



Rapport 2009 CWE/SANS des 25 erreurs de programmation les plus dangereuses

Mi-janvier 2009, le SANS (SystAdmin, Audit, Network, Security) Institute publiait un rapport sur les 25 erreurs de programmation concernant la sécurité informatique, et donc de la mauvaise sécurité des logiciels. Le rapport pointe les erreurs les plus communes, leurs impacts et les recommandations pour les réduire. Plus de 30 organisations, éditeurs, institutions européennes et américains ont participé à sa rédaction.

Programmez ! vous en offre une traduction intégrale.

Source du rapport (anglais) : <http://www.sans.org/top25errors/>

Traduit de l'anglais par Gilbert Vidal.

Liste des 25 erreurs les plus fréquentes en conception ou programmation de logiciels

Table des matières

Introduction.....	3
Liste abrégée des 25 Erreurs	3
Interaction non sécurisée entre composants.....	3
Gestion des ressources à risque.....	3
Défense poreuse	4
Construction et sélection de la Liste des 25.....	4
Organisation de la Liste 25	5
Autres types d'information	5
Interaction non sécurisée entre composants.....	6
CWE-20: Mauvaise validation des entrées	6
CWE-116: Codage incorrect des sorties	7
CWE-89: Injection SQL.....	9
CWE-79: Incapacité à protéger une structure de page Web (ou 'script intersites').....	10
CWE-78: Incapacité à préserver une structure de commande de l'OS (ou 'injection de commande système d'exploitation').....	13
CWE-319: Transmission en clair d'information sensible.....	14
CWE-352: Requête frauduleuse intersites (CSRF).....	15
CWE-362: Condition de course	16
CWE-209: Fuite d'information de message d'erreur	17
Gestion des Ressources à Risque	18
CWE-119: Incapacité à limiter des opérations à l'intérieur d'un tampon mémoire.....	19
CWE-642: Contrôle externe de données critiques d'état	20
CWE-73: Contrôle externe de nom de fichier ou chemin d'accès	21
CWE-426: Chemin de recherche non sur	22
CWE-94: Echec du contrôle de la génération de code (injection de code).....	23
CWE-494: Téléchargement de code sans vérification d'intégrité	24
CWE-404: Fermeture ou libération de ressources incorrectes.....	25
CWE-682: Calcul incorrect.....	27
Défenses Poreuses.....	28
CWE-285: Contrôle d'accès incorrect (Autorisation)	28
CWE-327: Utilisation d'un algorithme cryptographique cassé ou risqué	30
CWE-259: Mot de passe codé en dur.....	31
CWE-732: Assignation de permission non-sécurisée pour des ressources critiques.....	32
CWE-330: Utilisation de valeurs pas assez aléatoires	33
CWE-250: Exécution avec privilèges non nécessaires	34
CWE-602: Gestion coté client de la sécurité coté serveur.....	35
Appendice A : Critères de sélection et champs de définition.....	36
Prévalence de la vulnérabilité	36
Conséquences.....	37
Fréquence des attaques.....	37
Facilité de détection	37
Cout de réparation.....	38
Sensibilisation de l'attaquant	38
CWE Liées	38
Appendice B : Modèle de menace de l'attaquant entraîné et déterminé.....	38

Introduction

Le rapport 2009 CWE/SANS des 25 erreurs de programmation les plus dangereuses est une liste des erreurs de programmation les plus importantes pouvant fragiliser la sécurité des logiciels, On les trouve fréquemment, et elles sont aisées à trouver et aisées à exploiter. Elles sont dangereuses car elles permettent souvent aux attaquants de prendre le contrôle total du logiciel, voler des données, ou empêchent le logiciel de fonctionner,

Cette liste est issue de la collaboration entre les instituts SANS, MITRE et un grand nombre d'experts en matière de sécurité logiciel aux USA et en Europe. Elle s'appuie sur le développement des 20 vecteurs d'attaque SANS (<http://www.sans.org/top20>) et sur la liste des faiblesses les plus communes (CWE) développée par MITRE (<http://cwe.mitre.org>), MITRE assure la maintenance du site web CWE, avec l'aide de la division sur la sécurité informatique du département US de la Sécurité Intérieure, Cette liste présente des descriptions détaillées des 25 erreurs de programmation avec les solutions pour les éviter ou en diminuer les conséquences. Le site CWE contient également des informations sur plus de 700 erreurs de programmation additionnelles et les erreurs d'architecture qui peuvent permettre d'en exploiter les vulnérabilités.

Le but principal de la liste des 25 est d'éduquer les programmeurs afin d'éviter les erreurs à la source, et d'éliminer les problèmes les plus fréquents avant même la distribution du logiciel. Cette liste sera un outil qui permettra aux programmeurs d'éviter les erreurs majeures qui sont la plaie de cette industrie. Les utilisateurs de logiciel pourraient utiliser la liste afin d'exiger des logiciels plus surs. Finalement les responsables des développements et CIO peuvent utiliser la liste afin de contrôler l'évolution de la sécurité des logiciels.

Liste abrégée des 25 Erreurs

La liste des 25 est organisée en 3 catégories contenant plusieurs CWE

Interaction non sécurisée entre composants

Ces faiblesses sont liées à l'échange de données non sécurisées entre des composants, modules, , programmes, processus ou systèmes,

- . CWE-20: Mauvaise validation des entrées
- . CWE-116: Mauvais codage des sorties
- . CWE-89: Incapacité à protéger une structure de requête SQL (ou 'injection SQL')
- . CWE-79: Incapacité à protéger une structure de page Web (ou 'script intersites')
- . CWE-78: Incapacité à préserver une structure de commande de l'OS (ou 'injection de commande système d'exploitation')
- . CWE-319: Transmission en clair d'information sensible
- . CWE-352: Requête frauduleuse intersites (CSRF)
- . CWE-362: Condition de course
- . CWE-209: Fuite d'information de message d'erreur

Gestion des ressources à risque

Les faiblesses regroupées dans cette catégorie sont liées à une mauvaise gestion par le logiciel de la création, utilisation, transfert ou destruction de ressources systèmes importantes.

- . CWE-119: Incapacité à limiter des opérations à l'intérieur d'un tampon mémoire
- . CWE-642: Contrôle externe de données critiques d'état
- . CWE-73: Contrôle externe de nom de fichier ou répertoire
- . CWE-426: Chemin de recherche non fiabilisé
- . CWE-94: Incapacité à contrôler la génération de code (ou 'injection de code')
- . CWE-494: Téléchargement de code sans contrôle d'intégrité
- . CWE-404: Arrêt ou libération de ressource incorrect
- . CWE-665: Initialisation incorrecte
- . CWE-682: Calcul incorrect

Défense poreuse

Les faiblesses de cette catégorie sont liées aux techniques de défense qui sont souvent mal utilisées, abusées ou juste ignorées,

- . CWE-285: Contrôle d'accès incorrect (Autorisations)
- . CWE 327: Utilisation d'un algorithme cryptographique risqué ou cassé
- . CWE-259: Mot de passe en dur
- . CWE-732: Assignation non sécurisée d'autorisation à des ressources critiques
- . CWE-330: Utilisation de valeurs insuffisamment aléatoires
- . CWE-250: Exécution de code avec privilèges insuffisants
- . CWE-602: Gestion coté client de la sécurité coté serveur

Construction et sélection de la Liste des 25

Cette liste a été développée fin 2008. 40 sociétés de sécurité informatique ont contribué à son élaboration, comprenant des éditeurs de logiciels, des développeurs d'outils de scan, des consultants en sécurité, des représentants gouvernementaux et des professeurs d'université. La représentation fut internationale. Plusieurs versions intermédiaires ont été envoyées pour révision avant de finaliser la liste. Plus détails dans la page concernant Le Processus Liste 25.

Pour aider à la caractérisation et au classement les entrées de la liste, un modèle de menace a été développé qui identifie un attaquant ayant des compétences techniques solides et la détermination nécessaire pour investir le temps nécessaire à l'attaque d'une organisation. Plus de détails dans l'Appendice B.

Les faiblesses de la Liste 25 ont été choisies en fonction de deux critères principaux:

- . La prédominance de la vulnérabilité: quel pourcentage de présence trouve t'on dans les logiciels qui n'ont pas été développés avec l'intégration de la sécurité dans le cycle de développement.
- . Les conséquences: les conséquences typiques de l'exploitation de la vulnérabilité comme une exécution de code non prévue, des pertes de données ou suppression de service.

La prévalence a été déterminée à partir des estimations des nombreux contributeurs de la liste, puisque qu'aucune statistique officielle n'est disponible.

De futures versions de La Liste 25 évolueront pour couvrir de nouvelles faiblesses conformes à ces critères.

Voir Appendice A pour plus de détails sur les critères de sélection.

Organisation de la Liste 25

Pour chaque vulnérabilité, des informations complémentaires sont apportées. L'audience visée est celle des programmeurs et concepteurs de logiciels.

- CWE ID et nom
- Autres types d'information: informations supplémentaires sur la vulnérabilité utiles aux décisionnaires pour affecter des priorités.
- Présentation: courte et informelle sur la nature de la vulnérabilité et ses conséquences
- Prévention et atténuation: étapes à suivre par les développeurs pour atténuer les conséquences de la vulnérabilité ou la supprimer. Les développeurs peuvent choisir une ou plusieurs techniques d'atténuation suivant leurs besoins. Notez que l'efficacité de ces techniques varie et que plusieurs peuvent être combinées pour de meilleurs résultats.
- CWE liées: autres faiblesses liées. Cette liste n'est la qu'à but informatif, et n'est pas complète.
- Types d'attaques: références CAPEC des attaques pouvant être conduites contre la vulnérabilité. Notez que cette liste peut être incomplète.

Autres types d'information

Chaque entrée de la Liste 25 contient des informations sur la prévalence de la vulnérabilité et ses conséquences. Elle contient également les informations suivantes:

- Fréquence des attaques liées à la vulnérabilité
- Facilité de détection de la vulnérabilité par les attaquants
- Coût de réparation: quel effort sera nécessaire pour supprimer la vulnérabilité
- Sensibilisation de l'attaquant: la probabilité que l'attaquant connaisse cette vulnérabilité, ses méthodes de détection et d'exploitation.

Voir Appendice A pour plus de détails.

Interaction non sécurisée entre composants

CWE-20: Mauvaise validation des entrées

Résumé

Prévalence de la vulnérabilité	Élevée	Conséquences	Exécution de Code Suppression de service Perte de données
Coût de réparation	Faible	Facilité de détection	Facile à difficile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

C'est le tueur numéro un des logiciels sains. Vous vous attirez des ennuis si vous ne vérifiez pas que les entrées sont conformes aux spécifications. Par exemple une entrée numérique ne doit pas pouvoir contenir de lettres. Ou le prix d'une voiture neuve ne saurait être un Euro, même dans l'économie d'aujourd'hui. Les applications ont souvent des contraintes beaucoup plus complexes que ce simple exemple. Une mauvaise validation des entrées entraînera des vulnérabilités quand les attaquants modifieront les valeurs de manière imprévue. La plupart des vulnérabilités actuelles pourrait être éliminées ou au moins réduite rien qu'en validant correctement les entrées.

Prévention et atténuation des conséquences

Conception et Architecture	Utilisez un cadre de validation des entrées tel Struts ou l'API de validation OWASP ESAPI. Si vous utilisez Struts, soyez conscient des faiblesses décrites dans la catégorie CWE-101
	Comprendre toutes les possibilités de corruption de vos données d'entrée: paramètres, arguments, cookie et tout ce qui est lu en provenance du réseau, variable d'environnement, en-tête de requête et contenu, composants d'URL, courrier électronique, fichiers, bases de données, et tout système externe transférant des données à l'application. Validez les entrées dans des interfaces bien validées;
	Assumez que toute entrées est malicieuse. Utilisez les mécanismes standards de validation des entrées pour vérifier les longueurs, types, syntaxe et règles de gestion avant d'accepter ou stocker les données. Un exemple de règle de gestion: « bateau » peut être correct car ne contenant que des lettres, mais non valide si vous attendiez une couleur ! Utilisez une stratégie d'acceptation des valeurs connues, rejetez toute valeur ne correspondant pas à vos critère ou transformez là en quelque chose qui convienne.
	Dupliquez toutes les vérifications coté Client sur le Serveur pour éviter CWE-602. Ces vérifications non seulement diminuent le temps passé par le serveur pour les utilisateurs normaux qui ne connaissent pas le format de saisie. Les attaquants peuvent éviter ces mécanismes très simplement en interceptant les paramètres après la vérification coté Client, et les modifiant avant transmission au serveur. Ce devrait être simple à implémenter, et devrait éviter la probabilité d'utilisation de paramètres non sécurisés dans l'application. On ne doit pas ignorer les vérifications coté Client. Tout d'abord ils peuvent être utiles à la détection d'intrusion. Si le serveur reçoit

	des données qui auraient du être rejetées par le client, cela peut être un signe d'attaque. De plus les modèles AJAX nécessitent des validations d'entrées pour prévenir les problèmes DOM ou XSS.
	Ne pas se baser uniquement sur des listes noires pour détecter les entrées malicieuses ou pour encoder les sorties (CWE-184). Il existe trop de méthodes différentes pour encoder un caractère donné, et vous pourriez oublier une variante.
Implémentation	Quand votre application combine des données venant de sources multiples, validez-les après leur intégration. Les données individuelles peuvent être correctes, mais pas leur combinaison.
	Soyez spécialement prudent pour la validation des données quand vous appelez du code qui dépasse les limites de langage, comme celle entre un code interprété et Un code natif. Cela pourrait provoquer une interaction imprévue entre les limites des langages. Vérifiez que vous ne violez aucune des attentes du langage avec lequel vous vous interfacez. Par exemple, alors que Java n'est pas sensible au dépassement de tampon, fournir un argument trop grand lors d'un appel à du code natif peut provoquer un dépassement de tampon.
	Convertissez directement une entrée dans le type demandé par l'application, en utilisant par exemple une fonction de conversion de caractère en nombre. Après la conversion, vérifiez que le résultat est bien dans les bornes de validité et que l'intégrité multi-champs est bien maintenue.
	Les entrées doivent être décodées dans le format de représentation interne de l'application avant d'être validées (CWE-180 et CWE-181). Vérifiez bien que votre application ne décode pas la même entrée deux fois (CWE-174). Une telle erreur pourrait court-circuiter un schéma de liste de validation, en introduisant des valeurs dangereuses après la validation. Utilisez des bibliothèques telles OWASP ESAPI.
	En échangeant des données entre les composants, vérifiez que les composants utilisent le même type de codage des caractères. Indiquer explicitement le codage à utiliser chaque fois que le protocole le permet.

CWE liées

CWE-184: liste noire incomplète

CWE-74: Injection

CWE-79: Script multi-sites (XSS)

CWE-89: Injection SQL

CWE-95: Injection Eval

Type d'attaques

CAPEC-ID:

CWE-116: Codage incorrect des sorties

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Exécution de code Perte de données
Coût de réparation	Bas	Facilité de détection	Facile à modérée
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Les ordinateurs ont l'habitude dérangeante de faire ce que vous dites et non ce que vous voulez. L'insuffisance de codage des sorties est l'enfant ignoré des mauvaises validations des entrées, mais est à la racine des attaques basées sur l'injection, si fréquentes en ce moment. Un attaquant peut modifier les commandes que vous envoyez à un autre composant, entraînant la compromission totale de votre application – sans oublier la possibilité donnée à l'attaquant d'utiliser des failles d'autres composants auxquels il n'aurait pas du avoir accès. Cela transforme « fais ce que je souhaite » en « fais ce que l'attaquant dit ». Quand votre programme génère des sorties vers d'autres composants sous la forme de messages structurés comme des requêtes, il doit séparer l'information de contrôle et les métas-données des données réelles. On l'oublie trop souvent car beaucoup de paradigmes mélangent les données et les commandes dans le même envoi. Le Web 2.0 est un bon exemple qui fonctionne en atténuant ces limites. Cela expose aux attaques.

Prévention et atténuation des conséquences

Architecture et conception	Utilisez des procédures stockées ou d'autres mécanismes si disponibles. Ces procédures devraient typiquement coder des caractères spéciaux pour communiquer avec le composant.
	Comprenez bien le contexte dans lequel vos données vont être utilisées, spécialement aux frontières des composants. Comprenez bien comment vos données seront transmises entre les composants. Etudiez tous les protocoles de communication et les représentations de données pour déterminer les stratégies de codage. Soyez particulièrement attentif aux environnements tels que WWW, dans lequel le codage dépend de l'emplacement dans un même document. (script, URI, CSS, attributs, ou corps HTML)
	Appliquez l'encodage ad-hoc à chaque interface. Utilisez des bibliothèques qui utilisent le codage adéquat ou le gère automatiquement, comme l'API de codage ESAPI. Ou utilisez des fonctions intégrées, mais enrobez les au cas où une vulnérabilité serait découverte.
	Dans certains cas, la validation des entrées peut être une stratégie importante quand le contrôle du codage des sorties n'est pas une solution. Par exemple vous pouvez fournir une même sortie à des clients qui utilisent des codages ou représentations différents. Dans d'autres cas, vous pourriez avoir besoin d'autoriser des entrées contenant des contrôles comme des tags HTML qui supportent le formatage de Wiki ou de forum. Quand vous rencontrez ce type de demande, utilisez une liste blanche stricte pour limiter les séquences de contrôle utilisées. Vérifiez bien la structure syntaxique en entrée. Utilisez votre stratégie de codage normale pour le reste des entrées.
	Utilisez la validation des entrées comme mesure de défense approfondie pour réduire la probabilité d'erreur de codage des sorties (cf. CWE-20).
Conditions de base	Spécifier quels codes sont nécessaires pour les composants communiquant
Implémentation	Dans l'échange de données inter-composants vérifiez qu'ils utilisent bien le

	même type de codage. Spécifiez-le explicitement si le protocole le permet.
	Utilisez des outils d'analyse automatiques statiques et dynamiques qui ciblent ce type de vulnérabilité. Les techniques modernes utilisent des analyses de flux pour minimiser le nombre de faux positifs. Ca n'est pas parfait puisque la précision 100% est impossible.

CWE liées

CWE-74: Injection

CWE-78: Injection système d'exploitation

CWE-79: Script multi-sites (XSS)

CWE-88: Injection d'arguments

CWE-89: Injection SQL

CWE-93: Injection CRLF

Type d'attaques

CAPEC-ID:

CWE-89: Injection SQL

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquence	Perte de données Sécurité
Coût de réparation	Bas	Facilité de détection	Facile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Aujourd'hui il semble que le logiciel ne traite que des données: les ranger dans une base de données, les en extraire, les formater et les envoyer ailleurs pour le jeu ou le profit. Si des attaquants peuvent influencer sur le SQL que vous utilisez pour communiquer avec votre base, ils peuvent également faire des choses pas belles pour le jeu ou le profit. Si vous utilisez SQL pour la sécurité et les autorisations, les attaquants pourrait utiliser la logique de ces requêtes pour court-circuiter la sécurité. Ils pourront modifier les requêtes pour voler ou corrompre les données. Ils pourront même voler les données un octet à la fois, et ils en ont la patience et la connaissance.

Prévention et atténuation des conséquences

Conditions de base	Choisir les environnements de développement BDD et les langages permettant la mise en place des solutions listées ici.
Architecture et conception	Considérez l'utilisation des couches de persistance comme Hibernate ou Enterprise Java Beans, qui procureront une sécurité contre l'injection SQL si ils sont utilisés proprement.
	Traitez les requêtes SQL en utilisant des requêtes préparées, paramétrées ou des procédures stockées. Elles devront accepter des paramètres ou variables fortement typés. Ne pas construire ou exécuter des requêtes en utilisant

	« exec » ou assimilé, car vous introduiriez la possibilité d'une injection SQL.
	Suivez le principe du privilège minimum en créant les comptes utilisateurs de votre BDD. Les utilisateurs doivent avoir le minimum de privilèges pour utiliser leur compte. Si les besoins d'un système nécessite le droit pour chaque utilisateur de lire ou modifier ses données, alors limitez les autorisations pour qu'ils ne puissent pas modifier celles des autres. Utilisez les permissions les plus strictes sur les objets de la base telle que « exécuter uniquement » pour les procédures stockées.
	Dupliquez tous les filtres clients sur le serveur. Un attaquant peut court-circuiter les filtres coté client (CWE-602).
Implémentation	Si vous devez malgré tout utiliser des requêtes dynamiques, utiliser un codage correct. Au lieu de construire votre propre implémentation, utilisez celle présente dans la base ou le langage. Par exemple DBMS_ASSERT d'Oracle peut vérifier que les paramètres ont certaines propriétés qui les rendent moins vulnérables aux injections SQL. Pour MySQL la fonction mysql_real_escape_string() est disponible en C ou PHP.
	Alors que le codage propre des sorties est la solution la plus efficace pour éviter l'injection SQL, la validation des entrées peut procurer une défense en profondeur. Utilisez une vérification rigoureuse par liste blanche de toute entrée utilisateur pouvant être utilisée comme commande SQL. Définissez de manière étroite les caractères à utiliser en toute sécurité, en fonction des valeurs attendues des paramètres. Il vaut mieux interdire tout à fait les méta-caractères. Car une utilisation ultérieure des données enregistrées dans la base pourrait ne pas vérifier les méta-caractères avant utilisation. Cette approche ne vous protégera pas toujours de l'injection SQL surtout si vous devez intégrer des champs libres intégrant tous les caractères. O'Reilly devrait passer sans problème les étapes de validation car il s'agit d'un nom commun en Anglais. Toutefois il ne peut être inséré directement dans la base à cause de l'apostrophe. Dans ce cas supprimer l'apostrophe pourrait régler le problème au détriment de la qualité des données.

CWE liées

CWE-564: Injection SQL: Hibernate

CWE-566: Court circuit du contrôle d'accès en utilisant une clé primaire SQL

CWE-619: Injection Curseur

CWE-90: Injection LDAP

Type d'attaques

CAPEC-ID:

CWE-79: Incapacité à protéger une structure de page Web (ou 'script intersites')

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Exécution Code Sécurité compromise
Coût de réparation	Basse	Facilité de détection	Facile
Fréquences des attaques	Souvent	Sensibilisation Attaquant	Elevée

Présentation

Le script intersites (XSS) est l'une des vulnérabilités les plus fréquentes, obstinée et dangereuse dans les applications web. Elle est à peu près inévitable quand vous combinez la nature informelle de HTTP, le mélange de données et de script de HTML, le volume de données passant entre les sites, les différents codage de données, et les navigateurs aux fonctionnalités riches. Si vous n'êtes pas prudent, les attaquants pourront injecter du JavaScript ou autre code exécutable dans la page web générée par votre application. Votre page est alors vue par d'autres utilisateurs dont le navigateur exécutera le code malicieux, comme s'il provenait de votre application. D'un seul coup, votre site web fournit du code que vous n'avez pas écrit. L'attaquant peut alors utiliser toute une série de techniques pour introduire les données dans votre serveur, ou utilise une victime inconsciente comme intermédiaire.

Prévention et atténuation des conséquences

Architecture et conception	Utilisez des bibliothèques ou Framework qui facilitent la génération de sorties sécurisée en environnement web. Par exemple la bibliothèque anti-XSS de Microsoft, le module d'encodage OWASP ESAPI ou Wicket d'Apache
	Dupliquez les contrôles Client sur le Serveur, car les attaquants peuvent court-circuiter les contrôles coté client (CWE-602)
Implémentation	Quand vous codez une sortie vers une page web, directement ou indirectement, comprenez bien le contexte d'affichage, y compris le type d'encodage attendu. Le type de codage du positionnement de la sortie, dans le corps HTML, un attribut d'un élément, URI, section JavaScript, CSS etc..
	Utilisez un codage caractère fort, tel qu'ISO-8859 ou UTF-8. Quand le type de codage n'est pas spécifié, le navigateur peut choisir un autre type en essayant de deviner quel type de code est utilisé par la page web. Cela peut vous rendre vulnérable à certains types d'attaque XSS subtile liées à ce type d'encodage. Voir CWE-116 pour plus d'atténuation liées au codage.
	Avec Struts écrivez toutes les données des écrans beans avec le filtre bean positionné à 'vrai'.
	Pour atténuer les attaques XSS contre le cookie de session utilisateur, positionnez-le à HttpOnly. Dans les navigateurs qui gèrent HttpOnly (comme les dernières versions de Firefox et Internet Explorer), cet attribut peut permettre de protéger le cookie contre un accès nuisible d'un script coté client qui utiliserait le cookie document. Ca n'est pas une solution complète puisque tous les navigateurs ne gèrent pas HttpOnly. De plus les technologies XMLHttpRequest et autres technologies puissantes fournissent un accès à l'en-tête HTTP, y compris l'en-tête Set-Cookie dans lequel le drapeau HttpOnly est positionné.
	Le codage d'entité HTML peut être utilisé sur tous les caractères non alphanumériques dans des données reçues de l'utilisateur et écrites dans la

	requête. Toutefois cela ne fonctionnera que dans la partie de votre page écrite en HTML normal – et il y a beaucoup d'autres parties dans lesquelles le codage d'entité ne fonctionnera pas, comme les URL, les attributs d'éléments, CSS etc..
	Même si un codage des sorties convenable est la meilleure défense contre le XSS, la validation des entrées peut fournir une défense en profondeur. Vérifiez et filtrez soigneusement chaque entrée en utilisant une liste blanche définissant les caractères et formats autorisés. Toutes les entrées doivent être vérifiées et nettoyées, et pas uniquement les paramètres que l'utilisateur est censé envoyer, mais toutes les données de la requête y compris les champs masqués, les cookies, en-têtes, l'URL etc.. Une erreur fréquente qui entraîne la vulnérabilité XSS, est de ne vérifier que les champs qui doivent être affichés sur le site. Il est courant de voir des données inattendues et non prévues par l'équipe de développement en provenance de la requête, affiché par le serveur d'application ou l'application. Un champ qui n'est pas affiché actuellement pourra être utilisé par un développeur dans l'avenir. Donc il est obligatoire de valider TOUTES les parties de la requête HTTP. La validation des entrées est réellement une mesure de défense en profondeur, car elle restreint le contenu des données avant traitement, ce qui impacte les sorties. Malgré tout, c'est souvent insuffisant. Si vous développez une application riche en données, beaucoup d'entrées auront une liste non limitative de caractères en entrée, qui passeront vos contrôles, mais qui seront utilisables pour XSS. De plus, même si vous faites une erreur de validation (comme oublier l'un des 100 champs), un codage approprié devrait vous protéger d'attaques basées sur l'injection. Si vous ne l'utilisez pas de manière isolée, la validation des entrées reste une technique utile en diminuant votre surface exposée aux attaques, en plus des autres avantages non gérés par un codage correct. Vous devez vérifier les entrées dans des interfaces bien spécifiées de votre application. Cela la protégera, même si un composant est réutilisé ou déplacé.
Opération	Utilisez un pare-feu applicatif. Cela n'est pas une garantie de sécurité totale, mais détectera beaucoup d'attaque fréquente.
Implémentation	Utilisez des outils d'analyse automatiques statiques et dynamiques qui ciblent ce type de vulnérabilité. Les techniques modernes utilisent des analyses de flux pour minimiser le nombre de faux positifs. Ca n'est pas parfait puisque la précision 100% est impossible.
Tests	Utilisez la liste de vérification des failles XSS (voir référence) pour lancer une série d'attaques XSS contre votre application Web. La liste contient de nombreuses et subtiles variations XSS ciblés contre des défenses XSS faibles.

CWE liées

CWE-692 : Liste noire incomplète de script intersites

CWE-82 : Echec de nettoyage des Scripts dans les attributs d'étiquette IMG dans les pages web

CWE-85 : Doublement de caractères dans les manipulations XSS

CWE-87 : Echec du nettoyage de syntaxe XSS alternative

Type d'attaques

CAPEC-ID:

CWE-78: Incapacité à préserver une structure de commande de l'OS (ou 'injection de commande système d'exploitation')

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Exécution de code
Coût de réparation	Moyenne	Facilité de détection	Facile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Votre logiciel est souvent le pont entre un étranger sur le réseau et l'intérieur de votre système d'exploitation. Quand vous appelez un autre programme du système d'exploitation, mais que vous ne sécurisez pas la chaîne de caractères permettant cet appel, alors vous invitez les attaquants à franchir ce pont pour exécuter leurs propres commandes au lieu des vôtres.

Prévention et atténuation des conséquences

Architecture et conception	Si c'est possible, utilisez des bibliothèques en remplacement de processus externes pour recréer la fonctionnalité souhaitée.
	Exécuter la commande dans une « prison » ou un environnement similaire qui gère des frontières protégées entre le processus et le système d'exploitation. Par exemple Unix « chroot jail » et AppArmor. Cela n'est peut être pas possible, et certaines technologies ne protègent pas contre une commande nuisible qui endommagerait les ressources accessibles à l'application.
	Essayez de garder en dehors d'un contrôle externe, autant que faire se peut, toutes les données utilisées dans la génération de la commande. Par exemple dans une application web, cela pourrait consister à garder la commande localement dans l'état de session, plutôt que de l'envoyer au client dans un champ caché.
	Utilisez un Framework de codage des sorties qui gère les commandes OS, comme le contrôle de codage ESAPI.
Implémentation	Mettez entre guillemets et escape tous les caractères spéciaux dans les arguments. Si des caractères spéciaux sont nécessaires, mettez les entre guillemets et escape tous les autres caractères qui ne sont pas en liste blanche. Attention à l'injection d'arguments (CWE-88).
	Si le programme à exécuté autorise le transfert d'arguments via un fichier ou ligne d'entrée standard, voyez si vous pouvez utiliser ces techniques à la place de la ligne de commande.
	Certains langages offrent plusieurs fonctions utilisables pour l'exécution de commandes. Quand c'est possible, identifiez les fonctions qui appellent un environnement d'exécution en utilisant une seule chaîne de caractères, et remplacez les par une fonction qui nécessite des arguments individuels. En général ces dernières vérifient les paramètres. En C la fonction system() accepte une chaîne contenant l'intégralité de la commande à exécuter, alors

	que execl(), execve() et autres nécessite un tableau de chaînes, un pour chaque paramètre. Dans Windows, CreateProcess() accepte uniquement une commande à la fois, Si on fournit à la fonction system() de Perl un tableau d'arguments, elle les vérifiera tous.
	Utilisez une liste blanche stricte pour chaque paramètre, afin d'éliminer les syntaxes non appropriées des entrées. Cela limitera indirectement l'ampleur des attaques, mais cette technique est moins importante que le codage approprié des sorties et la délimitation des caractères potentiellement dangereux.
Opération	Exécutez le code dans un environnement qui vérifie automatiquement les propagations corrompues et évite l'exécution de toute commande utilisant des variables corrompues, comme le paramètre -T de Perl. Cela vous obligera à mettre en place une procédure de validation éliminant la corruption, sachant que vous devez valider vos entrées correctement afin de ne pas autoriser accidentellement des entrées corrompues. (vois CWE-183 et CWE-184).
	Appliquez une politique de vérification à l'exécution pour créer une liste blanche des commandes autorisées, et prohibez toute commande non présente dans cette liste. Des technologies telles AppArmor le permettent.
Configuration système	Etablir des autorisations système qui interdisent à l'utilisateur l'ouverture ou l'accès aux fichiers sensibles. Exécutez l'application avec le minimum de privilèges possibles. (CWE-250).

CWE liées

CWE-88: injection de paramètres

Type d'attaques

CAPEC-ID:

CWE-319: Transmission en clair d'information sensible

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Perte de données
Coût de réparation	Moyenne	Facilité de détection	Facile
Fréquence des attaques	Quelque fois	Sensibilisation attaquant	Elevée

Présentation

Si votre logiciel envoie des données privées ou d'authentification sur le réseau, cette information traversera plusieurs nœuds avant d'atteindre sa destination finale. Les attaquants peuvent renifler ces données et cela ne demande aucun effort particulier. Ils doivent juste contrôler l'un des nœuds de transit, un nœud du réseau du nœud de transit, ou se connecter à une interface disponible. Essayez de camoufler le trafic en utilisant des schémas basés sur Base64 et le codage d'URL n'offre aucune protection; ces codages sont utilisés pour les communications normalisées, par pour l'encryptage.

Prévention et atténuation des conséquences

Architecture et conception	Les informations secrètes ne doivent jamais être envoyées en texte clair. Utilisez un système fiable d'encryption des données avant envoi.
Implémentation	Pour une application Web utilisant SSL, utilisez SSL pour toute la session, du login au logout, et non pas juste pour la page de login initiale.
Operation	Configurez les serveurs pour l'utilisation de canaux cryptés pour les communications, ce qui peut inclure SSL ou tout autre protocole.

CWE liées

CWE-312: stockage en clair d'information sensible

CWE-614: Cookie sensible dans les sessions HTTPS sans l'attribut « sécurisation »

Type d'attaques

CAPEC-ID:

CWE-352: Requête frauduleuse intersites (CSRF)

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Perte de données Exécution de code
Coût de réparation	Elevée	Facilité de détection	Modérée
Fréquence des attaques	Souvent	Sensibilisation attaquant	Moyenne

Présentation

Vous n'accepteriez pas un coli d'un étranger dans un aéroport. Il pourrait contenir un contenu dangereux. De plus, si quelque chose ne se passe pas bien, vous apparaîtriez comme le coupable, car c'est vous qui portez le coli à l'embarquement. Une requête frauduleuse inter-site est comme ce coli, sauf que l'attaquant dupe un utilisateur pour qu'il active une requête allant sur votre site. Grâce aux scripts et au fonctionnement du web en général, l'utilisateur ne se rendra même pas compte de l'envoi de la requête. Mais à l'arrivée sur votre site, la requête proviendra apparemment de l'utilisateur et non de l'attaquant. Cela peut paraître innocent, mais l'attaquant s'est déguisé en utilisateur légitime et a obtenu les mêmes droits que l'utilisateur. Cela est bien pratique, surtout quand l'utilisateur a les droits d'un administrateur, ce qui entraînera la compromission complète de toutes les fonctionnalités de votre application. Combiné au XSS, les résultats peuvent être dévastateurs. Si vous avez entendu parler des vers XSS qui se ruent à travers les grands sites web en quelques minutes, ils sont souvent d'origine CSRF.

Prévention et atténuation des conséquences

Architecture et conception	Utiliser des solutions anti-CSRF comme OWASP CSRFGuard
Implémentation	Vérifiez que votre application ne contient aucun problème lié aux scripts intersites (CWE-79), car la plupart des défenses CSRF peuvent être court-circuitées en utilisant un script contrôlé par l'attaquant.
Architecture et conception	Générer un nonce unique pour chaque écran et vérifier le nonce. Vérifiez que le nonce n'est pas prévisible (CWE-330).
	Vérifiez chaque opération dangereuse

	Utilisez le double envoi des cookies comme décrit par Felten et Zeller
	Utilisez le contrôle de gestion de session ESAPI, qui incluse un composant CSRF.
	Ne pas utiliser la méthode GET pour les requêtes provoquant un changement d'état
Implémentation	Vérifiez l'en-tête HTTP Référent pour voir si la requête vient bien de la page prévue. Cela n'est pas très efficace par l'attaquant pourrait forcer le client à utiliser un Référent falsifié. Cela pourrait également casser une fonctionnalité légitime, car les utilisateurs ou proxies peuvent avoir débranché le Référent pour des questions de vie privée.
Test	Utilisez OWASP CSRFTester pour identifier les problèmes potentiels.

CWE liées

CWE-346: Erreur de validation origine

CWE-441: Proxy/Intermédiaire imprévu

Type d'attaques

CAPEC-ID:

CWE-362: Condition de course

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Interdiction de service Exécution de code Perte de données
Coût de réparation	Moyenne à élevée	Facilité de détection	Modérée
Fréquence des attaques	Quelque fois	Sensibilisation attaquant	Elevée

Présentation

Les accidents de la circulation se produisent lorsque des véhicules veulent utiliser la même ressource en même temps, par exemple la même partie de la route. Les conditions de course dans votre logiciel ne sont pas différentes, sauf que l'attaquant cherchera à les exploiter pour créer le chaos ou obliger votre application à produire quelque chose ayant de la valeur. Dans beaucoup de cas une condition de course impliquera des processus multiples ou l'attaquant aura le contrôle total de l'un des processus. Même si les conditions de course se produisent entre plusieurs sous-processus, l'attaquant peut être à même d'influer sur l'exécution de certains de ces sous-processus. Votre seul confort avec les conditions de courses est que la corruption des données et l'interruption de service est la norme. Aucune technique fiable pour l'exécution de code n'a – encore – été développée. Du moins pour certaines conditions de course. Pas très confortable... L'impact peut être local ou global, dépendant de ce qu'affectent les conditions de course: comme des variables d'état ou la logique de sécurité; et si elles se produisent à l'intérieur de plusieurs systèmes, processus ou sous-processus.

Prévention et atténuation des conséquences

Architecture et conception	Dans les langages qui le gère, utilisez les primitives de synchronisation. Simplement enveloppez-les autour du code critique pour diminuer les
----------------------------	--

	impacts sur les performances.
	Utilisez des capacités de sécurisation des sous-processus telle que l'abstraction des accès données dans Spring
	Si possible évitez d'utiliser des ressources partagées par plusieurs sous-processus.
Implémentation	Quand vous utilisez des sous-processus, utilisez uniquement des fonctions sécurisées sur les variables partagées
	Utilisez des fonctions atomiques sur les variables partagées. Evitez les constructions apparemment innocentes telles « x++ ». C'est en fait non-atomique, puisque cela utilise une lecture suivie d'une écriture.
	Si disponible utilisez un mutex, mais attention aux vulnérabilités telle CWE-412.
	Eviter les verrous à vérification double (CWE-609) et les autres erreurs d'implémentation qui se produisent quand on essaye d'éviter les surcoûts de synchronisation.
	Désactivez les interruptions ou signaux dans les parties critiques de votre application, mais vérifiez également les boucles infinies ou longues.
	Utilisez le modificateur de type volatile pour les variables critiques afin d'éviter les optimisations ou réarrangement imprévus du compilateur. Cela ne résoudra peut-être pas vos soucis de synchronisation, mais ça peut aider.
Tests	Tester le stress du code en l'appelant simultanément de plusieurs sous-processus ou processus, et chercher les indices de comportement imprévu. Mettre des points d'arrêt ou des délais entre les parties du code concerné afin d'élargir artificiellement la fenêtre de course pour en faciliter la détection.

CWE liées

CWE-364 : gestion de signal de condition de course

CWE-366 : Condition de course dans un sous-processus

CWE-367 : condition de course heure de vérification, heure d'utilisation (TOCTOU)

CWE-370 : condition de course lors de la vérification de la révocation de certificat

CWE-421 : condition de course lors de l'accès à un canal alternatif

Type d'attaques

CAPEC-ID:

CWE-209: Fuite d'information de message d'erreur

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Perte de données
Coût de réparation	Faible	Facilité de détection	Facile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Si vous utilisez des messages d'erreur diserts, vous pouvez alors fournir des informations secrètes à tout attaquant qui souhaiterait mal utiliser votre logiciel. Ces secrets peuvent couvrir une grande plage de données de valeur, comme des informations d'identification personnelles (PII), d'authentification et de configuration serveur. Quelques fois elles ressemblent à des informations sans dangers qui sont utiles à vos utilisateurs ou administrateurs, comme le chemin complet d'installation du logiciel. Même ces petits secrets peuvent simplifier grandement une attaque concertée qui génèrera de plus grandes récompenses, ce qui arrive continuellement dans le monde réel. Cela est une préoccupation, que vous envoyez des messages temporaires d'erreur à vos utilisateurs ou que vous les enregistriez dans un fichier.

Prévention et atténuation des conséquences

Implémentation	Assurez-vous que les messages d'erreur contiennent le minimum d'informations nécessaire au destinataire et personne d'autre. Les messages doivent être équilibrés entre être trop compréhensibles et incompréhensibles. Ils ne doivent pas forcément révéler la méthode utilisée pour détecter l'erreur. De telles informations détaillées peuvent aider l'attaquant à construire une autre attaque afin de court-circuiter les filtres de validation. Si les erreurs doivent être enregistrées, mettez les dans des fichiers journaux – mais examinez ce qui pourrait se produire si l'attaquant avait accès à ces journaux. Ne pas enregistrer des informations sensibles comme les mots de passe. Eviter les messages inconsistants qui pourrait informer un attaquant de l'état interne du système, comme la validité éventuelle d'un nom d'utilisateur.
Implémentation	Gérez les exceptions en interne, et ne pas afficher d'erreurs contenant des informations sensibles à un utilisateur.
Compilation et Edition de liens	Les informations de mise au point ne doivent pas être intégrées dans la version de production.
Configuration Système	Si c'est possible, configurer l'environnement pour utiliser des messages d'erreur moins diserts. Par exemple en PHP, désactiver <code>display_errors</code> pendant la configuration ou à l'exécution en utilisant la fonction <code>error_reporting()</code> .
	Créer des pages ou messages d'erreur qui ne révèlent aucune information sensible.

CWE liées

CWE-204 : différence de réponse dans une fuite d'information

CWE-210 : fuite d'information dans un message d'erreur généré

CWE-538 : fuite d'information concernant un fichier ou chemin d'accès

Type d'attaques

CAPEC-ID:

Gestion des Ressources à Risque

CWE-119: Incapacité à limiter des opérations à l'intérieur d'un tampon mémoire

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Exécution de code Interdiction de service Perte de données
Coût de réparation	Bas	Facilité de détection	Facile à modérée
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Le débordement de tampon est le rappel de Mère nature de la loi physique suivante: si vous essayez de mettre plus de chose dans un contenant qu'il ne peut garder, vous allez créer un problème. Fléau des applications C durant des décennies, le débordement de tampon est resté remarquablement résistant à l'élimination. Une raison est qui ne provient pas seulement de la mauvaise utilisation de `strcpy()`, ou de la mauvaise vérification de la longueur de vos entrées. Les attaques et les techniques de détection s'améliorent et les variantes actuelles du débordement de tampon ne sont pas apparente à première ou même à seconde vue. Vous pouvez penser que vous êtes invulnérables aux débordements de tampon car vous utilisez un langage de plus haut niveau que C. Mais en quoi votre interpréteur « sur » est-il écrit ? Et les logiciels qui gèrent l'infrastructure de l'internet ? Et oui !

Prévention et atténuation des conséquences

Conditions de base	Utiliser un langage insensible aux débordements de tampon
Architecture et conception	Utilisez une bibliothèque d'abstraction pour éloigner les API à risque. Par exemple Safe C String Library (SafeStr) de Messier et Viega, et Strsafe.h de Microsoft. Ces bibliothèques fournissent des versions plus sûres de la gestion des chaînes de caractères qui sont sujettes au débordement de tampon. Ce n'est pas une solution complète, car beaucoup de débordement de tampons n'ont aucun lien avec les chaînes de caractères.
Compilation et édition de liens	Utilisez les mécanismes automatiques de débordement de tampons qui sont proposés par plusieurs compilateurs ou extensions. Par exemple utilisez le drapeau /GS de Visual Studio de Microsoft, FORTIFY_SOURCE GCC de Fedore/Red Hat, StackGuard et ProPolice. Cela ne représente pas une solution complète car ces mécanismes ne détectent que certains débordements. De plus une attaque par débordement de tampon peut toujours causer une interdiction de services, puisque le comportement logique de l'application dans ce cas est de se terminer.
Implémentation	Les programmeurs doivent se soumettre aux règles suivantes en matière de gestion mémoire: faites une double vérification de la taille de votre tampon. Si vous utilisez une fonction qui copie une suite d'octet, comme <code>strcpy()</code> , vérifiez que si la taille du tampon destinataire est bien la même que le tampon d'origine, il pourrait ne pas terminer la chaîne par NULL. Vérifiez les frontières du tampon si vous appelez la fonction dans une boucle et soyez certains qu'il n'existe aucun danger d'écrire au delà de l'espace alloué. Si nécessaire, coupez toutes les chaînes d'entrée à une taille acceptable avant de les passer aux fonctions de copie et concaténation.
Opération	Utilisez un dispositif comme Espace Aléatoire d'Adressage (ASLR). Ca

	n'est pas une solution complète. Malgré tout cela oblige l'attaquant à deviner une valeur inconnue qui change à chaque exécution.
Opération	Utiliser un CPU et Système d'Exploitation qui offre une protection des données à l'exécution (NX) ou son équivalent. Ca n'est pas une solution complète puisque le débordement de tampon peut être utilisé pour surcharger des variables proches pour modifier l'état du logiciel d'une manière dangereuse. De plus elle ne peut être utilisée dans le cas de code auto-modifié.

CWE liées

CWE-120: Débordement de tampon classique

CWE-129: Index de tableau non vérifié

CWE-130: Echec de la gestion des inconsistances de longueur de paramètres

CWE-131: Erreur de calcul de taille de tampon

CWE-415: Double libre

CWE-416: utilisation après libre

Type d'attaques

CAPEC-ID:

CWE-642: Contrôle externe de données critiques d'état

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Brèche de Sécurité Perte de données Exécution de code
Coût de réparation	Moyenne	Facilité de détection	Facile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Il existe plusieurs méthodes de stockage de données évitant le surcout d'une base de données. Malheureusement si vous stocker les données à un endroit où l'attaquant peut les modifier, cela réduit aussi le surcout d'un compromis à succès. Par exemple les données peuvent être stockées dans des fichiers de configuration, profils, cookies, clés de registre, champs cachés, variables d'environnement ou d'autres endroits ou elles peuvent être modifiées par l'attaquant. Dans des protocoles sans états comme HTTP, certain états utilisateurs doivent être enregistrés à chaque requête, et donc sont exposés sans nécessités à un attaquant. Si vous exécutez des opérations de sécurité critiques à partir de ces données (comme indiquant que l'utilisateur est un administrateur), vous pouvez parier que quelqu'un modifiera les données pour duper votre application et lui faire faire quelque chose que vous n'aviez pas prévu.

Prévention et atténuation des conséquences

Architecture et conception	Comprenez tous les emplacements qui sont accessibles aux attaquants. Par exemple un programmeur pourra assumer que les cookies et champs cachés
----------------------------	---

	ne peuvent pas être modifiés par l'utilisateur, ou peut oublier qu'une variable d'environnement peut être modifiées avant l'exécution d'un programme privilégié.
	Ne pas garder d'information d'état sur le client sans utiliser de cryptage et de vérification d'intégrité, ou alors implémenter un mécanisme sur le serveur pour détecter la falsification d'état. Utilisez un algorithme de code d'authentification de message (MAC) tel que Hash Message Authentication Code (HMAC). Appliquez-le à la variable d'état que vous devez exposer, ce qui pourra garantir l'intégrité des données. c.à.d. Que les données n'ont pas été modifiées. Vérifiez que votre algorithme utilise une fonction de hachage forte (CWE-328).
	Stocker les informations d'état uniquement coté serveur. Validez que le système suit son propre état et celui de ses utilisateurs et qu'il a des règles pour les changements d'états légaux. N'autorisez aucune application à modifier un état directement, sans passer par des actions légales entraînant ces changements d'état.
	Avec un protocole sans état comme HTTP, utilisez un Framework qui maintient l'état pour vous. Par exemple View State d'ASP.NET et la gestion de session OWASP ESAPI. Faites attention au dispositif du langage qui gère l'état, car il peut être considéré comme une aide à la programmation et peut ne pas être sur.
Implémentation	Si vous utilisez PHP, configurer votre application pour qu'elle n'utilise pas register_globals. Pendant l'implémentation, développez votre application pour qu'elle n'utilise pas cette fonctionnalité, mais faites attention si vous en implémenter une émulation sujette à la faiblesse CWE-95, CWE-621 ou similaires.

CWE liées

CWE-472: contrôle externe de paramètres web considérés comme inamovibles

CWE-565: utilisation des cookies dans des décisions concernant la sécurité

Type d'attaques

CAPEC-ID:

CWE-73: Contrôle externe de nom de fichier ou chemin d'accès

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Exécution de code perte de données
Coût de réparation	moyenne	Facilité de détection	Facile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevée

Présentation

Alors que l'échange données utilise souvent des fichiers, vous ne souhaitez pas montrer tous les fichiers de votre système ce faisant. Quand vous utilisez une entrée externe pour construire un nom de fichier, le chemin résultant pourrait pointer en dehors du répertoire souhaité. Un attaquant

pourrait combiner plusieurs « .. » ou des séquences similaires pour entrainer le système d'exploitation à naviguer en dehors des chemins spécifiés. D'autres attaques sont simplifiées par le contrôle externe d'un nom de fichier, comme le suivi d'un lien symbolique, qui entraine votre application à lire ou modifier un fichier auquel l'attaquant n'a pas accès directement. La même chose se produit si votre programme utilise des autorisations élevées et accepte des noms de fichier en entrée. Et si vous autorisez quelqu'un de l'extérieur à spécifier une URL arbitraire et à télécharger du code et à l'exécuter, vous invitez juste des vers.

Prévention et atténuation des conséquences

Architecture et conception	Quand l'ensemble des noms de fichier est limité ou connu, faites une correspondance d'en ensemble de valeurs d'entrée fixes (comme un ID numérique) vers les noms de fichier, et rejetez les autres. Par exemple ID 1 pourrait correspondre à « inbox.txt » et ID 2 à « profile.txt ». AccessReferenceMap de ESAPI fournit cette fonctionnalité.
Installation	Utilisez les niveaux de sécurité du système d'exploitation et fonctionnez à privilèges minimums pour limiter la portée de l'attaque.
Architecture et conception	Créer une prison chroot() ou équivalent qui restreint l'accès à un certain répertoire. Attention aux vulnérabilités des prisons telle que CWE-243
Implémentation	Utiliser des listes blanches strictes limitant les caractères utilisables dans les noms de fichier. Si possible, n'autoriser qu'un « . » par nom de fichier. Utiliser des listes noires pour rejeter les noms incorrects.
	Utilisez une fonction intégrée de canonisation (comme realpath() en C) qui produira la version canonique du chemin, ce qui enlèvera les séquences « .. » et les liens symboliques

CWE liées

CWE-22: chemin transverse

CWE-434: chargement de fichier non restreint

CWE-59: Suivi de lien

CWE-98: Inclusion de fichier distant

Type d'attaques

CAPEC-ID:

CWE-426: Chemin de recherche non sur

Résumé

Prévalence de la vulnérabilité	Faible	Conséquences	Exécution de code
Coût de réparation	Moyen	Facilité de détection	Facile
Fréquence des attaques	Rarement	Sensibilisation attaquant	Elevée

Présentation

Votre logiciel dépend de vous ou de votre environnement pour lui fournir un chemin de recherche, pour qu'il puisse retrouver des ressources critiques, tels des bibliothèques de code ou des fichiers de

configuration. Si le chemin de recherche est sous le contrôle de l'attaquant, alors celui-ci peut le modifier pour pointer vers des ressources à sa convenance. Cela entraînera le logiciel à accéder à la mauvaise ressource au mauvais moment. Le même risque existe si un seul des éléments du chemin de recherche est sous le contrôle de l'attaquant, comme par exemple le répertoire de travail courant.

Prévention et atténuation des conséquences

Architecture et conception	Coder le chemin de recherche en dur avec des valeurs sûres, ou permettez à un administrateur de les spécifier dans un fichier de configuration. Ne permettez pas leurs modifications par un tiers. Evitez les vulnérabilités CWE-427 et CWE-428.
Implémentation	Si vous appelez d'autres programmes, spécifiez leur nom en utilisant des chemins complètement remplis.
	Nettoyez votre environnement avant d'appeler d'autres programmes. Cela inclut la variable d'environnement PATH, LD_LIBRARY_PATH et d'autres réglages qui identifient la disposition des bibliothèques de code et tous les chemins de recherche de votre application.
	Vérifier vos chemins de recherche avant utilisation et retirez en tout élément non sûr, comme le répertoire de travail actuel ou un répertoire de fichiers temporaires.
	Utilisez d'autres fonctions nécessitant un chemin explicite. C'est un moyen sûr d'éviter ce problème. Par exemple system() en C ne nécessite pas de chemin explicite, puisque le shell peut le gérer, alors que execl() et execv() demandent un chemin explicite.

CWE liées

CWE-427: Elément de chemin de recherche non contrôlé

CWE-428: Elément ou chemin de recherche sans guillemets

Type d'attaques

CAPEC-ID:

CWE-94: Echec du contrôle de la génération de code (injection de code)

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Exécution de code
Coût de réparation	Elevé	Facilité de détection	Modérée
Fréquence des attaques	Quelques fois	Sensibilisation attaquant	Moyenne

Présentation

Pour faciliter les développements, vous ne pouvez vous empêcher quelques fois d'utiliser un couple de lignes de code pour y mettre beaucoup de fonctionnalités. C'est même mieux si vous gérez le code dynamiquement. Bien sur il est difficile de battre le côté sexy du code généré dynamiquement, mais vos attaquants peuvent également le trouver intéressant. Cela devient une vulnérabilité majeure si votre code peut être appelé par des tiers non autorisés. Si des entrées externes peuvent déterminer quel code doit être exécuté, ou (horreur) si ces entrées sont directement injectées dans le

code lui-même. Les implications sont évidentes: votre code leur appartient.

Prévention et atténuation des conséquences

Architecture et conception	Refaites votre code pour ne pas en générer dynamiquement.
	Exécutez votre code dans une prison ou un environnement similaire qui maintiendra une frontière sûre entre le processus et le système d'exploitation. Cela peut être impossible, et limite simplement l'impact au système d'exploitation; le reste de votre application pourrait être compromise.
Implémentation	Utilisez un mélange approprié de listes noires et blanches pour filtrer la syntaxe inappropriée de toute entrée qui ne devrait pas contenir de code.
Opération	Exécuter le code dans un environnement qui empêche automatiquement la propagation des corruptions, et évite l'exécution de commandes comprenant des variables corrompues, comme l'option -T de Perl. Cela vous obligera à implémenter des étapes de validation pour enlever la corruption, mais soyez prudent lors de la validation des entrées pour marquer certaines comme correctes alors qu'elles ne le sont pas. (CWE-183 et 184)

CWE liées

CWE-470: réflexion non sûre

CWE-95: Injection d'évaluation\$

CWE-96: Injection de code statique

CWE-98: Inclusion de fichier distant

Type d'attaques

CAPEC-ID:

CWE-494: Téléchargement de code sans vérification d'intégrité

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Exécution de code
Coût de réparation	Moyen à élevé	Facilité de détection	Modérée
Fréquence des attaques	Rarement	Sensibilisation attaquant	Basse

Présentation

Vous ne devez pas être un voyant pour comprendre que si vous téléchargez du code et l'exécutez, vous faites confiance à la source. Vous pourriez peut être ne télécharger que du code en provenance d'une source sûre, mais des attaquant peuvent utiliser toute sorte de trucs pour modifier le code avant qu'il ne vous atteigne. Ils peuvent modifier le site d'origine, le copier en modifiant le DNS ou en empoisonnant un tampon, convaincre le système de se dérouter vers un autre site, ou même modifier le code lors de son transit sur le réseau. Ce scénario peut même s'appliquer au cas où votre logiciel télécharge et installe ses propres mises à jour. Quand cela arrive, votre logiciel fini par exécuter un code inattendu, ce qui est mauvais pour vous mais bon pour les attaquants.

Prévention et atténuation des conséquences

Implémentation	Faites des vérifications DNS avant et arrière pour détecter la personification de DNS. Ce n'est qu'une solution partielle, car cela n'empêche pas la modification à la volée à travers le réseau ou sur le site d'origine.
	Utilisez des canaux cryptés qui vérifie l'intégrité pour transférer le code du serveur hôte. Cela ne détectera pas personification DNS ou la modification du code sur le site hôte.
Architecture et conception	Si vous fournissez le code qui doit être téléchargé, comme des mises à jour automatique s de votre logiciel, utilisez des signatures cryptographiques pour votre code et modifier le client de téléchargement pour qu'il vérifie ces signatures. Vérifiez que votre code ne contient pas les vulnérabilités CWE-295, 320, 347 et celles liées.
	Utiliser une technologie de signature de code comme Autheticode. Voir références.

CWE liées

CWE-247: Confiance dans les recherches DNS dans les décisions de sécurité

CWE-292: Faire confiance un nom DNS auto-reporté

CWE-346: Erreur de validation d'origine

CWE-350: confiance mal accordée au DNS inverse

Type d'attaques

CAPEC-ID:

CWE-404: Fermeture ou libération de ressources incorrectes

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Interdiction de service Exécution de code
Coût de réparation	Moyen	Facilité de détection	Facile à modérée
Fréquence des attaques	Rarement	Sensibilisation attaquant	Basse

Présentation

Quand vis précieuses ressources systèmes sont arrivées en fin de vie, vous devez en disposer correctement. Sinon votre environnement deviendra sérieusement congestionné ou contaminé. Cela s'applique à la mémoire, aux cookies, structures de données, sessions, canaux de communication, etc.. Les attaquants peuvent exploiter des fermetures incorrectes, pour maintenir leur contrôle sur ces ressources bien après que vous pensiez vous en être débarrassé. Cela entrainera une consommation de ressources excessive, car en fait rien ne sera rendu au système. Si vous ne nettoyez pas vos poubelles avant de les éliminer, vos attaquants vont les examiner, pour rechercher des trésors comme des données confidentielles non effacées. Ils pourraient également réutiliser vos ressources, ce qui peut paraître bien dans un monde « vert », sauf que, dans le monde virtuel, ces ressources peuvent avoir encore une valeur substantielle.

Prévention et atténuation des conséquences

Exigences	Utilisez des technologies qui nettoient automatiquement les poubelles
Implémentation	C'est un bon usage de libérer toutes les ressources utilisées et d'être cohérent quant à la manière de libérer la mémoire dans une fonction. Si vous allouez de la mémoire que vous avez l'intention de rendre à la fin d'une fonction, vous devez vérifier cette libération à tous les points de sortie, y compris dans la gestion des erreurs.
	La mémoire doit être allouée et relâchée en utilisant des fonctions appariées comme malloc/free, new/delete et new[]/delete[]
	En relâchant un objet ou une structure complexe, vérifiez que vous avez bien relâché tous les éléments et non juste la structure elle-même

CWE liées

CWE-14 : suppression de code d'effacement de tampon par le compilateur

CWE-226 : information sensible non effacée avant libération

CWE-262 : non-utilisation de mot de passe à expiration

CWE-299: Echec de vérification de révocation de certificat

CWE-401: Fuite de mémoire

CWE-415: Libre double

CWE-416: Utilisation après libre

CWE-568: Méthode finalize() sans super.finalize()

CWE-590: libération de pointeurs invalides pas dans la pile

Type d'attaques

CAPEC-ID:

CWE-665: Mauvaise Initialisation

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Exécution de code Perte de données
Coût de réparation	Bas	Facilité de détection	Facile
Fréquence des attaques	Quelque fois	Sensibilisation attaquant	Basse

Présentation

Comme vous devez démarrer la journée avec un bon petit-déjeuner, une initialisation correcte permettra à votre logiciel de fonctionner sans s'effondrer au milieu d'un événement important. Si vous n'initialisez pas correctement vos données et vos variables, un attaquant le fera pour vous, ou extraira des informations provenant d'une session précédente. Quand ces variables ont utilisées dans des procédures de sécurité, comme une décision d'autorisation, elles pourraient alors être modifiées pour court-circuiter votre sécurité. Une initialisation correcte peut arriver n'importe où, mais en général le plus souvent dans des conditions rares entraînant votre code à ne pas faire la procédure d'initialisation sans vous en rendre compte.

Prévention et atténuation des conséquences

Exigences	Utilisez un langage obligeant le programmeur à initialiser ses variables avant toute utilisation
Architecture et conception	Identifiez les variables et magasins de données qui reçoivent leur informations à partir d'une source externe, et validez les entrées afin de vous assurer de la pertinence des données.
Implémentation	Initialisez explicitement vos variables et magasins de données soit à la déclaration, soit à la première utilisation.
	Faites très attention aux conditions complexes qui affectent l'initialisation, car certaines conditions pourraient ne pas faire l'initialisation.
	Evitez les conditions de courses (CWE-362) pendant l'initialisation
Compilation et édition de liens	Si vous utilisez un langage qui ne nécessite pas d'initialisation comme C, Perl ou PHP, alors exécuter ou compiler votre programme avec des réglages générant des avertissements concernant les données non initialisées.
Tests	Utilisez le fuzzing (utilisation de données aléatoires) ou toute autre technique similaire qui pourra générer une grande variété d'erreurs pouvant être produites par de données non initialisées.
	Identifiez des erreurs rares et déclenchez-les. Vous pouvez déclencher une erreur non gérée ou un bug similaire, mais si l'erreur est gérée sans problème, il est alors possible qu'une initialisation soit incomplète ou inexistante.

CWE liées

CWE-453: Initialisation par défaut de variable non sécurisée

CWE-454: Initialisation externe de variable de confiance

CWE-456: Initialisation manquante

Type d'attaques

CAPEC-ID:

CWE-682: Calcul incorrect

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Interdiction de service Perte de données Exécution de code
Coût de réparation	Bas	Facilité de détection	Facile à difficile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Moyenne

Présentation

Les ordinateurs peuvent faire des calculs qui ne semblent avoir aucune logique mathématique. Par exemple, si vous multipliez deux grands nombres entiers, le résultat peut être un nombre plus petit suite à un débordement d'entier. Dans d'autre cas le programme peut ne pas pouvoir faire l'opération, comme une division par zéro. Quand les attaquants ont un certains contrôle des entrées

utilisés dans des calculs numériques, la vulnérabilité peut avoir des conséquences en matière de sécurité. Elle peut entraîner une mauvaise décision de sécurité. Elle peut vous faire allouer beaucoup plus de ressources que prévues – ou beaucoup moins, dans le cas où un débordement d'entier provoque un débordement de tampon du fait d'une allocation de mémoire insuffisante. Elle peut aussi violer une règle de gestion, comme un calcul produisant un prix négatif<; finalement, une interdiction de service est aussi possible comme un arrêt brutal provoqué par une division par zéro.

Prévention et atténuation des conséquences

Implémentation	Comprenez bien la représentation sous-jacente de votre langage de programmation, et comment elle interagit avec les calculs numériques. Faites attention aux différences de tailles d'octets, précision, distinction signé/non signé, troncature, conversion, calculs entre « non numériques » et comment votre langage gère les nombres trop grands ou trop petits pour sa représentation sous-jacente.
	Vérifier que les nombres en entrées sont bien dans la fourchette de valeurs prévue.
	Utilisez le type approprié pour l'action souhaitée. Par exemple en C/C++ n'utilisez les valeurs non signées que pour des nombres qui ne peuvent jamais être négatifs comme la hauteur, largeur ou une quantité.
	Utilisez un environnement de gestion des entiers sur comme SafeInt(C++) ou IntergerLib (C ou C++)
Tests	Vérifiez l'algorithme de calcul en profondeur en utilisant toute sorte de fourchettes de valeurs et de types en entrée.

CWE liées

CWE-131: Mauvais calcul de taille de tampon

CWE-135: Mauvais calcul de la longueur d'une chaîne multi-caractères

CWE-190: Débordement ou bouclage d'entier

CWE-193: Erreur d'une unité

CWE-369: Division par zéro

CWE-467: Utilisation de sizeof() sur un type pointeur

CWE-681: Mauvaise conversion entre types numériques

Type d'attaques

CAPEC-ID:

Défenses Poreuses

CWE-285: Contrôle d'accès incorrect (Autorisation)

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Brèche de Sécurité
Coût de réparation	Bas à moyen	Facilité de détection	Modérée

Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevé
------------------------	---------	------------------------------	-------

Présentation

Imaginez que vous organisiez une soirée pour quelques amis intimes et leurs invités. Vous les faites entrer au salon, et pendant que vous discutez avec un ami, l'un des invités vide votre réfrigérateur, regarde dans votre armoire à pharmacie et s'interroge sur le contenu de votre table de chevet. Les logiciels ont le même type de problème d'autorisation qui peut entraîner des conséquences les plus graves. Si vous ne vérifiez pas que vos utilisateurs ne font que ce qu'ils ont l'autorisation de faire, alors les attaquants profiteront de cette faiblesse et utiliseront des fonctionnalités prévues uniquement pour un groupe restreint d'utilisateurs.

Prévention et atténuation des conséquences

Architecture et conception	Réduisez la surface d'attaque en faisant correspondre avec précision les rôles avec les données et les fonctionnalités. Divisez votre application en zones anonyme, normale, privilégiée et administrative. Utilisez un contrôle d'accès basé sur les rôles (RBAC) pour garantir les rôles aux frontières appropriées. Notez que cette approche pourrait ne pas vous protéger contre les autorisations horizontales c.à.d. qu'un utilisateur pourra en attaquer un autre avec un rôle identique.
	Vérifiez que vous contrôlez les accès en fonction de vos règles de gestion. Ils peuvent être différents des contrôles d'accès que vous appliquez aux ressources gérant vos règles de gestion.
	Utilisez un cadre d'autorisation tel JAAS Authorization Framework et le contrôle d'accès OWASP ESAPI.
Configuration système	Pour les applications web, vérifiez bien que les mécanismes de contrôle d'accès sont exécutés correctement du côté serveur à chaque page. Les utilisateurs ne doivent pas pouvoir accéder à des informations qui leur sont interdites, simplement en appelant directement la page. L'une des manières de le réaliser est de s'assurer que les pages contenant des informations sensibles ne sont pas stockées en cache, et que toutes ces pages restreignent leur accès à l'utilisation d'un jeton de sécurité actif, associé à un utilisateur qui a les autorisations d'accès pertinentes. Utilisez les contrôles d'accès de votre système d'exploitation et environnement serveur, et définissez vos listes de contrôle d'accès en conséquence. Utilisez une politique « refus par défaut » en définissant vos listes d'accès.

CWE liées

CWE-425 : Requête directe ('Navigation forcée')

CWE-749 : Méthode exposées non sécurisées

Type d'attaques

CAPEC-ID:

CWE-327: Utilisation d'un algorithme cryptographique cassé ou risqué

Résumé

Prévalence de la vulnérabilité	Elevée	Conséquences	Perte de données Brèche de Sécurité
Coût de réparation	Moyen à élevé	Facilité de détection	Modérée
Fréquence des attaques	Rarement	Sensibilisation attaquant	Moyen

Présentation

Si vous gérez des données sensibles ou devez protéger un canal de communication, vous utilisez peut-être la cryptographie pour éviter les attaques. Vous pourriez être tentés de développer votre propre schéma cryptographique pour rendre les attaques plus difficiles. Cette méthode est la bienvenue pour les attaquants. La cryptographie est juste extrêmement complexe. Si des mathématiciens et des informaticiens brillants à travers le monde n'y arrivent pas (et ils arrivent toujours à casser leurs propres méthodes), alors vous non plus. Vous pourriez penser avoir inventé un algorithme que personne ne pourra déchiffrer, mais il est plus probable que vous ayez réinventé la roue qui se cassera juste avant le défilé.

Prévention et atténuation des conséquences

Architecture et conception	Ne développez pas vos propres algorithmes de cryptographie. Ils seront sans nul doute exposés à des attaques bien connues des cryptographes. Les techniques de décodage sont au point. Si votre algorithme peut être compromis et si les attaquants peuvent comprendre comment il fonctionne, alors il est vraiment faible.
	Utilisez un algorithme considéré comme particulièrement fort par les experts. Choisissez une implémentation testée et utilisée. Par exemple le gouvernement US demande une certification FIPS 140-2. Comme tous les mécanismes cryptographiques, le code source doit être disponible pour analyse. Vérifiez que vous n'utilisez pas une cryptographie obsolète. Certains vieux algorithmes dont on pensait qu'il faudrait des milliards d'années à casser, peuvent l'être en quelques jours voire heures. Par exemple MD4, MD5, SHA1, DES ou autres.
	Concevez votre logiciel pour pouvoir aisément remplacer un algorithme par un autre. Cela facilitera la mise à jour.
Implémentation	Gérez et protégez soigneusement les clés cryptographiques (CWE-320). Si les clés peuvent être devinées ou volées, alors la force de la cryptographie n'a plus aucun intérêt. Chaque fois que c'est possible, utiliser une implémentation standard. Cela vous fera gagner en temps de développement, et vous évitera beaucoup d'erreurs d'implémentation. Examinez les fonctionnalités cryptographiques ESAPI. Quand vous utilisez des techniques cryptographiques approuvées, vous devez les utiliser correctement. N'essayez pas de gagner du temps en sautant des étapes coûteuses en ressources (CWE-325). Ces étapes sont souvent indispensables à la prévention des attaques communes.

CWE liées

CWE-320 : Erreurs de gestion des clés

CWE-329 : Ne pas utiliser de IV aléatoire en mode CBC

CWE-331 : Entropie insuffisante

CWE-338 : Utilisation de PRNG faible cryptographiquement

Type d'attaques

CAPEC-ID:

CWE-259: Mot de passe codé en dur

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Brèche de Sécurité
Coût de réparation	Elevé	Facilité de détection	Moyenne
Fréquence des attaques	Rarement	Sensibilisation attaquant	Elevé

Présentation

Coder en dur un compte secret et un mot de passe dans votre application est extrêmement commode – pour des experts en décodage. Bien que cela puisse diminuer vos budgets de tests et support, cela peut réduire à néant la sécurité de vos clients. Si le mot de passe est le même pour tout votre logiciel, alors tous vos clients seront vulnérables si (ou plutôt quand) ce mot de passe sera identifié. Comme il est codé en dur c'est extrêmement difficile aux administrateurs systèmes de régler le problème. Et vous savez comme ils aiment ce type de problème quand leur réseau est attaqué à 2 heures du matin... à peu près autant que vous quand vous devrez faire face à une horde de clients enragés et des articles incendiaires quand votre petit secret sera éventé. L'origine de la plupart des erreurs de la Liste des 25 sont de bonnes fois ; pour celle-ci vos clients ne l'entendront pas ainsi.

Prévention et atténuation des conséquences

Architecture et conception	Pour les authentifications sortantes : stockez les mots de passe dans un fichier de configuration crypté ou une base de données protégée de tout accès indésirable, y compris les autres utilisateurs du système. Protégez correctement la clé (CWE-320). Si vous ne pouvez utiliser un fichier crypté, réduisez ses accès au minimum possible.
	Pour les authentifications entrantes : plutôt que de coder en dur un nom d'utilisateur et mot de passe en dur pour la première utilisation, utilisez un mode « premier login » qui oblige l'utilisateur à entrer un mot de passe « fort ».
	Limitez les entités utilisant l'accès au mot de passe en dur. Par exemple à partir de la console système uniquement.
	Pour les authentifications entrantes : utilisez des hash unidirectionnels forts pour vos mots de passe et stockez-les dans un fichier de configuration ou une base de données sécurisée. Ainsi le vol de cette information obligera l'attaquant à tenter de craquer les mots de passe. Quand vous recevez un mot de passe en entrée, prenez le hash de l'entrée et comparez le au hash stocké. Cela augmentera l'ampleur des moyens que l'attaquant devra

employer pour attaquer brutalement votre système, et limitera l'efficacité de la méthode d'attaque dite table arc-en-ciel.

Pour des connexions frontal vers serveur : trois solutions sont possibles, quoiqu'aucune ne soit complète. Première suggestion : utilisez les mots de passe générés qui sont changés automatiquement et doivent être entrés régulièrement par l'administrateur système.. Ces mots de passe seront stockés en mémoire et ne seront valides que pour l'intervalle de temps considéré. Deuxièmement : les mots de passes utilisés devront être limités sur les serveurs aux actions valides sur le frontal, et non pas avoir un accès illimité. Finalement les messages devront être marqués et encodés avec des valeurs liées à l'heure pour éviter les attaques basées sur la réexécution.

CWE liées

CWE-256 : stockage des mots de passe en texte simple

CWE-257 : Stockages des mots de passe en format récupérable

CWE-260 : Mots de passe dans des fichiers de configuration

Cwe6 »é&/ Utilisation de clé cryptographique codée en dur

Type d'attaques

CAPEC-ID:

CWE-732: Assignation de permission non-sécurisée pour des ressources critiques

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Perte de données Exécution de code
Coût de réparation	Bas à élevé	Facilité de détection	Facile
Fréquence des attaques	Souvent	Sensibilisation attaquant	Elevé

Présentation

Il est impoli de prendre quelque chose sans en demander la permission, mais des utilisateurs impolis (les attaquants) sont prêts à passer un peu de temps pour voir ce qu'ils peuvent récupérer. Si vous avez des programmes critiques, magasins de données, ou fichiers de configuration avec des permissions permettant au monde entier de les lire ou écrire, eh bien c'est exactement ce qui se passera. Alors que ce problème pourra ne pas être traité pendant la conception ou l'implémentation, c'est pourtant lors de ces phases qu'une solution doit être appliquée. En laisser la responsabilité à un administrateur système déjà surchargé est loin d'être optimal et souvent impossible.

Prévention et atténuation des conséquences

Architecture et conception	En utilisant une ressource critique telle un fichier de configuration, vérifiez si cette ressource a des permissions non sécurisées (comme une modification possible par un utilisateur lambda) et générez une erreur ou une sortie de l'application si la ressource peut être modifiée par un tiers non autorisé.
	Définissez des groupes d'utilisateurs, privilèges et rôles qui vous

	permettront de maintenir un niveau de granularité fin sur vos ressources.
Implémentation	Quand votre application démarre, définissez explicitement les permissions par défaut pour ne pas hériter de réglages non sécurisés de l'utilisateur ayant lancé votre application.
Configuration système	Pour tous les fichiers de configuration, vérifiez qu'il n'est lisible ou modifiable que par l'administrateur système.
Documentation	Ne suggérer pas de configuration non sécurisée dans votre documentation., spécialement si ces configurations vont étendre à des ressources et d'autres logiciel en dehors du cadre de votre application.
Installation	Pendant l'installation régler les autorisations de façon explicite pour les ressources que vous installez au lieu d'utiliser les autorisations par défaut de l'utilisateur qui installe le logiciel. Cela vous permettra de ne pas hériter de niveaux d'autorisations non sécurisés. Ne supposez pas que l'administrateur changera manuellement les réglages des configurations aux niveaux recommandés par les manuels.
Tests	Vérifiez que votre logiciel fonctionne correctement sous la norme FDCC (Configuration Fédérale de base des ordinateurs de bureau) ou son équivalent, ce qui est utilisé pour beaucoup d'organisations afin de limiter la surface d'attaque et les risques potentiels des applications déployées.

CWE liées

CWE-276 Permissions par défaut non sécurisées

CWE-277 : Permissions héritées non sécurisées

CWE-279 : Permissions assignées à l'exécution non sécurisées

Type d'attaques

CAPEC-ID:

CWE-330: Utilisation de valeurs pas assez aléatoires

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Brèche de Sécurité Perte de données
Coût de réparation	Moyen	Facilité de détection	Facile à difficile
Fréquence des attaques	Rarement	Sensibilisation attaquant	Moyen

Présentation

Imaginez la vitesse à laquelle un casino de Vegas ferait faillite si les joueurs pouvaient prédire les prochains jets de dés, tours de roulette ou cartes. Si vous utilisez des réglages de sécurité qui nécessitent de bonnes valeurs aléatoires, mais que vous ne les fournissez pas, alors vos attaquants riront en vous dépouillant. Vous pouvez même être dépendant de valeurs aléatoires sans le savoir, comme par exemple en gérant des identifiants de session ou des noms de fichiers temporaires. Des générateurs de nombres pseudo aléatoires (PRNG) sont utilisés couramment, mais beaucoup de choses peuvent mal se passer. Une fois qu'un attaquant a déterminé la nature de l'algorithme utilisé, il peut lancer une attaque avec un nombre minimum d'essais. Après tout, si vous êtes à Vegas et que

vous déterminez qu'un jeu ayant une probabilité de 1 sur 1000 peut être réduite à 1 sur 10 après avoir examiné quelques tirages, cela vaudrait sans doute le coup de continuer à jouer jusqu'au gain final.

Prévention et atténuation des conséquences

Architecture et conception	Utilisez des générateurs fiables de nombres pseudo-aléatoires, avec des sources de longueurs suffisantes. En général si un algorithme PRNG n'est pas connu comme étant cryptographiquement sûr, il est sans doute de nature statistique et ne devrait pas être utilisé en environnement sécurisé. PRNG peuvent produire des nombres prévisibles si le générateur est connu et que la source peut être devinée.
Implémentation	Faites des tests FIPS 140-1 sur des données pour identifier des problèmes d'entropie.
	Envisagez un PRNG qui se ressource lui-même, ce qui est indispensable à toute sortie pseudo-aléatoire de qualité, comme les périphériques.

CWE liées

CWE-329 : ne pas utiliser d'IV aléatoire en mode CBC

CWE-331 entropie insuffisante

CWE-334 : espace de valeurs aléatoires trop petit

CWE-336 : source identique en PRNG

CWE-338 : utilisation d'algorithme PRNG faible cryptographiquement

CWE-341 prévisions à partir d'états observables

Type d'attaques

CAPEC-ID:

CWE-250: Exécution avec privilèges non nécessaires

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Exécution de code
Coût de réparation	Moyen	Facilité de détection	Modérée
Fréquence des attaques	Quelques fois	Sensibilisation attaquant	Elevé

Présentation

La devise de Spiderman est : avec de grands pouvoirs viennent de grandes responsabilités. Votre logiciel a peut être besoin de privilèges spéciaux pour certaines opérations, mais garder ces privilèges plus longtemps que nécessaire est extrêmement risqué. Quand elle utilise des privilèges élevée votre application a accès à des ressources inaccessibles en temps normal par l'utilisateur. Vous pouvez par exemple lancer un programme séparé, et ce programme permet à l'utilisateur de spécifier u fichier à ouvrir ; cela est particulièrement le cas dans des utilitaires d'aide ou des éditeurs. L'utilisateur peut alors avoir accès à des fichiers sensibles grâce à ce programme. L'exécution de commande peut se passer de la même manière. Même si vous ne lancer pas d'autres programmes, des vulnérabilités additionnelles de votre logiciel, peuvent avoir des conséquences

plus funestes que si vous utilisiez un niveau de privilège plus bas.

Prévention et atténuation des conséquences

Architecture et conception	Identifiez les fonctionnalités qui ont besoin de privilèges additionnels, comme l'accès à certaines ressources du système d'exploitation. Enrobez et centralisez ces fonctionnalités si possible et isolez autant que faire ce peut le code à haute priorité du reste de l'application. Augmentez la priorité le plus tard possible et baissez là dès que possible. Evitez les faiblesses comme CWE-288 et CWE-420 en protégeant tous les canaux de communication possibles qui peuvent interagir avec votre code privilégié, comme un canal secondaire que vous réservez aux administrateurs.
Implémentation	Validez de façon exhaustive les entrées de tout code à haute priorité qui doit être exposé à l'utilisateur et rejetez tout ce qui ne correspond pas à vos strictes conditions.
	Vérifiez que vous abandonnez vos privilèges dès que possible (CWE-271) et que vous vérifiez bien leur abandon effectif.
Tests	Si les circonstances vous obligent à une exécution à privilèges élevés, alors déterminez le minimum requis. Tout d'abord identifiez les différentes permissions dont ont besoin le logiciel et son utilisateur pour les actions considérées, comme les droits de lecture et écriture des fichiers, des canaux réseau etc. Puis autorisez spécifiquement ces actions en interdisant toutes les autres. Vérifiez les entrées soigneusement pour ne pas ajouter de vulnérabilité supplémentaire. Cette atténuation est plus sujette aux erreurs que l'abandon de privilèges. Vérifiez que votre logiciel fonctionne correctement sous la norme FDCC (Configuration Fédérale de base des ordinateurs de bureau) ou son équivalent, ce qui est utilisé pour beaucoup d'organisations afin de limiter la surface d'attaque et les risques potentiels des applications déployées.

CWE liées

CWE-272 : Viol du privilège le plus faible

CWE-273 : Echec de vérification d'abandon de privilège

CWE-653 : Compartimentalisation insuffisante

Type d'attaques

CAPEC-ID:

CWE-602: Gestion coté client de la sécurité coté serveur

Résumé

Prévalence de la vulnérabilité	Moyenne	Conséquences	Brèche de sécurité
Coût de réparation	Elevé	Facilité de détection	Modérée
Fréquence des attaques	Quelques fois	Sensibilisation attaquant	Elevé

Présentation

Un client riche peut rendre un attaquant encore plus riche, et un client plus pauvre, si vous faites confiance au code client pour assurer la sécurité au lieu du serveur. Rappelez-vous que sous la surface de l'interface utilisateur à la mode, ça n'est que du code. Les attaquants peuvent décoder votre code client et développer le leur, en éliminant des caractéristiques gênantes comme les contrôles de sécurité. Les conséquences dépendront de ce que protègent vos contrôles de sécurité., mais les cibles les plus courantes sont les authentifications, autorisations et validations d'entrées. Si vous avez implémenté de la sécurité dans vos serveurs, faites attention à ne pas uniquement faire confiance à votre code client pour la gérer.

Prévention et atténuation des conséquences

Architecture et conception	Les mécanismes de protection doivent être gérés sur le serveur. Aucune interaction ou entrée en provenance du client ne doit être considérée comme sûre. Quelques schémas de validation peuvent être dupliqués dans le code client, comme la validation des entrées, car ils sont utilisés comme vérification générale des erreurs.
	Si un certain niveau de confiance est nécessaire entre deux entités, alors utilisez les vérifications d'intégrité et l'authentification forte pour vérifiez que les entrées proviennent d'une source sûre. Concevez les produits afin que cette confiance soit gérée de façon centralisée, spécialement s'il existe des canaux de communications nombreux et complexes, de façon à réduire les risques que l'implémentation oublie par accident une vérification dans un seul chemin de code. .

CWE liées

CWE-20 : Validation des entrées insuffisante

CWE-642 : Contrôle externe de données d'état critique.

Type d'attaques

CAPEC-ID:

Appendice A : Critères de sélection et champs de définition

Les entrées de la liste des 25 ont été choisies sur deux critères : la prévalence de la vulnérabilité et la dangerosité. La dangerosité se manifeste dans les conséquences habituelles de la vulnérabilité.

Prévalence de la vulnérabilité

Combien de fois trouve-t-on cette vulnérabilité dans un logiciel qui n'a pas été écrit en intégrant la sécurité dans le cycle de développement (SDLC). La prévalence ne s'applique qu'aux applications qui lui sont potentiellement sensibles. Par exemple la prévalence de l'injection SQL n'est évalué que pour les applications utilisant une base de données.

La valeur de prévalence est déterminée en se basant sur des estimations de contributeurs multiples à la Liste des 25, y compris les données de tendance CVE. Les 25 contributeurs les plus actifs ont souhaité se baser sur des statistiques plus précises, mais celles-ci ne sont pas disponibles ni en couverture, ni en pertinence. La plupart des travaux de suivi des vulnérabilités se font à haut niveau d'abstraction. Par exemple les données de tendance CVE peuvent suivre les débordements de tampon, mais les rapports publics de vulnérabilité mentionnent rarement le problème précis à

l'origine du débordement. Quelques éditeurs peuvent suivre les problèmes à bas niveau, mais ils sont rarement prêts à partager leurs informations.

- **Elevée** : la faiblesse a 50% de probabilité de se produire dans les logiciels potentiellement affectés
- **Moyenne** : la faiblesse a 10% à 50% de probabilité de se produire dans les logiciels potentiellement affectés
- **Faible** : la faiblesse a moins de 10% de probabilité de se produire dans les logiciels potentiellement affectés

Conséquences

Quand cette faiblesse existe dans un logiciel pour produire une vulnérabilité, quelles en sont les conséquences typiques ?

- **Exécution de code** : un attaquant peut exécuter du code ou des commandes
- **Pertes de données** : un attaquant peut voler, modifier ou corrompre des données sensibles
- **Interdiction de Service** : un attaquant peut entraîner un arrêt ou un ralentissement du logiciel, et empêcher ainsi les utilisateurs légitimes d'accéder au logiciel
- **Brèche de sécurité** : un attaquant peut passer au travers d'un mécanisme de protection ; les conséquences dépendront de ce que le mécanisme était censé protéger.

Fréquence des attaques

Combien de fois cette faiblesse se produit dans des vulnérabilités qui sont visées par un attaquant déterminé et entraîné ?

Considérez un ordinateur exposé qui peut être : un serveur connecté à Internet, un Client utilisant Internet, un système multiutilisateurs avec des utilisateurs peu surs, ou un système à plusieurs étages qui franchit les limites de plusieurs organisations ou niveaux de sécurité. Considérez également qu'un attaquant déterminé peut combiner des attaques sur plusieurs systèmes afin d'atteindre une machine ciblée.

- **Souvent** : une machine exposée pourra voir une attaque quotidienne
- **Quelques fois** : une machine exposée pourra voir une attaque mensuelle ou plus
- **Rarement** : une machine exposée pourra voir une attaque mensuelle ou moins

Facilité de détection

Est-il aisé à un attaquant déterminé et entraîné de trouver cette faiblesse, soit en utilisant des méthodes à base de listes blanches ou noires, manuelles ou automatiques ?

- **Facile** : des outils ou techniques automatiques existent pour détecter cette faiblesse, ou elle peut être trouvée rapidement en utilisant de simples manipulation (comme taper `<script>` dans un champ de saisie)
- **Modérée** : il n'existe qu'un support partiel de méthodes automatiques ; peut demander une compréhension de la logique de l'application ; peut n'exister qu'en de rare situation qui peuvent ne pas être directement sous le contrôle de l'attaquant (comme des conditions de mémoire insuffisante).
- **Difficile** : nécessite des méthodes manuelles coûteuses en ressources, ou un support intelligent semi-automatique, et une expertise de l'attaquant.

Cout de réparation

Est-il couteux de régler le problème quand il se produit ? Cela ne peut être quantifié de manière générale, car chaque développeur est différent. Pour l'objet de cette liste, le cout est défini comme :

- Faible : changement du code dans un seul bloc ou fonction
- Moyen : changement de code ou d'algorithme, sans doute en local dans un seul fichier ou composant
- Elevé : nécessite un changement significatif de conception ou d'architecture, ou bien le comportement vulnérable est nécessaire aux clients en aval, par exemple un problème de conception dans l'une des fonctions d'une bibliothèque.

Cette sélection ne tient pas compte d'autres facteurs de cout, comme des changements procéduraux, formation, déploiements des modifications, assurance qualité ou autres.

Sensibilisation de l'attaquant

Quelle probabilité existe t'il pour qu'un attaquant entraîné et déterminé connaisse cette faiblesse particulière, ses méthodes de détection et d'exploitation ? Cela suppose la connaissance par l'attaquant de la configuration et de l'environnement utilisés.

- Elevée : l'attaquant est capable de détecter ce type de vulnérabilité et de l'exploiter de façon fiable pour les plateformes et configurations les plus courantes.
- Moyenne : l'attaquant est conscient de la faiblesse par le suivi régulier de bulletins de sécurité ou de bases de données des vulnérabilités, mais ne l'a pas étudiée en détail ; des environnements ou techniques automatisés d'exploitation n'existent pas ou ne sont pas disponibles.
- Basse : l'attaquant ne connaît pas cette vulnérabilité ou ne l'a pas étudiée en détail, ou bien la faiblesse demande une certaine expertise technique que n'a pas l'attaquant (mais qu'il pourrait obtenir).

CWE Liées

D'autres entrées CWE liées à celle étudiée. Cela comprend des variantes de bas niveau ou des CWE qui peuvent se produire si celle-ci est présente.

Cette liste est indicative mais pas complète.

Appendice B : Modèle de menace de l'attaquant entraîné et déterminé

La sélection de la Liste des 25 suppose que l'utilisateur veut rendre longue et difficile une attaque du logiciel par un attaquant entraîné et déterminé.

Bien que plusieurs races d'attaquants existent, on présume qu'il a les caractéristiques suivantes :

Compétences :

- Possède une connaissance technique approfondie des vulnérabilités les mieux documentées

- Peut détecter et exploiter ces vulnérabilités avec une bonne chance de succès, en utilisant des méthodes à boîtes noires ou blanches.
- Peut combiner des attaques sur plusieurs systèmes pour accéder plus profondément à l'intérieur d'une organisation.

Détermination :

- Souhaite voler des données ou un logiciel complet pour ses capacités opérationnelles, indépendamment du motif (financier, militaire, politique ou autre).
- Ne fait pas forcément partie d'un groupe important ou riche, mais peut collaborer avec un petit nombre d'autres individus ; et
- Est prêt à investir au moins 20 heures pour voler un logiciel

La compétence et la détermination d'un attaquant seront en général supérieures à celle d'un gamin écrivant un script, mais inférieures à celles d'un état ou organisation criminelle.

Notez que ce modèle ne considère pas que l'interdiction de service soit la préoccupation principale de l'attaquant. Cela est contraire au modèle utilisé dans le commerce électronique ou la protection des infrastructures critiques, où les temps d'arrêt des systèmes peuvent avoir des conséquences catastrophiques.

Notez également que ce modèle fut développé assez tard dans la période de révision de la Liste des 25, et n'a donc eu que peu d'influence sur la sélection des éléments de la liste. Il est toutefois inclus ici pour indiquer dans quel contexte les valeurs des champs de spécification ont été choisies. Les auteurs des Liste « N » ultérieures pourraient envisager de rendre leur modèle de menace plus explicite, ce qui garantirait une hiérarchisation appropriée pour les environnements choisis.