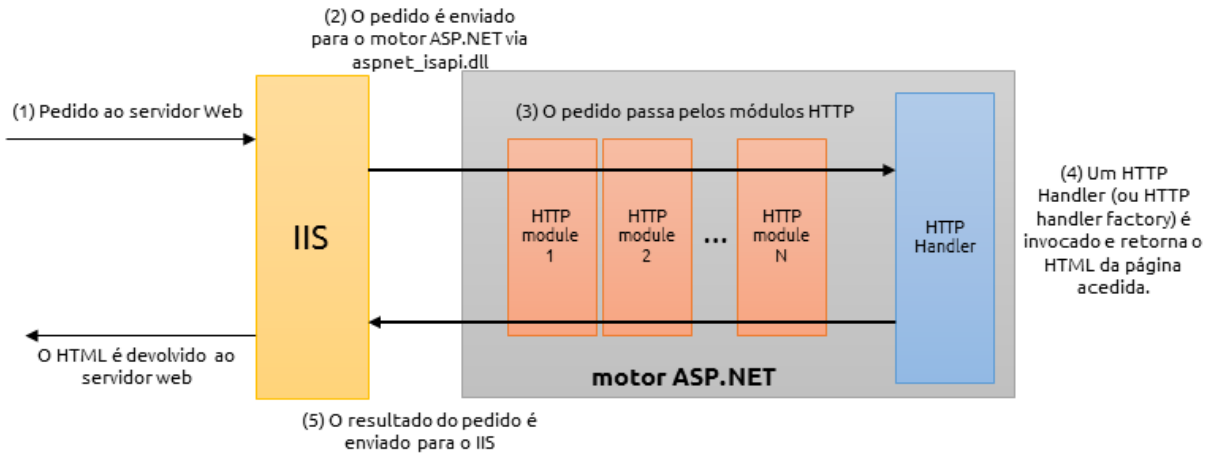

Figura 11 - Processamento de pedidos IIS e ASP.NET

O ciclo de vida de uma página ASP.NET começa com a chamada do método *ProcessRequest*. Este é o método que inicia os controlos da página. De seguida, a página e os seus controlos seguem um conjunto pré-definido de passos, que são essenciais à execução de uma página em ASP.NET. Estes passos incluem a gestão do *ViewState*, processamento de eventos de *PostBack* e geração do *HTML*:

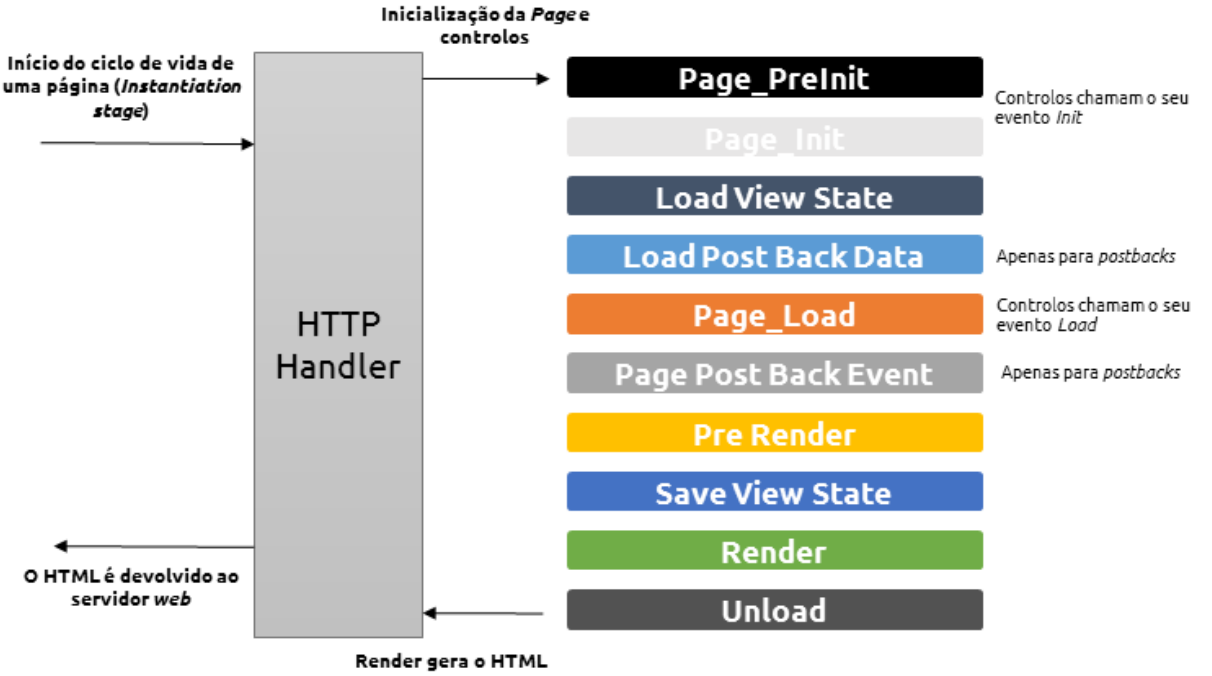

Figura 12 - Eventos no ciclo de vida de uma página ASP.NET

Ponto fulcral a reter é que cada vez que é efetuado um pedido a uma página ASP.NET, o pedido percorre sempre a mesma sequência de eventos; logo um evento que contenha um conjunto de instruções a serem efetuadas, estas serão sempre executadas de cada vez que a página é requerida. Um dos exemplos práticos da utilização de um evento para execução de certas tarefas, e também dos mais frequentes na programação em ASP.NET, é a execução de tarefas no evento *Page_Load*, permitindo executar tarefas antes de gerar o HTML dos

controlos, como por exemplo popular listas, grelhas, verificações de autenticação ou autorização, entre outras tarefas. Nesta dissertação, este evento é utilizado para efetuar verificações de acesso e verificação da existência de chave de encriptação, entre outras.

4.4. Utilização de controlos do ASP.NET

Os controlos de ASP.NET são objetos em páginas da *web* ASP.NET que são executados quando a página é solicitada e que são processados em HTML para o *browser*. Muitos controlos ASP.NET são semelhantes aos elementos equivalentes em HTML, tal como botões, caixas de texto, e outros. No entanto, existem controlos mais complexos em relação aos quais não há um equivalente direto nos elementos existentes em HTML; são exemplos controlos que podem ser utilizados para aceder a fontes de dados, grelhas, controlos de calendário, entre outros controlos (Microsoft, 2014e). No caso dos controlos que têm representação gráfica, isto é, que aparecem na página *web*, a plataforma encarrega-se de gerar o HTML que representar o controlo. Os controlos que não têm representação gráfica na página, serão executados do lado do servidor e a sua interação com os restantes controlos ou com motores de bases de dados, por exemplo, é refletida antes da geração do HTML final da página que é enviada para o utilizador.

4.5. ASP.NET *Postback*

Esta funcionalidade é o mecanismo de submissão de uma página ASP.NET para ela própria, enviando dados de volta para o servidor *web*, para processamento de determinados eventos. Este mecanismo permite a implementação de métodos de validação de dados introduzidos em formulários e respetiva atualização com sinalização de erros. Permite ainda a utilização da mesma página para finalizar o processo, por exemplo como inserção em base de dados.

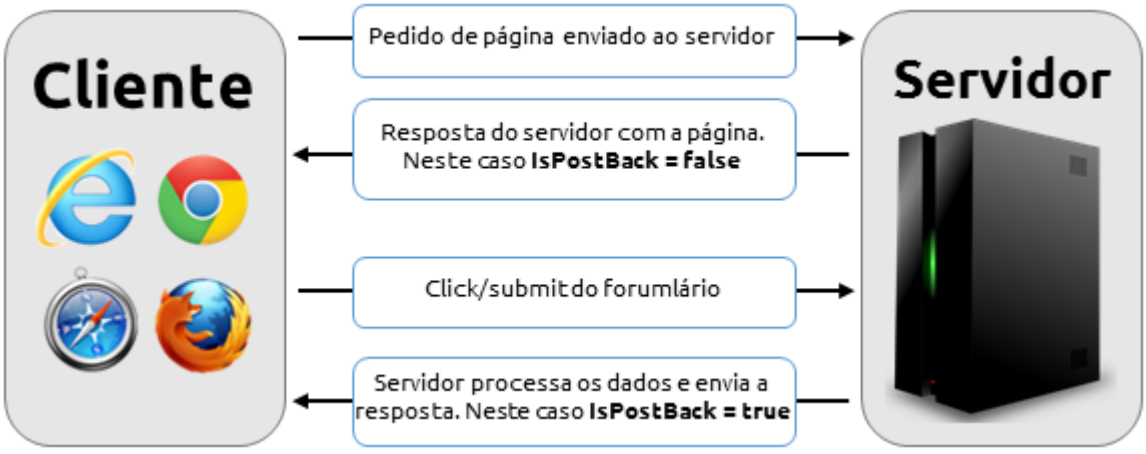

Figura 13 - Passagem de dados entre cliente e servidor num PostBack

Programaticamente este mecanismo é divido nos seguintes elementos (Raja, 2011):

- *um elemento FORM que indica a página para onde é submetido o formulário (para além do PostBack também por um botão, por exemplo);*
- *um Hidden Field, traduzido num elemento INPUT, com o nome __EVENTTARGET;*
- *um Hidden Field, traduzido num elemento INPUT, com o nome __EVENTARGUMENT;*
- *uma função em JavaScript __doPostBack, que submete o formulário e recebe dois argumentos eventTarget e eventArgument.*

```
<form method="post" action="/Default.aspx?lk=yEGM5xMxGEGeGGXxn1aieQSxyC2/Mn7mnn" id="formMain">
…
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
…
<script type="text/javascript">
…
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
//]]>
</script>
```

Qualquer controlo que tenha a propriedade *AutoPostBack*, fica ligado à função *__doPostBack* utilizando os atributos onClick ou onChange. Estes atributos indicam ao *browser* a que ações deve responder do lado do cliente, através de eventos de *JavaScript* onClick e onChange. Na prática, o motor ASP.NET transforma automaticamente estes eventos em código *JavaScript* que é executado do lado do cliente, utilizando a função *__doPostBack* (Raja, 2011). A sequência de eventos está descrita na Figura 12.

A utilização de *JavaScript* permite implementar os mecanismos do lado dos clientes, para que a plataforma possa ser acedida, funcionando corretamente e do mesmo modo em todos os *browsers*, incluindo os dispositivos móveis.

4.6. ASP.NET ViewState

No ponto 3 foi referido, de uma forma genérica, que o protocolo *HTTP* é *stateless;* existem técnicas que permitem que existam dados de forma persistente. O modo de gestão utilizado pelo ASP.NET é chamado *ViewState*. Existem duas categorias e várias técnicas de gestão de estado em ASP.NET, tal como referido na Figura 4.

É possível alterar o modo como o *ViewState* funciona, de modo a permitir a utilização de bases de dados ou outro tipo de suporte de dados (Microsoft, 2014b), em que o mecanismo que determina o comportamento é chamado de *provedor (provider)*.

Por omissão, quando o estado atual da página e controlos é processado, gera-se uma *hash* (neste caso uma *string* em base 64) que é guardado num *Hidden Field* da página com o

nome *__VIEWSTATE*. Sempre que existe uma submissão do formulário o conteúdo deste campo é enviado como parte do conteúdo do POST (Bhimani, 2015).

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="ZRYcgjir7ye0y5L3FmE40RSK03cMilzsiwGvbip0BmZZ46UeAOPa/KIojWe6mJ+DKAZgNJ44BMEXgx9XQGJl9POc8kOFs
UqXABD8snW2jcuFX9AuURi+qOgKOunvK3Ht8PDSzDvHXleUia6+vdtUywLPvKBjEFvLSSUIO+dVW6Vx4vKXu5arEoARjWjIEqhVt
n9fnNSIgY7pnXEKCE/3PDRQ3c6GsL/kHdVfblVEPq4ztTKtWq6hoCrdgv8RNvRepCmZx3Gcvxwf4svhyxLYI0WLXFo8Ty+2vqLwO
rZx/FSv+pE0T4M/nPCImvfDWe6lx+KsIvd6h4JnS+V7i0KJ+Si7aTyKEuiF06wVo6lAkxI0M4p6GT8q2w1TT4qPSWHFVKeG4VvID
QyBvjw4JHpNO1N5Giuisr4D+Bb7Fe9aSyvqliu6Skkf3Q5aYOq07YlEwhqwMwDq+cSpV8Gy6Fm43TIMsEAuovrr2Pzrhr0zydAgW
0u0cD/j9Y+jqy742upNCgYDbzWMFXXD+E1VYS11w6tHT67aduuX5GH3HfJ5S2XFUmO+tFZd1Cmvj1TPMk8gj/cpM5py/gYxTt2q2
lRnahvdz2/AyWiQoftAi0CA0eL3eWrV6OsySvgyCxlmWXg/YRIjAZR57hlMffKdga9QqhhX0R53+/FOb5mj/wbuzDlEs/2Dex+
…
```

Cada controlo é responsável por guardar a informação do seu estado através da propriedade *ViewState* definida na classe *System.Web.UI.Control* (Mitchell, 2004a). No entanto, é também possível adicionar dados.

4.7. Encriptação em ASP.NET

Quando nos referimos a encriptação, é necessário compreender que existem duas técnicas distintas: encriptação simétrica e encriptação assimétrica.

4.7.1. Encriptação assimétrica

A encriptação assimétrica é constituída por duas chaves, uma pública e outra privada, conhecida pela sigla PKI (*Public Key Infrastructure*).

Este mecanismo de segurança é normalmente utilizado quando existem dois intervenientes diferentes que comunicam um com o outro, como por exemplo, um *browser* a comunicar com um servidor *web*, em que um fornece um serviço público, encontrando-se, portanto, exposto.

A chave pública será a chave que é partilhada, utilizada pelos clientes para encriptar os dados que são enviados para o servidor, que posteriormente serão desencriptados utilizando a chave privada existente no servidor. Há, portanto, uma relação entre as duas chaves, sendo, no entanto, diferentes uma da outra. É necessário haver uma entidade que certifique a autenticidade e confidencialidade dos dados trocados.

4.7.2. Encriptação simétrica

Este tipo de encriptação consiste na existência de uma única chave que permite a encriptação e desencriptação. Por norma é utilizada para proteger dados quando estes são guardados num dispositivo físico ou como forma de completar outras medidas de segurança. É muito mais rápida que a utilização de chaves assimétricas, sendo utilizada por exemplo em bases de dados.

Nesta dissertação, será gerada uma chave que será guardada do lado do servidor na sessão do utilizador. A chave do utilizador é utilizada por exemplo para encriptar o *URL* e/ou a *query string* enviada para o cliente, e no percurso inverso quando um pedido é recebido pelo servidor para desencriptar os mesmos dados de modo a servir a informação a que o cliente deseja aceder.

4.7.3. Protocolos de encriptação

Existem vários protocolos de encriptação disponíveis para a utilização na *internet* e na plataforma *ASP.NET*. Neste momento o algoritmo *standard* do *NIST* (*National Institute of Standards and Technology* dos Estados Unidos da América, EUA) é o *AES* (*Advanced Encryption Standard*).

Tal como os algoritmos de encriptação simétricos, o *AES* recebe o conteúdo que se deseja encriptar, que por norma se trata de texto simples, e também a chave de encriptação. O resultado do processo de encriptação é um conjunto de números em sistema binário, como tal não consegue ser interpretado. É comum incluir no algoritmo de encriptação do *AES,* no processo inicial de encriptação, uma medida adicional de segurança, chamada o vetor de inicialização (*IV*). Do mesmo modo é uma boa norma incluir na parte de *hashing* um mecanismo adicional chamado sal (*salt*). Ambos os mecanismos permitem ofuscar o resultado da encriptação, de modo a tornar ainda mais complexo o processo de desencriptação e, aumentando, deste modo, a proteção contra potenciais ataques.

Figura 14 - Efeito da utilização de salt na saída do algoritmo AES (Cogley, 2007)

O *AES* recebe, na entrada, blocos de tamanho fixo de 128 bits, em conjunto com a chave de tamanho variável.

O *AES* funciona manipulando uma matriz bidimensional de *bytes,* com 4x4 posições. Para encriptar, cada iteração do *AES* consiste em quatro estágios: *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey* (Gürkaynak, 2014), como demonstrado na Figura 15.

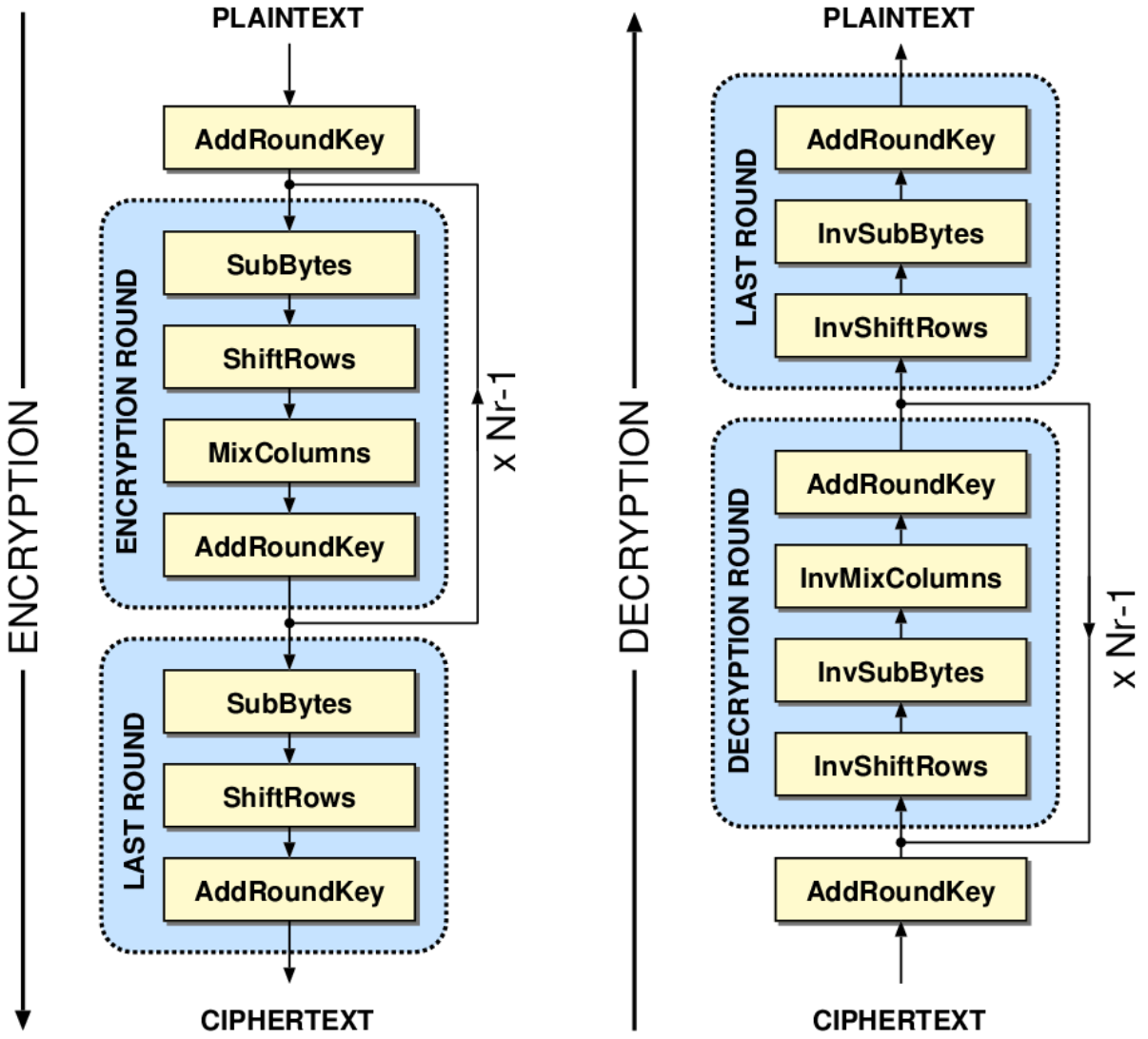

Figura 15 - Estrutura do algoritmo AES: esquerda encriptação, direita desencriptação

Embora em ASP.NET estejam disponíveis implementações de algoritmos mais antigos, tais como DES, 3DES e RC2, o algoritmo AES é o que a Microsoft recomenda de modo a proteger dados (Microsoft, 2014a). Existem inúmeras bibliotecas na *internet*, com exemplos de código utilizando diferentes implementações com AES. Nesta dissertação é utilizado um destes exemplos que utiliza os mecanismos *IV* e *salt*.

4.8. ASP.NET Redirecionamentos

O mecanismo de redirecionamento é crítico para o desenvolvimento desta dissertação. Sem a sua utilização seria muito difícil ocultar o endereço original. No entanto, é essencial perceber, que do lado do servidor, para uma página ser acedida, é essencial obter o seu endereço e respetiva *query string* (caso esta exista). Se por exemplo, acedermos ao endereço *http://localhost:88888/WebSiteMCE/Default.aspx?QueryString1=1&QuerrString2=2*, programaticamente é possível obtermos acesso à informação da seguinte forma:

Código *Resultado*

HttpContext.Current.Request.Url.Host	localhost
HttpContext.Current.Request.Url.Authority	localhost:88888
HttpContext.Current.Request.Url.AbsolutePath	/WebSiteMCE/Default.aspx
HttpContext.Current.Request.ApplicationPath	/WebSiteMCE
HttpContext.Current.Request.Url.AbsoluteUri	http://localhost:88888/WebSiteMCE/Default.aspx?QueryString1=1&QuerrString2=2
HttpContext.Current.Request.Url.PathAndQuery	/WebSiteMCE/Default.aspx?QueryString1=1&QuerrString2=2

Tabela 1 - Código em C# para obter o URL e/ou query string

4.8.1. Response.Redirect

Nas várias implementações práticas do *website* utilizadas para demonstrar esta dissertação, com maior ênfase na primeira delas, o *website* utiliza a funcionalidade *Response.Redirect* que não é mais do que um redirecionamento de uma página para outra. A Microsoft define este método como o redirecionando do cliente para um novo *URL*, podendo determinar se a execução da página atual termina ou não. Este tipo de redirecionamento é efetuado através de especificações da norma *HTTP/1.0* e *HTTP/1.1*, com códigos na casa dos 300 (a própria *RFC* contém a indicação – *Redirection 3xx*), normalmente com o código 302 (*302 Found*). Nesta dissertação, este tipo de redirecionamento é, por exemplo, utilizado quando tentamos aceder aos detalhes de um produto. Como o *ID* é enviado por *query string*, este é validado se existe, se é um inteiro ou se o seu tamanho é menor que o tamanho máximo (do tipo) de objetos do tipo Int32. Se por ventura alguma destas validações falhar, o pedido será encaminhado para a página principal do *website* (*/Default.aspx*). Os códigos indicados pelo servidor, para o exemplo referido, podem ser verificados na figura 16.

```
if (Request.QueryString["id"] != null && Int32.TryParse(Request.QueryString["id"], out ProdNum) &&
ProdNum <= Int32.MaxValue)
{
...
}
else
{
    Response.Redirect("/Default.aspx", true);
}
```

Tab	Request ID	Type	Time	Status	Method	Hostname	URL
× 1392	62958	main_frame	17:38:21 (136ms)	200 OK	GET	localhost	/Default.aspx
× 1392	62958	main_frame	17:38:18 (2848ms)	302 Found	GET	localhost	/ProdutoDetalhes.aspx 1
× 1392	62948	main_frame	17:37:58 (1846ms)	200 OK	GET	localhost	/ProdutoDetalhes.aspx 1
× 1392	62947	xmlhttprequest	17:37:58 (89ms)	200 OK	POST	localhost	/Default.aspx

Figura 16 - Pedido e resposta utilizando Response.Redirect

Este método aceita um segundo parâmetro do tipo *bool*, que determina o *endResponse*, que indica se a página atual deve terminar a sua execução ou não. Como é óbvio, se o utilizador

estiver a ser redirecionado para outra página, então como modo de preservar/otimizar recursos, o segundo parâmetro deverá ser definido como verdadeiro. Este tipo de redirecionamento é efetuado pelo *browser* do utilizador, logo é possível verificar a alteração do *URL* na barra de navegação e aceder à página que originou o redirecionamento, se se voltar à página anterior.

4.8.2. Server.Response

Nesta dissertação são efetuadas inúmeras verificações e, por norma, em situações de falha das mesmas, serão utilizados os redirecionamentos. Utilizando o mecanismo do ponto 4.8.1, a tentativa de ocultação de navegação do cliente cai por terra, permitindo que, através de análise do tráfego entre cliente e servidor, seja possível perceber o endereço para o qual o utilizador é direcionado.

A *framework ASP.NET* possui um outro mecanismo, *Server.Response*, que permite, ao invés de indicar ao *browser* que deve aceder a outro endereço, entregar diretamente o conteúdo dessa nova página ao cliente, como se lhe estivesse a aceder diretamente. Este método permite diminuir a quantidade de pedidos efetuados ao servidor, para além de manter o mesmo *URL*. Este método apenas pode ser utilizado para transferir páginas do próprio servidor e não páginas existentes em servidores remotos. Esta restrição existe, porque o contexto do pedido é transmitido para a página final; deste modo o objeto que representa o pedido permanece o mesmo, sendo necessário que o processamento fique na mesma aplicação *web*. No caso da página, para onde se deseja redirecionar o utilizador, existir noutra aplicação *web* ou domínio, seria necessário efetuar um novo pedido e, portanto, utilizar o mecanismo do ponto 4.8.1. Este método aceita um segundo parâmetro do tipo *bool*, que determina o *preserveForm*, que tal como o próprio nome indicia quando definido como verdadeiro, as variáveis do formulário e *query string* estão disponíveis na página final que irá ser transferida (Moore, 2004).

4.9. Routing

No ponto 2.2 foi descrito o mecanismo de *URL ReWrite*. O *Routing* é um mecanismo cujo resultado final poderá ser considerado equivalente, com funcionamento diferente, podendo ser considerado como uma evolução do *URL ReWrite*. Em ASP.NET, o *Routing* é um mecanismo que permite associar um determinado *URL* a um *HTTP Handler* que o possa processar. Esta associação é efetuada registando as rotas que definem qual *HTTP Handler* deve ser invocado para atender ao pedido de um determinado diretório de um *URL* (Yakushev, 2008). Quando é efetuado um pedido ao servidor *web* ASP.NET, o *Routing* verifica se o caminho do *URL* solicitado se encontra na lista de rotas registadas. Se o percurso for encontrado e a rota existir, o *HTTP Handler* correspondente é invocado para processar este pedido (Yakushev, 2008).

Programaticamente é possível executar o *URL ReWrite* de um pedido que seja intercetado, isto é, para todos os pedidos do mesmo género será necessário a sua interceção de modo a utilizar este mecanismo (de modo estático é possível através da definição de regras no ficheiro *web.config*). O *Routing* é mais adaptado de modo a ser utilizado de forma dinâmica,

permitindo adicionar e remover rotas, que permanecem estáticas, intercetando os pedidos que chegam ao servidor *web*. É necessário perceber que estas rotas não são estáticas e que, em caso de um *restart* do servidor, as rotas serão apagadas. Do mesmo modo, a existência de um *cluster* de servidores, isto é, vários servidores diferentes a servirem o mesmo endereço, será necessário implementar um sistema de sincronização de rotas entre os vários servidores.

4.10. Quando se deve utilizar *Routing* ou *URL ReWrite*

- *O URL ReWrite é utilizado para manipular caminhos do URL antes de o pedido ser processado pelo IIS. O módulo de URL ReWrite não sabe qual o HTTP Handler que eventualmente processará o URL que foi reescrito. Do mesmo modo, o próprio HTTP Handler é alheio ao facto de o URL ter sido reescrito.*
- *O Routing é utilizado para enviar um pedido para um HTTP Handler baseado no caminho do URL que foi solicitado. Ao contrário do URL ReWrite, o módulo de Routing sabe quais HTTP Handlers existem e seleciona o que deve gerar uma resposta para o URL solicitado. Podemos ver o Routing em ASP.NET como um mecanismo avançado de manipulação do mapeamento.*

O processamento de um *URL* pelo *IIS*, implica a verificação da existência do documento requerido, no caminho indicado pelo *URL*. Este processamento passa, por além de verificar a existência física do documento, pela utilização dos módulos de *URL Rewrite* e de rotas para verificar se o *URL* pedido se encontra definido, correspondendo a uma outra localização do que a indicada no *URL*. Apenas no caso de este documento não existir em nenhuma das circunstâncias indicadas, é enviado um erro 404 (*not found*).

O processamento de uma rota e do *URL Rewrite*, difere em termos de *IIS* e arquitetura do ASP.NET; estes processos correm em diferentes estágios do *pipeline*. É necessário entender que o *URL ReWrite* é um modulo nativo que é adicionado ao conjunto de módulos do *IIS*, que processa os pedidos que passam no *pipeline* nos eventos *Pre-Begin Request* ou *Begin Request*, avaliando o *URL* através das regras de *rewrite* existentes. Por outro lado, o *Routing* funciona de modo diferente, sendo operado de modo programático, processando todos os pedidos que passam pelo *pipeline*, nos eventos *Resolve Cache* (*PostResolveRequestCache*) e no evento *Map Handler* (*PostMapRequestHandler*), como se pode verificar na figura 17.

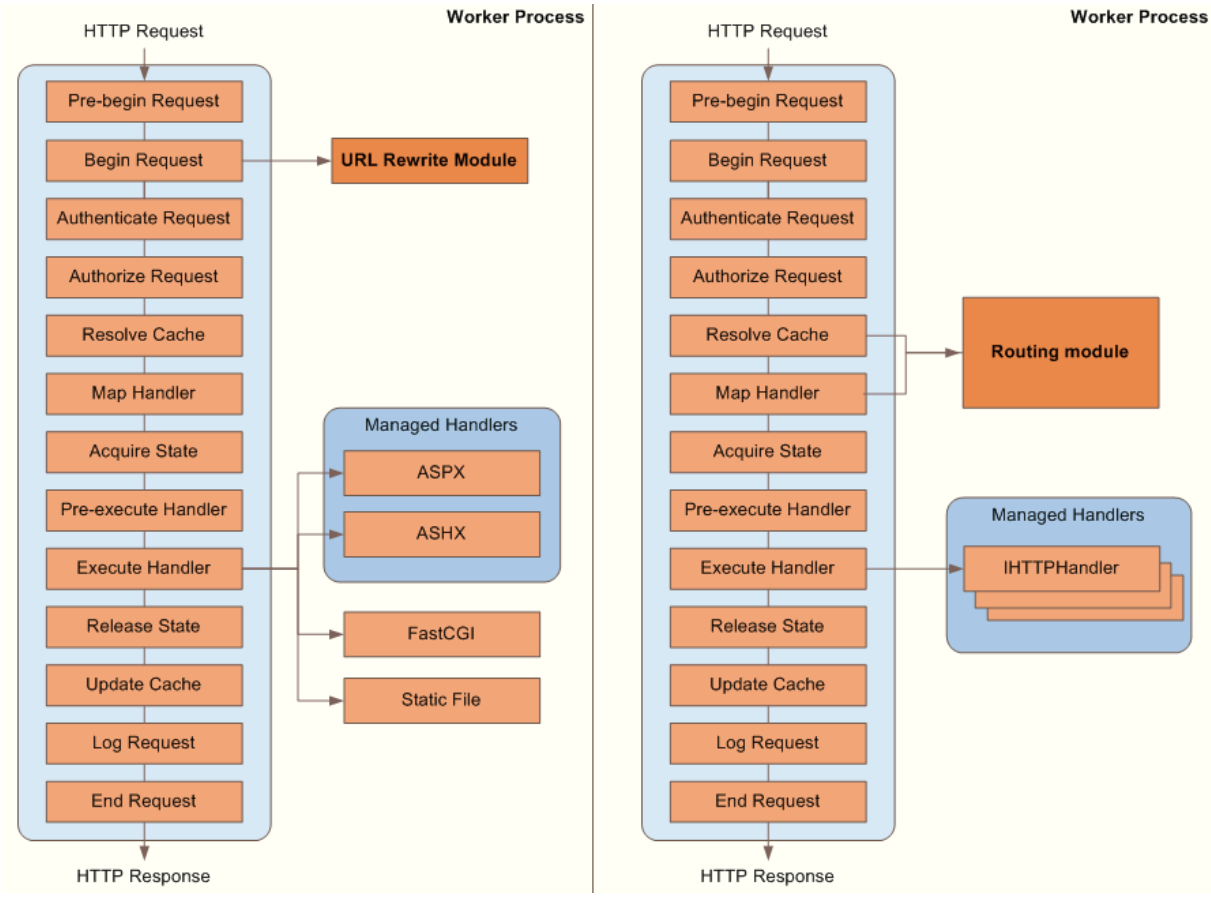

Figura 17 - Routing HTTP Application pipeline

5. Inspeção de tráfego

Nesta dissertação, de modo a testar a validade das implementações, é necessário escutar o tráfego entre o cliente e servidor. Para tal, foram escolhidas duas aplicações: uma que permite verificar o tráfego que passa na rede entre os dois intervenientes e outra que permite verificar o conteúdo recebido pelo cliente (*browser*). Foram escolhidas as ferramentas mais populares entre utilizadores:

- *Web Sniffer*
- *WireShark*

5.1. Web Sniffer

O *Web Sniffer* é uma extensão para o *brower* Google Chrome que permite verificar todos os pedidos e respostas trocados entre o cliente (*browser*) e o servidor, podendo listar estes pedidos por cada uma das abas abertas no *browser* (os *browsers* modernos permitem múltiplas abas com uma página diferente por cada uma aba), método na troca de informação (*GET*, *POST*, etc.), hora do pedido, identificação, entre outros.

Type	Time	Status	Method	Hostname	URL
xmlhttprequest	08:12:00 (96ms)	200 OK	GET	0.client-channel.google.com	/client-channel/channel/bind
xmlhttprequest	08:11:59	200 OK		0.client-channel.google.com	/client-channel/channel/bind
xmlhttprequest	08:11:46 (300ms)	200 OK	POST	plus.google.com	/u/0/_/n/gcosuc
xmlhttprequest (cache)	08:11:46 (356ms)	200 OK	POST	plus.google.com	/u/0/_/n/gcosuc
xmlhttprequest (cache)	08:11:46 (96ms)	200 OK	GET	apis.google.com	/_/scs/abc-static/_/js/k=gapi.g

Figura 18 - Resultado da utilização do Web Sniffer

A utilização da aplicação tem a vantagem de permitir verificar o tráfego *HTTPS* após ser desencriptado à chegada ao *browser,* podendo verificar o tráfego como se o túnel SSL tivesse sido quebrado.

5.2. WireShark

O *Wireshark* é uma ferramenta de análise protocolar, que permite a captação em tempo real de pacotes de dados, apresentando esta informação no seu formato original e num formato legível para o utilizador. O processo de captura de tráfego é realizado através da colocação da placa de rede em modo promíscuo (possibilidade de capturar todos os pacotes, independentemente do endereço de destino) (Pinto, 2014). Para esta dissertação, este programa permitiu filtrar o tráfego entre o cliente e o servidor, filtrando também por porto utilizado, o que permite filtrar os protocolos *HTTP* e *HTTPS*, sobre os quais funcionam a implementação prática.

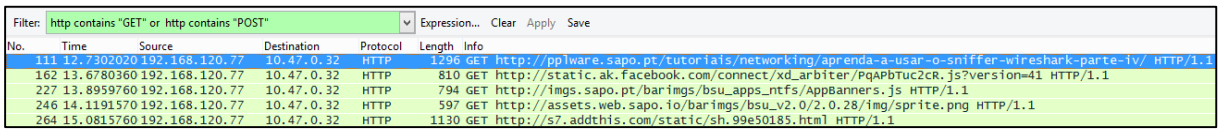

Figura 19 - Captura de tráfego HTTP com o WireShark

O servidor utilizado para as várias implementações desta dissertação e onde se encontra a correr o *IIS*, tem o *IP* 10.204.1.200 e que serve os protocolos *HTTP* e *HTTPS*. A filtragem do tráfego é efetuada da seguinte forma:

Filtro	*Resultado*
http contains "GET" or http contains "POST"	Filtra o tráfego mostrando apenas o tráfego *HTTP* com *GET* ou *POST* no corpo
ip.dst == 10.204.1.200 and tcp.port==443	Filtra o tráfego *HTTPS* para o *IP* do servidor mce-godinho.isec.pt
ip.dst == 10.204.1.200 and tcp.port==80	Filtra o tráfego *HTTP* para o *IP* do servidor mce-godinho.isec.pt
ip.dst == 10.204.1.200 and (tcp.port==80 or tcp.port == 443)	Filtra o tráfego *HTTP* e *HTTPS* para o *IP* do servidor mce-godinho.isec.pt

Tabela 2 - Filtros aplicados ao WireShark

6. Exemplos de Implementações práticas

A implementação prática desta dissertação, partiu do desenvolvimento de um *website* base, construído sem mecanismos de ocultação da navegação ou medidas extras de encriptação (exceto a exigência da utilização de *HTTPS*). Tendo como base esta versão, foram desenvolvidas outras cinco, sendo implementadas novas funcionalidades e, no final de cada versão, testadas de modo a verificar da sua funcionalidade.

O objetivo pessoal por detrás do desenvolvimento desta dissertação, foi sempre o de conseguir produzir uma versão funcional. A necessidade da implementação de diferentes versões surge devido à falta de sucesso no percurso de desenvolvimento, aos problemas que a maior parte das versões produziu, falhando o seu objetivo final. Por este motivo, as várias versões são a inclusão e mistura de vários mecanismos, na tentativa de resolução do problema.

As versões desenvolvidas foram as seguintes:

- *v1.mce-godinho.isec.pt – versão base do website em ASP.NET;*
- *v2.mce-godinho.isec.pt – versão com recurso a encriptação da query string;*
- *v3.1.mce-godinho.isec.pt – versão com recurso a encriptação do URL, através de uma classe de encriptação e redirecionamentos;*
- *v4.mce-godinho.isec.pt – versão com recurso a encriptação do URL, através de URL Rewrite utilizando a interface iHttpModule;*
- *v5.mce-godinho.isec.pt – versão com recurso a encriptação do URL, através de URL Rewrite utilizando a interface iHttpHandler;*
- *v6.mce-godinho.isec.pt – versão com recurso a encriptação do URL, através da utilização de Routing.*

6.1. BeGamer V1 – versão base

Nesta versão base do *website* serão utilizadas sessões para o carrinho de compras e *query strings* para a restante informação que transite entre páginas, como por exemplo na gestão de produtos e edição de um produto, com a passagem do *id* da base de dados. Capturando o tráfego entre o cliente e o servidor obtemos o seguinte resultado:

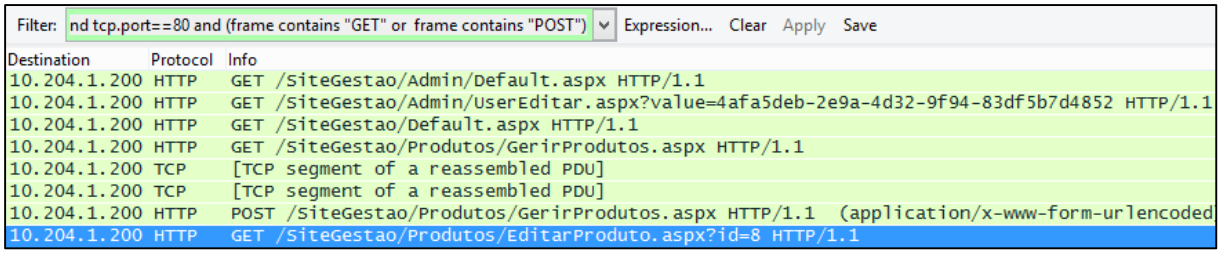

Figura 20 - Captura de tráfego HTTP ao aceder à versão 1

Na figura 20 encontram-se os pacotes de rede, que transitam entre o cliente e o servidor. É possível verificar pelo pacote assinalado com o fundo a azul, que existe um pedido *HTTP* com o *URL /SiteGestao/Produtos/EditarProduto.aspx?id=8*, de onde se pode obter a *query*

string (*id=8*), que corresponde a editarmos o produto que tem o *ID* com o valor *'8'* na base de dados.

Podemos verificar através da figura 21 que, efetuando os mesmos pedidos que foram efetuados através do protocolo *HTTP* (na figura 20), o protocolo *HTTPS* ofusca o tráfego, não sendo possível obter o tipo de método de acesso ao servidor (métodos do protocolo *HTTP*, *GET* e *POST*), nem que páginas estão a ser acedidas.

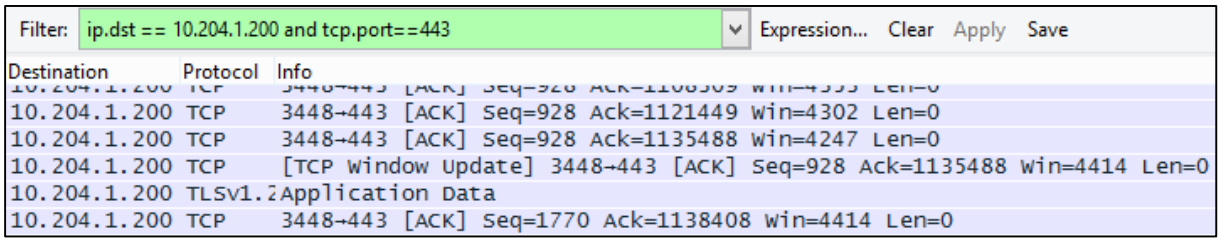

Figura 21 - Captura de tráfego HTTPS ao aceder à versão 1

Pelo resultado da captura de tráfego ao porto 443 (protocolo *HTTPS*) não é legível a informação que passa entre cliente e servidor, a que endereços o utilizador aceder, cookies, ...

Podemos concluir que, nesta dissertação, elevamos a fasquia quando falamos de ocultar a navegação, visto ser algo que se irá tentar implementar mesmo que o acesso seja por protocolo *HTTP*. Por outras palavras, partimos do pressuposto que o túnel encriptado entre o cliente e servidor (protocolo *HTTPS*) pode ser comprometido e que será possível verificar o conteúdo trocado entre os dois intervenientes. A implementação desta dissertação tenta que, mesmo nesta situação, não seja possível verificar a navegação do utilizador.

6.2. BeGamer V2 – Encriptação da *query string*

Para esta segunda versão foi criada uma classe chamada *"MestradoEncriptacao"* que irá permitir gerar chaves que serão utilizadas para encriptar e desencriptar *strings* utilizando o algoritmo de encriptação *AES* (algoritmo descrito em 4.7.3). O método de geração de chaves implementado para esta versão é o mesmo que existe nas versões posteriores.

Para gerar *strings* que servem como chaves, recorre-se ao método *Membership.GeneratePassword* que é normalmente utilizado para gerar palavras-chaves quando utilizamos o modelo *Membership* da *framework ASP.NET*. Este método serve para gerar uma *string*, recebendo como argumento dois inteiros, o primeiro dos quais define o seu tamanho e o segundo argumento o número de caracteres alfanuméricos. O mesmo método verifica se o utilizador possui uma chave atribuída; se esta não existir, uma chave será então gerada, sendo guardada na sessão do utilizador com o nome *"key"*.

```
if (HttpContext.Current.Session["Key"] == null)
{
    // Este método gera uma string de 12-chars com 4 non-alphanumeric.
    HttpContext.Current.Session["Key"] = Membership.GeneratePassword(12, 4);
}
```

É com a chave gerada para o utilizador, que encriptamos os valores dos parâmetros que constituem a *query string*. Tomando como exemplo a página de administração de produtos, quando se clica para editar um produto (contido dentro de um componente *GridView*), existe um evento afeto ao comando de editar do botão:

```
if (e.CommandName == "EditarProd")
{
    String encrypt_id =
MestradoEncriptacao.Encrypt(ProdutosGrid.DataKeys[rowIndex]["id"].ToString(),
BeGamer.GetSessionKey());
    Response.Redirect("/SiteGestao/Produtos/EditarProduto.aspx?id=" + encrypt_id, true);
}
```

Capturando o tráfego com o *WireShark*, na página de administração de produtos, quando clicamos no botão editar de um produto verificamos:

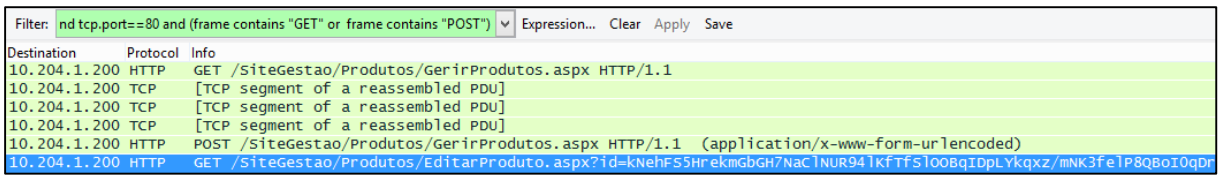

Figura 22 - Captura de tráfego HTTP ao aceder à versão 2

Foi efetuado o mesmo pedido que se encontra assinalado na Figura 20, a diferença é a encriptação do *ID* passado na *query string*.

Versão 1	*/SiteGestao/Produtos/EditarProduto.aspx?id=8*
Versão 2	/SiteGestao/Produtos/EditarProduto.aspx?id=kNehFS5HrekmGbGH7NaClNUR94lKfTfSlOOBqIDpLYkqxz/mNK3felP8QBoI0qDr

Tabela 3 - Diferença de URL entre a 1ª e 2ª versões

É possível, portanto, tendo em conta que uma chave é única para cada utilizador, ocultar a navegação de um utilizador. Imaginemos que mesmo que um atacante consiga quebrar o túnel *SSL/TLS* do protocolo *HTTPS* e deste modo obter o *URL* a que o utilizador acedeu, ao tentar aceder ao mesmo *URL*, será com uma outra chave que, a ser utilizada para desencriptar o *ID*, irá falhar em obter o valor original, não conseguindo aceder à página original.

6.3. BeGamer V3 - Encriptação do *URL* original com *Server.Response*

Nesta terceira versão foi implementada a encriptação do *URL* e através do mecanismo *Server.Response* para entregar a página pretendida pelo utilizador.

O fluxo de informação de um pedido *HTTP*, efetuado para uma página de um *website* e utilizando como exemplo o pedido efetuado nos pontos 6.1 e 6.2, será o seguinte:

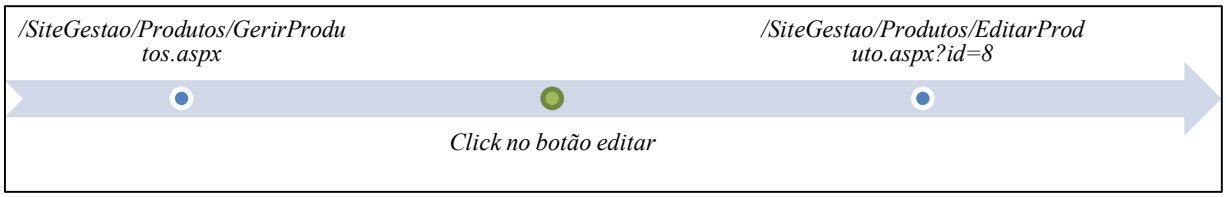

Figura 23 - Pedido e resposta a um evento de uma página ASPX

A chave de encriptação é obtida através da classe MestradoEncriptacao, gerada do mesmo modo referido no ponto 6.2. Com essa chave encriptamos o endereço original numa *string*, sendo este passado por argumento na *query string*, como valor do parâmetro *"lk"*. A página principal (*/Default.aspx*) recebe todos os pedidos com a respetiva *query string*, processa o pedido desencriptando o parâmetro *"lk"*. Se o parâmetro existir e se for válido, o pedido é processado e a página é servida ao cliente; caso contrário será servida a página inicial do *website*. Esquematicamente o fluxo de informação será o seguinte:

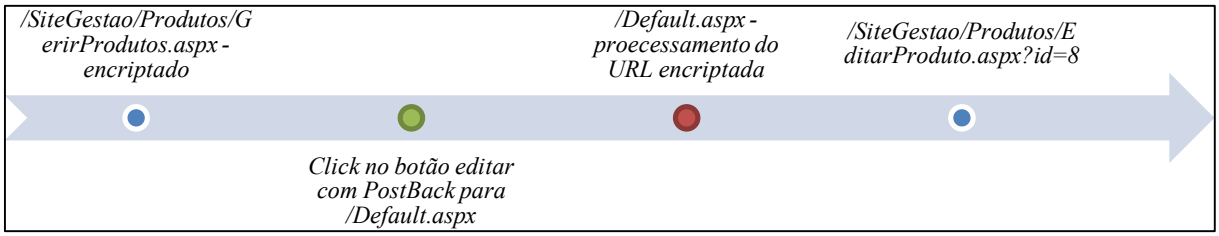

Figura 24 - Pedido e resposta na versão 3

6.3.1. Implementação

Na classe MestradoEncriptacao, criada para a implementação do ponto 6.2, foram desenvolvidas novas funcionalidades necessárias a esta versão. Foram acrescentados os seguintes métodos:

- *EncodeURL - método que encripta o URL e devolve um link para o documento inicial;*
- *CheckUrlEncrypted - método que verifica se existe um URL encriptado, isto é, se começa por /Default.aspx?lk=....;*
- *CheckDirectAccess - método que verifica se o documento é acedido diretamente.*

```
if (!IsPostBack)
{
    try
    {
        if (
            !MestradoEncriptacao.CheckDirectAccess(
                HttpContext.Current.Request.UrlReferrer.PathAndQuery.ToString()))
            Response.Redirect("/Default.aspx", true);
    }
    catch (Exception)
    {
```

```
            Response.Redirect("/Default.aspx", true);
        }
    }
```

6.3.2. O problema do *PostBack*

Tendo em conta o comportamento do mecanismo *PostBack*, explicado em detalhe no ponto 4.5, em que o formulário é submetido para o próprio documento, o modo como foi implementada esta versão e com o fluxo de informação existente, o *PostBack* torna-se problemático. Este problema do mecanismo de *PostBack* do ASP.NET acontece por este ser gerado automaticamente pela plataforma, não refletindo o *URL* que aparece na barra de endereços do *browser*. Isto é, o *PostBack* ao ser efetuado para a mesma página irá denunciar o *URL* original, visto ser efetuado para a página (documento) com o endereço não encriptado. No entanto, a alteração do comportamento dos controlos quebra o seu normal funcionamento.
Se por exemplo acedermos à página inicial do nosso *website*, os produtos são mostrados através de um controlo *Repeater*, podendo ser visualizados em detalhe clicando num botão e através de uma segunda página. Para tal, é utilizada a propriedade *OnItemCommand* do controlo *Repeater*, que permite a chamada da segunda página com o parâmetro. Utilizando o *WireShark* podemos verificar o nosso problema:

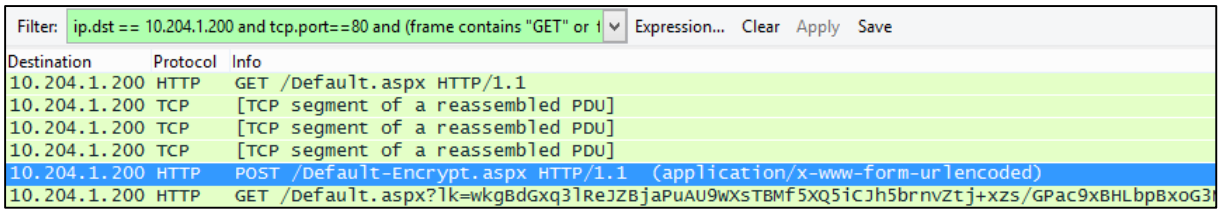

Figura 25 - Captura de tráfego na versão 3

Na Figura 25 encontra-se a sequência de pedidos trocados entre o cliente e o servidor após o *click* no botão "detalhes". O último pedido (linha) mostra um GET ao endereço correto, com o respetivo valor no parâmetro "lk" *(/Default.aspx?lk=wkgBdG...)*. No entanto, é possível verificar na penúltima linha, no *POST* assinalado a azul, que o *PostBack* em vez de ser efetuado para o *URL* existente no *browser* - */Default.aspx* -, expõe o *URL* real - */Default-Encrypt.aspx*. O mesmo problema ocorre na utilização de controlos do tipo *GridView*, se por exemplo utilizarmos a página de gestão de categorias de produtos, que permite listar as categorias existentes, editar, apagar e adicionar categorias. Todas as ações nesta página são executadas através dos eventos do tipo *OnRowCommand* ou na *DropDownList* existente dentro desta para a edição de categorias, no evento *OnSelectedIndexChanged*. Tal como na página inicial, a utilização do *PostBack* dos controlos ASP.NET, colocam a descoberto o endereço que tentamos ocultar, como podemos verificar na imagem 26 na coluna *URL*.

Request ID	Type	Status	Method	Hostname	URL
8783	xmlhttprequest	200 OK	POST	v3.mce-godinho.isec.pt	/SiteGestao/Categorias/(1
8779	xmlhttprequest	200 OK	POST	v3.mce-godinho.isec.pt	/SiteGestao/Categorias/(1
8777	xmlhttprequest	200 OK	POST	v3.mce-godinho.isec.pt	/SiteGestao/Categorias/(1
8775	main_frame	200 OK	GET	v3.mce-godinho.isec.pt	/Default.aspx 1

Figura 26 - Captura de tráfego quando se utiliza a gestão de categorias

Tornou-se óbvio que seria necessário alterar o *URL* de *PostBack* dos controlos de modo a corrigir este problema, isto é, forçar a que o *PostBack* seja efetuado para o *URL* "camuflado".

6.3.3. *PostBack* dos controlos

A primeira abordagem para solucionar este problema passou por tentar alterar o modo como cada um dos controlos se comporta em relação ao *PostBack*. No caso de botões e controlos semelhantes (*LinkButton* e *HtmlAnchor*), a abordagem passou por definir a propriedade *PostBackUrl*, podendo ser definido quer no controlo, quer no *code behind*.

```
<a href='<% Response.Write(MestradoEncriptacao.EncodeURL("/SiteGestao/Produtos/GerirProdutos.aspx"));
%>'>Gerir Produtos</a>
```

A abordagem a este tipo de controlos é simples, visto permitirem a utilização desta propriedade. No entanto, este tipo de controlos são dos mais simples que existem na plataforma ASP.NET. Os maiores problemas surgem quando se trata de controlos mais complexos, como por exemplo do tipo *GridView*.

Na área de cliente, a página de entrada mostra um histórico das encomendas do mesmo. Esta página é constituída por uma *GridView* que lista este referido histórico. Nesta página é introduzido um botão para permite visualizar os detalhes de uma determinada encomenda através do evento *OnRowCommand*.

6.3.4. Evento *OnRowCommand*

Este *website* utiliza inúmeras vezes o controlo *GridView*, sendo este um dos controlos mais frequentemente utilizados em *websites* desenvolvidos em ASP.NET. Como tal, torna-se importante detalhar o tratamento do evento *OnRowCommand*, evento que é essencial para o funcionamento das *GridViews*. Este é o evento gerado quando se clica dentro de um botão contido pela *GridView*, permitindo indicar por parâmetro um comando, o qual é utilizado para distinguir quais as instruções a serem executadas por um botão específico. O procedimento costuma ser implementado através da instrução *Switch*, que permite um conjunto de comparações, que são os vários comandos que implementados (que não são mais do que um conjunto de *strings* que de forma sugestiva costumam ser: inserir, editar ou apagar).

```
protected void GridViewHistorico_OnRowCommand(object sender, GridViewCommandEventArgs e)
{
    this.Form.Action = HttpContext.Current.Request.Url.PathAndQuery;
    switch (e.CommandName.ToString())
    {
        case "Detalhes":

Response.Redirect(MestradoEncriptacao.EncodeURL(string.Format("/ContaUtilizador/Encomendas.aspx?valu
e={0}", e.CommandArgument.ToString())));
            break;
    }
}
```

É mais fácil se descrevermos o fluxo de dados, verificando o percurso desde a página das encomendas, o clicar no botão, até à página onde se mostra ao utilizador os pormenores de uma encomenda. O fluxo de dados é descrito na figura 27.

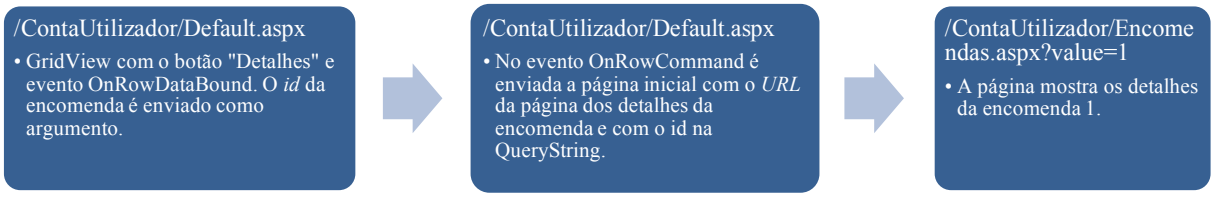

Figura 27 - Fluxo de dados para visualizar a encomenda com o id 1

O resultado da interação com o utilizador será o descrito na figura 28.

Figura 28 - Resultado de se selecionar a visualização de uma encomenda

Embora no *browser* os endereços estejam mascarados, a verdade é que uma análise do tráfego mostra que um potencial atacante tem acesso a parte do histórico da navegação. Utilizando a extensão *Web Sniffer* do Chrome, é possível verificar a resposta do *browser* ao clique no botão; podemos confirmar o resultado na figura 29.

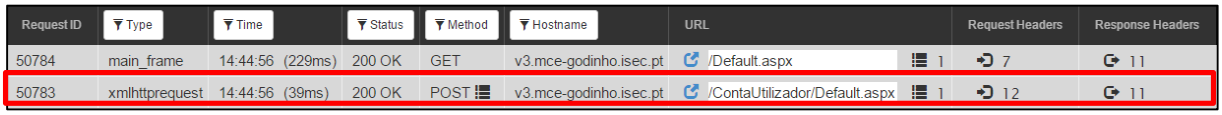

Figura 29 - Resultado do Web Sniffer quando se seleciona 'ver detalhes' de uma encomenda

Podemos verificar que, após o evento de resposta ao clique no botão de detalhes, o utilizador é enviado para a página principal (na imagem como pedido com o *ID* 50784), mas o *PostBack* que trata do evento do botão, põe a descoberto o pedido, através de *POST* de

informação para o servidor, na imagem como pedido com o *ID* 50783. Do mesmo modo, utilizando uma ferramenta de análise de tráfego de rede, neste caso o *WireShark*, obtemos o seguinte tráfego entre o PC do utilizador e o servidor:

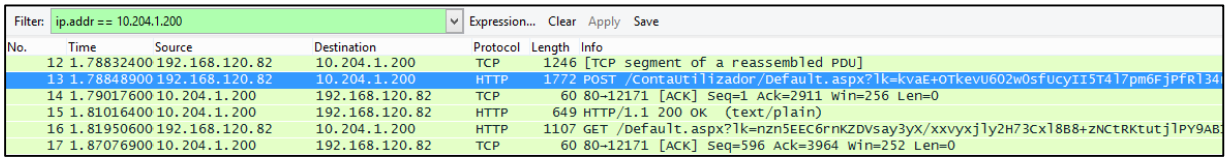

Figura 30 - Resultado do WireShark quando se selecciona 'ver detalhes' de uma encomenda

Podemos concluir que a utilização deste tipo de evento em controlos do tipo *GridView*, não permite a ocultação dos dados de navegação, tornando impossível assegurar a pretendida confidencialidade do histórico do utilizador, que se tenta implementar nesta dissertação.

6.3.5. Evento *OnRowDataBound*

No ponto anterior verificou-se que a utilização do evento *OnRowCommand* não permite ocultar a navegação devido ao mecanismo de *PostBack* dos botões que se encontram incorporados em controlos do tipo *GridView*. No entanto, é possível alterar o endereço do *PostBack* do botão de modo a tentar solucionar este problema, utilizando o método do ponto 6.2 para encriptar o endereço, e construindo a *query string* com o *ID* de cada. Ao gerar a informação de cada linha da *GridView*, quando os seus dados são preenchidos, pode realizar-se esta operação, alterando deste modo o botão *"BtnDetalhes"* modificando o endereço para o qual este efetua a submissão dos dados. O tratamento do evento *OnRowDataBound* passa então a ser o seguinte:

```
protected void GridViewHistorico_OnRowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        DataRowView drv = (DataRowView)e.Row.DataItem;
        LinkButton btnSelect = ((LinkButton)e.Row.FindControl("BtnDetalhes"));
        btnSelect.PostBackUrl =
MestradoEncriptacao.EncodeURL(string.Format("/ContaUtilizador/Encomendas.aspx?value={0}",
drv["id"].ToString()));
    }
}
```

Após esta alteração, quando se tenta verificar os detalhes de uma das encomendas, é possível verificar que é efetuado o *PostBack* mostrando corretamente o detalhe da encomenda. Verifica-se também, tanto pelo *Web Sniffer* como pelo *WireShark*, que a navegação é ocultada corretamente, sendo apenas visível o acesso à página principal do *website* (/Default.aspx), como podemos verificar pelas figuras 31 e 32.

Tab	Request ID	Type	Time	Status	Method	Hostname	URL	Request Headers	Response Headers
× 939	56490	main_frame	16:16:41 (222ms)	200 OK	POST	v3.mce-godinho.isec.pt	/Default.aspx 1	9	11
× 939	56488	stylesheet (cache)	16:16:36 (17ms)	200 OK	GET	v3.mce-godinho.isec.pt	/css/jquery.dataTables.min.cs	7	10
× 939	56487	main_frame	16:16:35 (768ms)	200 OK	GET	v3.mce-godinho.isec.pt	/Default.aspx 1	7	11

Figura 31 - Resultado do Web sniffer após alteração do OnRowCommand

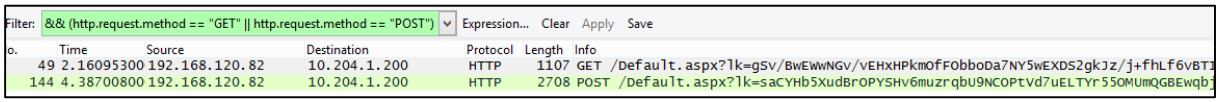

Figura 32 - Resultado do WireShark após alteração do OnRowCommand

Na inserção de produtos, esta situação foi resolvida através da utilização de um *UpdatePanel* e colocação do conteúdo do formulário dentro deste. Deste modo, a plataforma assegura o correto *PostBack*, embora simplificado por se tratar de um *PostBack* parcial da página. No entanto, novo problema: o controlo *FileUpload* (utilizado para carregar a imagem do produto) deixa de funcionar corretamente. Foi necessário adicionar ao *UpdatePanel*, um *Trigger*, associado do botão de submissão do produto.

```
<Triggers>
          <asp:PostBackTrigger ControlID="btInsertProd" />
</Triggers>
```

6.3.6. *PostBack* da página completa

Numa página ASP.NET ao ser gerada para o cliente, isto é, ao ser gerado o *HTML* de resposta ao pedido do *browser*, é incorporado um elemento do tipo *form*, que envolve o conteúdo do corpo da página e que indica o endereço para onde deve ser submetido o conteúdo. Neste ponto foi verificado algo bastante importante: qualquer tentativa de ocultação do *URL* teria obrigatoriamente de passar por alterar o endereço de submissão do formulário, caso contrário, o *URL* original é incluído no *HTML* e legível por qualquer interveniente. É possível verificar este problema através da propriedade *action* do elemento *form*.

```
<body class="l-body headertype_sticky headerpos_top">
    <form method="post"
action="SiteGestao/Categorias/GerirCategorias.aspx?lk=kTJwXJl5hWNnNuaXnX3y%2fpDvOsro92paRxtpua747VNY
iKsLiDxeQykEG2PSWYn7Ihg9pfjnWPjiV9Ql6LN1NSFIkd5W7dUCeuZTxGYIkk8%3d" id="formMain">
```

O modo de alterar o *URL* do formulário tem de ser efetuado no evento *Page_Load*, isto é, mal a página é gerada, com o seguinte comando:

```
Form.Action = HttpContext.Current.Request.Url.PathAndQuery;
```

Tomando esta opção, partiu-se para a implementação da solução desta versão. Infelizmente e mais uma vez, o modo como a plataforma ASP.NET gera o código, limita a implementação das opções de *PostBack* e de reação a eventos.

Tomando como exemplo a página de edição de um produto: o seu fluxo normal de dados seria um *PostBack* para o próprio documento, que trataria do evento do clique do botão, alterando os dados na base de dados. Uma simples análise de tráfego confirma o normal funcionamento, como se verifica na figura 33.

Figura 33 - Resultado do PostBack obtido por Web Sniffer

De forma esquemática, o comportamento de uma página em ASP.NET sem as alterações realizadas no âmbito desta dissertação, teria o fluxo de dados descrito na figura 34.

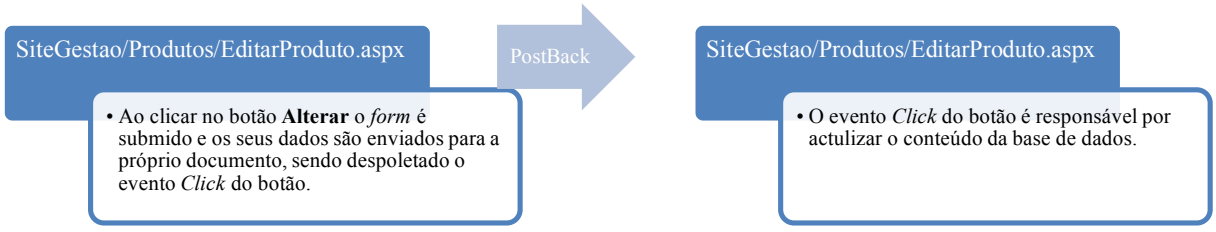

Figura 34 - Fluxo de dados do PostBack de uma página ASP.NET

Com as alterações implementadas nesta dissertação, como o endereço de submissão do formulário, o fluxo de dados será o descrito na figura 35.

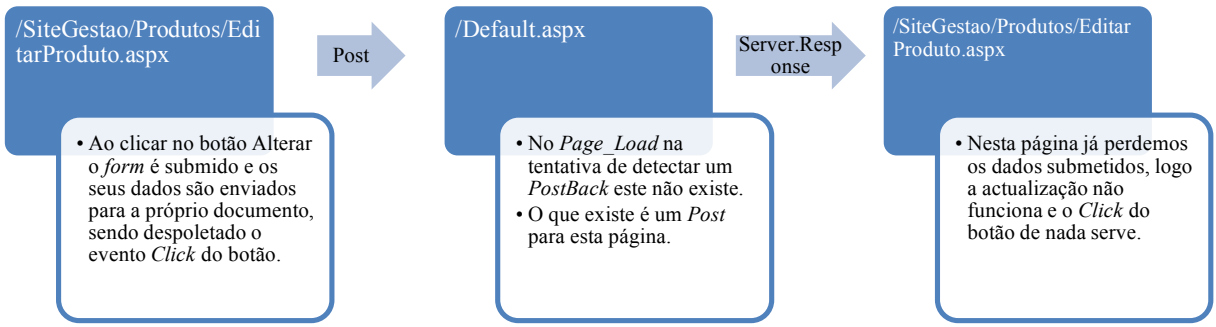

Figura 35 - Fluxo de dados do PostBack após a alteração no Page_Load

Como se pode verificar pelas figuras 34 e 35, a alteração do endereço de submissão do formulário é indispensável. Numa análise no caso de esta funcionalidade não ser implementada, e em caso de quebra da encriptação do *HTTPS*, o endereço ficaria exposto no código *HTML* do corpo da página.

```
<form method="post" action="EditarProduto.aspx?id=1" onsubmit="javascript:return
WebForm_OnSubmit();" id="formMain" enctype="multipart/form-data">
```

No entanto, com esta alteração, ainda que na análise do tráfego se verifique que se conseguiu mascarar o tráfego, as alterações anulam as funcionalidades dos controlos das páginas. Com o *PostBack* a ser efetuado para a página principal do *website*, e posteriormente efetuar o redirecionamento, a página final não é aquela para onde foi submetido o formulário. Deste modo, perdem-se os dados submetidos da página original.

Como é óbvio, alterando o modo como se comportam os controlos da plataforma ASP.NET, seria de esperar que algumas funcionalidades não funcionassem em pleno, ou que outras não funcionassem e ainda que algumas, para além de não funcionarem, pudessem dar erros. A utilização de eventos de paginação é um exemplo das funcionalidades que não funcionam. Na segunda versão (no ponto 6.2) a paginação era efetuada sem qualquer tipo de problema e com recurso à propriedade *EnableSortingAndPagingCallbacks*: no entanto esta opção apenas funciona se a *GridView* apenas contiver *BoundFields*. Se, e como é o caso desta implementação, os campos tiverem sido alterados, como acontece com o elemento da imagem em que este campo é *array* de *bytes* carregado para o elemento *img* de *HTML*, a referida opção da *GridView* deixa de funcionar. Nesta implementação, quando se tenta utilizar a paginação da *Gridview*, é possível observar o seguinte erro na consola do *browser*:

```
Uncaught Sys.WebForms.PageRequestManagerServerErrorException:
Sys.WebForms.PageRequestManagerServerErrorException: Input string was not in a correct format.

ScriptResource.axd?d=DT3YJR8QaqV61-teuz0hghG2OQ5yPPI0saNvHryenZtL7jkst-
67GSq_CnqDBJN96iOMJE7hHPvtQ5cCfPb0Ya1_HJ_RzAMHeRht-
CxhXRrQbSOI8lEuKGHwm3U0FjKB28jYzEU5LBSM_fM037WrAg2&t=ffffffffda74082d
```

A solução encontrada para ultrapassar este problema consistiu em carregar todos os campos para a *GridView* e na utilização de *DataTables* em *JQuery*. No entanto, esta solução é impraticável quando a quantidade de registos for significativa. Do mesmo modo, na utilização do tratamento do evento *OnSelectedIndexChanged* não funciona. Nas páginas de edição de utilizador ou na página de gestão das categorias, para além de componentes do tipo *GridView* este evento é utilizado em controlos *DropDownList* e *CheckBox*. Ao utilizarmos este evento nos referidos controlos, podemos verificar o seguinte erro na consola do *browser*:

```
Uncaught Sys.WebForms.PageRequestManagerParserErrorException:
Sys.WebForms.PageRequestManagerParserErrorException: The message received from the server could not
be parsed.
```

A questão do *PostBack* tornou-se problemática devido ao tempo necessário para resolução de todos os problemas que surgiram. Um dos exemplos é a página que permite adicionar um produto. Ao submeter o formulário o *PostBack* envia para o documento principal

(*/Default.aspx*), não sendo efetuada a inserção na base de dados. Foram tentadas as seguintes soluções, sem sucesso:

- Verificação dos valores *Page.PreviousPage, Page.PreviousPage.IsPostBack e Request.Form*; apenas este último nos devolve algum conteúdo;
- Alteração do *URL* de *PostBack*, atrás descrita. O endereço de *PostBack* é a página principal sem o parâmetro "lk";
- Utilização da propriedade *PostBackUrl* no botão de submissão do novo produto*;*
- Verificação no *PageLoad* se existiu ou não um *PostBack* e tratar os dados do formulário.

Em termos práticos a inserção de produtos deixa de funcionar, mas a funcionalidade de apagar através da *GridView* funciona, sem mostrar, no entanto, mensagens de aviso ao utilizador. Outras áreas da página também deixam de funcionar devido a esta alteração. O mecanismo de *PostBack* em ASP.NET tem entre outras a vantagem de mostrar mensagens ao utilizador (tal como descrito no ponto 4.5), como por exemplo mensagens de sucesso ou de erro quando se tenta inserir um produto na base de dados.

6.3.7. Resultado da implementação

A alteração do mecanismo de *PostBack* nos controlos invalida o normal funcionamento de controlos dos tipos *GridView*, *DropDownList* e *CheckBox*, não permitindo a ocultação dos dados de navegação e frequentemente quebrando as funcionalidades dos próprios controlos, como, por exemplo, na funcionalidade de paginação das *GridViews*, sendo necessário muito mais trabalho para conseguir implementar esta funcionalidade, neste caso através de *JQuery*. Embora este tipo de implementação solucionasse o problema da ocultação do *URL*, não pode ser aplicada a todo o tipo de controlos e é propensa a erros. É preciso ter em conta que basta um controlo que não tenha esta solução implementada para deitar por terra todo o esforço de ocultação da informação. Torna-se necessário procurar uma solução mais abrangente. A alteração do *PostBack* do formulário da página também tem o efeito nefasto de não permitir a normal interação com o utilizador e de várias áreas da página simplesmente deixarem de funcionar.

Podemos concluir que esta implementação, não produz uma solução viável para os objetivos desta dissertação, visto não só não produzir a ocultação da navegação, como em muitos casos quebrar a função da própria página, para além de deixar de permitir a interação com o utilizador.

6.4. BeGamer V4 - *HttpModule*

A Implementação do ponto 6.2 partiu da encriptação dos parâmetros da *query string*. Este trabalho exige o cuidado de encriptar todas as áreas em que os *ID's* são enviados para o utilizador e efetuar o processo inverso quando se recebe os valores do utilizador, utilizando-os posteriormente em pedidos efetuados às bases de dados. Se existir apenas um erro, poderá ser o suficiente para expor um *ID* do cliente que efetuou o pedido ao *website*.

No ponto 4.1 foram referidos os elementos do *pipeline* do *IIS* e no ponto 4.2 foram descritos os elementos *HTTP Module* e *HTTP Handler* e como estes componentes intercetam o tráfego, podendo ser utilizados para processarem o seu conteúdo. Nesta implementação, foi construído um *HTTP Module*, que permita intercetar o pedido à entrada com o *ID* encriptado e que o desencripte ao passar os dados para o cliente.

Para a construção do *HTTP Module* é necessário o seguinte:

- *criar uma classe que implemente a interface IHttpModule, dentro da pasta App_Code;*
- *registar a classe no IIS para o módulo ficar ativo. Nesta implementação e como se trata de um sistema moderno, assumimos que o IIS corre apenas em modo integrado, ao invés de versões antigas do IIS. Para o registo do módulo é necessário incluir na configuração da aplicação no web.config:*

```
<system.webServer>
   <modules>
      <add name="URLRewriter" type="URLRewriter" />
   </modules>
</system.webServer>
```

6.4.1. Módulo URLRewriter

Ao implementar a interface *iHTTPModule*, é obrigatório definir os métodos *Init* e *Dispose*, sendo no método *Init* que registamos os eventos que o nosso módulo irá escutar. Na Figura 3 encontra-se referido (de forma incompleta) a sequência de eventos do *HTTP pipeline*, em que o primeiro evento é o *BeginRequest*, sendo neste evento que se regista o módulo, através do seguinte código:

```
public void Init(HttpApplication context)
{
    context.BeginRequest += new EventHandler(Application_BeginRequest);
}
```

Esta linha de código determina que o método *Application_BeginRequest* é executado quando o evento *BeginRequest* é despoletado. Este método, construído para esta versão, recebe um *buffer* de dados e através expressões regulares, efetua a substituição do conteúdo numérico que se encontra após a *string "id="*, este número é o *ID* da base de dados, é encriptado e substituído no conteúdo que é enviado para o utilizador.

No ponto 2.2 é descrito o mecanismo de *URL ReWrite* e a sua utilização tradicional. O principal problema nesta implementação advém do facto de se utilizarem chaves que são geradas para cada utilizador, sendo estas guardadas na sessão do utilizador. No decorrer desta implementação, percebeu-se que é impossível aceder à sessão do utilizador nos eventos iniciais

do *pipeline*, visto que a utilização deste mecanismo é apenas executada no início da geração da página, logo torna-se impossível aceder a estas chaves.

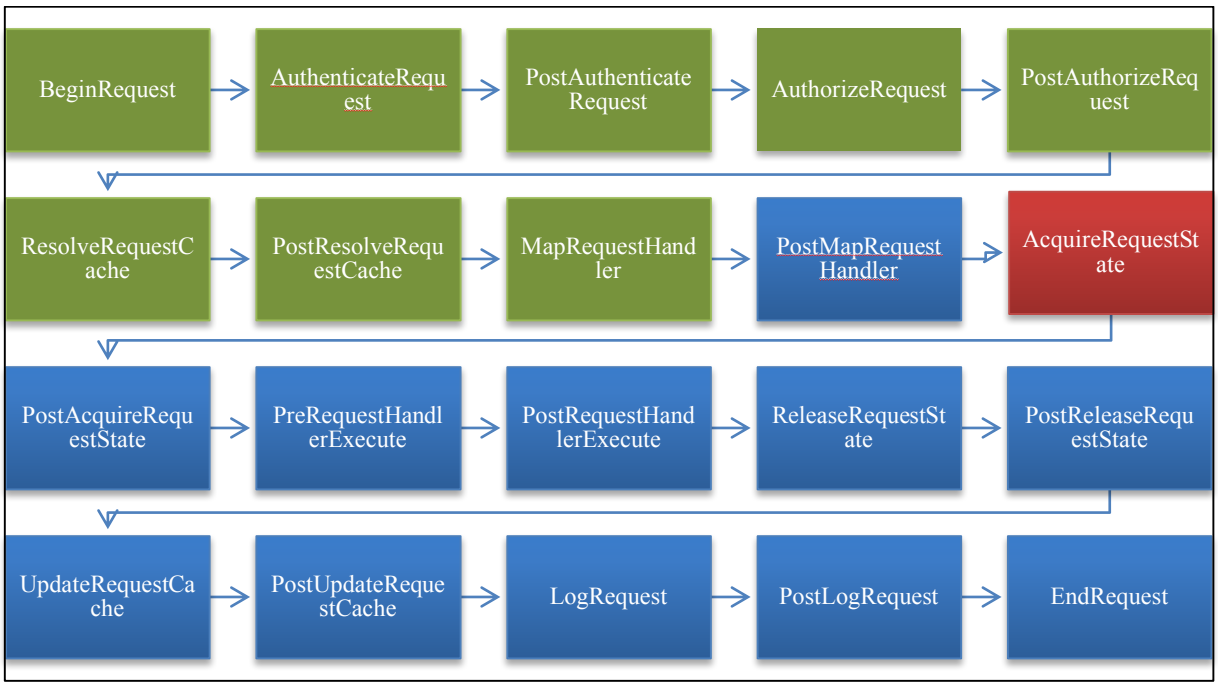

Figura 36 - Eventos da HTTP Application pipeline em que o ReWrite funciona

Na Figura 36 foi assinalado a verde os eventos em que foram efetuados com sucesso *URL ReWrite*. O objetivo desta dissertação de mestrado é utilizar chaves diferentes/independentes por cliente, sendo esta chave gerada e guardada na sessão do utilizador; portanto a utilização do mecanismo de *URL ReWrite* por si só não cumpre o nosso objetivo. No entanto, a sessão do utilizador apenas fica disponível no evento *AcquireRequestState*, assinalado a vermelho na Figura 36, sendo demasiado tarde para se alterar o mapeamento dos *URLs*, tornando o uso do *URL ReWrite* e desta implementação impossível.

6.4.2. Implementação com chave fixa

Para ser possível testar este tipo de implementação que poderá ser utilizada numa noutra circunstância, criou-se um campo na configuração do projeto com uma chave que foi gerada e gravada no ficheiro *web.config*.

```
<appSettings>
  <add key="EncKey" value="HA6]1$j{%^2)" />
</appSettings>
```

O módulo criado processa o *stream* de dados, utilizando expressões regulares para verificar se existem parâmetros com o padrão *"id=NÚMERO-INTEIRO"*, substituindo o

número inteiro por uma *string* encriptada, como referido na Versão 3. Tal como na referida implementação, os controlos não reagem corretamente aos eventos de *PostBack*.

Não foi impossível colocar esta implementação em funcionamento, mesmo abdicando da necessidade da utilização de chaves dinâmicas, o que por si só já invalidaria qualquer solução encontrada. Do mesmo modo, foi decidido abandonar a tentativa de implementação através de um *HTTP Handler* (seria a Versão 5), visto que os mesmos problemas que se encontraram na implementação com um módulo seriam os mesmos problemas com um *HTTP handler*.

6.5. BeGamer V6 - *Routing*

A impossibilidade do normal funcionamento dos controlos e utilização do *URL* encriptado nas versões anteriores, e tendo em conta o objetivo da ofuscação da navegação, deram origem à ideia de utilizar a funcionalidade de *Routing* para atingir este objetivo, funcionalidade descrita no ponto 4.9.

Por norma, as rotas são adicionadas através do evento *Application_Start* no ficheiro *Global.asax*. Esta abordagem permite que as rotas estejam disponíveis quando a aplicação é iniciada. O exemplo seguinte demonstra uma rota a ser adicionada que define dois parâmetros enviado por *query string*, com os nomes *action* e *categoryName*. Os pedidos para o *URL* com este padrão são enviados para a página que existe com o nome *Categories.aspx* (Microsoft, 2015b):

```
protected void Application_Start(object sender, EventArgs e)
{
    RegisterRoutes(RouteTable.Routes);
}

public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapPageRoute("",
        "Category/{action}/{categoryName}",
        "~/categoriespage.aspx");
}
```

6.5.1. Implementação

Nas diferentes versões da implementação prática desta dissertação, foram testadas várias hipóteses que utilizam chaves de encriptação geradas para cada utilizador, quando este inicia a navegação no *website* (descrito no ponto 6.2). Utilizando o *Routing*, foi tentada uma implementação com o mesmo efeito da implementação do ponto 6.3. O seu objetivo é encriptar o *URL* relativo (*URL* sem o domínio, mas com a *query string*) e acrescentar a extensão *.aspx*. A extensão é acrescentada de modo a que o pedido seja encaminhado para o *HTTP handler* que processa os documentos .aspx, sem a necessidade de forçar o reencaminhamento do pedido.

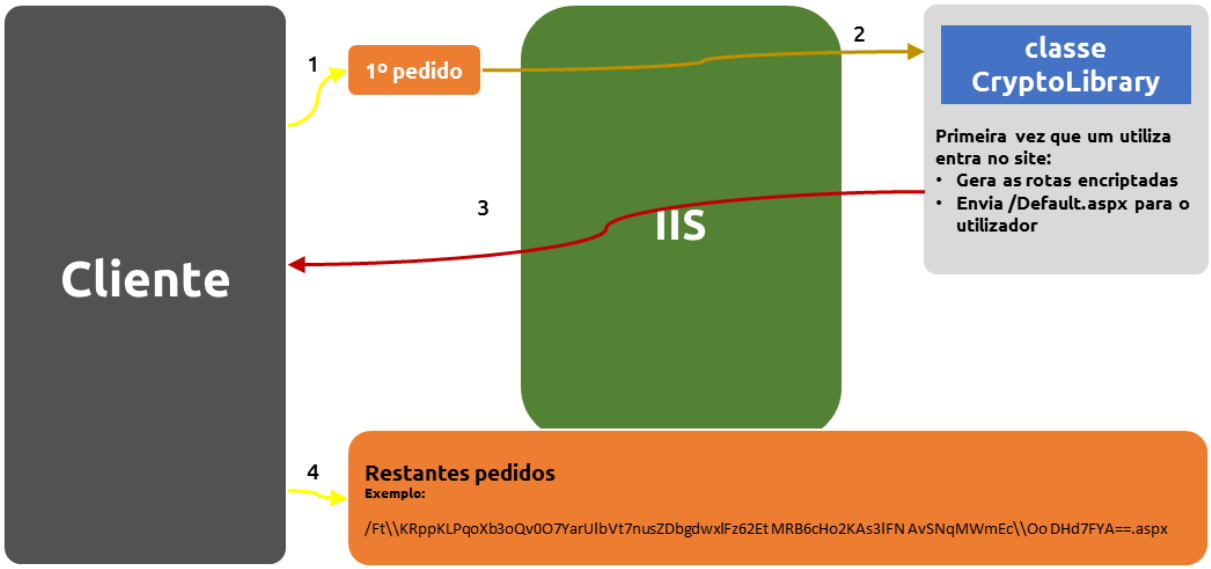

Figura 37 - Fluxo de dados utilizando Routing e encriptação

6.5.2. Encriptação

Quando o utilizador inicia a navegação no *website*, é verificada a existência de uma sessão válida e de uma chave de encriptação que se encontra guardada na sessão. Caso esta não exista, é pressuposto que o utilizador está a iniciar a sua navegação no *website*, como tal, é gerada uma chave de encriptação. É neste ponto, e imediatamente após a geração da chave do utilizador, que é definido o *Routing*, isto é, que são criadas as rotas do utilizador.

```
public static String EncryptKey()
{
    if (HttpContext.Current.Session["Key"] == null)
    {
        HttpContext.Current.Session["Key"] = Membership.GeneratePassword(12, 4);
        AdicionairRouting();
    }
    return HttpContext.Current.Session["Key"].ToString();
}
```

A implementação da encriptação nesta dissertação, desde a sua versão 3, utiliza um mecanismo chamado *salt (sal)*, que é um conjunto de *bytes* que são introduzidos no algoritmo para gerar as *strings* encriptadas, sendo gerado um *sal* diferente para cada chave, tal como referido na Figura 14. O resultado final da utilização destes *bytes* a gerar conteúdo encriptado, é que, mesmo quando é utilizada a mesma chave em conjunto com o mesmo conteúdo a encriptar cem vezes consecutivas, é possível obter cem resultados diferentes. Como é óbvio tratando-se de um algoritmo de encriptação de chave simétricas, o resultado da desencriptação é o mesmo para todos os resultados que saem do algoritmo da encriptação. Este é um grande problema no início desta implementação: é que ao serem geradas as rotas, é necessário que o endereço a que o utilizador acede esteja registado nas rotas, de modo a que o pedido seja

encaminhado para a página de destino. Se o resultado da encriptação for diferente a cada iteração, o resultado é completamente imprevisto e nunca irá resultar com sucesso. Se por exemplo, ao ser encriptado duas vezes o endereço *"/Autenticacao/Login.aspx"* o resultado de saída é diferente das duas vezes, isto quer dizer que o *link* na página entregue ao utilizador, será diferente da rota existente que permitir aceder a essa página. O utilizador ao tentar aceder ao *link* que existe na página ocorre um erro do tipo 404 (*not found*). É, portanto, necessário alterar o mecanismo de encriptação de modo a que, para uma mesma *string* de entrada, em conjunto com uma mesma chave, a saída seja sempre igual.

6.5.3. Caracteres permitidos no *URL*

Como está a ser utilizado um algoritmo de encriptação, a *string* da saída gera todo o tipo de caracteres e a alguns destes não é permitida a sua utilização no *URL*. As *RFCs* 1738 (*Request for Comments 1738 - Uniform Resource Locators (URL)*) e *RFC* 3986 (*Request for Comments 3986 - Uniform Resource Identifier (URI): Generic Syntax*) descrevem o conjunto de caracteres que são permitidos, e aqueles que são considerados reservados. Existem caracteres reservados que quando utilizados resultam em erros do tipo 404. A utilização do caracter *'/'* indica uma pasta, também introduzindo erros do mesmo tipo, por pesquisa de documentos em pastas não existentes, impossibilitando esta implementação.

Não seria correto, implementar um algoritmo de encriptação com desempenho esperado de ocultação do conteúdo original e ao mesmo tempo restringir o número de caracteres que a sua saída poderia conter. Este tipo de restrição diminui a robustez do mecanismo de encriptação, mas é imperativa para esta implementação. O resultado final foi correr uma simples substituição de caracteres na *string* de saída, de modo a que estes caracteres não sejam por ela contida.

```
//Encriptação
return CipherUtility.Encrypt<AesManaged>(plainText, HttpContext.Current.Session["Key"].ToString(),
"salt").Replace("/", "~").Replace("+", "-");

//Desencriptação
ciphertext = ciphertext.Replace("~", "/").Replace("-", "+");
return CipherUtility.Decrypt<AesManaged>(ciphertext, HttpContext.Current.Session["Key"].ToString(),
"salt");
```

Do mesmo modo é impossível encriptar o *URL* completo incluindo a *query string*, na edição de produtos: na edição de produto o *ID* é encriptado e enviado por *query string*.

Originalmente o código era:

```
String encrypt_id = MestradoEncriptacao.Encrypt(ProdutosGrid.DataKeys[rowIndex]["id"].ToString(),
BeGamer.GetSessionKey());
Response.Redirect("/SiteGestao/Produtos/EditarProduto.aspx?id=" + encrypt_id, true);
```

Esta implementação apenas poderia funcionar se todos os resultados com os diversos *ID's* fossem adicionados às rotas. Por exemplo, o produto final da encriptação dos *URLs* *EditarProduto.aspx?id=123* e *EditarProduto.aspx?id=456* é totalmente diferente, sendo necessário que a parte da *query string* seja adicionada à parte comum do *URL* e que deste modo exista na lista de rotas. A opção final recaiu por encriptar as duas partes (*URL* e *query string*) de forma independente, juntando as duas para formar o *URL* final. O exemplo seguinte é utilizado na gestão de produtos, quando se clica no botão editar produto:

```
String encrypt_id = CryptoLibrary.Encrypt(ProdutosGrid.DataKeys[rowIndex]["id"].ToString());
Response.Redirect(CryptoLibrary.EncodeURL("/SiteGestao/Produtos/EditarProduto.aspx")+"?id=" +
encrypt_id);
```

Deste modo, a parte referente aos parâmetros da *query string* como que são encriptados duas vezes. Uma primeira vez tal como no ponto 6.2, e uma segunda vez o *URL* completo com a *query string*.

6.5.4. Definição de rotas

Como mostrado na Figura 37, a adição de rotas é efetuada no instante em que é gerada a chave de encriptação. O método que gera a *password* de encriptação é chamado diretamente em todas as páginas, através da *Master Page* do *website* e indiretamente em todas as páginas, visto que vários outros métodos e/ou secções das páginas, necessitam da *password* e acabam por chamar o método que a vai gerar. O método que gera as rotas efetua uma listagem de todos os ficheiros com a extensão *.aspx* existentes na aplicação *web*, efetuando o seu mapeamento. É um método recursivo, que percorre todos os diretórios e para cada um dos documentos encontrados gera a rota. O método seguinte foi o desenvolvido:

```
private static void GetFiles(DirectoryInfo info)
{
    foreach (DirectoryInfo info1 in info.GetDirectories())
    {
        GetFiles(info1);
    }

    foreach (FileInfo finfo in info.GetFiles())
    {
        if (finfo.FullName.EndsWith(".aspx"))
        {
            String urlClear = "/" + finfo.FullName.Substring(IniStr).Replace("\\", "/");

            var charsToRemove = new string[] { " ", "/", ".aspx" };
            String routeName = urlClear;
            foreach (var c in charsToRemove)
            {
                routeName = routeName.Replace(c, string.Empty);
            }
            RouteTable.Routes.MapPageRoute(routeName, EncodeURL(urlClear), "~/" + urlClear);
```

```
        }
    }
}
```

O modo como o nome de cada uma das rotas é gerado, é o caminho completo para o ficheiro mais o seu nome, sem a sua extensão e sem caracteres especiais. Exemplo de rotas geradas:

Nome	Endereço
AutenticacaoLogin	/Autenticacao/Login.aspx
SiteGestaoAdminUserApagar	/SiteGestao/Admin/UserApagar.aspx
SiteGestaoProdutosEditarProduto	/SiteGestao/Produtos/EditarProduto.aspx

Tabela 4 - Definição dos nomes das várias rotas

O primeiro problema encontrado vem da dificuldade de ser necessário efetuar uma verificação das rotas existentes. O ASP.NET não permite efetuar facilmente *debug* das rotas existentes ou até simplesmente o *output* numa página das rotas existentes e das suas definições. Para tal foi implementada uma página (*http://v6.mce-godinho.isec.pt/ListarFiles.aspx*) que mostra para uma determinada sessão, a chave de encriptação e os comandos enviados para a plataforma para definição das rotas. Foi instalada uma aplicação chamada Glimpse, que é uma plataforma que permite efetuar diagnósticos à plataforma ASP.NET, tornando-se útil para este trabalho, sendo apenas utilizada para permitir obter alguma informação sobre as rotas existentes.

6.5.4.1. Problema da duplicação de rotas

Quando um segundo cliente efetua o seu primeiro pedido ao *website*, verifica-se que as rotas para ele definidas não funcionam, quando este tenta aceder aos endereços encriptados que lhe são gerados, é gerado um erro 404. Foi possível perceber que as rotas a partir do primeiro utilizador não eram definidas. Este problema deriva de não puderem existir duas regras com o mesmo nome. Como tal será necessário criar rotas unívocas para cada utilizador. A solução encontrada, consistiu em gerar uma pequena chave para cada cliente que se liga ao *website*, juntando esta chave ao nome de cada rota, criando assim nomes diferentes para cada utilizador.

```
HttpContext.Current.Session["ClientID"] = Guid.NewGuid().ToString();

RouteTable.Routes.MapPageRoute(routeName + Session["ClientID"], EncodeURL(urlClear), "~/" +
urlClear);
```

6.5.4.2. Poluição das rotas

Após alguns pedidos de vários utilizadores, é possível verificar que existem rotas de utilizadores que já abandonaram o *website*, as rotas adicionadas no início da sessão do utilizador, permanecem ativas e não são retiradas.

É necessária a criação de um mecanismo de identificação das rotas, que permita identificar e eliminar as rotas dos utilizadores que já abandonam o *website* e que, portanto, já não são requeridas. Deste modo, foi alterado o nome das rotas (que já tinham sido alteradas no ponto anterior) de modo a que sejam identificadas com maior facilidade. A solução encontrada foi a de alterar o nome de cada rota, juntando o *ID* da sessão do utilizador:

```
RouteTable.Routes.MapPageRoute(routeName + HttpContext.Current.Session.SessionID,
EncodeURL(urlClear), "~/" + urlClear);
```

Para retirar as rotas desnecessárias existiam duas hipóteses distintas:

- *Iterar pelas sessões existentes e verificar as rotas que contêm ID's se sessões que já não estão ativas, implementando um mecanismo que corra ciclicamente.*
- *Limpar as rotas de um utilizador quando este abandona o website.*

A segunda opção é, sem dúvida, a opção mais lógica. O modo prático de o implementar será quando as rotas deixam de ser necessárias, isto é, quando o utilizador abandona o *website* ou quando a sua sessão expirar. Deverá ser no evento relativo ao fim da sessão que esta lógica vai ser implementada. Então, no ficheiro *Global.asax* e no evento *Session_End*, é implementado o método que irá retirar as rotas, sendo idêntico ao que as adiciona, apenas mudando a linha que as adiciona para o código que efetua a sua remoção, pelo seu nome:

```
void Session_End(object sender, EventArgs e)
{
    RemoverRouting();
}
…
RouteTable.Routes.Remove(RouteTable.Routes[routeName + HttpContext.Current.Session.SessionID]);
```

O modo de resolução deste problema exige a definição no ficheiro *web.config*, do modo de estado de sessão *InPRoc*. Este é o modo por omissão, e o único que permite o acesso aos eventos *Session_End e Session_OnEnd*:

```
<sessionState timeout="60" mode="InProc" />
```

6.5.5. Análise de tráfego

Após a alteração do *website* de modo a suportar os *URL*s encriptados, foi efetuado um teste de navegação do *website* e respetiva captura de tráfego para verificar o sucesso desta implementação. O resultado a análise de tráfego é a seguinte:

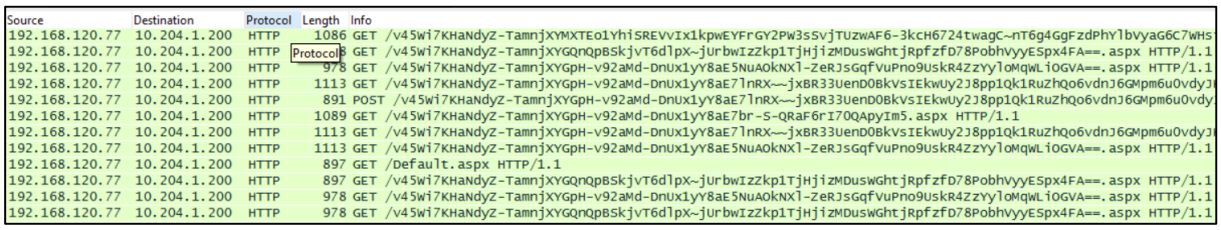

Figura 38 - Captura de tráfego na versão 6

É possível verificar que todos os pedidos e respostas são efetuados de e para endereços encriptados de forma correta, como se pode verificar pela captura de tráfego. O seu funcionamento é o mais transparente de todas as implementações até agora tentadas, com o *PostBack* a funcionar sem problemas. Do mesmo modo, quando se observa o código fonte da página, é possível observar que o endereço de *PostBack* do formulário está devidamente ocultado:

```
<body class="l-body headertype_sticky headerpos_top">
<form method="post"
action="./373kYKBNjV7Fahe63JyzTCGcPs12s8vUbpodExph9q0VBBEprIhUMOAnYc~qkNbbiaGdn2YYBz7oaXKtDzc7xQ==.a
spx" id="formMain">
```

No entanto, e este ponto é crítico na implementação desta dissertação, é necessário ter em conta que, quando uma rota existe, esta existe para qualquer cliente que aceda ao *website*; mesmo que possua uma chave de encriptação diferente poderá aceder a uma página através de uma rota existente e que tenha sido gerada para outro cliente. Será então necessário criar um mecanismo que permita verificar se uma rota pertence ao cliente que está a aceder a uma página.

6.5.6. Restrição rota-cliente

Como foi exposto no ponto 6.5.4, a manipulação de rotas em ASP.NET é um processo complicado, embora a sua adição seja fácil, não existe uma forma de iterar pelas rotas existentes. O seu acesso é complicado visto existir uma indexação interna para a plataforma e não para o utilizador. Existem algumas ferramentas que o poderiam permitir, mas apenas funcionam em ASP.NET MVC (Roth, 2015). No entanto existem algumas implementações (Haack, 2008) que permitem alterar o modo como as rotas são definidas, permitindo consultar se uma rota existe. Deste modo, foi alterado o modo de definição de rotas, passando a ser efetuada através da instrução *MapPageRouteWithName*, que adiciona a rota e permite a utilização do método *GetRouteName* nas restantes páginas, o que irá colmatar a lacuna que existia na anterior implementação.

```
public static void MapPageRouteWithName(RouteCollection routes, string routeName, string routeUrl,
string physicalFile, bool checkPhysicalUrlAccess = true,
        RouteValueDictionary defaults = default(RouteValueDictionary), RouteValueDictionary
constraints = default(RouteValueDictionary), RouteValueDictionary dataTokens =
default(RouteValueDictionary))
```

```
{
    if (dataTokens == null)
        dataTokens = new RouteValueDictionary();

    dataTokens.Add("route-name", routeName);
    routes.MapPageRoute(routeName, routeUrl, physicalFile, checkPhysicalUrlAccess, defaults,
constraints, dataTokens);
}

public static string GetRouteName(RouteData routeData)
{
    if (routeData.DataTokens["route-name"] != null)
        return routeData.DataTokens["route-name"].ToString();
    else return String.Empty;
}
```

A utilização destes métodos permite que seja possível dentro de cada página verificar se a rota que orienta o pedido para aquela página, contém o *ID* da sessão do utilizador. Para efetuar este processo foi acrescentado o seguinte código a cada página:

```
if (!CryptoLibrary.GetRouteName(this.RouteData).Contains(Session.SessionID))
{
    Server.Transfer("/Default.aspx");
}
```

Esta solução, sendo necessária à sua repetição por todas as páginas do *website*, é uma solução eficaz, mas pouco elegante. Deste modo, foi acrescentado à *master page* do *website* o seguinte código:

```
if
(!CryptoLibrary.GetRouteName(HttpContext.Current.Request.RequestContext.RouteData).Contains(Session.
SessionID))
{
    Server.Transfer("/Default.aspx");
}
```

Infelizmente este código não produziu os resultados esperados. Sem qualquer tipo de *output* para o programador e sem possibilidade de verificar o que se passou de errado o *website* deixou de responder. Ao observar o servidor onde corre o *website* foi possível observar que o processo *w3wp.exe* produziu um erro e no *EventViewer* do servidor é também possível observar que foi originado um erro.

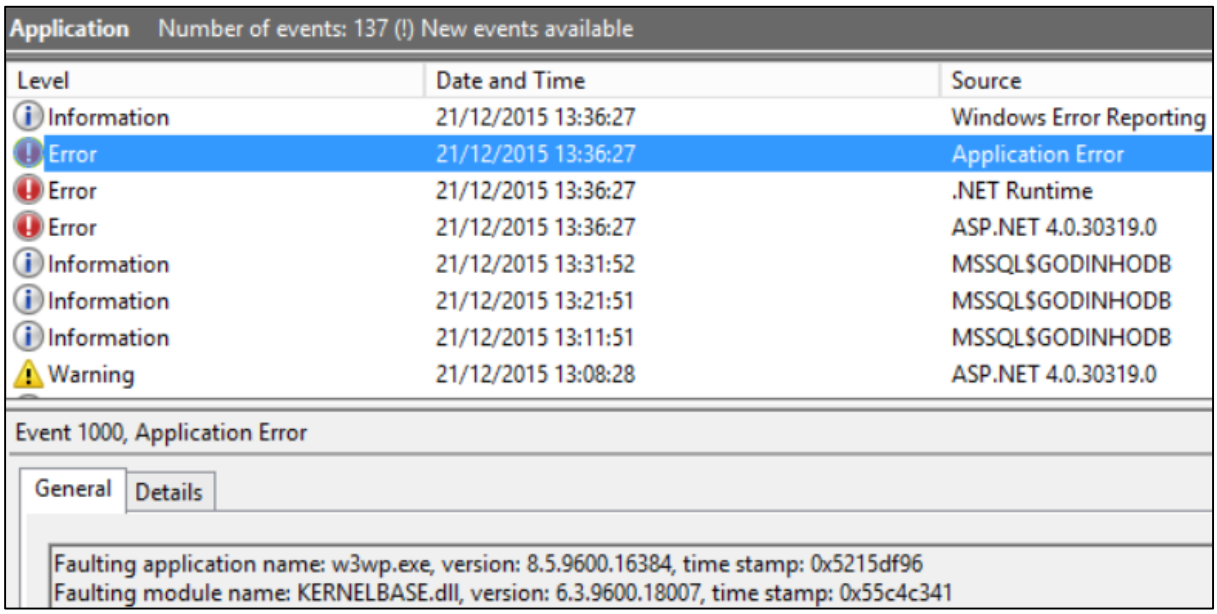

Figura 39 - Event Viewer do servidor onde corre o IIS

Deste modo, não existe alternativa para além de efetuar a verificação em todas as páginas existentes no projeto, e cada vez que uma nova página for acrescentada é necessário adicionar as linhas de código respetivas.

Nesta implementação, devido às rotas que vão sendo retiradas após a sessão expirar, existem várias situações em que é possível ocorrer erros do tipo 404. De modo a evitar este problema, tornou-se mais simples contornar a questão, e utilizar a página inicial como a página específica deste erro, para onde os utilizadores são encaminhados quando este tipo de erro ocorre.

```
<customErrors mode="RemoteOnly" defaultRedirect="~/Default.aspx">
  <error statusCode="404" redirect="~/Default.aspx" />
</customErrors>
```

7. Conclusão

Esta dissertação teve como objetivo o estudo de várias tecnologias de segurança, através da construção de um *website* que implementasse e que permitisse proteção da privacidade do utilizador e ocultação da navegação. Alguns dos mecanismos utilizados, poderão ser considerados apenas como boas normas de segurança a serem implementadas num *website* de *e-commerce*, outros serão o seu aprofundar. As várias versões desenvolvidas tentam implementar estes mecanismos, partindo do pressuposto que o túnel *SSL/TLS* do *HTTPS* poderá ser quebrado, expondo deste modo a informação trocada entre cliente e servidor. A nível de exemplo, o ISEC adquiriu uma *firewall* que permite obter informação sobre o tráfego que passa em algumas das ligações encriptadas, o que pressupõe que existe tecnologia capaz de quebrar os túneis *SSL/TLS* e permitir analisar o seu conteúdo. As várias implementações podem ser consideradas como camadas adicionais de segurança que devem ser utilizadas em cima do túnel encriptado, que deverá existir sempre entre cliente e servidor (ligação em *HTTPS*).

Conclui-se com relativa facilidade, que a implementação de um *website* sem medidas de segurança, e utilizando a passagem de parâmetros por *query string*, permite através de uma simples análise de tráfego, obter informação sobre a navegação do cliente, sendo possível utilizar essa informação para potenciais ataques ou violações de privacidade.

Desenvolveu-se uma implementação com encriptação aplicada aos parâmetros da *query string*, onde se conseguiu com sucesso, mascarar parcialmente a navegação do utilizador. Foi testada a alteração da normal navegação do *website* de modo a que todos os pedidos fossem efetuados ao documento inicial do *website*, */Default.aspx*. Nesta implementação, embora a encriptação permitisse encriptação da informação e ocultar a navegação do utilizador, não foi possível torná-la completamente funcional. Demonstrou-se que o modo como a tecnologia ASP.NET gera o código das páginas, previne o despoletar do evento de *PostBack*, deixando de ser chamado, não permitindo, por exemplo, alterações nas bases de dados e quebrando por completo o funcionamento de quase todas as páginas e os controlos.

Foi também desenvolvido um módulo *HTTP*, que permite intercetar os pedidos e respostas que chegam e saem do servidor *web (IIS)*, alterando os pedidos que possuam o parâmetro *id=''* na *query string*, e intercetando as respostas ao cliente, encriptando o parâmetro da *query string id=''*. Atuando como uma espécie de *man-in-the-middle*, alterando a informação entre cliente e servidor. Conclui-se que a utilização de um módulo deste género, quebra também o funcionamento do *PostBack* dos controlos. O desenvolvimento de um *HTTP Handler*, teria resultado igual, visto apenas mudar o mecanismo, mas obtendo os mesmos resultados práticos.

Às terceira, quarta e quinta implementações, embora produzindo resultados com relativo sucesso, talvez fosse possível torná-las completamente funcionais desde que, decompondo os controlos, isto é, não utilizando o mecanismo de *PostBack* e distribuindo por várias páginas o que a plataforma ASP.NET permite efetuar em apenas uma. Podemos concluir que para aplicar este tipo de funcionalidade, seria necessário abandonar o uso de bastantes dos controlos e alterar a estrutura das páginas, implementando a página como em outras linguagens *web* de *server side*

como o PHP. Conclui-se que esta abordagem não é lógica, visto este trabalho ser realizado em ASP.NET e tal solução seria abdicar das potencialidades que a plataforma disponibiliza.

O problema original, do qual surge de a ideia desta dissertação, tem origem num problema relacionado com *URL ReWrite*; no entanto não foi possível a utilização deste mecanismo, visto o ciclo de vida de uma página ASP.NET não o permitir, uma vez que o valor da chave se encontra guardado na sessão do utilizador e a sessão apenas estar disponível após o último evento que permite o *URL ReWrite*.

A última implementação tentada utilizou o mecanismo de *routing* da plataforma .NET, sendo a solução que produziu o resultado pretendido. Com o mecanismo de *Routing* foi possível, através da chave de encriptação gerada para cada um dos clientes, gerar rotas para todos os ficheiros .aspx do *website*, permitindo deste modo encriptar e ocultar a navegação de um utilizador. Os parâmetros, ao serem passados por *query string*, são encriptados do mesmo modo que foi desenvolvido para a segunda versão do *website*. Foi necessário alterar o mecanismo de encriptação de modo a não produzir saídas diferentes utilizando a mesma chave e o texto a encriptar, no entanto, alterando desde modo a robustez do algoritmo de encriptação. No entanto, tendo em conta que o tempo de navegação de um cliente é relativamente baixo e o tempo de vida das sessões é efémera, isto é, é um período de tempo relativamente curto quando comparado com o tempo médio necessário para ser possível desencriptar a informação enviada por *URL*, é uma opção imperativa que em princípio não compromete a segurança da informação trocada.

Conclui-se que é possível implementar uma solução que permita a ocultação e confidencialidade da navegação do utilizador, objetivo inicial desta dissertação. Esta solução foi conseguida através do mecanismo de *Routing*. No entanto, é necessário introduzir algumas alterações no mecanismo de encriptação desenvolvido, de modo a restringir os caracteres produzidos na saída, retirando os caracteres ilegais no uso dos *URLs*, com impacto negativo na sua robustez. Mantendo-se a dupla encriptação dos valores dos parâmetros da *query string*, minimizou-se este impacto na robustez.

A utilização futura das soluções propostas, em especial a sua utilização em *websites* com quantidades significativas de acessos, necessita ainda de um estudo profundo. Será necessário estudar as necessidades de recursos, especialmente de processamento e memória para a encriptação e desencriptação massiva de *strings*. Sendo geradas um elevado número de chaves, a sua utilização para encriptar e desencriptar os *URLs* e os valores introduzidos em *query string* é recorrente, os requisitos em termos de recursos de servidor, podem tornar esta solução incomportável financeiramente.

Deverá também ser testada a utilização de diferentes algoritmos de encriptação, quer a nível da encriptação dos *URLs* como também das *query strings*. A plataforma ASP.NET suporta outros algoritmos seguros para além do *AES*, que poderão ser utilizados em sua alternativa e mantendo o nível de segurança pretendido, verificando as necessidades de requisitos de *hardware*.

A utilização de sessões ao invés de *query string* ou *cookies*, é causadora do mesmo problema de requisito de recursos, requerendo, em princípio, maiores quantidades de memória, potenciando este problema. Será necessário, então, verificar o comportamento do *IIS* com a

utilização de rotas de forma massiva. Esta é uma área um pouco cinzenta no que trata a documentação e a estudo de *performance*.

A recente mudança de rumo da Microsoft em relação à ASP.NET e .NET Core, deverá também ser alvo de estudo. Existem diferenças profundas entre as duas plataformas, uma delas relativamente à utilização de rotas. Na plataforma *.NET Core*, o modo como o programador pode interagir com estas, sofreu profundas alterações, sendo muito mais fácil o seu manuseamento.

Retomando o que disse anteriormente, conclui-se que é possível construir *websites* seguros e confidenciais emASP.NET, mas é necessário, ainda, um estudo profundo, a realizar posteriormente.

REFERÊNCIAS BIBLIOGRÁFICAS

Abhijit, Jana. 2010. Beginner's Guide: How IIS Process ASP.NET Request. *Abhijit's World of .NET.* [Online] 2010. [Citação: 03 de 11 de 2015.] http://abhijitjana.net/2010/03/14/beginner%E2%80%99s-guide-how-iis-process-asp-net-request/.

Bhimani, Akash. 2015. Overview of State Management in ASP.Net. *C Sharp Corner.* [Online] 2015. [Citação: 18 de 08 de 2015.] http://www.c-sharpcorner.com/UploadFile/d0a1c8/overview-of-state-management-in-Asp-Net/.

Camargo, Wellington Balbo De. 2013. Conceitos e Exemplo Prático: Usando QueryString. *DevMedia.* [Online] 2013. [Citação: 10 de 1 de 2015.] http://www.devmedia.com.br/conceitos-e-exemplo-pratico-usando-querystring/18094.

Cogley, Jonathan. 2007. Symmetric Salting - remember that salt goes with more than just hash. *Jonathan Cogley's Blog.* [Online] 2007. [Citação: 25 de 12 de 2015.] http://weblogs.asp.net/jcogley/symmetric-salting-remember-that-salt-goes-with-more-than-just-hash.

D'Mello, Gwyn. 2014. Google reels under DDoS attack. *Diligent Media Corporation Ltd.* [Online] 2014. [Citação: 12 de 08 de 2015.] http://www.dnaindia.com/scitech/report-google-reels-under-ddos-attack-2040211.

Gürkaynak, Frank K. 2014. GALS System Design: Side Channel Attack Secure Cryptographic Accelerators. *Institut für Integrierte Systeme Integrated Systems Laboratory.* [Online] 2014. [Citação: 20 de 08 de 2015.] http://www.iis.ee.ethz.ch/~kgf/acacia/.

Haack, Phil. 2008. Using Routing With WebForms. *You've Been Haacked.* [Online] 2008. [Citação: 21 de 12 de 2015.] http://haacked.com/archive/2008/03/11/using-routing-with-webforms.aspx/.

Howard, Rob. 2000. ASP.NET Session State. *MSDN.* [Online] 2000. [Citação: 21 de 07 de 2015.] https://msdn.microsoft.com/en-us/library/ms972429.aspx.

IETF. 2014. RFC HTTP 1.1. *Internet Engineering Task Force (IETF).* [Online] 2014. [Citação: 26 de 07 de 2015.] http://tools.ietf.org/html/rfc7231#section-4.3.3.

Kenneth Schaefer, Jeff Cochran,Scott Forsyth,Rob Baugh,Mike Everest,Dennis Glendenning. 2011. *Profissional IIS 7.* s.l. : John Wiley & Sons, 2011.

Koirala, Shivprasad. 2009. The Two Interceptors: HttpModule and HttpHandlers. *Code Project.* [Online] 2009. [Citação: 04 de 11 de 2015.] http://www.codeproject.com/Articles/30907/The-Two-Interceptors-HttpModule-and-HttpHandlers.

Microsoft. 2015a. ASP.NET Application Life Cycle Overview for IIS 7.0. *Microsoft Developer Network.* [Online] 2015a. [Citação: 10 de 05 de 2015.] https://msdn.microsoft.com/en-us/library/bb470252(v=vs.140).aspx.

Microsoft. 2015b. ASP.NET Routing. *Microsoft Developer Network.* [Online] 2015b. [Citação: 14 de 11 de 2015.] https://msdn.microsoft.com/en-us/library/cc668201.aspx.

Microsoft. 2014a. .NET Framework Cryptography Model. *Microsoft Developer Network.* [Online] 2014a. [Citação: 20 de 08 de 2015.] https://msdn.microsoft.com/en-us/library/0ss79b2x(v=vs.110).aspx. a).

Microsoft. 2014b. ASP.NET View State Overview. *Microsoft Developer Network.* [Online] 2014b. [Citação: 19 de 08 de 2015.] https://msdn.microsoft.com/en-us/library/bb386448.aspx.

Microsoft. 2014c. Encrypting and Decrypting Configuration Sections. *MSDN.* [Online] 2014c. [Citação: 01 de 09 de 2015.] https://msdn.microsoft.com/en-us/library/zhhddkxy(v=vs.140).aspx.

Microsoft. 2014d. HTTP Handlers and HTTP Modules Overview. *Microsoft Developer Network.* [Online] 2014d. [Citação: 12 de 05 de 2015.] https://msdn.microsoft.com/en-us/library/bb398986.aspx.

Microsoft. 2014e. Introduction to ASP.NET Web Forms. *ASP.NET.* [Online] 2014e. [Citação: 14 de 08 de 2015.] http://www.asp.net/web-forms/what-is-web-forms.

Microsoft. 2008. Query String vs. Session Variable. *ASP.NET Forums.* [Online] 2008. [Citação: 10 de 1 de 2015.] http://forums.asp.net/t/1209639.aspx.

Mishra, Brij Bhushan. 2010. A Walkthrough to Application State. *Code Project.* [Online] 2010. [Citação: 11 de 04 de 2015.] http://www.codeproject.com/Articles/87316/A-walkthrough-to-Application-State.

Mitchell, Scott. 2004a. Understanding ASP.NET View State. *Microsoft Developer Network.* [Online] 2004a. [Citação: 18 de 08 de 2015.] https://msdn.microsoft.com/en-us/library/ms972976.aspx.

Mitchell, Scott. 2004b. URL Rewriting in ASP.NET. *Microsoft Developer Network.* [Online] 2004b. [Citação: 24 de 03 de 2015.] https://msdn.microsoft.com/en-us/library/ms972974.aspx.

Moore, Karl. 2004. Server.Transfer Vs. Response.Redirect. *Developer.com.* [Online] 2004. [Citação: 24 de 08 de 2015.] http://www.developer.com/net/asp/article.php/3299641/ServerTransfer-Vs-ResponseRedirect.htm.

Oneda, Ricardo. 2010. Gerenciamento de Sessão no ASP.NET. *Microsoft Developer Network.* [Online] 2010. [Citação: 8 de 3 de 2015.] https://msdn.microsoft.com/pt-br/library/gg454582.aspx.

OWASP. 2014. Testes para a poluição do parâmetro HTTP (OTG-INPVAL-004). *Open Web Application Security Project.* [Online] 2014. [Citação: 28 de 07 de 2015.] https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_(OTG-INPVAL-004).

OWASP. 2015. Testing for SQL Injection (OTG-INPVAL-005). *Open Web Application Security Project.* [Online] 2015. [Citação: 27 de 07 de 2015.] https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005).

Pinto, Pedro. 2014. Aprenda a usar o sniffer Wireshark (Parte IV). *pplware.* [Online] 2014. [Citação: 24 de 08 de 2015.] http://pplware.sapo.pt/tutoriais/networking/aprenda-a-usar-o-sniffer-wireshark-parte-iv/.

Raja, Prabhu. 2011. What is PostBack in ASP.NET. *C# Corner.* [Online] 2011. [Citação: 19 de 08 de 2015.] http://www.c-sharpcorner.com/uploadfile/2f73dd/what-is-postback-in-Asp-Net/.

Roth, Daniel. 2015. Recommended way to list all registered routes in Mvc website. *GitHub.* [Online] 2015. [Citação: 21 de Dezembro de 2015.] https://github.com/aspnet/Mvc/issues/2622.

Team, Google Security. 2014. This POODLE bites: exploiting the SSL 3.0 fallback. *Google - Online Security Blog.* [Online] 2014. [Citação: 22 de 12 de 2015.] https://googleonlinesecurity.blogspot.co.uk/2014/10/this-poodle-bites-exploiting-ssl-30.html.

Templin, Microsoft - Reagan. 2007. Introduction to IIS Architectures. *IIS .Net.* [Online] 2007. [Citação: 10 de 05 de 2015.]

Tero, Paul. 2011. Introduction To URL Rewriting. *Smashing Magazine.* [Online] 2011. [Citação: 24 de 03 de 2015.] http://www.smashingmagazine.com/2011/11/02/introduction-to-url-rewriting/.

W3Schools. 2012. ASP.NET 4 Tutorial. *W3Schools.* [Online] 2012. [Citação: 14 de 08 de 2015.] http://www.w3schools.com/aspnet/.

Wikipedia. 2015a. Advanced Encryption Standard. *Wikipedia.* [Online] 2015a. [Citação: 12 de 06 de 2015.] http://pt.wikipedia.org/wiki/Advanced_Encryption_Standard.

Wikipedia. 2015b. Data Encryption Standard. *Wikipedia.* [Online] 2015b. [Citação: 12 de 05 de 2015.] http://pt.wikipedia.org/wiki/Data_Encryption_Standard.

Wikipedia. 2015c. EFF DES cracker. *Wikipedia.* [Online] 2015c. [Citação: 12 de 05 de 2015.] http://en.wikipedia.org/wiki/EFF_DES_cracker.

Wikipedia. 2015d. HTTPS. *Wikipedia.* [Online] 2015d. [Citação: 01 de 05 de 2015.] http://pt.wikipedia.org/wiki/HTTPS.

Wikipedia. 2015e. POST (HTTP). *Wikipedia.* [Online] 2015e. [Citação: 26 de 07 de 2015.] https://pt.wikipedia.org/wiki/POST_(HTTP).

Yakushev, Ruslan. 2014. Creating Rewrite Rules for the URL Rewrite Module. *IIS.net.* [Online] Microsoft, 2014. [Citação: 07 de 05 de 2015.] http://www.iis.net/learn/extensions/url-rewrite-module/creating-rewrite-rules-for-the-url-rewrite-module.

Yakushev, Ruslan. 2008. IIS URL Rewriting and ASP.NET Routing. *IIS NET.* [Online] 2008. [Citação: 10 de 11 de 2015.] http://www.iis.net/learn/extensions/url-rewrite-module/iis-url-rewriting-and-aspnet-routing.