OWASP TOP 10

As 10 vulnerabilidades de segurança mais críticas em aplicações WEB

2007

VERSÃO: PORTUGUÊS (BRASIL)

Notas da versão

Esta versão do Top 10 2007 foi desenvolvida como parte das atividades do capítulo Brasil da OWASP em prol da comunidade de desenvolvedores e de segurança do Brasil. Participaram desta tradução:

- Cleber Brandão “Clebeer” - Analista de Controle de Qualidade - BRconnection
- Fabricio Ataides Braz
- Leonardo Cavallari Militelli – Especialista de Segurança - E-val Tecnologia
- Marcos Aurélio Rodrigues - Analista Segurança - BRconnection
- Myke Hamada
- Rodrigo Montoro “Sp0oKer” - Analista Segurança - BRconnection

Para saber mais sobre os eventos e atividades desenvolvidas pelo capítulo Brasil, acesse o site ou cadastre-se na lista de discussão OWASP-BR.

ÍNDICE

INTRODUÇÃO

Bem vindo ao OWASP TOP 10 2007! Esta edição completamente reescrita lista as mais sérias vulnerabilidades em aplicações WEB, discute como se proteger contra elas e provê links para mais detalhes.

OBJETIVO

O objetivo principal do OWASP TOP 10 é educar desenvolvedores, designers, arquitetos e organizações a respeito das conseqüências das vulnerabilidades mais comuns encontradas em aplicações WEB. O TOP 10 provê métodos básicos para se proteger dessas vulnerabilidades – um ótimo começo para a codificação segura de um programa de segurança.
Segurança não é um evento único. Não é suficiente considerar a segurança de código uma única vez. Em 2008, esse Top 10 será modificado e, caso não mude uma linha do código de sua aplicação, você poderá estar vulnerável. Portanto, revise as dicas em "Where to go from here" para mais detalhes.
Uma iniciativa de codificação segura deve abordar todos os estágios do ciclo de vida de um programa. As aplicações WEB seguras são possíveis apenas quando um SDLC seguro é utilizado. Os programas seguros são seguros por concepção, durante seu desenvolvimento e por padrão. Existem no mínimo 300 problemas que afetam a segurança das aplicações WEB como um todo. Estes 300 ou mais são detalhados no Guia OWASP, cuja leitura é essencial para qualquer um que se interesse por desenvolver aplicações WEB.
Este documento é, primeiramente, um recurso de estudo, não um padrão. Não adote este documento como uma política ou padrão sem falar conosco primeiro! Se você precisa de uma política de codificação e desenvolvimento seguro, a OWASP possui este tipo de documentos além de projetos de padronização em andamento. Por favor, considere a possibilidade de se associar ou sustentar financeiramente estas iniciativas.

AGRADECIMENTOS

Nós gostaríamos de agradecer o MITRE por tornar público e gratuitamente os dados da "*Vulnerability Type Distribution*" no CVE. O projeto OWASP Top 10 é liderado e patrocinado pela Aspect Security.
Lider do projeto: Andrew van der Stock (Diretor Executivo, OWASP Foundation)
Coautores: Jeff Williams (Chair, OWASP Foundation), Dave Wichers (Conference Chair, OWASP Foundation).
Nós gostaríamos de agradecer nossos revisores:

- Raoul Endres por ajudar em manter o Top 10 ativo novamente e por seus valiosos comentários
- Steve Christey (MITRE) por sua extensiva revisão e adição das informações do MITRE CWE
- Jeremiah Grossman (White Hat Security) pelas revisões e contribuições valiosas sobre as formas automatizadas de detecção.
- Sylvan Von Stuppe por sua revisão exemplar
- Colin Wong, Nigel Evans, Andre Gironda, Neil Smithline pelos comentários enviados por email.

SUMÁRIO

A1 – Cross Site Scripting (XSS)	Os furos XSS ocorrem sempre que uma aplicação obtém as informações fornecidas pelo usuário e as envia de volta ao navegador sem realizar validação ou codificação daquele conteúdo. O XSS permite aos atacantes executarem scripts no navegador da vítima, o qual pode roubar sessões de usuário, pichar sites Web, introduzir worms, etc.
A2 – Falhas de Injeção	As falhas de injeção, em especial SQL Injection, são comuns em aplicações Web. A injeção ocorre quando os dados fornecidos pelo usuário são enviados a um interpretador com parte do comando ou consulta. A informação maliciosa fornecida pelo atacante engana o interpretador que irá executar comandos mal intencionados ou manipular informações.
A3 – Execução maliciosa de arquivos	Os códigos vulneráveis à inclusão remota de arquivos (RFI) permite ao atacante incluir código e dados maliciosos, resultando em ataques devastadores, como o comprometimento total do servidor. Os ataques de execução de arquivos maliciosos afeta PHP, XML e todos os frameworks que aceitem nomes de arquivo ou arquivos dos usuários.
A4 – Referência Insegura Direta à Objetos	Uma referência direta à objeto ocorre quando um desenvolvedor expõe a referência a um objeto implementado internamente, como é o caso de arquivos, diretórios, registros da base de dados ou chaves, na forma de uma URL ou parâmetro de formulário. Os atacantes podem manipular estas referências para acessar outros objetos sem autorização.
A5 – Cross Site Request Forgery (CSRF)	Um ataque CSRF força o navegador da vítima, que esteja autenticado em uma aplicação, a enviar uma requisição pré-autenticada à um servidor Web vulnerável, que por sua vez força o navegador da vítima a executar uma ação maliciosa em prol do atacante. O CSRF pode ser tão poderoso quanto a aplicação Web que ele ataca.
A6 – Vazamento de Informações e Tratamento de Erros Inapropriado	As aplicações podem divulgar informações sobre suas configurações, processos internos ou violar a privacidade por meio de uma série de problemas na aplicação, sem haver qualquer intenção. Os atacantes podem usar esta fragilidade para roubar informações consideradas sensíveis ou conduzir ataques mais estruturados.
A7 – Autenticação falha e Gerenciamento de Sessão	As credenciais de acesso e token de sessão não são protegidos apropriadamente com bastante freqüência. Atacantes comprometem senhas, chaves ou tokens de autenticação de forma a assumir a identidade de outros usuários.
A8 – Armazenamento Criptográfico Inseguro	As aplicações Web raramente utilizam funções criptográficas de forma adequada para proteção de informações e credenciais. Os atacantes se aproveitam de informações mal protegidas para realizar roubo de identidade e outros crimes, como fraudes de cartões de crédito.
A9 – Comunicações inseguras	As aplicações freqüentemente falham em criptografar tráfego de rede quando se faz necessário proteger comunicações críticas/confidenciais.
A10 – Falha de Restrição de Acesso à URL	Frequentemente, uma aplicação protege suas funcionalidades críticas somente pela supressão de informações como links ou URLs para usuários não autorizados. Os atacantes podem fazer uso desta fragilidade para acessar e realizar operações não autorizadas por meio do acesso direto às URLs.

Tabela 1: TOP 10 vulnerabilidade de aplicações Web para 2007

METODOLOGIA

Nossa metodologia para o TOP 10 de 2007 foi simples: pegamos o estudo de Tendência de Vulnerabilidades para 2006 do MITRE e extraímos as 10 principais vulnerabilidade relativas à aplicações Web. O resultado é apresentado a seguir:

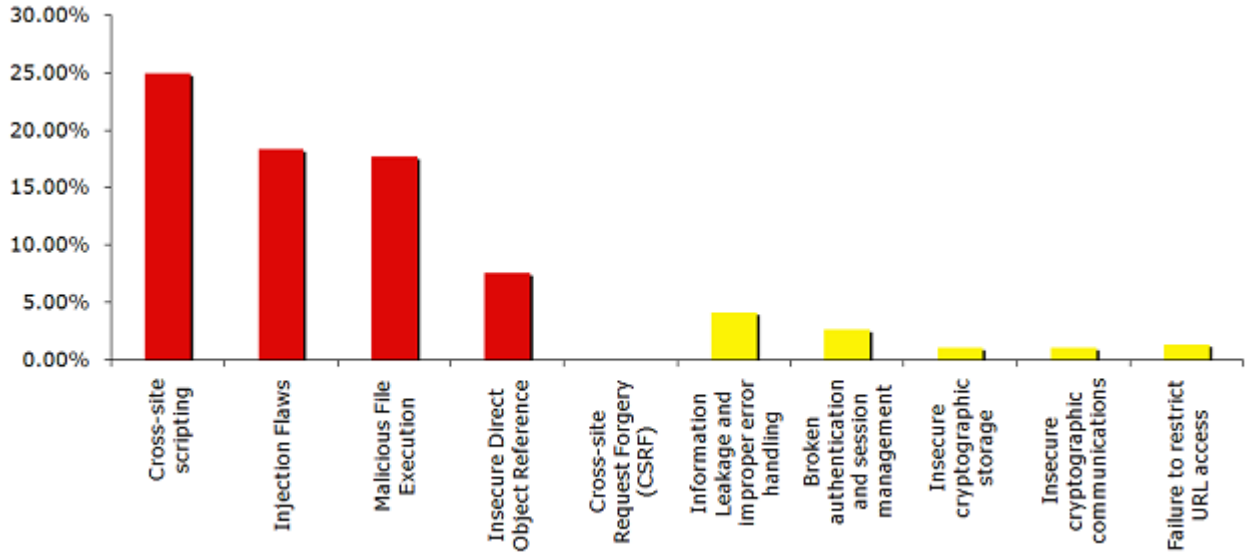

Figura 1: Dados do MITRE para as Top 10 vulnerabilidades de aplicações Web para 2006

Apesar da tentativa de preservação do mapeamento um a um das estatísticas de vulnerabilidades do MITRE para nomear cada seção do documento, nós mudamos deliberadamente algumas categorias com o intuito de mapear de forma mais apropriada as causas principais. Se você está interessado nas estatísticas finais originais do MITRE, nós incluímos uma planilha Excel na página do projeto OWASP Top 10.

Todas as recomendações de proteção provêm soluções para os três mais prevalentes frameworks de aplicação Web: Java EE, ASP .NET e PHP. Outros frameworks utilizados, como Ruby on Rails e Perl podem adaptar facilmente as recomendações para satisfazer necessidades específicas.

POR QUE NÓS DESCARTAMOS ALGUNS PROBLEMAS IMPORTANTES

Entrada de dados não validados é o maior desafio para qualquer time de desenvolvimento e é a origem dos problemas de segurança de muitas aplicações. De fato, muitos dos itens presentes na lista recomendam a validação de entrada como parte da solução. Nós recomendamos criar um mecanismo de validação centralizado como parte de sua aplicação. Para maiores informações, leia os seguintes documentos de validação de dados da OWASP:

- http://www.owasp.org/index.php/Data_Validation
- http://www.owasp.org/index.php/Testing_for_Data_Validation

Problemas de estouro de pilhas, estouro de inteiros e formato de strings são vulnerabilidades extremamente sérias para programas escritos em linguagem C ou C++. A remediação para estes tipos de problemas são tratados por comunidades de segurança de aplicações tradicionais, como o SANS, CERT e pelos fornecedores de linguagem de programação. Se o seu código é escrito em uma linguagem que é passível a estouros de pilha, nós encorajamos você a ler os conteúdos a este respeito no site da OWASP:

- http://www.owasp.org/index.php/Buffer_overflow
- http://www.owasp.org/index.php/Testing_for_Buffer_Overflow

Gerenciamento de configuração insegura afeta todos os sistemas em alguma extensão, particularmente o PHP. No entanto, o ranking do MITRE não nos permite incluir este problema neste ano. Quando implementando sua aplicação, você deve consultar a última versão do OWASP Guide e o OWASP testing Guide para informações detalhadas a respeito do gerenciamento de configuração segura e testes:

- http://www.owasp.org/index.php/Configuration
- http://www.owasp.org/index.php/Testing_for_infrastructure_configuration_management

POR QUE NÓS ADICIONAMOS ALGUNS PROBLEMAS IMPORTANTES

Cross Site Request Forgery (CSRF) é a maior nova inclusão nesta edição do OWASP Top 10. Embora ocupe a 36ª posição na classificação original, nós acreditamos que ela seja de tamanha importância, que as aplicações devem iniciar seus esforços de proteção hoje, particularmente para aplicações de alto risco e criticidade. O CSRF é mais prevalente do que sua atual classificação e pode ser mais perigoso.
Criptografia. O uso incorreto da criptografia não ocupam as posições 8 e 9 da classificação, conforme as informações do MITRE, porém representam a origem de muitos problemas de quebra de privacidade e conformidade (particularmente a conformidade com o PCI DSS 1.1).

VULNERABILIDADES, NÃO ATAQUES

A edição anterior do Top 10 continha um misto de ataques, vulnerabilidades e contramedidas. Esta vez, nós focamos unicamente em vulnerabilidades, embora a terminologia utilizada comumente combine vulnerabilidades e ataques. Se organizações usam este documento para tornar suas aplicações seguras e, consequentemente, reduzir o risco para seus negócios, será possível observar uma redução direta nos seguintes pontos:
Ataques de *phishing*, que podem explorar qualquer uma dessas vulnerabilidades, particularmente, XSS e problemas de autenticação e autorização (A1,A4,A7,A10)
Violação de privacidade devido à validação fraca, regras de negócio e verificações de autorização fracas (A2, A4, A6, A7, A10)
Roubo de identidade por meio de controles de criptografia fracos ou não existentes (A8 e A9), inclusão de arquivo remoto (A3) e autenticação, regras de negócio e verificação de autorização (A4, A7, A10)
Comprometimento de sistema, alteração de informações e destruição de dados por ataques de injeção (A2) e inclusão de arquivo remoto (A3)
Perda financeira por meio de transações não autorizadas e ataques CSRF (A4, A5, A7, A10)
Perda de reputação devido à exploração de qualquer uma das vulnerabilidades acima (A1 à A10)
Uma vez que a organização se distancie da preocupação de controles reativos e vai de encontro à pro - atividade na redução de riscos de seu negócio, ela melhorará a conformidade com regimes regulatórios, reduzirá custos operacionais, e certamente terá um sistema mais robusto e seguro como resultado.

DIRECIONAMENTO

A metodologia descrita acima necessariamente direciona o Top 10 a favor das descobertas da comunidade de pesquisadores de segurança. A forma de descoberta de falhas é similar ao método utilizado em ataques reais, em especial, no que se refere à pseudo-atacantes (“script kiddies”). Protegendo sua aplicação contra as vulnerabilidades do Top 10 proverá uma proteção módica contra as formas mais comuns de ataque e mais, ajudará a traçar um rumo para melhoria da segurança de seu software.

MAPEAMENTO

Nesta versão do Top 10 houve mudanças nos títulos das vulnerabilidades, mesmo quando o conteúdo se relaciona fortemente ao conteúdo anterior. Nós não utilizamos mais o esquema de nomenclatura WAS XML pelo fato deste não se manter atualizado com as vulnerabilidades modernas, ataques e contramedidas. A tabela abaixo representa como esta edição se relaciona com o Top 10 2004 e a classificação do MITRE:

OWASP Top 10 2007	OWASP Top 10 2004	Classificação do MITRE 2006
A1. Cross Site Scripting (XSS)	A1. Cross Site Scripting (XSS)	1
A2. Falhas de Injeção	A6. Falhas de Injeção	2
A3. Execução Maliciosa de Arquivos (NOVO)		3
A4. Referência Insegura Direta à Objetos	A2. Controle de Acesso Falho (dividido no TOP 10 2007)	5
A5. Cross Site Request Forgery (CSRF) (NOVO)		36
A6. Vazamento de Informações e Tratamento de Erros Inapropriados	A7. Tratamento inapropriado de erros	6
A7. Falha de Autenticação e Gerenciamento de Sessão	A3. Falha de Autenticação e Gerenciamento de Sessão	14
A8. Armazenamento Criptográfico Inseguro	A8. Armazenamento Inseguro	8
A9. Comunicações inseguras	Discutido em A10. Gerenciamento Inseguro de Configuração	8
A10. Falha de Restrição de Acesso à URL	A2. Controle de Acesso Falho (dividido no TOP 10 2007)	14
<Removido em 2007>	A1. Entrada não Validada	7
<Removido em 2007>	A5. Estouro de buffer	4, 8, e 10
<Removido em 2007>	A9. Negação de Serviço	17
<Removido em 2007>	A10. Gerenciamento Inseguro de Configuração	29

Tabela 2: Relacionamento das vulnerabilidades das edições 2004 e 2007 do Top 10 e a classificação do MITRE.

A1 – CROSS SITE SCRIPTING

O Cross Site Scripting, mais conhecido como XSS, é de fato um subconjunto de inserções HTML. XSS é a questão de segurança em aplicações web mais prevalente e perniciosa. Os furos XSS ocorrem em aplicações quaisquer que receba dados originados do usuário e o envie ao navegador sem primeiramente validar ou codificando aquele conteúdo.
O XSS permite atacantes executarem script no navegador da vítima, que pode seqüestrar sessões de usuários, desfigurar web sites, inserir conteúdo hostil, conduzir ataques de roubo de informações pessoais (*phishing*) e obter o controle do navegador do usuário usando um *script* mal intencionado (*malware*). O *script* malicioso é freqüentemente em Java Script, mas qualquer linguagem de script suportada pelo navegador da vítima é um alvo potencial para este ataque.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicação web são vulneráveis a Cross Site Scripting (XSS).

VULNERABILIDADE

Existem três tipos bem conhecidos de XSS: refletido, armazenado e inserção DOM. O XSS refletido é o de exploração mais fácil – uma página refletirá o dado fornecido pelo usuário como retorno direto a ele:

```
echo $_REQUEST['userinput'];
```

O XSS armazenado recebe o dado hostil, o armazena em arquivo, banco de dados ou outros sistemas de suporte à informação e então, em um estágio avançado mostra o dado ao usuário, não filtrado. Isto é extremamente perigoso em sistemas como CMS, blogs ou fóruns, onde uma grande quantidade de usuários acessará entradas de outros usuários.
Com ataques XSS baseados em DOM, o código Java Script do site e as variáveis são manipulados ao invés dos elementos HTML.
Alternativamente, os ataques podem ser uma mistura ou uma combinação dos três tipos. O perigo com o XSS não está no tipo de ataque, mas na sua possibilidade. Comportamentos não padrão do navegador pode introduzir vetores de ataque sutis. O XSS é também potencialmente habilitado a partir de quaisquer componentes que o browser utilize.
Os ataques são freqüentemente implementados em Java Script, que é uma ferramenta poderosa de *scripting*. O uso do Java Script habilita atacante a manipular qualquer aspecto da página a ser renderizada, incluindo a adição de novos elementos (como um espaço para *login* que encaminha credenciais para um site hostil), a manipulação de qualquer aspecto interno do DOM e a remoção ou modificação de forma de apresentação da página. O Java Script permite o uso do XmlHttpRequest, que é tipicamente usado por sites que usam a tecnologia AJAX, mesmo se a vítima não use o AJAX no seu site. O uso do XmlHttpRequest permite, em alguns casos, contornar a política do navegador conhecida como "*same source origination*" – assim, encaminhando dados da vítima para sites hostis e criar *worms* complexos e zumbis maliciosos que duram até o fechamento do navegador. Os ataques AJAX não necessitam ser visíveis ou requerem interação com o usuário para realizar os perigosos ataques cross site request forgery (CSRF) (vide A-5).

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar que todos os parâmetros da aplicação são validados e/ou recodificados antes de ser incluídos em páginas HTML.

Abordagens automatizadas: ferramentas de teste de penetração automatizadas são capazes de detectar XSS de reflexão a partir de injeção de parâmetro, mas freqüentemente falha na localização de XSS persistente, particularmente se a saída do vetor de injeção XSS é prevenida via verificação de autorização (como por exemplo, se um usuário fornece dados de entradas maliciosos que são vistos posteriormente apenas pelos administradores). Ferramentas de análise de código fonte podem encontrar pontos APIs com falhas ou perigosas, mas é comum não poderem determinar se a validação ou recodificação foram aplicadas, que pode resultar em falsos positivos. Nenhuma ferramenta é capaz de encontrar XSS baseado em DOM, isso significa que aplicações em Ajax estarão sempre em risco se somente forem aplicados testes automatizados.
Abordagens manuais: caso um mecanismo centralizado de validação e recodificação for usado, a maneira mais eficiente de verificar a segurança é verificar o código. Se uma implementação distribuída for usada, então a verificação demandará esforço adicional considerável. O teste demanda muito esforço, pois a superfície de ataque da maioria das aplicações é muito grande.

PROTEÇÃO

A melhor proteção para XSS está na combinação de validação de "lista branca" de todos os dados de entrada e recodificação apropriada de todos os dados de entrada. A validação habilita a detecção de ataques e a recodificação previne qualquer injeção de script bem sucedida de ser executada no navegador. A prevenção de XSS ao longo da aplicação como um todo requer uma abordagem arquitetural consistente.

- **Validação de entrada**: utilize mecanismo padrão de validação de entrada para validar todas as entradas quanto ao tamanho, tipo, sintaxe e regras de negócio antes de aceitar que o dado seja mostrado ou armazenado. Use uma estratégia de validação "aceite o conhecido como bom". Rejeite entrada inválida ao invés da tentativa de corrigir dados potencialmente hostis. Não se esqueça que as mensagens de erro podem também incluir dados inválidos.
- **Forte codificação de saída**: garanta que qualquer dado de entrada do usuário esteja apropriadamente codificado (tanto para HTML ou XML dependendo do mecanismo de saída) antes da renderização, usando a abordagem de codificação de todos os caracteres, com exceção de um subconjunto muito limitado. Esta é a abordagem da biblioteca Microsoft Anti-XSS e a biblioteca prevista OWASP PHP Anti-XSS. Adicionalmente, configure a codificação de caracteres para cada página a ser produzida como saída, que diminuirá a exposição a algumas variações.
- **Especifique a codificação de saída** (como ISO 8859-1 ou UTF-8): Não permita que o atacante escolha isso para seus usuários.
- **Não use validação de "lista negra"** para detectar XSS na entrada ou codificação de saída. A procura ou troca de poucos caracteres ("<" ">" e outros caracteres similares ou frases como "*script*") é fraco e tem sido explorado com sucesso. Mesmo uma tag "<b>" não verificada é insegura em alguns contextos. O XSS possui um conjunto surpreendente de variantes que torna simples ultrapassar validações de "lista negra" (*Blacklist)*.
- **Cuidado com os erros de conversão**. As entradas devem ser decodificadas e convertidas para a representação interna corrente antes de ser validada. Certifique-se que sua aplicação não decodifica a mesma entrada duas vezes. Tais erros podem ser usados para ultrapassar esquemas de "lista branca" pela introdução de entradas perigosas após serem checados.

Recomendações específicas por linguagem:

- **Java:** use mecanismo de saída *struts* como <bean:write … >, ou use o padrão JSTL escapeXML="true" attribute in <c:out … >. Não use <%= … %> não aninhado (isto é, fora de um mecanismo de apropriado de saída codificada).

- **.NET**: use a biblioteca Microsoft Anti-XSS 1.5 disponível sem custo do MSDN. Não atribua campos de formulário diretamente do objeto Request: *username.Text = Request.QueryString("username");* sem usar esta biblioteca. Entenda quais controles .NET codificam automaticamente o dado de saída.
- **PHP**: garanta que a saída passe pelo htmlentities() ou htmlspecialchars() ou use a biblioteca PHP Anti-XSS que está para ser lançada pela OWASP. Desabilite register_globals, caso este não tenha sido desabilitado.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4206
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3966
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5204

REFÊRENCIAS

- CWE: CWE-79, Cross-Site scripting (XSS)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- OWASP – Cross site scripting, http://www.owasp.org/index.php/Cross_Site_Scripting
- OWASP – Testing for XSS, http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- OWASP Stinger Project (A Java EE validation filter) – http://www.owasp.org/index.php/Category:OWASP_Stinger_Project
- OWASP PHP Filter Project - http://www.owasp.org/index.php/OWASP_PHP_Filters
- OWASP Encoding Project - http://www.owasp.org/index.php/Category:OWASP_Encoding_Project
- RSnake, XSS Cheat Sheet, http://ha.ckers.org/xss.html
- Klein, A., DOM Based Cross Site Scripting, http://www.webappsec.org/projects/articles/071105.shtml
- .NET Anti-XSS Library - http://www.microsoft.com/downloads/details.aspx?FamilyID=efb9c819-53ff-4f82-bfaf-e11625130c25&DisplayLang=en

A2 – FALHAS DE INJEÇÃO

As falhas de Injeção, particularmente injeção SQL, são comuns em aplicações web. Existem muitos tipos de injeção: SQL, LDAP, XPath, XSLT, HTML, XML, comando de sistema operacional e muitas outras. Falhas de Injeção acontecem quando os dados que o usuário dá de entrada são enviados como parte de um comando ou consulta. Os atacantes confundem o interpretador para o mesmo executar comandos manipulados enviando dados modificados. As falhas de Injeção habilitam o atacante a criar, ler, atualizar ou apagar arbitrariamente qualquer dado disponível para a aplicação. No pior cenário, estes furos permitem ao atacante comprometer completamente a aplicação e os sistemas relacionados, até pelo contorno de ambientes controlados por firewall.

AMBIENTES AFETADOS

Todos os frameworks de aplicação web que usem interpretadores ou invoquem outros processos são vulneráveis a ataques por injeção. Isto inclui quaisquer componentes do *framework* que possam usar interpretadores como *back-end*.

VULNERABILIDADE

Caso uma entrada de usuário seja fornecida a um interpretador sem validação ou codificação, a aplicação é vulnerável. Verifique se a entrada de usuário é fornecida à *queries* dinâmicas, como por exemplo:

```
PHP:
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id’] . "’";
Java:
String query = "SELECT user_id FROM user_data WHERE user_name = '" +
req.getParameter("userID") + "' and user_password = '" + req.getParameter("pwd") +"'";
```

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar se os dados do usuário não possam modificar os comandos e *queries* enviadas a interpretadores invocadas para a aplicação.
Abordagens automatizadas: muitas ferramentas de varredura de vulnerabilidades localizam falhas de Injeção, particularmente Injeção SQL. Ferramentas de análise estática que localizam o uso de APIs de interpretadores não seguras são úteis, mas freqüentemente não podem verificar que uma validação ou codificação adequada possa estar em uso para proteger contra tal ameaça. Caso a aplicação gerencie erros internos de servidor 501 / 500 ou erros de banco de dados detalhados, isto pode atrapalhar a varredura por ferramentas automatizadas, mas o código ainda continuará em risco. Ferramentas automatizadas são capazes de detectar injeções LDAP / XML / XPath.
Abordagens manuais: a abordagem mais eficiente e precisa é verificar o código que invoca interpretadores. O revisor deve verificar o uso de API segura ou que validação e/ou codificação apropriada acontece. O teste pode ser extremamente demorado com baixa cobertura devido ao fato da superfície de ataque na maioria das aplicações ser grande.

PROTEÇÃO

Evite o uso de interpretadores quando possível. Caso invoque um interpretador, o método chave para evitar injeções está no uso de APIs seguras, como por exemplo, *queries* parametrizadas e bibliotecas de mapeamento objeto relacional (ORM).

Estas interfaces manipulam todas as fugas dados, ou aquelas que não demandam fuga. Note que enquanto interfaces seguras resolvem o problema, validação é ainda recomendada para detectar ataques.
O uso de interpretadores é perigoso, portanto são recomendados cuidados extras, como os seguintes:

- **Validação de entrada**: utilize mecanismo padrão de validação de entrada para validar todas as entradas quanto ao tamanho, tipo, sintaxe e regras de negócio antes de aceitar que o dado seja mostrado ou armazenado. Use uma estratégia de validação "aceite o reconhecido como bom". Rejeite entrada inválida ao invés da tentativa de checar dados potencialmente hostis. Não se esqueça que as mensagens de erro podem também incluir dados inválidos.
- **Use APIs de query fortemente tipado** com substituidores de espaços reservados, mesmo quando usar *stored procedures.*
- **Imponha o menor privilégio** quando conectar a banco de dados ou outros sistemas de suporte.
- Evite mensagens de erros detalhadas que sejam úteis ao atacante.
- **Use *stored procedures*** uma vez que elas são geralmente seguras contra injeção SQL. Entretanto, seja cuidadoso, pois elas podem ser injetáveis (como por exemplo, via o uso do exec()) ou pela concatenação de argumentos dentro da *stored procedured.*
- **Não use interfaces de query dinâmicas** (como por exemplo, mysql_query() ou similares).
- **Não use funções de fuga simples**, como a addslashes() do PHP ou funções de substituição de caracteres como str_replace("'", "''"). Elas são fracas e têm sido exploradas com sucesso por atacantes. Para PHP, use mysql_real_escape_string() para MySQL ou preferencialmente use PDO que não requer fuga.
- **Quando usar mecanismo simples de fuga,** note que funções simples de fuga não podem escapar nomes de tabelas! Os nomes de tabelas devem ser SQL legal e assim completamente sem sentido para entrada fornecida pelo usuário.
- **Localize erros de conversão**. As entradas devem ser decodificadas e convertidas para a representação interna corrente antes de ser validada. Certifique-se que sua aplicação não decodifica a mesma entrada duas vezes. Tais erros podem ser usados para ultrapassar esquemas de "lista branca" pela introdução de entradas perigosas após sua validação.

Recomendações específicas por linguagem:

- Java EE – use PreparedStatement fortemente tipado ou ORMs como Hibernate ou Spring.
- NET – use queries parametrizadas fortemente tipadas, como SqlCommand com SqlParameter ou um ORM como o Hibernate.
- PHP – use PDO com parametrização fortemente tipada (using bindParam())

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5121
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4953
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4592 REFERENCES
- CWE: CWE-89 (SQL Injection), CWE-77 (Command Injection), CWE-90 (LDAP Injection), CWE-91 (XML Injection), CWE-93 (CRLF Injection), others.
- WASC Threat Classification:
  http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml
  http://www.webappsec.org/projects/threat/classes/sql_injection.shtml
  http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP, http://www.owasp.org/index.php/SQL_Injection
  OWASP Guide, http://www.owasp.org/index.php/Guide_to_SQL_Injection

A3 – EXECUÇÃO MALICIOSA DE ARQUIVO

As vulnerabilidades de execução de arquivos são encontradas em muitas aplicações. Os desenvolvedores têm por hábito usar diretamente ou concatenar entradas potencialmente hostis com funções de arquivo ou *stream*, ou confiar de maneira imprópria em arquivos de entrada. Em muitas plataformas, *framework*s permitem o uso de referências a objetos externos, como referências a URLs ou a arquivos de sistema. Quando o dado é insuficiente verificado, isto pode levar a uma inclusão arbitrária remota que será processado ou invocado um conteúdo hostil pelo servidor web.
Isto permite ao atacante realizar:

- Execução de código remoto.
- Instalação remota de *rootkit* e comprometimento total do sistema.
- Em Windows, comprometimento interno do sistema pode ser possível a partir do uso de PHP's SMB file wrappers.

Este ataque é particularmente prevalecente em PHP e cuidado extremo deve ser aplicado com qualquer sistema ou função de arquivo para garantir que a entrada fornecida pelo usuário não influencie os nomes de arquivos.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicação web são vulneráveis a execução maliciosa de arquivo se eles aceitam nomes de arquivos ou arquivos do usuário. Exemplos típicos incluem: *assemblies* .NET que permitem argumentos de nome de arquivos ou código que aceita a escolha do arquivo pelo usuário de modo a incluir arquivos locais.
O PHP é particularmente vulnerável a ataque de inclusão de arquivo remota (RFI) a partir de manipulação de parâmetro com qualquer API baseada em arquivo ou *streams*.

VULNERABILIDADE

Uma vulnerabilidade comum construída é:

```
include $_REQUEST['filename'];
```

Isto não somente permite a avaliação de *scripts* hostis remotos, mas pode ser usado para acessar arquivos locais do servidor (caso o PHP seja hospedado no Windows) devido ao suporte SMB nos PHP's file system wrappers. Outros métodos de ataque incluem:

- Upload de dados hostis a arquivos de sessões, dados de log e via imagens (típico de software de fórum).
- Uso de compressão ou *streams* de áudio, como por exemplo, zlib:// ou ogg:// que não inspecione a flag interna do PHP e então permite o acesso remoto a recursos, mesmo que *allow_url_fopen* ou *allow_url_include* esteja desabilitado.
- Usando PHP *wrappers*, como por exemplo, php://input e outros para coletar entrada da requisição de dados POST ao invés de um arquivo.
- Usando o PHP's *data: wrapper*, como por exemplo, *data:;base64,PD9waHAgcGhwaW5mbygpOz8+*.

Uma vez que essa lista é extensa (e muda com periodicidade), é vital que o uso de uma arquitetura desenhada apropriado para segurança e design robusto quando manipulamos entradas fornecidas pelo usuário que influenciem a escolha de nomes de arquivos e acesso no lado do servidor.
Apesar de fornecidos alguns exemplos em PHP, este ataque é também aplicável de maneiras diferentes em .NET e J2EE. As aplicações desenvolvidas nestes *frameworks* necessitam de atenção particular aos mecanismos de segurança de acesso ao código para garantir que os nomes de arquivos fornecidos ou

influenciados pelos usuários não habilitem que controles de segurança sejam desativados. Por exemplo, é possível que documentos XML submetidos por um atacante terá um DTD hostil que force o analisador XML a carregar um DTD remoto e analisar e processar os resultados. Uma empresa Australiana de segurança demonstrou esta abordagem para varredura portas para firewalls. Veja [SIF01] nas referências deste artigo para maiores informações.
O dano causado por essa vulnerabilidade está diretamente associado com os pontos fortes dos controles de isolamento da plataforma no *framework*. Como o PHP é raramente isolado e não possui o conceito de caixa de areia "*sandbox*" ou arquitetura segura, o dano é muito pior do que comparado com outras plataformas com limitação ou confiança parcial, ou são contidos em uma sandbox confiável como, por exemplo, quando uma aplicação web é executada sob um JVM com um gerenciador de segurança apropriado habilitado e configurado (que é raramente o padrão).

VERIFICAÇÃO DE SEGURANÇA

Abordagens automatizadas: ferramentas de localização de vulnerabilidades possuem dificuldades em identificar os parâmetros que são usados para entrada do arquivo ou a sintaxe que os façam isso. As ferramentas de análise estática podem localizar o uso de APIs perigosas, mas não podem verificar se a validação ou codificação apropriada foi implementada para proteger contra a vulnerabilidade.
Abordagens manuais: uma revisão de código pode localizar código que possa permitir que um arquivo seja incluído na aplicação, mas existem muitos erros possíveis de ser reconhecidos. Testes podem detectar tais vulnerabilidades, mas identificar parâmetros particulares e a sintaxe correta pode ser difícil.

PROTEÇÃO

A prevenção falhas de inclusão de arquivos remotos exige um planejamento cuidadoso nas fases de arquitetura e design, avançando para os testes. Em geral, uma aplicação bem desenvolvida não usa entrada de usuário para nomes de arquivos para nenhum recurso de servidor (como imagens, documentos XML e XSL ou inclusão de *script*), e terá regras de firewall estabelecidas para prevenir conexões de saída para a internet ou internamente como retorno a qualquer outro servidor.
Entretanto, muitas aplicações legadas continuarão a ter a necessidade de aceitar entrada fornecida pelo usuário.
Dentre as mais importantes considerações, inclui-se:

- **Use um mapeamento indireto de objetos de referência** (veja a seção A4 para mais detalhes). Por exemplo, quando um nome de arquivo parcial é uma vez usado, considere um *hash* para a referência parcial. Ao invés de:

```
<select name=”language”>
  <option value=”English”>English</option>
```

  use

```
 <select name=”language”>
<option value=”78463a384a5aa4fad5fa73e2f506ecfc”>English</option>
```

- Considere o uso de *salts* para prevenir força bruta em referência indireta do objeto. Alternativamente, use somente valores de índices como 1, 2, 3 e garanta que os limites dos vetores são verificados para detectar manipulação de parâmetros.
- **Use mecanismos explícitos de verificação de corrupção**, caso sua linguagem suporte isso. Caso contrário, considere um esquema de nomeação variável para auxiliar na verificação de corrupção.

```
$hostile = &$_POST; // refer to POST variables, not $_REQUEST
$safe[‘filename’]= validate_file_name($hostile[‘unsafe_filename’]); // make it safe
```

Conseqüentemente, qualquer operação baseada em entrada hostil se torna imediatamente óbvia:

```
✘ require_once($_POST['unsafe_filename'] . 'inc.php');
✔ require_once($safe['filename'] . 'inc.php');
```

- **Forte validação de entrada de usuário** usando uma estratégia de validação "aceite o reconhecido como bom".
- **Adicione regras no firewall** para prevenir servidores web estabelecerem novas conexões externas com sites web e sistemas internos. Para sistemas de alto valor, isole o servidor web em sua própria VLAN ou uma sub-rede privada.
- **Certifique-se que os arquivos ou nomes de arquivos fornecidos** não desativam outros controles, como por exemplo, dados corrompidos no objeto da sessão, avatares e imagens, relatórios PDF, arquivos temporários, etc.
- **Considere uma implementação de um *"chroot jail"*** ou outros mecanismos de sandbox como virtualização para isolar aplicações umas das outras.
- **PHP:** desabilite *allow_url_fopen* e *allow_url_include* no php.ini e considere a construção de PHP localmente sem a inclusão dessa funcionalidade. Poucas aplicações necessitam desta funcionalidade e assim estas configurações devem ser configuradas de acordo com a aplicação.
- **PHP**: desabilite *register_globals* e use *E_STRICT* para localizar variáveis não inicializadas.
- **PHP**: garanta que todas as funções de arquivo ou de *stream* (stream_*) são cuidadosamente moderadas. Certifique-se que a entrada do usuário não seja fornecida a qualquer função que use o nome do arquivo como argumento, incluindo:

```
include() include_once() require() require_once() fopen() imagecreatefromXXX() file()
file_get_contents() copy() delete() unlink() upload_tmp_dir() $_FILES move_uploaded_file()
```

- **PHP**: Seja extremamente cauteloso caso o dado seja passado para system() eval() ou `.
- **Com o J2EE,** certifique-se que o gestor de segurança seja habilitado e configurado apropriadamente e que a aplicação demande as permissões apropriadas.
- **Com ASP.NET**, recorra à documentação referente à ***confiança parcial*** e desenhe sua aplicação de forma a ser segmentada por confiança, de forma que ela exista sob os estados mais baixos possíveis de segurança.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0360
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5220
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4722

REFERÊNCIAS

- CWE: CWE-98 (PHP File Inclusion), CWE-78 (OS Command Injection), CWE-95 (Eval injection), CWE-434 (Unrestricted file upload)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP Guide, http://www.owasp.org/index.php/File_System#Includes_and_Remote_files
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP PHP Top 5, http://www.owasp.org/index.php/PHP_Top_5#P1:_Remote_Code_Execution
- Stefan Esser, http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html
- [SIF01] SIFT, Web Services: Teaching an old dog new tricks, http://www.ruxcon.org.au/files/2006/web_services_security.ppt
- http://www.owasp.org/index.php/OWASP_Java_Table_of_Contents#Defining_a_Java_Security_Policy
- Microsoft - Programming for Partial Trust, http://msdn2.microsoft.com/en-us/library/ms364059(VS.80).aspx

A4 – REFERÊNCIA INSEGURA DIRETA A OBJETO

Uma referência direta a um objeto acontece quando um desenvolvedor expõe uma referência a um objeto de implementação interna, como por exemplo, um arquivo, diretório, registro na base de dados ou chave, uma URL ou um parâmetro de um formulário. Um atacante pode manipular diretamente referências a objetos para acessar outros objetos sem autorização, a não ser que exista um mecanismo de controle de acesso.
Por exemplo, em aplicações de *Internet Banking* é comum o uso do número da conta como a chave primária. Conseqüentemente, pode ser tentador usar o número da conta diretamente na interface web. Mesmo que os desenvolvedores tenham usado *queries* SQL parametrizadas para prevenir inserções de comandos SQL (SQL injection), e caso não exista uma verificação adicional para garantir que o usuário é o proprietário da conta e que está autorizado a ver a conta, um atacante pode manipular a partir do parâmetro do número da conta e possivelmente pode ver e modificar todas as contas.
Este tipo de ataque aconteceu no site da Australian Taxation Office's GST Start Up Assistance em 2000, onde um usuário legítimo, mas hostil, simplesmente modificou o ABN (identificador da empresa) presente na URL. O usuário se apossou de cerca de 17.000 registros de empresas cadastrados no sistema, e então enviou para cada uma das 17.000 empresas detalhes do ataque. Este tipo de vulnerabilidade é muito comum, porém não testada largamente em muitas aplicações.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicações web estão vulneráveis a ataques a referências diretas inseguras a objetos.

VULNERABILIDADE

Muitas aplicações expõem referências a objetos internos aos usuários. Atacantes manipulam parâmetros a fim de modificar as referências e violarem a política de controle de acesso de forma intencionalmente e sem muito esforço. Freqüentemente, estas referências apontam para arquivos do sistema e banco de dados, mas qualquer aplicação exposta pode estar vulnerável.
Por exemplo, se o código permite especificação de nomes de arquivos ou caminhos a partir da entrada do usuário, isto pode permitir que atacantes acessem diretórios da aplicação que não estão publicados e acessem outros recursos.

```
<select name="language"><option value="fr">Français</option></select>
…
require_once ($_REQUEST['language'."lang.php");
```

Tal código pode ser atacado usando uma *string* como "../../../../etc/passwd%00" usando injeção de um byte nulo (vide OWASP Guide para mais detalhes) para acessar qualquer arquivo no sistema de arquivo do servidor web.
Similarmente, referências as chaves de banco de dados são freqüentemente expostas. Um atacante pode atacar estes parâmetros simplesmente chutando ou procurando por outra chave válida. Geralmente, elas são seqüenciais por natureza. No exemplo a seguir, mesmo que a aplicação não apresente um link qualquer para um carrinho não autorizado e nenhuma injeção SQL seja possível, um atacante pode ainda modificar o parâmetro cartID para qualquer outro desejado.

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

VERIFICAÇÃO DE SEGURANÇA

O objetivo consiste em verificar se a aplicação não permite que referências diretas a objetos sejam manipuladas por atacantes.
Abordagens automatizadas: scanners de vulnerabilidades possuem dificuldades de identificar os parâmetros susceptíveis a manipulação ou se a manipulação aconteceu. As ferramentas de análise estática não podem saber quais parâmetros devem ter uma verificação de controle de acesso antes de ser usado.
Abordagens manuais: uma revisão de código pode localizar parâmetros críticos e identificar se eles são susceptíveis a manipulação em muitos casos. Testes de penetração podem verificar também quando tal manipulação é possível. Entretanto, ambas as técnicas são dispendiosas e podem não ser suficientes.

PROTEÇÃO

A melhor proteção é evitar a exposição direta de referências a objetos a usuários usando um índice, mapa de referência indireta ou outro método indireto que seja fácil de validar. Caso uma referência direta a objeto pode ser usada, garanta que o usuário esteja autorizado ante do uso.
O estabelecimento de uma forma padrão para referenciar objetos da aplicação é importante, pois:

- **Evita a exposição de referências de objetos privados a usuários** sempre que possível, como chaves primárias e nomes de arquivos.
- **Valide cada referência privada a objeto** através da abordagem “aceite o reconhecido como bom”.
- **Verifique a autorização de todos os objetos referenciados**.
- A melhor solução é usar um valor de índice ou um mapa de referência para prevenir ataques de manipulação de parâmetro.

```
http://www.example.com/application?file=1
```

Caso necessite expor diretamente referências às estruturas de banco de dados, certifique-se que as declarações SQL e outros métodos de acesso à base de dados permitam somente sejam mostrados registros autorizados:

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
User user = (User)request.getSession().getAttribute( "user" );
String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND userID=" +
user.getID();
```

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0329
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4369
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0229

REFERÊNCIAS

- CWE: CWE-22 (Path Traversal), CWE-472 (Web Parameter Tampering)
- WASC Threat Classification:
  http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
  http://www.webappsec.org/projects/threat/classes/insufficient_authorization.shtml
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_business_logic
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP, http://www.owasp.org/index.php/Category:Access_Control_Vulnerability
- GST Assist attack details, http://www.abc.net.au/7.30/stories/s146760.htm

A5 – CROSS SITE REQUEST FORGERY (CSRF)

Cross site request forgery não é um novo ataque, mas é simples e devastador. Um ataque CSRF força o navegador logado da vítima a enviar uma requisição para uma aplicação web vulnerável, que realiza a ação desejada em nome da vítima.
Esta vulnerabilidade é extremamente disseminada, uma vez que qualquer aplicação web:

- Não tenha verificação de autorização para ações vulneráveis
- Execute uma ação caso um login padrão seja enviado na requisição (ex. http://www.example.com/admin/doSomething.ctl?username=admin&passwd=admin)
- Autorize requisições baseadas somente em credenciais que são automaticamente submetidas como, por exemplo, *cookie* de sessão, caso logada corretamente na aplicação, ou a funcionalidade "Relembrar-me", se não logado na aplicação, ou um token Kerberos, se parte de uma Intranet que tenha o logon integrado com o Active Directory.

Este tipo de aplicação estará em risco. Infelizmente, hoje, a maioria das aplicações web confia exclusivamente em credenciais submetidas automaticamente, como por exemplo, *cookies* de seção, credenciais de autenticação básica, endereço de IP de origem, certificados SSL ou credenciais de um domínio Windows.
Esta vulnerabilidade é também conhecida por outros diversos nomes incluindo *Session Riding*, Ataques *One-Click*, *Cross Site Reference Forgery, Hostile Linking* e *Automation Attack*. O acrônimo XSRF é freqüentemente usado. Ambos a OWASP e o MITRE padronizaram o uso do termo *Cross Site Request Forgery* e CSRF.

AMBIENTES AFETADOS

Todos os frameworks de aplicações web estão vulneráveis à CSRF.

VULNERABILIDADE

Um ataque típico CSRF contra um fórum pode ter a forma de direcionar o usuário a invocar alguma função, como por exemplo, a página de *logout* da aplicação. A seguinte tag em qualquer página web vista pela vítima gerará uma requisição que encerra sua seção:

```
<img src="http://www.example.com/logout.php">
```

Caso um banco permita sua aplicação a processar requisições, como a transferência de fundos, um ataque similar pode permitir:

```
<img src="http://www.example.com/transfer.do?frmAcct=document.form.frmAcct
&toAcct=4345754&toSWIFTid=434343&amt=3434.43">
```

Jeremiah Grossman em sua palestra na BlackHat 2006 "Hacking Intranet Sites from the outside", demonstrou ser possível forçar o usuário a modificar seu roteador DSL sem seu consentimento; mesmo que o usuário não saiba que o roteador possua uma interface web. Jeremiah usou um nome de conta padrão do roteador para realizar o ataque.
Todos estes ataques funcionam, pois a credencial de autorização do usuário (tipicamente um *cookie* de sessão) é automaticamente incluída em requisições do navegador, mesmo que o atacante não forneça tal credencial.

Caso a tag contendo o ataque possa ser postada em uma aplicação vulnerável, então a probabilidade de encontrar vítimas autenticadas é incrementada significativamente, similar ao incremento do risco entre as falhas XSS armazenadas e refletidas. Falhas XSS não são necessárias para um ataque CSRF funcionar, apesar de que qualquer aplicação com falhas XSS esteja susceptível a CSRF, pois um ataque CSRF pode explorar uma falha XSS para roubar qualquer credencial não fornecida de forma automática que possa estar em execução para proteger contra um ataque CSRF. Muitos *worms* de aplicação têm usado ambas as técnicas de forma combinada.
Quando estiver construindo defesas contra ataques CSRF, deve-se focar também na eliminação de vulnerabilidades XSS na aplicação, uma vez que tais vulnerabilidades podem ser usadas para subverter a maioria das defesas contra CSRF aplicadas.

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar se a aplicação está protegida contra ataques CSRF pela geração e então requisição de algum tipo de *token* de autorização que não seja automaticamente submetido pelo browser.
Abordagens automatizadas: hoje poucos scanners podem detectar CSRF, mesmo que a detecção de CSRF seja possível para a inteligência dos *scanners* de aplicação. Entretanto, caso seu scanner de vulnerabilidade localize uma vulnerabilidade XSS e não haja proteções anti-CSRF, é muito provável que esteja em risco de ataques CSRF encubados.
Abordagens manuais: teste de penetração é uma maneira rápida de verificar que uma proteção CSRF esteja em operação. A verificação de código é a maneira mais eficiente para se verificar que o mecanismo está seguro e implementado apropriadamente.

PROTEÇÃO

As aplicações devem se certificar que não estão se baseando em credenciais ou *tokens* que são automaticamente submetidos pelos navegadores. A única solução é utilizar um *token* personalizado que o navegador não "lembrará" e então incluir automaticamente em um ataque de CSRF.
As seguintes estratégias devem estar em todas as aplicações web:

- Garanta que não existam vulnerabilidades XSS em sua aplicação (Vide A1 – XSS).
- Insira *tokens* randômicos personalizados em todos os formulários e URL que não seja automaticamente submetido pelo browser. Por exemplo:

```
<form action="/transfer.do" method="post"> <input type="hidden" name="8438927730"
value="43847384383">
…
</form>
```

  e então verifique se o *token* submetido é correto para o usuário corrente. Tais *tokens* podem ser únicos para tal função ou página particular para o referido usuário, ou simplesmente único para a sessão como um todo. Quanto mais focado o *token* for para uma função particular e/ou conjunto particular de dados, mais forte será a proteção, mas mais complicado será seu desenvolvimento e manutenção.

- **Para dados sensíveis ou transações de valores, re-autentique ou use assinatura de transação** para garantir que a requisição é genuína. Configure mecanismos externos como, por exemplo, contato por e-mail ou telefone de maneira a verificar requisições ou notificar o usuário das requisições.
- Não use requisições GET (URLs) para dados sensíveis ou para realizar transações de valores. Use somente métodos POST quando processar dados sensíveis do usuário. Entretanto, a URL pode conter *token* randômico à medida que este cria uma URL única, que torna o CSRF quase impossível de se realizar.
- Um único POST é insuficiente para proteção. Deve-se também combinar com *tokens* randômicos, autenticação por outros meios ou re-autenticação para proteger apropriadamente contra CSRF.

- Para ASP.NET, configure o valor ViewStateUserKey. (Vide referências). Isto prove um tipo similar de verificação a um *token* randômico como descrito anteriormente.
- Enquanto estas sugestões diminuirão drasticamente a exposição, ataques avançados de CSRF podem contornar muitas destas restrições. A técnica mais forte é usar *tokens* únicos e eliminar todas as vulnerabilidades XSS em sua aplicação.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0192
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5116
- MySpace Samy Interview: http://blog.outer-court.com/archive/2005-10-14-n81.html
- An attack which uses Quicktime to perform CSRF attacks http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9005607&intsrc=hm_list

REFERÊNCIAS

- CWE: CWE-352 (Cross-Site Request Forgery)
- WASC Threat Classification: No direct mapping, but the following is a close match: http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
- OWASP CSRF, http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- OWASP Testing Guide, https://www.owasp.org/index.php/Testing_for_CSRF
- OWASP CSRF Guard, http://www.owasp.org/index.php/CSRF_Guard
- OWASP PHP CSRF Guard, http://www.owasp.org/index.php/PHP_CSRF_Guard
- RSnake, "What is CSRF?", http://ha.ckers.org/blog/20061030/what-is-csrf/
- Jeremiah Grossman, slides and demos of “Hacking Intranet sites from the outside” http://www.whitehatsec.com/presentations/whitehat_bh_pres_08032006.tar.gz
- Microsoft, ViewStateUserKey details, http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers_topic2

A6 – VAZAMENTO DE INFORMAÇÕES E TRATAMENTO DE ERROS INAPROPRIADO

Diversas aplicações podem sem intenção vazar informações sobre suas configurações, funcionamento interno, ou violar privacidade através de diversos problemas. Aplicações podem vazar o funcionamento interno via tempo de resposta para executar determinados processos ou respostas diferentes para entradas diversas, como exibindo mesma mensagem de erro mas com código de erros diferentes. Aplicações Web freqüentemente vazarão informações sobre seu funcionamento interno através de mensagens de erros detalhadas ou debug. Freqüentemente, essa informação pode ser o caminho para lançar ataques ou ferramentas automáticas mais poderosas.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicação web são vulneráveis ao vazamento de informações e tratamento de erros inapropriado.

VULNERABILIDADE

Aplicações freqüentemente geram mensagens de erros e as mostram para os usuários. Muitas vezes essas informações são úteis para os atacantes, visto que elas revelam detalhes de implementações ou informações úteis para explorar uma vulnerabilidade. Existem diversos exemplos comuns disso:

- Manipulação de erro detalhada, onde se induzirmos alguns erros serão mostradas muitas informações, como o rastreamento da pilha, validações, falhas de SQL, ou outras informações de debug.
- Funções que produzem diferentes saídas baseado-se em diferentes entradas. Por exemplo, substituindo o mesmo nome de usuário com senhas diferentes deveria produzir o mesmo texto como usuário inexistente, ou password inválido. Entretanto, muitos sistemas geram diferentes códigos de erros.

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar se a aplicação não vaza informações via mensagens de erros ou outros meios.
Métodos Automáticos: Ferramentas de scanner de vulnerabilidades geralmente ocasionarão mensagens de erros. Ferramentas de análise estática podem procurar pelo uso de API que vazam informações, mas não terão capacidade de verificar o significado dessas mensagens.
Métodos manuais: Em uma revisão de código podemos procurar por manipulações inapropriadas de erros e outros fatores que vazam informações, mas demandará um consumo de tempo elevado.

PROTEÇÃO

Desenvolvedores deveriam usar ferramentas como OWASP's WebScarab para tentar fazer suas aplicações gerarem erros. Aplicações que não foram testadas dessa maneira quase que certamente gerarão erros de saída inesperados. Aplicações também deveriam incluir um padrão de exceção de manipulação para prevenir que informações desnecessárias vazem para os atacantes.
Prevenir vazamento de informações requer disciplina. As seguintes práticas têm provado ser eficientes:

- Tenha certeza que toda a equipe de desenvolvimento de software compartilha o mesmo método para manipular erros.
- Desabilite ou limite o detalhamento na manipulação de erros. Em particular, não mostre informações de debug, rastreamento de pilha ou informação de caminhos(path) para usuários finais.

- Tenha certeza que caminhos (paths) seguros que tenha múltiplos resultados, retornem mensagens de erros similares ou idênticas. Se não for possível, considere colocar um tempo de espera randômico para todas as transações para esconder esse detalhe dos atacantes.
- Várias camadas podem retornar resultados excepcionais ou fatais, como camadas de banco de dados, servidores web (IIS, Apache, etc). É vital que erros de todas as camadas sejam adequadamente checados e configurados para prevenir que mensagens de erros sejam exploradas por atacantes.
- Tenha consciência que frameworks comuns retornam diferentes códigos HTTP dependendo se é um erro customizado ou erro do framework . É muito valioso criar um manipulador de erros padrão que retorne uma mensagem de erro já checada para maioria dos usuários em produção para os erros de caminho(path).
- Sobrepor o manipulador padrão de erros para que ele sempre retorne "200" (OK) reduz a probabilidade de ferramentas de testes automáticos descobrirem se alguma falha grave ocorre. Isso é segurança através da obscuridade e pode se prover uma camada extra de defesa.
- Algumas organizações maiores têm escolhido incluir códigos de erros randômicos / únicos em todas suas aplicações. Isto pode ajudar o suporte a encontrar a solução correta para um erro particular, mas isso pode também permitir que atacantes determinem exatamente onde a aplicação falhou.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4899
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3389
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0580

REFÊRENCIAS

- CWE: CWE-200 (Information Leak), CWE-203 (Discrepancy Information Leak), CWE-215 (Information Leak
- Through Debug Information), CWE-209 (Error Message Information Leak), others.
- WASC Threat Classification:
- http://www.webappsec.org/projects/threat/classes/information_leakage.shtml
- OWASP, http://www.owasp.org/index.php/Error_Handling
- OWASP, http://www.owasp.org/index.php/Category:Sensitive_Data_Protection_Vulnerability

A7 – FURO DE AUTENTICAÇÃO E GERÊNCIA DE SESSÃO

Autenticação e gerência de sessão apropriadas são criticas para a segurança na web. Falhas nesta área geralmente envolvem a falha na proteção de credenciais e nos *tokens* da sessão durante seu tempo de vida. Estas falhas podem estar ligadas à roubo de contas de usuários ou administradores, contornando controles de autorização e de responsabilização, causando violações de privacidade.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicações web são vulneráveis a furos de autenticação e de gerência de sessão.

VULNERABILIDADE

Furos no mecanismo principal de autenticação não são incomuns, mas falhas são geralmente introduzidas a partir de funções menos importantes de autenticação como *logout*, gerência de senhas, *timeout*, recordação de dados de *logon*, pergunta secreta e atualização de conta.

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar se o aplicativo autentica corretamente os usuários e protege as identidades das credenciais associadas.
Abordagens automatizadas: ferramentas de localização de vulnerabilidade têm dificuldade em esquemas de autenticação e de sessão personalizados. As ferramentas de análise estáticas provavelmente também não detectarão problemas em códigos personalizados para autenticação e gerência de sessão.
Abordagens manuais: revisão de código e testes, especialmente combinados, são muito efetivos para a verificação de autenticação, gerencia de sessão e funções secundárias estão todas implementadas corretamente.

PROTEÇÃO

A autenticação depende da comunicação segura e de armazenamento de credenciais. Primeiramente, assegure-se que o SSL é a única opção para todas as partes autenticadas do aplicativo (veja A9) e que todas as credenciais estão guardadas de uma forma encriptada ou em *hash* (veja A8).
Prevenir falhas de autenticação requer um planejamento cuidadoso. Algumas das considerações importantes são:

- **Use somente mecanismos padrão para gerência de sessão**. Não escreva ou use gerenciadores secundários de sessão em qualquer situação.
- Não aceite novos identificadores de sessão, pré-configurados ou inválidos na URL ou em requisições. Isto é chamado de ataque de sessão fixada.
- **Limite ou limpe seu código de *cookies*** personalizados com propósito de autenticação de gerência de sessão, como funções 'lembrar do meu usuário' ou funções domésticas de autenticação centralizadas como o S*ingle Sign-On* (SSO). Isto não se aplica às soluções de autenticação federadas robustas ou SSO reconhecidas.
- **Use um mecanismo único de autenticação** com dimensão e número de fatores apropriados. Certifique-se que este mecanismo não estará facilmente sujeito à ataques ou fraudes. Não faça esse mecanismo complicado demais, pois ele pode se tornar alvo de seu próprio ataque.

- **Não permita que o processo de login comece de uma página não encriptada**. Sempre inicie o processo de login de uma segunda página encriptada ou de um novo código de sessão, para prevenir o roubo de credenciais ou da sessão, *phishing* e ataques de fixação de sessão.
- Considere gerar uma nova sessão após uma autenticação que obteve sucesso ou mudança do nível de privilégio.
- **Assegure-se que todas as páginas tenham um link de *logout***. O logout deve destruir todas as sessões e cookies de sessão. Considere os fatores humanos: não pergunte por confirmação, pois usuários acabarão fechando a aba ou janela ao invés de sair com sucesso.
- Use **períodos de expiração de prazo** que automaticamente dão *logout* em sessões inativas, bem como o conteúdo das informações que estão sendo protegidas.
- Use **somente funções de proteção secundárias eficientes** (perguntas e respostas, reset de senha), pois estas credenciais são como senhas, nomes de usuários e *tokens*. Aplique *one-way hash* nas respostas para prevenir ataques nos quais a informação pode ser descoberta.
- **Não exponha nenhum identificador de sessão ou qualquer parte válida das credenciais** em URLs e logs (não regrave ou armazene informações de senhas de usuários em logs).
- Verifique a **senha antiga** do usuário quando ele desejar mudar a senha.
- Não confie em **credenciais falsificáveis** como forma de autenticação, como endereços de IP ou máscaras de rede, endereço de DNS ou verificação reversa de DNS, cabeçalhos da origem ou similares.
- Cuide-se **quando enviar segredos para endereços** de e-mail (procure por RSNAKE01 nas referências) como um mecanismo de reset de password. Use números randômicos *limited-time-only* para resetar acesso e envie um e-mail de retorno assim que a senha for reconfigurada. Cuide para quando permitir que usuários registrados mudem seus endereços de e-mail – envie uma mensagem para o e-mail anterior antes de efetuar a mudança.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6229
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6528

REFERÊNCIAS

- CWE: CWE-287 (Authentication Issues), CWE-522 (Insufficiently Protected Credentials), CWE-311 (Reflection attack in an authentication protocol), others.
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/insufficient_authentication.shtml
- http://www.webappsec.org/projects/threat/classes/credential_session_prediction.shtml
- http://www.webappsec.org/projects/threat/classes/session_fixation.shtml
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Authentication
- OWASP Code Review Guide, http://www.owasp.org/index.php/Reviewing_Code_for_Authentication
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_authentication
- RSNAKE01 - http://ha.ckers.org/blog/20070122/ip-trust-relationships-xss-and-you

A8 – ARMAZENAMENTO CRIPTOGRTAFICO INSEGURO

Proteger dados sensíveis com criptografia tem sido parte chave da maioria das aplicações Web. Simplesmente não criptografar dados sensíveis é muito comum. Ainda, aplicações que adotam criptografia freqüentemente possuem algoritmos mal concebidos, usam mecanismos de cifragem inapropriados ou cometem sérios erros usando cifragem fortes.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicação web são vulneráveis ao armazenamento criptográfico inseguro.

VULNERABILIDADE

Prevenir falhas de criptografia requer planejamento cuidadoso. Os problemas mais comuns são:

- Não criptografar dados sensíveis
- Uso inseguro de algoritmos fortes
- Uso de algoritmos caseiros
- Continuar usando algoritmos que provadamente são fracos (MD5, SHA-1, RC3, RC4, etc.)
- Difícil codificação de chaves, e armazenar chaves em sistemas de armazenamento desprotegidos

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar se as aplicações corretamente armazenam informações sensíveis em sistemas de armazenamento.
Abordagens automáticas: Ferramentas de pesquisas de vulnerabilidades não verificam a criptografia em sistemas de armazenamento em geral. Ferramentas de pesquisa de códigos podem detectar o uso de APIs de criptografias conhecidas, mas não podem detectar se ela esta sendo usada corretamente ou se a criptografia é realizada em um mecanismo externo.
Abordagem manual: Como ferramentas de pesquisas, testes não podem verificar o armazenamento criptografado. Revisão de código é a melhor forma de verificar que a aplicação criptografa dados sensíveis e tem mecanismos e armazenamento de chaves corretamente implementados. Isto em alguns casos pode envolver examinar as configurações de sistemas externos.

PROTEÇÃO

O aspecto mais importante é assegurar que tudo que deve ser criptografado está realmente criptografado. Então você deve assegurar que a criptografia esta corretamente implementada. Como existem vários métodos de incorretamente usar a criptografia, as seguintes recomendações devem ser seguidas como parte de seus testes para ajudar a assegurar o manuseamento seguro de mecanismos de criptografia:

- **Não crie algoritmos de criptografia**. Somente use algoritmos aprovados publicamente como, AES, Criptografia de chaves publicas RSA, SHA-256 ou melhores para *hash.*
- **Não use algoritmos fracos**, como MD5/SHA1. Use mecanismos mais seguros como SHA-256 ou melhores.
- Crie chaves *offline* e armazene chaves privadas com extremo cuidado. Nunca transmita chaves privadas em canais inseguros.
- Assegure que credenciais de infra-estrutura como credenciais de banco de dados ou detalhes de filas de acessos MQ estão corretamente seguras (por meio de rígidos sistemas de arquivos e controles), criptografados de forma adequada e não podem ser decriptografados por usuários locais ou remotos.

- **Assegure que dados armazenados criptografados no disco não são fáceis de decriptografar**. Por exemplo, criptografia de banco de dados é inútil se a conexão de banco de dados permite acessos não criptografados.
- Sobre o requisito 3, de Padrões de Dados Seguros PCI, você deve proteger dados dos titulares das informações. O cumprimento do PCI DDS é obrigatório até 2008 por comerciantes e qualquer um que lidar com cartões de crédito. **Uma boa prática é nunca armazenar dados desnecessários**, como informações sobre a fita magnética ou o número da conta primária (PAN, também conhecida como o numero do cartão de crédito). Se você armazenar o PAN, o requisito para o cumprimento do DSS são significativos. Por exemplo, NUNCA permitir armazenar o numero CVV (O numero de três dígitos nas costas do cartão) sobre quaisquer circunstâncias. Para mais informações, por favor leia o PCI DSS *Guidelines* e implemente controles como necessários.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1664
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1101 (True of most Java EE servlet containers,too)

REFÊRENCIAS

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded
- Cryptographic key), CWE-325 (Missing Required Cryptographic Step), others.
- WASC Threat Classification: No explicit mapping
- OWASP, http://www.owasp.org/index.php/Cryptography
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Cryptography
- OWASP, http://www.owasp.org/index.php/Insecure_Storage
- OWASP, http://www.owasp.org/index.php/How_to_protect_sensitive_data_in_URL's
- PCI Data Security Standard v1.1,
- https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Bruce Schneier, http://www.schneier.com/
- CryptoAPI Next Generation, http://msdn2.microsoft.com/en-us/library/aa376210.aspx

A9 – COMUNICAÇÕES INSEGURAS

Aplicações geralmente falham na hora de encriptar tráfego de rede quando é necessário proteger comunicações sensíveis. A encriptação (geralmente SSL) deve ser usada em todas as conexões autenticadas, especialmente páginas web com acesso via internet, mas também conexões com o *back-end*. Senão, o aplicativo irá expor uma autenticação ou o *token* de sessão. Adicionalmente, a autenticação deve ser usada sempre que dados sensíveis, assim como cartões de crédito ou informações de saúde são transmitidos. Aplicações cujo modo de encriptação possa ser subvertido são alvos de ataques.
Os padrões PCI requerem que todas as informações de cartões de credito que são transmitidas pela internet sejam encriptadas.

AMBIENTES AFETADOS

Todos os *frameworks* de aplicações web são vulneráveis às comunicações inseguras.

VULNERABILIDADE

Falha na hora de encriptar informações sensíveis significa que um invasor que possa escutar o tráfego da rede poderá ter acesso à conversa, incluindo quaisquer credenciais ou informações sensíveis transmitidas. Considerando que redes diferentes terão mais ou menos suscetibilidade a escuta. Entretanto, é importante notar que eventualmente um servidor será comprometido em praticamente qualquer rede, e que invasores instalarão rapidamente uma escuta para capturar as credenciais de outros sistemas.
O uso de SSL para comunicação com usuários finais é critico, pois é muito provável que eles utilizem formas inseguras de acessar os aplicativos. Porque HTTP inclui credenciais de autenticação ou um *token* de sessão para cada pedido, toda autenticação do tráfego deve ir para o SSL, não só os pedidos de *login*.
A encriptação de informações com servidores de *back-end* também é importante. Mesmo que estes servidores sejam naturalmente mais seguros, as informações e as credenciais que elas carregam são mais sensíveis e mais impactantes. Portanto, usar SSL no *back-end* também é muito importante.
A encriptação de informação sensível, assim como cartões de crédito e informações de previdência, se tornou um regulamento financeiro e de privacidade para várias empresas. Negligenciar o uso de SSL para o manuseio de conexões de informações cria um risco de não conformidade.

VERIFICAÇÃO DE SEGURANÇA

O objetivo é verificar se as aplicações encriptam corretamente toda a comunicação autenticada e sensível.
Abordagens automatizadas: ferramentas de localização de vulnerabilidade podem verificar se o SSL é utilizado na interface do sistema e pode localizar muitas falhas relacionadas à SSL. Entretanto, elas não têm acesso às conexões no *back-end* e não podem verificar se elas são seguras. As ferramentas de análise estática podem ajudar com análises de algumas ligações no *back-end*, mas provavelmente não entenderão a lógica customizada requerida para todos os tipos de sistemas.
Abordagens manuais: testes podem verificar se o SSL é utilizado e localizar muitas falhas relacionadas à SSL na interface, mas as abordagens automatizadas são provavelmente mais eficientes. A revisão de código é muito eficiente para verificar o uso de SSL em conexões no *back-end*.

PROTEÇÃO

A parte mais importante da proteção é usar o SSL em todas as conexões autenticadas ou em qualquer informação sensível em transmissão. Há inúmeros detalhes envolvendo a configuração do SSL para aplicações web, então é importante entender e analisar seu ambiente. Por exemplo, o IE7 (internet Explorer 7) provê uma barra verde para certificados SSL altamente confiáveis, mas isto não é um controle apropriado que demonstre por si só o uso seguro da criptografia.

- Use SSL para todas as conexões que são autenticadas ou que transmitam informações sensíveis ou de valor, assim como credenciais, cartões de crédito, e outros.
- Assegure que as comunicações entre os elementos da infra-estrutura, como servidores de web e sistemas de banco de dados, estão propriamente protegidas pelo uso de camadas de transporte de segurança ou de encriptação de nível de protocolo para credenciais e informações de valor intrínseco.
- Dentro dos requisitos de segurança PCI 4, deve-se proteger as informações dos proprietários de cartões de crédito em transmissão. A conformidade com as normas PCI DSS é mandatória até 2008 para todos os comerciantes e qualquer um que lide com cartões de crédito. Em geral, clientes, parceiros, funcionários e acesso administrativo online aos sistemas devem ser encriptados via SSL ou similar.

EXEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6430
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4704
- http://www.schneier.com/blog/archives/2005/10/scandinavian_at_1.html

REFERÊNCIAS

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded cryptographic key), CWE-325 (Missing Required Cryptographic Step), others.
- WASC Threat Classification: No explicit mapping
- OWASP Testing Guide, Testing for SSL / TLS, https://www.owasp.org/index.php/Testing_for_SSL-TLS
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Cryptography
- Foundstone - SSL Digger, http://www.foundstone.com/index.htm?subnav=services/navigation.htm&subcontent=/services/overview_s3i_des.htm
- NIST, SP 800-52 Guidelines for the selection and use of transport layer security (TLS) Implementations, http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf
- NIST SP 800-95 Guide to secure web services, http://csrc.nist.gov/publications/drafts.html#sp800-95

A10 – FALHA AO RESTRINGIR ACESSO À URLS

Comumente, a única proteção para uma URL é não mostrar o link para usuários não autorizados.
No entanto, um motivado, hábil ou apenas um sortudo atacante pode ser capaz de achar e acessar estas páginas, executar funções e visualizar dados. Segurança por obscuridade não é suficiente para proteger dados e funções sensíveis em uma aplicação. Verificações de controles de acesso devem ser executadas antes de permitir uma solicitação a uma função sensível, na qual garante que somente o usuário autorizado acesse a respectiva função.

AMBIENTES AFETADOS

Todos os frameworks de aplicações web estão vulneráveis a falhas de restrição de acesso a URLs.

VULNERABILIDADE

O principal método de ataque para esta vulnerabilidade é chamado de "navegação forçada" ("*forced browsing*"), na qual envolve técnicas de adivinhação de links ("*guessing*") e força bruta ("*brute force*") para achar páginas desprotegidas. É comum que aplicações utilizem códigos de controle de acesso por toda a aplicação, resultando em um modelo complexo que dificulta a compreensão para desenvolvedores e especialistas em segurança. Esta complexidade torna provável a ocorrência de erros e algumas páginas não serão validadas, deixando a aplicação vulnerável.
Alguns exemplos destas falhas incluem:

- URLS "escondidas" e "especiais", mostradas apenas para administradores ou usuários privilegiados na camada de apresentação, porém acessível a todos os usuários caso tenham conhecimento que esta URL existe, como /admin/adduser.php ou /approveTransfer.do. Estas são particularmente comuns em códigos de menus.
- Aplicações geralmente permitem acesso a arquivos "escondidos", como arquivos XML estáticos ou relatórios gerados por sistemas, confiando toda segurança na obscuridade, escondendo-os.
- Códigos que forçam uma política de controle de acesso desatualizada ou insuficiente. Por exemplo, imagine que /approveTransfer.do foi disponibilizado uma vez para todos usuários, mas desde que os controles da SOX foram adotados, ele supostamente só pode ser acessível por usuários aprovadores. Uma possível correção seria não mostrar a URL para usuários não autorizados, no entanto o controle de acesso ainda não estaria implementado na requisição para esta página.
- Códigos que validam privilégios no cliente (browser) e não no servidor, como neste ataque na MacWorld 2007, que aprovava para "Platinum" passes que valiam $1700 via Java Script no browser ao invés de validar no servidor.

VERIFICANDO A SEGURANÇA

O objetivo é verificar que o controle está forçado constantemente na camada de apresentação e nas regras de negócio para todas as URLs da aplicação.
Abordagem automatizada: Scanners de vulnerabilidades e ferramentas de análise manual, ambos possuem dificuldades em verificar o controle de acesso na URL por diferentes razões. Scanners de vulnerabilidades possuem dificuldade em adivinhar páginas escondidas e determinar qual página deveria ser permitida para cada usuário, enquanto mecanismos de análise estática tentam identificar controles de acessos personalizados no código e ligam a camada de apresentação com as regras de negócios.
Abordagem manual: A abordagem mais eficiente e precisa está em utilizar a combinação da revisão do código e dos testes de segurança para verificar os mecanismos de controles de acesso. Se o mecanismo

é centralizado, a verificação pode ser bastante eficiente. Se o mecanismo é distribuído através de uma completa base de código, a verificação pode se tornar dispendiosa. Se o mecanismo está forçado externamente, a configuração deve ser examinada e testada.

PROTEÇÃO

Tendo o tempo para planejar a autorização criando uma matriz para mapear as regras e as funções da aplicação é o passo primordial para alcançar a proteção contra acessos não autorizados. Aplicações web devem garantir controle de acesso em cada URL e funções de negócio. Não é suficiente colocar o controle de acesso na camada de apresentação e deixar a regra de negócio desprotegida. Também não é suficiente verificar uma vez o usuário autorizado e não verificar novamente nos passos seguintes. De outra forma, um atacante pode simplesmente burlar o passo onde a autorização é verificada e forjar o valor do parâmetro necessário e continuar no passo seguinte.
Habilitar controle de acesso na URL necessita de um planejamento cuidadoso. Dentre as considerações mais importantes podemos destacar:

- Garanta que a matriz do controle de acesso é parte do negócio, da arquitetura e do design da aplicação
- Garanta que todas URLs e funções de negócio são protegidas por um mecanismo de controle de acesso efetivo que verifique as funções e direitos do usuário antes que qualquer processamento ocorra. Certifique-se que este processo é realizado em todos os passos do fluxo e não apenas no passo inicial de um processo, pois pode haver vários passos a serem verificados.
- Realize um teste invasão (penetration test) antes do código entrar em produção a fim de garantir que a aplicação não poderá ser utilizada de má fé por um atacante motivado ou com conhecimentos avançados.
- Preste muita atenção em arquivos de includes/bibliotecas, especialmente se eles possuem extensões executáveis como .php. Sempre que possível, devem ser mantidos fora da raiz web. Devem ser verificados se não estão sendo acessados diretamente, por exemplo, verificando por uma constante que pode somente ser criada através de uma biblioteca do chamador.
- Não suponha que usuários não estarão atentos acessar URLs ou APIs escondidas ou especiais. Sempre se assegure que ações com privilégios altos e administrativos estarão protegidos.
- Bloqueie acesso a todos os tipos de arquivos que a sua aplicação não deva executar. Este filtro deve seguir a abordagem "*accept known good*" na qual apenas são permitidos tipos de arquivos que a aplicação deva executar, como por exemplo .html .pdf, .php. Isto irá bloquear qualquer tentativa de acesso a arquivos de log, arquivos XML, entre outros, aos quais se espera nunca serem executados diretamente.
- Mantenha o antivírus e as correções de segurança atualizados para componentes como processadores XML, processadores de texto, processadores de imagem, entre outros que manipulam arquivos fornecidos por usuários.

EXEMPLOS

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0147
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0131
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1227

REFERÊNCIAS

- CWE: CWE-325 (Direct Request), CWE-288 (Authentication Bypass by Alternate Path), CWE-285 (Missing or Inconsistent Access Control)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/predictable_resource_location.shtml
- OWASP, http://www.owasp.org/index.php/Forced_browsing
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Authorization

AONDE IR A PARTIR DAQUI

O OWASP Top 10 é apenas o primeiro passo rumo a segurança das suas aplicações web.
Podemos dividir as 6 Milhões de pessoas que vivem no mundo em dois grupos: O primeiro grupo é formado pelas pessoas que sabem porque a grande maioria das empresas de software lançam seu produtos com Bugs conhecidos e o segundo grupo é formado pelas pessoas que não sabem porque isto ocorre. As pessoas do primeiro grupo geralmente têm seu otimismo juvenil estragado pela dura realidade. Às vezes encontramos pessoas que pertencem aos dois grupos; são as pessoas que estão chocadas por ver que existem empresas lançando seus produtos antes de testá-los e corrigir todos os bugs.
(http://www.guardian.co.uk/technology/2006/may/25/insideit.guardianweeklytechnologysection) Eric Sink, Guardian 25 maio de 2006.
A maioria dos seus clientes e usuários está no primeiro grupo. O modo com o qual você encara este problema é uma oportunidade de aumentar a segurança de suas aplicações web no modo geral pois Bilhões de dólares são perdidos todos os anos e milhões de pessoas sofrem com fraude e roubo de identidade devido a vulnerabilidades discutidas neste documento.

AOS WEB-DESIGNERS

Para garantir a segurança das suas aplicações você precisa saber o que você está protegendo, conheça todas as ameaças e riscos de insegurança e classifique-os de forma estruturada. O Desenvolvimento de qualquer tipo de aplicação requer uma boa dose de segurança.

- Certifique-se que você está aplicando segurança baseando-se no modelo de ameaça de risco, no entanto como os modelos de normas (SOX, HIPAA, Basel,…) estão cada vez mais caros torna-se mais conveniente investir tempo e recurso para satisfazer o mínimo necessário para os dias atuais, porém as normas mais conhecidas são bem mais rígidas.
- Faça perguntas sobre necessidades empresariais, principalmente sobre requisitos não funcionais.
- Trabalhe seguindo o contrato de segurança de Software OWASP.
- Incentivar o desenvolvimento de Software seguro inclui uma defesa profunda e uma construção simples do código utilizando o modelo de ameaça de risco. (ver [HOW1] no livro referências)
- Certifique-se de ter considerado a confidencialidade, integridade, disponibilidade e não-repúdio.
- Certifique-se de que seus Web-Designers são coerentes com a política de segurança e normas, tais como COBIT ou PCI DSS 1,1.

AOS DESENVOLVEDORES

Muitos desenvolvedores já possuem uma boa base sobre segurança no desenvolvimento de aplicações Web porém para garantir uma segurança efetiva no desenvolvimento de aplicações Web requer muita experiência, pois qualquer leigo pode atacar um sistema.

- Faça parte da comunidade e OWASP e freqüente as reuniões regionais.
- Procure por treinamentos sobre desenvolvimento de código seguro.
- Desenvolva suas aplicações com segurança, Crie códigos simples e com profunda segurança.
- Desenvolva com aplicações que favoreçam a segurança do código.
- Reconstrua o código de forma segura de acordo com a sua plataforma utilizando pesquisas otimizadas.
- Leia o guia OWASP e comece a aplicar controles mais seguros a seu código, diferente do outros guias ele é desenvolvido para ajudá-lo a criar aplicações seguras e não a quebrá-las.
- Faça os testes de segurança e defeitos do seu código e torne esta prática constante no seu dia-a-dia.

- Revise o livro de referências e veja se existem opções que se aplicam ao seu ambiente de trabalho.

PARA PROJETOS DE CÓDIGO ABERTO

O código aberto é um desafio particular para a segurança de aplicações web. Existem milhares de projetos de código aberto tanto pessoais como o Apache (www.apache.org) e Tomcat (tomcat.apache.org/) quanto em larga escala como PostNuke (www.postnuke.com).

- Faça parte da comunidade e OWASP e freqüente as reuniões regionais.
- Se o seu projeto possui mais de quatro desenvolvedores, deixe pelo menos um deles responsável pela segurança.
- Desenvolva suas aplicações com segurança, Crie códigos simples e com boa segurança.
- Desenvolva com normas que favoreçam a segurança do código.
- Divulgue a política de segurança de forma responsável para garantir que os problemas de segurança estão sendo tratados corretamente.
- Revise o livro de referências e veja se existem opções que se aplicam ao seu ambiente de trabalho.

PARA OS PROPRIETÁRIOS DE APLICAÇÃO

Os proprietários de aplicações comerciais geralmente têm tempo e recursos reduzidos. Os Proprietários de aplicações devem:

- Trabalhar com o contrato de segurança de software OWASP anexo com os produtores de softwares.
- Garantir que os requisitos de negócio incluem requisitos não-funcionais (non-functional requirements, NFRs), tais como requisitos de segurança.
- Estimule os desenvolvedores a criar aplicações com código simples e com boa segurança.
- Contrate (ou treine) desenvolvedores com bons conhecimentos em segurança.
- Faça testes de segurança em todo o projeto, desenvolvimento, criação, testes e implementação.
- Reserve recurso e tempo no orçamento do projeto para cuidar de questões de segurança.

AOS DIRETORES EXECUTIVOS

Sua empresa precisa ter um ciclo de vida de desenvolvimento seguro. As vulnerabilidades são mais fáceis de serem corrigidas durante o desenvolvimento do que após o produto já ter sido lançado. Possuir um ciclo de vida de desenvolvimento seguro não inclui somente os testes para o Top 10, também inclui:

- Ao vender software assegure-se de estar incluindo políticas e contratos com termos de segurança.
- Para código customizado adote codificação segura, principalmente nas suas políticas e normas.
- Para código customizado adote codificação segura, principalmente nas suas políticas e normas.
- Para código customizado adote codificação segura, principalmente nas suas políticas e normas.
- Notifique seus produtores de software sobre a importância da segurança para os resultados da empresa.
- Treine seus web-designers e projetistas nos fundamentos de segurança em aplicações web.
- Considere a possibilidade do código ser auditado por terceiros para que haja uma análise mais independente.
- Adote práticas responsáveis de divulgação e construa um processo para responder adequadamente aos relatórios de vulnerabilidade dos seus produtos.

REFERÊNCIAS

PROJETOS DA OWASP

OWASP é o site principal sobre segurança de aplicações web. O site da OWASP hospeda diversos projetos, fóruns, blogs, apresentações, ferramentas e artigos. A OWASP organiza duas grandes conferências de segurança por ano e mais de 80 capítulos locais.
Os seguintes projetos da OWASP são mais comuns de serem utilizados:

- OWASP Guide to Building Secure Web Applications
- OWASP Testing Guide
- OWASP Code Review Project (in development)
- OWASP PHP Project (in development)
- OWASP Java Project
- OWASP .NET Project

LIVROS

Por necessidade, esta não é uma lista exaustiva. Use-a como referência para encontrar a área apropriada em sua livraria local e selecione alguns títulos (incluindo um ou mais dos seguintes) que satisfaçam suas necessidades:

- [ALS1] Alshanetsky, I. “php|architect's Guide to PHP Security”, ISBN 0973862106
- [BAI1] Developing more secure ASP.NET 2.0 Applications”, ISBN 978-0-7356-2331-6
- [GAL1] Gallagher T., Landauer L., Jeffries B., "Hunting Security Bugs", Microsoft Press, ISBN 073562187X
- [GRO1] Fogie, Grossman, Hanse Cross Site Scripting Attacks: XSS Exploits and Defense”, ISBN 1597491543
- [HOW1] Howard M., Lipner S., "The Security Development Lifecycle", Microsoft Press, ISBN 0735622140
- [SCH1] Schneier B., "Practical Cryptography", Wiley, ISBN 047122894X
- [SHI1] Shiflett, C. “Essential PHP Security”, ISBN 059600656X
- [WYS1] Wysopal et al, The Art of Software Security Testing: Identifying Software Security Flaws, ISBN 0321304861

WEB SITES

- OWASP, http://www.owasp.org
- MITRE, Common Weakness Enumeration – Vulnerability Trends, http://cwe.mitre.org/documents/vuln-trends.html
- Web Application Security Consortium, http://www.webappsec.org/
- SANS Top 20, http://www.sans.org/top20/
- PCI Security Standards Council, publishers of the PCI standards, relevant to all organizations processing or
- Holding credit card data, https://www.pcisecuritystandards.org/
- PCI DSS v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Build Security In, US CERT, https://buildsecurityin.us-cert.gov/daisy/bsi/home.html