Avant-propos

Quand j'ai commencé ma carrière, dans les années 1980, l'informatique personnelle en était à ses balbutiements. Les premiers ordinateurs disposaient d'une mémoire interne très réduite, les fichiers, voire le programme de lancement, étaient stockés sur des disquettes souples, de très faible capacité. Les interfaces graphiques n'existaient pas ou commençaient tout juste à être inventées. Les échanges avec les autres utilisateurs s'effectuaient par l'intermédiaire de disquettes, et pour les systèmes informatiques plus puissants, par l'échange de bandes magnétiques. L'accès aux ordinateurs centraux était réalisé à partir de terminaux passifs, reliés par des câbles dédiés (un câble par terminal). On parlait alors de *liaisons série*.

A partir des années 1990, suite à l'augmentation de puissance des machines, le réseau a progressivement remplacé les connexions filaires directes, et les terminaux ont été remplacés par des PC. Les ordinateurs ont pu alors commencer à dialoguer entre eux dans les entreprises. En revanche, les échanges entre les sites étaient réduits, les coûts de communication étant alors exorbitants et les débits très faibles. Les applications étaient écrites pour les systèmes d'exploitation visés, l'environnement était parfaitement maîtrisé : les écrans avaient tous les mêmes caractéristiques, les ordinateurs (les PC) également.

A la fin des années 1990, le développement du réseau Internet et la création des premiers sites web ouverts au public ont modifié profondément le paysage. La mutation vers la généralisation des applications web, en lieu et place de celles dédiées aux machines, a bouleversé les approches. Désormais, n'importe qui pouvait se connecter à n'importe quel logiciel, accéder à de l'information, commander des produits, puis gérer sa messagerie, etc.

Les années 2000, avec la généralisation du sans fil (le wifi n'a commencé vraiment à s'implanter de façon massive en France qu'au milieu des années 2000), ont également été le début d'une prise de conscience des problèmes de sécurité induits

par le nouvel environnement. Auparavant, les problèmes étaient assez circonscrits, d'une part parce que les informaticiens étaient rares, et d'autre part parce que l'environnement technique était totalement maîtrisé. Cela ne veut pas dire que la sécurité n'était pas prise en compte (le chiffrement a toujours existé) mais, en général, seules les sociétés ou les organismes qui manipulaient des informations sensibles (banques, militaires, etc.) y apportaient une attention soutenue. Pour la plupart des entreprises, le risque de perte de données était très souvent le seul pris en compte, et encore parfois de manière très légère. On vérifiait également l'identité de l'utilisateur avec un *login* et un mot de passe non chiffré, les risques d'interception étant alors assez faibles. Les machines n'étaient alors de toute façon pas assez puissantes pour intégrer du chiffrement « à la volée ».

Mais des individus, voire des organisations, ont vite compris les gains financiers, puis politiques, qu'ils pouvaient obtenir en s'attaquant à ces applications web : Internet n'était plus statique comme dans ses débuts (les pages n'étaient alors pas modifiables), mais dynamique. C'est l'utilisateur qui induit le comportement du serveur et les informations qui lui seront présentées.

Les applications web présentent un risque nouveau : l'information qui arrive du navigateur doit être vérifiée systématiquement. Des comportements non prévus peuvent être occasionnés si l'utilisateur envoie des données modifiées dans ce but. C'est le cas, par exemple, de l'injection SQL où, sans protection particulière, il est possible de faire faire au serveur des opérations non prévues par le programmeur.

Aujourd'hui, le développement de logiciels web nécessite des compétences techniques très variées (langage de programmation comme PHP, HTML pour dessiner les pages, CSS pour leur donner un style unique, JavaScript, et ses déclinaisons comme JQuery, pour ajouter de l'interaction, SQL pour accéder aux bases de données, etc.).

De nombreux programmeurs apprennent ces langages, mais n'ont pas forcément conscience des risques qu'ils induisent. La formation initiale insiste rarement sur ces aspects (sauf formations spécialisées), ou n'en traite que certains, préférant se concentrer sur les apprentissages fondamentaux. Il n'est pas rare de rencontrer des développeurs fraîchement diplômés qui n'ont que de rares notions sur la manière de sécuriser leurs logiciels. C'est également souvent le cas des autodidactes, qui commencent à programmer quand le besoin s'en fait sentir, par exemple pour mettre en place un site d'association, ou qui écrivent une interface permettant d'alimenter une petite base de données.

Pourtant, les risques sont bien là : sans protection particulière, les données peuvent être corrompues, le site peut servir de porte d'entrée à un pirate qui pourra prendre la main dans le système d'information de l'entreprise, ou être défiguré à des fins politiques.

Prendre le temps de réfléchir à la sécurité de son application est donc particulièrement important. Cela commence par une estimation du risque : une application bancaire n'a pas la même sensibilité qu'une réservation de salles de réunion dans un établissement.

Il faut ensuite tenir compte de l'environnement d'exécution du logiciel. Il ne peut être intrinsèquement sûr : si les machines qui l'hébergent ne sont pas elles-mêmes protégées, toutes les protections mises en place dans le code ne serviront pas à grand-chose. De la même manière, les échanges entre le serveur et les utilisateurs doivent être chiffrés pour éviter une écoute intempestive, voire une modification à la volée. Bien sûr, on s'éloigne un peu du codage proprement dit de l'application, mais les mécanismes de chiffrement sont très souvent utilisés dans de nombreuses routines, notamment en raison de la garantie qu'ils apportent pour certaines opérations.

Connaître les risques qui menacent le code est compliqué, ne serait-ce que par l'étendue des mécanismes d'attaques possibles. Il existe, heureusement, des projets comme l'OWASP qui éditent régulièrement les listes des attaques les plus fréquentes, et il suffit souvent de quelques lignes de code pour s'en protéger. L'ANSSI, l'Agence nationale de la sécurité des systèmes d'informations, promulgue également régulièrement des conseils et apporte, *via* la méthode EBIOS, une méthodologie permettant d'analyser le risque.

Un aspect important de la sécurité concerne la gestion des utilisateurs et de leurs droits d'accès. Il existe plusieurs mécanismes qui peuvent être mis en œuvre pour les identifier et leur permettre d'accéder aux informations dont ils ont besoin.

Concevoir une application est une opération complexe, et la seule manière d'être sûr que la sécurité soit bien prise en compte partout, c'est de structurer son code. Des méthodes d'organisation, comme le modèle MVC (modèle, vue, contrôleur), permettent de répondre à ces besoins et non seulement de fiabiliser le fonctionnement de l'application, mais également d'assurer sa sécurité.

Enfin, tester le logiciel créé avec des outils adaptés permettra de repérer les failles les plus criantes. C'est une étape importante avant la mise en production, qui garantira auprès des utilisateurs que les précautions ont bien été prises en regard des risques encourus.

De nombreux exemples sont présentés dans cet ouvrage. Ils ont été écrits principalement en PHP, un des langages les plus fréquemment utilisés pour créer des applications web. Ils peuvent bien sûr être adaptés à d'autres. Ce ne sont souvent que quelques lignes de code qui sont nécessaires pour boucher une faille, et l'algorithme utilisé ou la manière d'aborder le problème est souvent plus intéressant que le code lui-même.