Autenticação em HTTP Baseada em Desafio-Resposta

Augusto Jun Devegili[*], **Ronaldo Vasconcelos Parente**[1]

[1]Escola Técnica Federal de Palmas – Tocantins
AESE 34 Av. NS 10 s/n
77021-090 Palmas, TO

augusto@devegili.org, parente@netsgo.com.br

Abstract. *We describe the implementation of a challenge-response protocol suitable for Web servers and Web browsers without HTTP digest authentication. Simple primitives are used (pseudorandom numbers, hash functions), which allow for easy implementation in Web servers. Our implementation allows Web developers to keep their own authentication information databases.*

Resumo. *Nós descrevemos a implementação de um protocolo desafio-resposta para uso em servidores e navegadores Web que prescinde do modo de autenticação em resumo padronizado pelo HTTP. Esta implementação utiliza primitivas (números pseudo-randômicos, funções resumo) facilmente implementáveis em servidores Web e permite a um desenvolvedor de aplicações Web manter sua própria base de informações de autenticação.*

1. Introdução

O protocolo HTTP (Hypertext Transfer Protocol – Protocolo para Transferência de Hipertexto) é o protocolo padrão para a World Wide Web. Definido pela RFC 2616, tem como versão atual a 1.1 e seu objetivo principal é transportar documentos HTML. Devido à explosão do uso da Web, várias aplicações têm usado este protocolo como meio de comunicação entre o servidor e o cliente da aplicação. Em grande parte das aplicações, é necessário que haja um mecanismo de autenticação que permita a identificação de usuários aptos a usarem a aplicação.

A RFC 2616 define dois modos de autenticação: básico e resumo (*basic* e *digest*, respectivamente) [IETF, 1999]. Na autenticação básica, as informações de autenticação (nome de usuário, senha e domínio) são enviadas em texto plano usando a codificação base64. Desta forma, um adversário que esteja escutando o canal de comunicação facilmente obtém as senhas enviadas ao servidor Web. Na autenticação em resumo, não se envia a senha em texto plano, mas sim um resumo da senha concatenada a outras informações de autenticação.

O servidor Web Apache, o mais utilizado em ambientes Unix, está atualmente na versão 2.0.44, em que o suporte à autenticação em resumo é experimental [Apache, 2003]. O Microsoft Internet Information Service, possivelmente o servidor Web mais utilizado

[*]Parte deste trabalho foi realizada no Centro Universit´ario Luterano de Palmas

em ambientes Windows, está atualmente na versão 5.0 e suporta a autenticação em resumo. Todavia, este suporte está atrelado à autenticação do sistema operacional *(Active Directory)* [Microsoft, 2002], o que dificulta a autenticação em casos onde há várias aplicações com bases distintas de autenticação em um mesmo servidor.

Um desenvolvedor de aplicações Web encontra-se, portanto, em um dilema: para prover maior segurança aos dados de autenticação, deve usar SSL/TLS[1] ou a autenticação em resumo. SSL/TLS exigem a presença de um certificado digital X.509v3 no servidor Web, o que pode não ser financeiramente viável em todos os casos. A autenticação em resumo, por outro lado, pode ser experimental (Apache) ou estritamente integrada à base de autenticação do sistema operacional (IIS).

Neste trabalho, descrevemos algumas possibilidades de autenticação mais seguras do que a autenticação básica do HTTP sem a utilização da autenticação em resumo. Desta forma, conseguimos evitar o tráfego de senhas em texto plano e a aplicação está liberada para manipular seus próprios dados de autenticação de forma independente do servidor Web ou do sistema operacional. Além disso, nossa implementação trabalha com o conceito de sal, integrando-se melhor com bases de autenticação que usem este conceito.

Este artigo está organizado da seguinte forma: a seção 2 descreve o protocolo de autenticação baseado em desafio-resposta que foi utilizado neste trabalho; a seção 3 enumera possibilidades de armazenamento de informações no navegador Web; a seção 4 descreve como as informações de autenticação podem ser enviadas entre servidor e navegador Web; e a seção 5 discute os resultados alcançados por este trabalho.

2. Estratégia de Autenticação Baseada em Desafio-Resposta

Protocolos de desafio-resposta são comuns em protocolos de autenticação. De forma geral, para que um cliente seja autenticado, o servidor envia um desafio que só pode ser respondido pelo cliente caso este detenha as informações de autenticação corretas.

Para a implementação descrita neste trabalho, considera-se que a base de dados de autenticação contém as seguintes informações: nome do usuário, sal[2] e o resumo da senha do usuário concatenada ao sal.

Nossa implementação de desafio-resposta segue os passos abaixo, os quais são ilustrados pela figura 1:

1. O usuário, através de seu navegador Web, identifica-se perante o servidor Web. A identificação é feita com base em um nome de usuário.
2. O servidor Web, de posse do nome do usuário, verifica em sua base de autenticação qual a informação de autenticação associada ao nome de usuário informado. Esta informação de autenticação é composta pela concatenação da senha

[1] Secure Socket Layer/Transport Layer Security – Camada de Socket Seguro/Seguranç a da Camada de Transporte – s˜ao protocolos que prov^em confi dencialidade e autenticaç ˜ao atrav´es do uso de criptografi a e podem servir como base para a transmiss˜ao de pacotes HTTP.

[2] O sal ´e uma seqü^encia de caracteres que ´e concatenada `a senha antes do c´alculo do resumo. O sal possui dois objetivos: difi cultar um ataque de dicion´ario, em que um advers´ario possui uma base pr´e-calculada de resumos de palavras comuns, e tamb´em evitar que usu´arios diferentes com mesmas senhas possuam resumos iguais. [Schneier, 1996]

do usuário com o sal gerado quando da criação da senha, seguida pela aplicação de uma função de resumo: `infoaut = resumo(senha || sal)`. Para a função de resumo, recomenda-se a utilização de SHA-1[3].

3. O servidor Web gera um número randômico usando uma função de geração de números pseudo-randômicos. Este número randômico é o desafio propriamente dito. A resposta do desafio é gerada através da aplicação da função resumo à concatenação do número randômico com a informação de autenticação: `resposta = resumo(número_randômico || infoaut)`.
4. O servidor Web armazena o número randômico e o desafio que foram gerados para aquela sessão e envia o desafio e o sal ao navegador Web.
5. O navegador Web recebe o número randômico e o sal do servidor Web. De posse da senha do usuário (por exemplo, solicitada em um formulário Web), o navegador calcula a resposta do desafio através utilizando os mesmos passos efetuados pelo servidor Web.
6. O navegador Web, em sua próxima requisição ao servidor Web, envia a resposta do desafio.
7. O servidor Web verifica se a resposta do desafio recebida do navegador Web confere com a resposta armazenada para aquela sessão. Caso a resposta esteja correta, gera-se uma resposta conforme a requisição. Caso a resposta esteja errada, a sessão é cancelada e retorna-se ao passo 1.

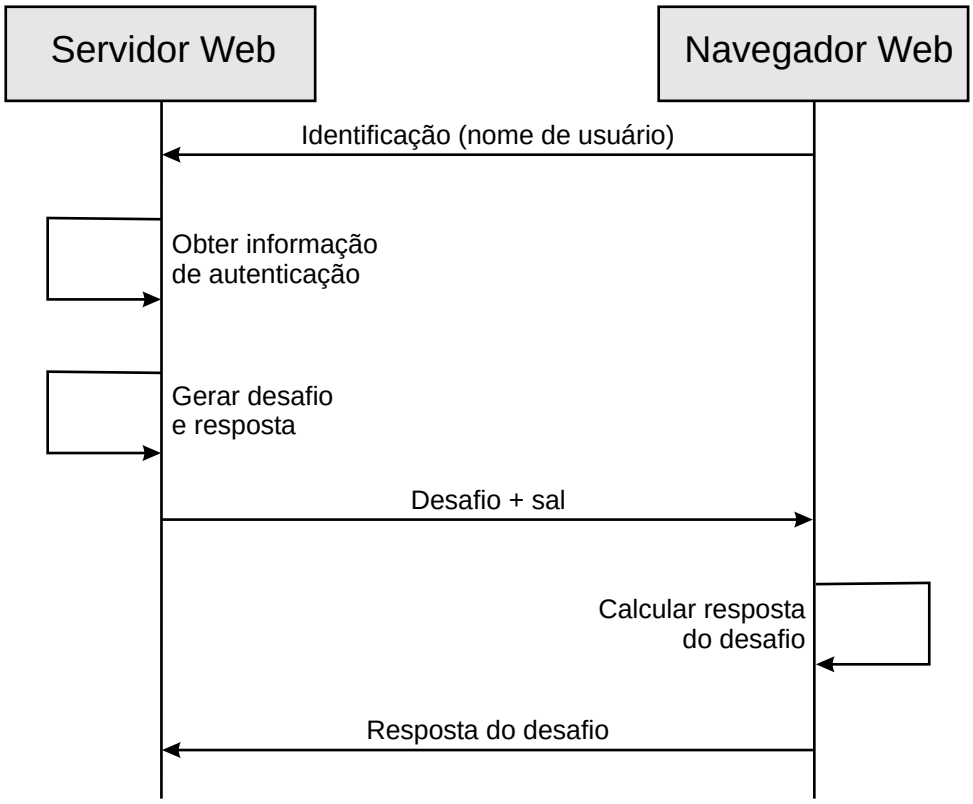

Figura 1: Passos do protocolo de desafio-resposta

Os passos de 3 a 7 são executados para cada requisição/resposta.

[3]Secure Hash Algorithm 1 - Algoritmo de Resumo Seguro 1

Considerando-se um adversário que consiga escutar os pacotes transmitidos entre o servidor e o navegador Web, verifica-se que em momento algum este adversário consegue obter a senha do usuário, já que a senha informada no navegador é concatenada ao sal, submetida a uma função resumo, sendo o resultado concatenado ao desafio e submetido a uma função resumo novamente. Como são usadas funções de resumo criptográficas, as quais possuem a propriedade de caminho único, um adversário não consegue, a partir do resumo, obter a informação original.

Para a implementação deste protocolo em uma aplicação Web, apresentam-se os seguintes requisitos:

- O servidor Web deve ser capaz de gerenciar sessões.
- A senha do usuário deve ser armazenada no cliente, já que ela é necessária para o cálculo da resposta ao desafio.
- A resposta do desafio precisa ser enviada ao servidor Web. Este envio pode ser feito através de dados de formulários, strings de consulta ou *cookies*.

Gerenciamento de sessões em aplicações Web é uma característica de algumas linguagens de programação como PHP, JSP e ASP. Caso a linguagem de programação não possua gerenciamento de sessões, o desenvolvedor pode criar seu próprio mecanismo. O armazenamento de dados no cliente Web e o envio de dados ao servidor Web são discutidos nas próximas seções.

3. Armazenamento de Dados no Cliente Web

No protocolo de desafio-resposta proposto, em um primeiro momento o usuário se identifica, fornecendo seu nome de usuário. A partir deste momento, cada requisição ao servidor Web precisa ser autenticada através da resposta ao desafio gerado pelo servidor. Para que o navegador Web consiga responder corretamente o desafio, é necessário que ele possua a senha do usuário. Como não é viável solicitar a senha do usuário para cada requisição, esta senha deve ser pedida uma única vez e então armazenada no navegador. Mais especificamente, não é estritamente necessário que a senha *per se* seja armazenada, mas sim o resultado do cálculo da função resumo sobre a concatenação da senha ao sal do usuário. As próximas seções discutem alternativas para este armazenamento.

3.1. *Cookies*

Cookies são um provimento do protocolo HTTP que permitem que um servidor Web envie uma informação a ser armazenada no navegador Web. Após isto, o navegador Web envia o valor do *cookie* em cada requisição feita a este servidor Web[4] [Kristol, 2001]. Uma outra possibilidade é o navegador definir o *cookie* sem a intervenção do servidor Web.

Cookies são suportados por virtualmente todos os navegadores Web. Todavia, devido à possibilidade de uso malicioso de cookies, há uma tendência (em alguns lugares, já concretizada) de bloquear *cookies* indistintamente. A iniciativa do P3P[5] do W3C[6] pode

[4]De forma mais precisa, existe a possibilidade se confi gurar um cookie para que seja enviado a qualquer servidor que pertenç a a um dom´ı nio.

[5]Platform for Privacy Preferences Project – Projeto de Plataforma para Preferˆencias de Privacidade

[6]World Wide Web Consortium

servir como base para a aceitação de *cookies* conforme políticas definidas pelo servidor Web [W3C, 2002].

Há que se considerar dois tipos de *cookies*: os persistentes e os de sessão. Os persistentes residem no navegador Web até que sejam explicitamente eliminados pelo usuário ou pelo servidor Web. Os de sessão existem apenas para uma determinada sessão entre navegador e servidor, sendo eliminados ao término da sessão. Desta forma, informações mais sensíveis ficam armazenadas por pouco tempo. Além disso, um *cookie* pode ser definido como *seguro*. Neste caso, ele só é enviado a um servidor Web caso a conexão esteja sendo feita através de SSL/TLS.

3.2. Persistência no Microsoft Internet Explorer

Persistência é um mecanismo implementado no navegador Web Microsoft Internet Explorer que permite que um desenvolvedor Web consiga criar uma página em que os dados possam ser armazenados de forma persistente [Microsoft, 2003] por um tempo indefinido ou então apenas durante a sessão que os criou. Estes dados seguem uma estrutura de segurança similar à dos *cookies*: dados cuja persistência foi solicitada por uma página de um determinador servidor Web não podem ser lidos por páginas de outros servidores Web. A desvantagem clara deste mecanismo é que só funciona em um navegador Web específico.

3.3. Janela de Autenticação

Outra possibilidade de armazenamento de informações no navegador Web são as próprias janelas do navegador Web. Conforme a hierarquia do DOM (Document Object Model – Modelo de Objetos do Documento), um navegador pode estar com várias janelas abertas. Em cada janela existe um e apenas um documento. Um documento, por outro lado, pode ter vários formulários, e dentro de um formulário pode haver vários campos.

Neste contexto, desenvolvemos um mecanismo onde a aplicação Web, do ponto de vista do navegador, contém uma *janela de autenticação*. Esta janela é responsável por manter os dados de autenticação necessários à geração da resposta correta ao desafio gerado pelo servidor Web, ou seja, o resumo da concatenação da senha ao sal e o desafio recebido do servidor Web. Quando é necessário gerar uma resposta ao desafio, esta janela é referenciada de forma a obter as informações necessárias.

4. Tráfego de Desafio e Resposta

Para o tráfego de desafio e resposta, deve-se considerar os dois sentidos da comunicação: o servidor precisa enviar o desafio ao navegador, e o navegador precisa enviar a resposta ao servidor.

O servidor pode usar os seguintes mecanismos para enviar o desafio ao cliente: corpo da resposta HTTP (possivelmente um campo escondido em um formulário ou uma variável em JavaScript), *cookie* e cabeçalho da resposta HTTP. No lado do navegador Web, entretanto, as opções são mais limitadas. Não é possível, através de JavaScript[7], manipular o cabeçalho HTTP da requisição que o navegador envia ao servidor Web. Portanto, restam as seguintes alternativas: corpo da requisição HTTP (campo de formulário), URL da requisição (campo na *string de consulta*) e *cookie*.

[7]Consideramos apenas o JavaScript porque ele ´e suportado pela ampla maioria dos navegadores Web.

5. Resultados e Discussão

O protocolo de desafio-resposta detalhado neste documento é mais seguro do que a autenticação básica provida pelo HTTP ou mecanismos de autenticação simplórios encontrados em aplicações Web, os quais trafegam a senha do usuário em texto plano a menos que se utilize SSL/TLS para o tráfego HTTP. Similar à autenticação em resumo especificada para o protocolo HTTP, nosso protocolo tem a vantagem de não exigir que o servidor e o navegador Web tenham suporte à autenticação em resumo original o que, conforme visto na seção 1, ainda não atingiram maturidade suficiente para aplicações Web.

Para a implementação deste protocolo, avaliamos as possibilidades de armazenamento de dados no navegador Web e tráfego de desafio e resposta. Os resultados estão sumarizados a seguir.

Quanto ao *armazenamento de dados no navegador Web*, as estratégias usando *cookies* e o mecanismo de persistência do Microsoft Internet Explorer são as que possuem implementação mais fácil. Todavia, *cookies* têm sido bloqueados indistintamente e a persistência limita-se a um navegador específico. A janela com informações de autenticação, por outro lado, não sofre destas limitações, porém peca na transparência visual: é necessário que esta janela esteja aberta e portanto visível ao usuário.

Em relação ao *tráfego de desafio e resposta*, os *cookies* são o mecanismo mais natural para a implementação. Um *cookie* pode ser gerado pelo servidor (no caso do protocolo, para enviar o desafio) e é sempre enviado do navegador para o servidor (no caso do protocolo, para enviar a resposta). De fato, uma das características dos *cookies* é justamente o armazenamento de informações de sessão já que o protocolo HTTP não mantém sessão (o HTTP/1.1 pode manter conexão, o que não se aplica neste caso). Todavia, após sua utilização maciça para identificação e rastreamento, gerou-se uma sensação de desconforto em relação a eles, de forma que é comum encontrar instalações onde são sumariamente bloqueados. A definição de políticas de privacidade com base em P3P tem o potencial de amenizar esta situação, porém a difusão desta especificação tem sido bastante pequena.

Outra possibilidade para este tráfego é a manipulação do cabeçalho HTTP. Apesar de isto ser trivial em várias linguagens de programação para servidores Web, o mesmo não se aplica ao navegador. Ou seja, para enviar o desafio ao navegador Web, o servidor Web pode adicionar campos no cabeçalho HTTP da resposta – todavia, este cabeçalho não será disponibilizado pelo navegador para *scripting* e, mais agravante, a linguagem JavaScript não permite a manipulação do cabeçalho HTTP da requisição, impedindo portanto o envio da resposta ao servidor Web.

A alternativa restante envolve o envio de campos ao servidor Web. O envio de um campo pode ser feito de duas formas: na própria URL da requisição ou como um campo do formulário, potencialmente escondido. Usar a URL como forma de enviar a resposta do desafio tem o inconveniente de expor a informação de autenticação ao usuário, além de evitar a possibilidade de utilização do endereço da URL como um marcador para posterior acesso rápido à página. O envio da resposta da requisição como um campo de formulário é mais transparente, porém há um inconveniente: caso a página original tenha um formulário, é necessária a inserção do campo escondido para enviar a resposta. Caso

a página original não tenha um formulário, é necessário criá-lo para que possa conter o campo de resposta. Na nossa implementação, foram criadas bibliotecas em PHP e JavaScript para minimizar este trabalho.

Nossa implementação, portanto, pode ser aplicada em situações onde não haja segurança de tráfego HTTP através de SSL/TLS (seja por motivos financeiros ou operacionais) e se queira evitar o tráfego de senhas em texto plano. Além disso, como uma característica a mais em relação à autenticação em resumo do HTTP, nós utilizamos o sal como parte das informações de autenticação, permitindo que a base de autenticação use esta técnica para minimizar ataques de dicionário ou identificação de senhas repetidas. É certo que há um esforço maior por parte do desenvolvedor – neste caso, é necessário analisar que este custo de desenvolvimento tem como contrapartida mais segurança quanto às informações de autenticação.

Quanto a trabalhos futuros, vislumbramos a possibilidade de se implementar um módulo para o Apache que automatize o trabalho que por enquanto deve ser de preocupação do desenvolvedor. Da mesma forma, para o Internet Information Services, seria possível construir uma extensão ISAPI para fazer essa automação. Além disso, é importante lembrar que mesmo que a autenticação em resumo do HTTP saia do estado "experimental" no Apache, nosso trabalho pode contribuir em ambos os servidores Web descritos (e potencialmente outros) através da liberdade do desenvolvedor em lidar com sua própria base de autenticação, incluindo a utilização do sal.

Referências

Apache Group. (2003). *Apache module mod_auth_digest*. `http://httpd.apache.org/docs-2.0/mod/mod_auth_digest.html`.

Internet Engineering Task Force. (1999). *RFC 2617 – HTTP Authentication: Basic and Digest Access Authentication*. `ftp://ftp.rfc-editor.org/in-notes/rfc2617.txt`.

Kristol, D. M. (2001). HTTP Cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology*, 1(2):151–198.

Microsoft Corporation. (2002). *Setting up Digest Authentication for Use with Internet Information Services 5.0*. `http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B222028`.

Microsoft Corporation. (2003). *Introduction to Persistence*. `http://msdn.microsoft.com/workshop/author/persistence/overview.asp`.

Schneier, B. (1996). *Applied Cryptography*. John Wiley & Sons, New York.

World Wide Web Consortium. (2002). *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. `http://www.w3.org/TR/P3P/`.