Relatório TP1 - OWASP Proactive Controls

Roberto Cachada (a81012) Pedro Gonçalves (a82313)

Braga, março de 2020

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Engenharia de Segurança
(2º Semestre/2019-2020)
Grupo 7

Conteúdo

1 Introdução

Nos dias que correm, os diferentes *softwares* utilizados pela população mundial são um dos alvos mais susceptiveis a ataques. Uma das razões para tal, é o facto de esses, muitas das vezes, serem desenvolvidos sem preocupações com segurança. Para tentar combater este facto, a OWASP(Open Web Application Security Project), desenvolveu uma lista de categorias técnicas/-medidas de segurança, *OWASP Proactive Controls*, que os desenvolvedores de *software* devem ter em mente aquando do processo de criação de um novo produto, de modo a minimizar os riscos de segurança do produto antes de o lançarem no mercado, daí serem medidas *proactive*.

Assim sendo, este relatório irá descrever essas medidas, e a forma como as mesmas ajudam a criar *software* mais seguro.

2 OWASP Proactive Controls

Esta lista possui 10 categorias, ordenadas de 1 a 10 consoante a ordem decrescente de importância. As 10 categorias, ou controlos, são as seguintes:

- C1: Define Security Requirements;
- C2: Leverage Security Frameworks and Libraries;
- C3: Secure Database Access;
- C4: Encode and Escape Data;
- C5: Validate All Inputs;
- C6: Implement Digital Identity;
- C7: Enforce Access Controls;
- C8: Protect Data Everywhere;
- C9: Implement Security Logging and Monitoring;
- C10: Handle All Errors and Exceptions.

2.1 Define Security Requirements

Segundo o OWASP, um *security requirement* é a constatação de uma funcionalidade de segurança que deva estar presente no *software*, de modo a assegurar certas propriedade de segurança, para que sejam eliminadas potenciais vulnerabilidades no *software*. Estes requisitos são geralmente derivados de *standards* da indústria, leis e do histórico de vulnerabilidades já conhecidas e existem para evitar a ocorrência de falhas de segurança já descobertas pela comunidade.

A importância de definir requisitos de segurança antes da construção de um novo produto de *software*, é que está a ser criada uma base de funcionalidades seguras mesmo antes de ser escrita uma linha de código, o que evitará a criação de vulnerabilidades durante o processo de desenvolvimento.

Para se tirar o maior proveito desta capacidade de prevenção do apareceimento de vulnerabilidades, é recomendado o uso de requisitos de segurança *standard*, tais como o *OWASP Application Security Verification Standard (ASVS)* e os presentes nas *Cheat-Sheets* do *OWASP Proactive Controls*.

2.2 Leverage Security Frameworks and Libraries

Tal como já foi anteriormente mencionado, é importante a implementação de funcionalidades de segurança durante o processo de construção de *software*. Para tal é recomendada a utilização de bibliotecas e *frameworks* que tratem da segurança do projeto, permitindo a poupança de recursos por parte da equipa de desenvolvimento. Outro motivo pelo qual é recomendado a utilização destas bibliotecas, é que essas são atualizadas aquando do aparecimento de novas vulnerabilidades, o que permite a obtenção de um *software* seguro ao longo do tempo, desde que essas mesmas bibliotecas sejam utilizadas de forma correta, isto é, aquando uma atualização nas bibliotecas se atualize o projeto de modo a que o mesmo continue seguro e também que essas bibliotecas sejam utilizadas de forma encapsulada, utilizando apenas o necessário para garantir a segurança do produto de *software*.

2.3 Secure Database Access

Esta secção serve para descrever as melhores práticas para implementar um acesso seguro a bases de dados pelo *software*.

Secure queries

Para evitar ataques, como injeções de SQL, aquando da execução de *queries* com a base de dados é preciso implementar parametrização das *queries*, que consiste na pré-compilação de declarações SQL que apenas necessitam de parâmetros, impedindo assim que sejam executadas *queries* maliciosas. No entanto, certas partes das bases de dados podem não permitir esta parametrização, sendo necessário fazer outro tipo de controlo de *input* de modo a evitar ataques de injeção SQL.

Secure Communication and Authentication

Durante a execução de *queries*, estas e os seus resultados devem ser sempre encriptados, de modo a não revelarem informações do sistema. Além disso, todos os acessos à base de dados devem ser autenticados através de canais seguros.

Secure Configuration

Por fim, antes da utilização de uma qualquer base de dados, devem ser ativados os controlos de segurança das mesmas, configurando-os de modo a proteger os dados do sistema.

2.4 Encode and Escape Data

Encoding and Escape são técnicas utilizadas com o objetivo de evitar ataques de injeção.

Encoding, também denominado *Output Encoding*, consiste na tradução de caracteres especiais para formas equivalentes mas que não sejam ameaças para o interpretador desses caracteres. Um exemplo de *encoding* consiste em traduzir o caractere < para **<** aquando da escrita de uma página HTML, evitando assim possíveis pontos para ataques de injeção.

Escaping consiste em acrescentar caracteres especiais antes de certos elementos, tais como "'", de modo a que sejam interpretados da maneira pretendida.

No entanto, este processo deve ser executado o mais tarde possível, de modo a evitar que não se possa utilizar esse conteúdo noutras partes do sistema.

2.5 Validate All Inputs

Validação de *inputs* é uma técnica utilizada com o intuito de garantir que todo o *input* de um programa esteja formatado corretamente. No entanto, é importante referir que mesmo validando o *input* utilizando as técnicas apresentadas em baixo, não significa que o mesmo não possa ser perigoso para a integridade do sistema.

Existem dois tipos de validação que devem ser verificados a cada *input*, sendo eles:

- Validade Sintática - Valida se os dados introduzidos estão na forma esperada, isto é, se for pedido um número com 4 dígitos, verifica se o *input* recebido se trata de um número com 4 dígitos.

  Existem duas abordagens para fazer este tipo de validação, sendo elas:

  - *Whitelisting* - Verifica se o *input* cumpre um dado conjunto de regras 'boas' pretendidas;
  - *Blacklisting* - Verifica se existe algum tipo de conteúdo malicioso no *input*. Esta é uma técnica onde é díficil cobrir todos os possíveis casos, mas ajuda a evitar ataques mais óbvios, por exemplo, procurando <**SCRIPT**> no *input*.

- Validade Semântica - Aceita apenas *inputs* que façam sentido no contexto da aplicação, por exemplo, uma data inicial deve ocorrer antes de uma data final.

Convém referir que é recomendado ter pelo menos algum tipo de *whitelisting* presente na validação de *inputs*, para limitar a superfície de ataques possíveis, visto ser possível evitar *blacklisting*.

No caso de a aplicação ser do tipo Cliente-Servidor, a validação de *input* deve ser sempre feita do lado do servidor por motivos de segurança, pois a validação do lado do Cliente pode ser facilmante ultrapassada, pois este não possui as ferramentas necessárias para validar corretamente o *input*.

Regular Expressions

Expressões Regulares (ER), podem ser utilizadas para validar se um *input* cumpre certos padrões, mas é necessário ter em atenção se as mesmas são implementadas corretamente, pois podem provocar **REDoS**, **DoS** provocado pelo facto de existirem *inputs* que fazem com que as ERs façam uma validação muito lenta, o que pode levar atacantes a criarem *inputs* longos de modo a atacarem o sistema. Existem no entanto ferramentas para verificar se uma ER é susceptível a este tipo de ataque.

Além das desvantagens referidas no parágrafo anterior, a utilização de ER pode ser de díficil compreensão para certos programadores, o que pode dificultar o seu uso e manutenção.

Serialized Data

Dados Serializados é um dos tipos de dados onde se torna complicado fazer validações de *input*. Assim sendo, sempre que possível, deve ser evitado o uso deste tipo de dados como forma de *input*. Caso não seja possível evitar o uso deste tipo de dados, devem ser implementadas as seguintes sugestões para validação destes dados:

- Implementar medidas para verificar a integridade deste tipo de dados, ou encriptar os mesmos para evitar a criação de objetos hostis ou até mesmo *data tampering*;
- Isolar, em ambientes com poucos privilégios (i.e *cointainers* temporários), partes de código que validem este tipo de dados;
- Restringir e monitorizar a conectividade na rede dos *cointainers* temporários responsáveis por validar este tipo de dados;
- Definir tipos de dados rigorosos durante a deserialização antes da criação de objetos. Existem no entanto formas de evitar esta técnica;

- Monitorizar os processos de deserialização, e notificar os administradores do sistema para quando um utilizador o estiver a fazer de forma constante;
- Fazer *logging* de erros/excepções que ocorram durante o processo de deserialização, por exemplo, quando o tipo de dados extrari não corresponder ao tipo esperado.

2.6 Implement Digital Identity

Antes de prosseguir é necessário definir dois conceitos importantes para este tipo de controlo:

- *Digital Identity* - Identidade digital corresponde à informação única, acerca de um agente que está envolvido numa qualquer transação *online*;
- *Session Management* - Processo no qual um servidor mantém o estado de autenticação de um dado utilizador, de forma a eveitar reautenticações constantes.

Authentication Assurance Levels (AALs)

AALs são níveis que definem o método de autenticação necessário para obter acesso a uma aplicação. Quanto mais alto o nível, maior a sensibilidade da informação presente.

AALs estão divididas em 3 níveis:

- Nível 1: *Passwords*;
- Nível 2: *Multi-Factor Authentication (MFA)*;
- Nível 3: *Cryptographic Based Encryption*.

Nível 1: *Passwords*

Passwords devem ser armazenadas de forma segura, e devem também poder ser alteradas. Para reforçar a segurança fornecida, estas devem cumprir alguns requisitos de modo a oferecerem maior resistência a ataques:

- Possuír, pelo menos, 8 caracteres se existir *Multi-Factor Authentication (MFA)*, caso contrário, este número deve ser aumentado para, pelo menos, 10 caracteres;

- Todos os caracteres ASCII que sejam imprimíveis (números, letras, acentos, pontuação, entre outros), devem ser aceites;
- Encorajar o uso de *passwords* e *passphrases* longas;
- Não utilizar requisitos de complexidade, pois já foi demonstrado possuírem efetividade limitada, e optar por adotar MFA ou *passwords* mais longas;
- Garantir que as *passwords* utilizadas não são *passwords* comuns, isto é, que já tenham sido comprometidas. Para tal, bloquear as 1000/10000 *passwords* mais comuns que cumpram os requisitos de comprimento e que se encontrem em listas de *passwords* já comprometidas.

No que toca à recuperação de *passwords*, este processo deve utlizar, sempre que possível, elementos de MFA. Por exemplo perguntas de segurança e/ou envio de *tokens* para um dispositivo que pertença ao utilizador.

Para além disso, as *passwords* armazenadas pelo sistema, devem estar guardadas de forma segura, existindo controlos criptográficos para que, caso sejam comprometidas, não permitir ao atacante acesso imediato a esses segredos.

Nível 2: Multi-Factor Authentication (MFA)

Geralmente, este nível de autenticação é utilizado em aplicações que contenham PII (*personal identifiable information*) disponível *online*.

MFA assegura a identidade de um utilizador, ao pedir que o mesmo se autentique utilizando uma combinação de:

- Algo que o utilizador conheça - *Password* ou PIN;
- Algo que pertença ao utilizador - *Token* ou telemóvel;
- Algo que faça parte do utilizador - Algum tipo de dados biométricos. Este tipo de dados, por si só, não oferece um nível de segurança aceitável, pois podem ser obtidos de forma relativamente fácil.

Ao pedir que a autenticação seja feita desta forma, o sistema torna-se mais robusto, pois o atacante precisa de ganhar acesso a mais do que um elemento de um utilizador para se fazer passar pelo próprio.

Nível 3: Cryptographic Based Authentication

Este nível de segurança é utilizado quando um sistema comprometido resulta em danos pessoais, financeiros, do interesse público ou se envolverem infrações civis ou criminais. A autenticação utilizada nestes casos, é baseada na prova de posse de uma chave através de protocolos criptográficos. Geralmente, é implementado usando módulos de hardware criptográfico, como por exemplo, o cartão de cidadão.

Session Management

O estado de autenticação do utilizador, é mantido numa sessão guardada num servidor. Um utilizador recebe um ID de sessão, para que saiba que sessão *server-side* possui a sua informação. O utilizador apenas precisa deste ID, o que implica que toda a informação sensível está do lado do servidor, o que é o recomendado.

De seguida, encontram-se algumas boas práticas para implementar *session management*:

- IDs de sessão devem ser longos, únicos e aleatórios;
- Deve ser gerado um novo ID de sessão durante a autenticação e re-autenticação;
- Após um certo período de inatividade, a sessão deve expirar e o utilizador deve-se autenticar de novo. O intervalo de tempo utilizado deve ser inversamente proporcional ao valor da informação protegida.

Browser Cookies

Cookies são métodos utilizado por muitas aplicações *web*, para armazenar IDs de sessão. No entanto, aquando da sua utilização, devem ser implementadas as seguintes defesas:

- Quando utilizadas para manter IDs de sessão, as *cookies* devem expirar em simultâneo com a sessão, ou logo de seguida;
- As *flags "secure"* e *"HttpOnly"* devem estar ativas, de forma a garantir que uma transferência é feita apenas através de uma conexão segura (TLS), e para impedir que as *cookies* sejam acedidas via JavaScript, respetivamente;

- Deve ser adicionado o atributo *"samesite"* para impedir que *browsers* modernos enviem *cookies* com *cross-site requests*, o que ajuda a evitar *cross-site request forgery* e fugas de informação.

Tokens

Para algumas aplicações, a utilização de sessões *server-side* pode ser um contra-tempo. Assim, para evitar carga extra nos servidores, essas aplicações *"stateless"* geram um *token*, com pouco tempo de validade, que pode ser utilizado para autenticar os pedidos do utilizador sem necessitar do reenvio das credenciais após a autenticação inicial.

2.7 Enforce Access Control

Controlo de Acesso consiste num processo cujo objetivo é conceder/-negar pedidos feitos por utilizadores, programas ou processos. É também responsável por gerir os privilégios de acesso ao sistema por parte dos vários utilizadores do mesmo.

Este tipo de controlo é importante, pois é indesejável permitir acesso indiscriminado à totalidade do sistema por parte de qualquer entidade. Assim sendo, antes da implementação de um protocolo de controlo de acessos, é necessário definir qual a política que o mesmo irá implementar. De seguida estão apresentadas as políticas de controlo de acesso mais populares:

- Discretionary Access Control (DAC) - Nesta política, é o dono do objeto que decide quem tem acesso ao mesmo;
- Mandatory Access Control (MAC) - Nesta política, os recursos aos quais se pretendem aceder possuem um determinado nível de autorização. Para se aceder a um dado recurso, é necessário possuir um nível de autorização igual ou superior ao do recurso;
- Role Based Access Control (RBAC) - Nesta política, o acesso a um recurso é determinado pela *role* de quem tenta aceder ao mesmo possui no sistema;
- Attribute Based Action Control (ABAC) - Por fim, esta política garante acesso aos recursos consoante atributos dos utilizadores, como por exemplo, ser necessário possuir mais de 18 anos para aceder ao recurso.

Depois de definida a política de acesso, é necessário implementar o protocolo de controlo de acesso, tendo em conta os seguintes passos:

- Os controlos de acesso devem ser planeados na sua totalidade antes de começar o processo de desenvolvimento pois, como estes tendem a evoluir para algo complexo, torna-se difícil fazer alterações após a implementação;
- Todos os pedidos devem passar por algum tipo de controlo de acesso;
- Deve ser *standard* negar todos os pedidos de acesso, a não ser que estes sejam especificamente permitidos;
- Dar o menor acesso possível a todos os utilizadores, programas e processos;
- Não definir as *roles* dos utilizadores de forma *hardcoded*, pois além de tornar a manutenção do sistema de controlo de acesso difícil e trabalhosa, pode levar a acessos indevidos ao sistema;
- Todos os eventos do sistema de controlo de acesso devem ser registados em *logs*, pois tentativas falhadas podem ser um sinal de *probing* malicioso.

2.8 Protect Data Everywhere

Esta categoria serve para indicar quais as medidas necessárias para manter a informação de um sistema segura.

Tendo em conta que nem todos os dados de um sistema possuem o mesmo grau de importância, é necessário criar uma espécie de classificação para os dados consoante a sua importância, de modo a ser possível mapear quais as medidas de segurança que serão necessárias implementar para proteger os diversos dados do sistema tendo em conta o seu grau de importância.

Aquando da necessidade da transmissão de dados, estas devem possuir encriptação *end-to-end*, de modo a que estejam protegidos no caso se serem interceptados por atacantes ativos.

Se o sistema necessitar de armazenar dados localmente, esses devem ser guardados de forma encriptada, de modo a que seja possível evitar que a mesma seja revelada ou modificada sem autorização.

Por fim, é também necessário garantir que as chaves utilizadas para proteger os dados são armazenadas de forma correta e que todos os certificados/chaves sejam renovados periódicamente, de modo a garantir uma maior segurança para os dados.

2.9 Implement Security Logging and Monitoring

A implementação de um sistema de *logging*, para além de ser uma ferramenta muito útil para *debugging*, se for corretamente implementada, permite servir como uma barreira de defesa para identificar atividade suspeita no sistema, por exemplo através da utilização de um IDS, aumentando assim a segurança da aplicação.

Apesar de útil, se um sistema deste tipo não for corretamente implementado, pode servir como uma fonte para atacantes obterem informações sobre o sistema, por isso é necessário ter em conta certos fatores aquando da construção de um sistema de *logging*:

- Não se deve armazenar informação sensível nos *logs*, tais como *passwords*;
- Não se deve armazenar informação desnecessária nos *logs*, apenas o essencial para identificação de potenciais ataques;
- Os formatos dos *logs* das várias partes do sistema devem ser consistentes entre si, e aquando da sua sincronização é necessário ter em atenção aos seus *timestamps*;
- Os *logs* de sistemas distribuídos devem ser encaminhados para um sistema central, de modo a permitir uma monotorização central e resiliência no caso da perda de um nodo.

2.10 Handle all Errors and Exceptions

A ocorrência de erros é algo normal durante a utilização de um *software*, por isso é crucial que os mesmos sejam tratados de forma correta através do uso de excepções.

Aquando do tratamento de erros, é preciso ter alguns fatores em consideração de modo a não comprometer a segurança do sistema, nem a revelar informação crítica acerca da aplicação. Assim sendo, algumas recomendações para o correto tratamento de erros são as seguintes:

- Tratar as excepções de forma centralizada de modo a evitar duplicação de código;
- Assegurar que existem excepções capazes de 'apanhar' qualquer comportamento inesperado no sistema;

- Fazer *logging* das excepções com informação suficiente do erro ocurrido, de modo a permitir aos desenvolvedores perceberem o que se passou de errado no sistema;
- Após a deteção de um erro, corrigir o mesmo;
- Testar o código que trata de 'apanhar' erros, para garantir que funciona da maneira esperada.

3 Conclusão

Com este relatório o grupo considera que os seus conhecimentos sobre o aspeto de segurança na produção de *software* aumentou, tendo aprendido quais as áreas mais importantes a proteger durante a sua produção, formas de as proteger e as razões pelas quais essas áreas devem ser protegidas.

No entanto consideramos qua ainda há, como haverá sempre, espaço para aprender mais. Como é mencionado no própio documento do *OWASP Top 10 Proactive Controls*, muitas das áreas discutidas são bastante vastas e há ainda muito que não foi nele explorado.

Contudo, este documento serve como uma boa base para criação de *software* seguro, e o grupo recomenda a utilização do mesmo aquando da criação de novo *software*.

Referências

[1] Anton, K; Manico, J; Bird, J.: OWASP Top 10 Proactive Controls V3 (2018)

[2] OWASP.: owasp.org/www-project-proactive-controls (2020)

[3] OWASP.: owasp.org/www-project-application-security-verification-standard (2020)

[4] OWASP.: github.com/OWASP/CheatSheetSeries (2020)