DESENVOLVENDO APLICAÇÕES SEGURAS EM AMBIENTE HTML/HTTPS

Keesje Duarte Pouw[®]
BankBoston
Rua Líbero Badaró, 425/31-GSP
01009-000 São Paulo - SP
+55 11 3118-6101 kdpouw@bkb.com

Paulo Lício de Geus
Instituto de Computação - IC
Universidade Estadual de Campinas - UNICAMP
13083-970 Cx. Postal 6167 Campinas - SP
+55 19 788-5865 paulo@ic.unicamp.br

RESUMO

Este trabalho apresenta diretrizes para o desenvolvimento de aplicações seguras em ambiente HTML/HTTPS e altamente integradas a sistemas legados, como ilustrado pelas aplicações do setor financeiro. A separação em camadas bem definidas (apresentação, lógica de aplicação e dados), com funções de segurança não sobrepostas, juntamente com uma série de cuidados inerentes ao serviço sendo oferecido, permitem atingir o desejado grau de segurança para aplicações com transferências de valores, destacando-se Internet banking e comércio eletrônico em geral.

ABSTRACT

This paper introduces guidelines to develop secure applications over HTML/HTTPS environments and well integrated to legacy systems, as well illustrated by the finantial industry. A well structured three-tier application layer, well defined security functions, along with special defined measurements to this environment, can provide the strict security requirements demanded by fund transfers application in the Web, as found in Internet Banking and Electronic Commerce applications.

1 INTRODUÇÃO

O advento do protocolo HTTPS – Protocol HTTP (Hyper Transfer Transmission Protocol) (Stevens, 1996) sobre SSL (Secure Socket Layer) (Freier, 1996) representou um divisor de águas no desenvolvimento de aplicações que requerem premissas básicas de segurança, notoriamente os conceitos de privacidade, integridade e autenticação. Desde a sua proposição inicial pela Netscape em meados de 1996, o protocolo SSL tornou-se um padrão de fato para construção de canais de segurança de maneira transparente para as aplicações, em especial àquelas desenvolvidas sobre o paradigma HTTP.

No entanto, a construção de aplicações críticas sobre este paradigma, tais como banco eletrônico e comércio eletrônico em geral, envolve questões mais sutis que o simples uso do protocolo SSL e mecanismos nativos disponíveis no browsers comerciais. A integração com os sistemas legados, autenticação de clientes e formalização de não repúdio são premissas importantes, porém não nativas nos sistemas atuais (browsers e servidores comercias), e serão analisadas no escopo principal deste artigo.

Finalmente vale ressaltar que as técnicas aqui apresentadas resumem de maneira sistemática a infraestrutura de segurança utilizada na aplicação de Internet Banking do BankBoston. O resultado deste modelo propiciou um elevado nível de segurança para a camada de aplicação e sistemas legados, além de possibilitar a construção e disponibilização de novos serviços financeiros de maneira dinâmica e flexível. Tais características – segurança e dinamismo – são de elevado fator competitivo no mercado financeiro.

2 ARQUITETURA DE APLICAÇÕES WEB

O cenário de aplicações na Internet, analisado por este artigo, envolve o protocolo HTTP sobre SSL. Tipicamente a interface do usuário é codificada em HTML, e lógicas simples no cliente são implementas através de *scripts* que possuem interface nativa com HTML nos browser comerciais, tal como JavaScript (Flanagan, 1998) e VBScript (Smith, 1997). Lógicas de aplicação mais sofisticadas somente podem ser obtidas através de mecanismos dependentes do browser utilizado. Notadamente, a tecnologia de Plugin para o browser Communicator e ActiveX para o browser Internet Explorer. A figura 1 ilustra a camada de protocolos na arquitetura TCP e arquitetura genérica de aplicação descritos nas seções seguintes.

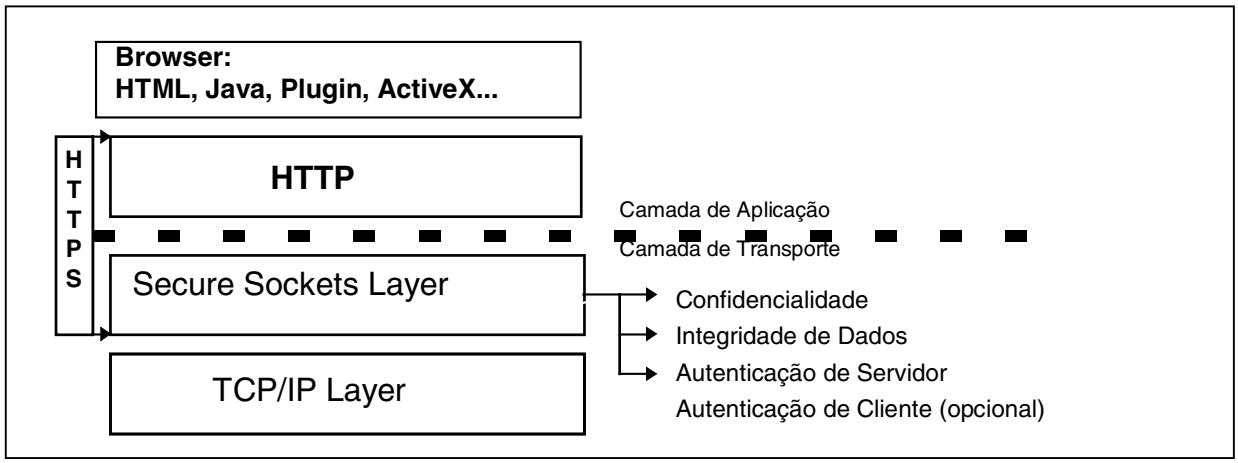

Figura1: Relação entre as arquiteturas TCP/IP e a genérica de aplicação.

2.1 O Protocolo SSL

O protocolo SSL provê uma abstração de canal seguro de comunicação, ou *stream* seguro. Os princípios de integridade, privacidade, autenticação de servidor, são oferecidos à interface HTML de maneira transparente. Para criar um canal seguro com um servidor qualquer, a URL (Unified Resource Locator) a ser referenciada deve ser precedida pelo prefixo HTTPS, por exemplo, HTTPS://www.unicamp.br.

Autenticação de clientes, no entanto, requer a geração de certificados digitais de maneira específica para cada browser de mercado. De maneira geral, a autenticação de clientes baseadas em certificados SSL é muito pouco difundida, mesmo para aplicações críticas que envolvam transferência de valores na Internet. A falta de padronização na interface de uso e geração de certificados, além da complexidade de implementação e uso em comparação a utilização de senhas convencionais, têm inibido o uso mais extensivo desta técnica de autenticação.

Por outro lado, o princípio de não repúdio não tem como ser implementado de maneira transparente pelo protocolo SSL, e como veremos adiante requer um alto grau de integração e infra-estrutura de segurança sofisticada por parte da aplicação em si.

Ainda assim a principal deficiência da arquitetura de segurança HTTPS está na fraca associação, por parte dos browsers comerciais, entre uma conexão segura e o servidor seguro propriamente dito. Tal vulnerabilidade é denominada *link spoofing*, e será descrita a seguir.

2.2 Link Spoofing

Na arquitetura SSL a autenticação de servidores é efetuada através de certificados públicos emitidos por entidades certificadoras (Certificate Authority - CA), cujos próprios certificados encontram-se estaticamente armazenados nas aplicações clientes (browsers). Este esquema oferece uma maneira segura de associar uma URL, por exemplo https://wwws.companiax.com, ao seu respectivo servidor seguro. Neste caso, um canal seguro de comunicação é estabelecido entre o servidor citado na URL e o usuário que a referenciou.

No entanto, uma prática extremamente comum na Internet é "navegar" de referência em referência (link) até uma página de "login" segura obtida através do protocolo HTTPS. Como a página que referencia o link seguro é transferida através do protocolo HTTP, esta está sujeita às mesmas vulnerabilidades do protocolo TCP/IP. Ou seja, a referência ao link seguro original pode ser alterada em trânsito ou através de técnicas de personificação de máquinas descritas em (Pouw e Geus, 1996). Assim, um usuário ao acessar a referência adulterada (possivelmente também uma referência HTTPS) pode ser conduzido a uma outra página, provavelmente em outro servidor e possivelmente com o mesmo aspecto da referência original.

Se a referência adulterada for também uma referência HTTPS, então o indicador de conexão segura nos browsers comerciais mostrar-se-á como o esperado. Neste caso, a única proteção oferecida ao usuário é a visualização dos atributos de segurança (certificado SSL do servidor) da URL em questão, o que constitui prática pouco comum entre a maioria dos usuários.

2.3 O Protocolo HTTP-HTML

O HTTP é um protocolo simples orientado à mensagens. O cliente estabelece uma conexão TCP com um servidor, envia um comando, e recebe os dados do servidor. A conexão TCP é finalizada pelo servidor assim que os dados são transmitidos[1]. A mensagem retornada pelo servidor é normalmente um do-

[1] A versão 1.1 do protocolo HTTP cria o conceito de conexões persistentes, onde várias mensagens são enviadas em uma única conexão TCP (Stevens, 1996).

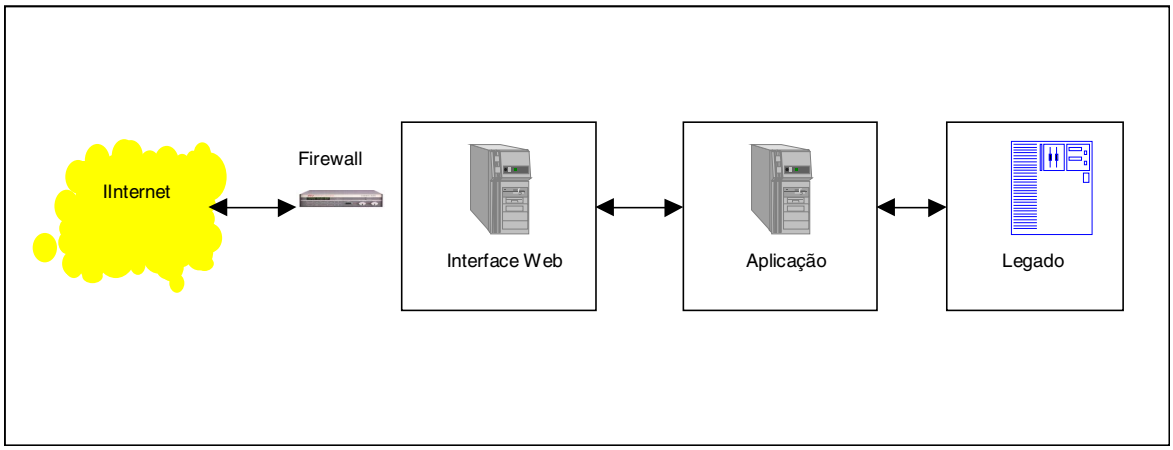

Figura 2: Arquitetura de aplicação Web segura em ambiente típico.

cumento HTML com referências (links) para outros documentos HTML, imagens, arquivos PostScript, arquivos texto etc.

Os principais comandos HTTP são os métodos GET e POST. O método GET retorna qualquer dado associado a URL em questão. O método POST, por sua vez, é usado para enviar correio eletrônico (e-mail), e principalmente formulários eletrônicos pre-enchidos interativamente pelos usuários. Este é o único comando que envia parâmetros de requisição.

Os formulários eletrônicos construídos através da linguagem HTML, linguagem Java, e componentes ativos formatam os parâmetros em uma sintaxe conhecida informalmente como label-valor. Nesta estrutura cada parâmetro recebe um identificador ao qual um determinado valor será associado. O caracter "=" associa um label a um determinado valor e o caracter "&" separa os pares label-valor. O objetivo deste padrão é disponibilizar um "protocolo universal" de troca de dados entre aplicações cliente-servidor sem estrutura pré-definida.

A interface entre o servidor Web e as aplicações é feita através de componentes conhecidos genericamente como CGI 's (Common Gateway Interface). Os cabeçalhos HTTP, comando e parâmetro (comando POST) são normalmente obtidos através de variáveis de ambiente e repassados às aplicações que interpretam e tratam a estrutura label-valor. Os CGI's podem ser fisicamente executáveis, extensões dos servidores Web na forma de bibliotecas compartilhadas (Microsoft ISAPI e Netscape NS-API), dentre outras tecnologias que fogem ao escopo deste artigo.

2.4 Componentes Ativos

Entende-se por componentes ativos todo e qualquer código atualizado dinamicamente nos clientes Internet (*browsers*) que tenha acesso a recursos de sistema, tais como os recursos de entrada e saída (I/O) e arquivos. Claramente, tais conteúdos, normalmente embutidos no contexto de páginas HTML, podem introduzir vírus, cavalos de Tróia e outros elementos nocivos em seus sistemas hospedeiros. Os componentes ativos mais difundidos são os controles ActiveX (Denning, 1997) e Plugins (Morgan, 1997).

Normalmente a atualização/instalação de componentes ativos nos *browsers* somente é permitida se os mesmos forem eletronicamente assinados com a chave privada de alguma entidade confiável (*Certificate Authority*—CA). No entanto, a exibição do certificado da entidade emissora do conteúdo ativo somente prova sua autenticidade, mas não diz a nada a respeito do conteúdo em si. Assim a melhor proteção é simplesmente desabilitar a execução de conteúdos ativos nos *browsers*, ou filtrá-los nos Firewalls quando estes estiverem disponíveis.

3 ARQUITETURA DE APLICAÇÕES WEB SEGURA

Infelizmente o processo de construção de aplicações Web seguras é menos determinístico do que se apresenta em uma primeira análise. Entretanto, algumas diretrizes básicas podem tornar este trabalho mais palpável no contexto de aplicações Web.

A arquitetura, projeto e programação em um ambiente qualquer influi diretamente na qualidade e segurança do sistema desenvolvido. De uma maneira geral o ambiente Web é constituído de uma interface Web, um servidor de aplicação e os sistemas legados, conforme ilustrado na figura 2.

3.1 Interface Web

Refere-se a todos os aplicativos e serviços acessíveis via Internet, tais como o servidor Web propriamente dito, servidor FTP, correio eletrônico etc. Como estes elementos têm contato direto com o mundo externo e com a rede interna, eles devem estar instalados em máquinas fortificadas (*bastion host*s) e isolados em uma **zona desmilitarizada** (Pouw, 1999).

Desta forma, apenas os serviços Web (HTTP, FTP etc) terão permissão de acesso definida no Fi-

rewall para as máquinas na zona desmilitarizada, e estas por sua vez acessarão apenas os serviços de aplicação definidos na rede interna. Assim, mesmo que a interface Web seja comprometida por falhas de *software* (*bug* no servidor Web, por exemplo), os sistemas legados e rede interna não serão comprometidos. Um Firewall que disponha de uma sub-rede exclusiva para os *bastion hosts*, com controle de acesso tanto para a Internet quanto para a rede interna, é tecnicamente denominado de *screened subnet* (Pouw, 1999).

Esta última premissa de segurança somente será verdadeira se a **camada de apresentação**, que corresponde na prática à construção de páginas HTML dinâmicas, estiver perfeitamente isolada do contexto de aplicação na interface Web. Isto porque os mecanismos de autenticação e de acesso ao banco de dados e sistemas legados não devem ser acessíveis pela camada de apresentação. Além disso, como o processo de alteração de "visual" é muito dinâmico e freqüente, erros neste processo poderiam ocasionar falhas de segurança, caso alguma lógica de aplicação estivesse acoplada na camada de apresentação.

3.2 Servidor de Aplicação

O servidor de aplicação deve ser responsável por disponibilizar e processar as informações e serviços requeridas pela Web, assegurando integridade e segurança às mesmas. Novamente, para assegurar o cumprimento de suas responsabilidades este serviço deve, preferencialmente, ser estruturado em camadas.

Este deverá efetuar três verificações básicas: obter o contexto da requisição (controlar sessão), verificar as permissões de acesso e instanciar/executar o serviço requerido. Além disso, serviços básicos podem ser disponibilizados aos componentes específicos e à camada de controle de acesso. Tipicamente temos os serviços de autenticação, e camada de acesso ao banco de dados. A Figura 3 ilustra esta arquitetura.

Note que nesta arquitetura as funções do sistema somente poderão ser executadas através do servidor de aplicação, que automaticamente cuidará de efetuar as devidas verificações de segurança e controle de acesso. Desta forma, novos serviços podem ser agregados automaticamente sem riscos de falhas de segurança e sem o custo de implementação de mecanismos de segurança específicos. Isto reduz drasticamente os custos inerentes aos rígidos procedimentos de testes necessários para aplicações críticas do ponto de vista de segurança., aumentado a produtividade de desenvolvimento neste ambiente crítico.

3.3 Controle de Sessão

Esta camada cria a abstração de sessão para os protocolos não orientados à conexão (tipicamente HTTP) e comandos assíncronos. Associa o contexto de sessão (sessionID) com os parâmetros de autenticação (usuário, método de autenticação, bilhete de acesso), *timeout* e área comum de dados entre aplicações e métodos. A camada de controle de acesso utiliza-se diretamente dos parâmetros de autenticação para determinar se uma dada requisição tem permissões de execução ou não (Figura 4).

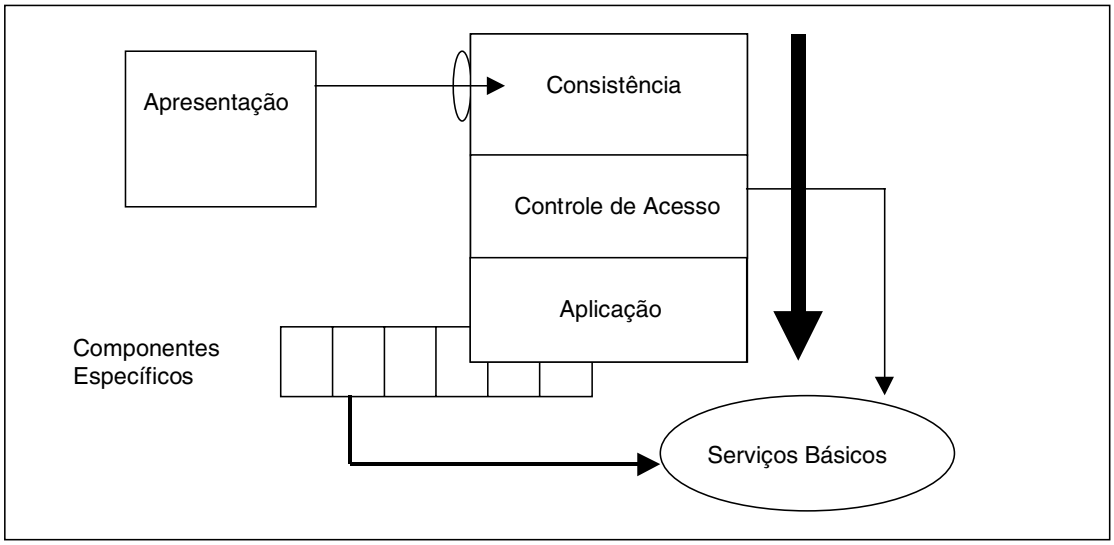

Figura 3: Arquitetura de servidor de aplicação.

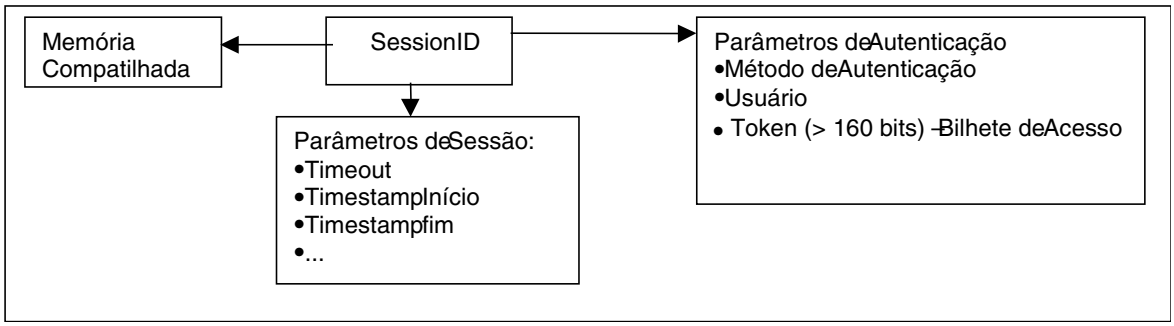

Figura 4: Camada de Controle de Acesso.

3.4 Autenticação e Não Repúdio

Devido à certa facilidade existente em personificar um servidor Web seguro (*spoofing*), técnicas de autenticação baseadas em senhas reutilizáveis tornam-se extremamente frágeis.

Vejamos, por exemplo, o seguinte cenário de ataque: em uma página institucional insegura (`http://companhiax.com.br`, por exemplo) aparece o link para a "página transacional segura", digamos,

`https://wwws.companhiax.com.br/scripts/internet2.dll?Pag=Login`.

Como a página institucional é insegura, o *link* anterior poderia ser substituído para, digamos, `https://wwws.ciax.com.br/scripts/internet2.dll?Pag=Login`.

Obviamente, o intruso teria que registrar um domínio (no caso, `wwws.ciax.com.br`) para tal fim[2]. Ao entrar no *link* transacional falso, o usuário poderia efetuar um "login" falso e enviar sua senha inadvertidamente. Após capturar o segredo em questão, o servidor atacante poderia simplesmente interromper a conexão, simulando uma falha de comunicação.

Portanto, no ambiente HTTPS, aplicações críticas do ponto de vista de segurança devem implementar métodos de autenticação baseados no conceito de *"one time passwords"*. Dentre estes métodos destacam-se os baseados em mecanismo de desafio/resposta (Haller, 1995) e aqueles baseados em assinaturas digitais (Pouw, 1999). Estes últimos, quando derivados de algoritmos assimétricos e devidamente implementados, garantem também a propriedade de não repúdio, especialmente necessária em transações que envolvam transferência de valores.

A implementação de não repúdio exige integração (interface) direta da aplicação com os módulos de criptografia para o cálculo/verificação da assinatura digital de mensagens específicas. Ou seja, o cliente Web (*browser*) deve ser capaz de identificar quais mensagens precisam ser assinadas e o servidor (controle de acesso) identificar quais mensagens deveriam vir assinadas e verificar tais assinaturas. Além disso, seqüenciadores aleatórios devem estar associados a cada mensagem de maneira a evitar ataques de repetição de mensagens, e todas as mensagens assinadas devem também ser armazenadas para eventual verificação.

Vale ressaltar que o protocolo SSL apresenta mecanismos nativos para autenticação de clientes baseados em certificados digitais (assinatura digital), de maneira transparente para a camada de aplicação. No entanto, os mecanismos de autenticação de cliente do SSL não provêem recursos nativos para a implementação de assinaturas digitais sob requisição da aplicação, e consequentemente para não repúdio.

Recentemente a última versão do *browser* Communicator (4.5) dispôs mecanismos para assinar requisições sob demanda (*Form Signing*) (Netscape, 1999), utilizando-se da chave privada e certificado público de clientes SSL.

3.5 Integração com Sistemas Legados

As mensagens recebidas pela camada de aplicação Web seguem o formato label-valor descrito anteriormente. No entanto, sistemas pré-existentes que interajam com as aplicações Web podem ter estrutura de mensagens próprias para as quais os primeiros devem ser convertidos. Como regra geral, este processo deve ser feito de maneira muito criteriosa.

Como a estrutura label-valor pode ser facilmente editada e modificada pelos usuários, todos os *labels* recebidos devem ser cuidadosamente consistidos para cada requisição recebida. Preferencialmente, cada requisição (comando) deve saber identificar exatamente quais *labels* são esperados e quais devem ser consistidos, além de identificar *labels* repetidos e valores fora da faixa esperada.

Suponhamos o seguinte cenário: a camada de controle de acesso verifica e valida o identificador de sessão (sessionID) e bilhete de acesso recebidos. Em seguida, a identidade do usuário (userID) dono da sessão em questão seria acrescentada à estrutura label-valor, antes desta ser repassada para a camada responsável por convertê-la no formato de mensa-

[2]Cabe aos órgãos responsáveis na Internet o zelo com relação ao registro de nomes, principalmente os similares. No Brasil, a FAPESP passou há algum tempo a exigir CGC para o registro de nomes no domínio com.br.

gem esperado pelo sistema legado. Este fluxo é representado na Figura 5.

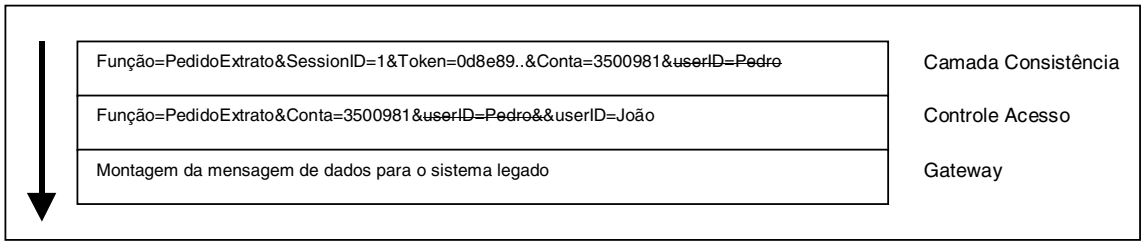

Figura 5: Fluxo de Informação.

Neste cenário temos uma potencial vulnerabilidade que é descrita a seguir. Suponhamos que um usuário mal intencionado altere a requisição original de, por exemplo, uma aplicação de Internet *banking*, trocando a informação da conta requisitada por uma outra pertencente a outro usuário, e acrescente ao final da requisição o identificador de usuário (userID) do proprietário da conta alterada.

Seguindo o modelo da estrutura acima, o "dono verdadeiro" da sessão é acrescentado à estrutura *label*-valor antes de ser repassada ao Gateway de acesso. Se a camada de Controle de acesso não identificar a pré-existência do *label* userID, e o Gateway não identificar a duplicidade de *labels* e procurar pelo primeiro *label* userID, a mensagem enviada ao sistema legado incluirá as informações da conta adulterada e não da do usuário que de fato tenha se autenticado. Naturalmente, esta é uma situação específica que pressupõe um conhecimento prévio do sistema a ser atacado. Contudo, isto ilustra as nuances existentes no processo de validação/autorização de requisições, bem com no tratamento da estrutura de dados e interfaces com os sistemas legados.

4 CONCLUSÃO

A existência do protocolo SSL de maneira nativa nos *browsers* comerciais, e a recente liberação por parte do governo norte-americano à exportação de algoritmos simétricos para instituições financeiras[3] (Microsoft, 1999), trouxe grande impulso ao desenvolvimento de aplicações sensíveis do ponto de vista de segurança, principalmente no que tange à transferência de valores.

No entanto, a existência de um canal seguro entre clientes e servidores na Internet não é suficiente para garantir todas as premissas de segurança necessárias em aplicações do gênero (financeiras). Autenticação forte, baseada em assinaturas digitais ou mecanismos de desafio/resposta, é estritamente necessária devido à facilidade que o ambiente apresenta para personificar servidores seguros (*server spoofing*). Ainda assim, mecanismos de assinatura digital são cruciais para obter a propriedade de não repúdio, que por si requer uma infra-estrutura de sequenciamento de mensagens e armazenamento das mesmas.

Além disso, de pouco vale o uso de técnicas avançadas de segurança e criptografia se não forem respeitados os princípios básicos, de qualidade, arquitetura e engenharia de *software*, com foco em segurança de sistemas, e não apenas em Firewalls ou canais seguros (criptografia).

[3] A referência aqui é obviamente aos algoritmos de criptografia forte (128 bits), em comparação com os anteriores liberados à exportação, de apenas 40 bits.

REFERÊNCIAS BIBLIOGRÁFICAS

STEVENS, W. Richard. TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the Unix Domain Protocols. Addison-Wesley, 1996.

FREIER, Alan O. Freier et al. The SSL Protocol Version 3.0. Netscape. November, 1996.

FLANAGAN, David. Javascript: The Definitive Guide, 3rd Edition. O'Reilly & Associates. 1998.

SMITH, Eric et al. Inside VBScript with ActiveX. New Riders Publishing. 1997.

POUW, Keesje D. e GEUS, Paulo L. de. Uma Análise das Vulnerabilidades dos Firewalls. WAIS´96 (Workshop sobre Administração e Integração de Sistemas), Fortaleza. Maio, 1996.

DENNING, Adam. ActiveX Controls Inside Out. Microsoft Press, 1997.

MORGAN, Mike. Netscape Plug-Ins Developer's Kit. 1997.

POUW, Keesje Duarte. *Segurança na Arquitetura TCP/IP: de Firewalls a Canais Seguros.* Campinas: IC-Unicamp. Janeiro, 1999. Tese de Mestrado.

HALLER N.. The S/KEY One-time Password System. RFC 1760. Bellcore. February, 1995.

NETSCAPE. Http://developer.netscape.com/docs/manuals/security/sgntxt/index.htm

MICROSOFT. Microsoft Corp. Server Gated Cryptography. http://www.microsoft.com/windows/ie/security/gc.as