

DOSSIER

S'opposer à la collecte de données par des robots moissonneurs

Juin 2026

linc.cnil.fr

| Romain Darous, Ingénieur au Service de l'Intelligence Artificielle

Le développement de modèles d'IA à usage général nécessite la création d'immenses bases de données, souvent constituées de données publiques collectées en ligne. Les propriétaires de sites internet qui font l'objet de cette collecte disposent d'un ensemble de moyens pouvant les aider à maîtriser les données prélevées et les usages qui peuvent en être fait. Les articles de ce dossier introduisent tout d'abord les concepts relatifs au moissonnage de données en ligne, avant de présenter les méthodes qui peuvent être mises en place par l'éditeur du site internet pour maîtriser ce moissonnage.

SOMMAIRE DU DOSSIER

ARTICLE INTRODUCTIF	4
INTRODUCTION	4
POURQUOI ENCADRER LES PRATIQUES DE MOISSONNAGE ?	5
DES RISQUES POUR LE BON FONCTIONNEMENT DES SITES, ...	5
... POUR LA PROPRIETE INTELLECTUELLE, ...	5
... ET POUR LA PROTECTION DES DONNEES PERSONNELLES DES PERSONNES CONCERNEES	5
FAUT-IL MODULER SON OPPOSITION ?	6
DES METHODES FAILLIBLES, INADAPTEES AUX DONNEES PERSONNELLES	6
MODULER SON OPPOSITION EN FONCTION DE LA FINALITE DU ROBOT	6
DES ALTERNATIVES A L'OPPOSITION AU MOISSONNAGE	7
CRAWLING, SCRAPING, TDM DE QUOI PARLE-T-ON ?	8
WEB CRAWLING	8
WEB SCRAPING	10
LES ROBOTS DE SCRAPING ET DE CRAWLING	11
EXEMPLE DE SCRAPING	11
TDM (TEXT AND DATA MINING)	12
APPLICATION A LA CREATION DE GRANDES BASES DE DONNEES POUR ENTRAINER DES LLMs	13
S'OPPOSER PAR DES METHODES DECLARATIVES	14
LE PROTOCOLE RFC 9309, OU « ROBOTS.TXT »	14
LES TAGS <META>	16
DES ALTERNATIVES EMERGENTES, SPECIFIQUES AU MOISSONNAGE	16
LE PROTOCOLE « AI.TXT »	16
LE PROTOCOLE TDMREP	17
DISCUSSION SUR LA PORTEE DE CES METHODES	19
UNE INTEGRATION INEGALE	19
DES MESURES AUX GARANTIES LIMITEES	20
L'EXPERIENCE UTILISATEUR RESTE INTACTE	20
S'OPPOSER EN BLOQUANT L'ACCES AU SITE INTERNET	21
LE DYNAMIC PAGE LOADING POUR DEVELOPPER LE SITE INTERNET	21
UNE MESURE A PORTEE LIMITEE	22
METTRE EN PLACE DES RESTRICTIONS D'ACCES	22
UTILISER DES CAPTCHAS	23
L'IMPACT SUR L'EXPERIENCE UTILISATEUR	24
UNE METHODE LOIN D'ETRE INFALLIBLE	24
EXAMINER LES REQUETES HTTP DU CLIENT	25
UTILISER DES PARE-FEUX OU DES CDNS	26

DES METHODES IMPARFAITES ET PARFOIS INTRUSIVES	26
ARTICLE BONUS : LE « FINGERPRINTING »	27
RAPPEL SUR LES PROTOCOLES DE COMMUNICATION WEB	27
LES PROTOCOLES DE COMMUNICATION WEB	27
HTTP/2 ET HTTP/3, QUELLES DIFFERENCES ?	28
INFORMATIONS DE LA COUCHE TRANSPORT TCP	29
MISE EN PLACE DU PROTOCOLE DE SECURITE	30
POIGNEE DE MAIN QUIC	31
INFORMATIONS OBTENUES DANS LES EN-TETES HTTP	32
IDENTIFIER LE NAVIGATEUR PAR L'EXECUTION COTE CLIENT	33
L'ACCESSIBILITE DES INFORMATIONS	34
UTILISER CES INFORMATIONS : LES EMPREINTES	35
IMPLEMENTATION DES BIBLIOTHEQUES D'EMPREINTE TCP, TLS ET HTTP	35
IMPLEMENTATION DES BIBLIOTHEQUES D'EMPREINTE NAVIGATEUR	37
CONCLUSION	37

Article introductif

Introduction

Lorsqu'un éditeur met son site internet en ligne, il le fait pour atteindre une audience et lui proposer du contenu (articles, vidéos, podcasts), vendre un produit, ou encore offrir des services en ligne. Or, les utilisateurs ne connaissent pas l'adresse exacte (appelée nom de domaine) de tous les sites qu'ils souhaitent visiter. Ils s'appuient donc sur un moteur de recherche, qui suggère des pages web à partir des mots-clés saisis. Pour fournir des résultats pertinents, les moteurs de recherche utilisent des programmes automatisés qui parcourent le Web, analysent les sites internet et les indexent. Ce processus est essentiel pour les deux parties : il permet aux moteurs de recherche de proposer des résultats de recherche utiles à l'utilisateur, et aux éditeurs de rendre leur contenu visible auprès de lecteurs, clients ou d'utilisateurs potentiels. Les entreprises qui développent des moteurs de recherche et les éditeurs de site peuvent donc chacun y trouver un bénéfice.

Aujourd'hui, certains grands modèles de langage peuvent perturber ce partage des bénéfices. Leur entraînement repose souvent sur une collecte massive de données en ligne, mais les bénéfices tirés par les propriétaires de ces sites internet sont souvent moins évidents. D'après [The Wall Street Journal](#) l'essor des agents conversationnels et des résumés générés par IA intégrés aux résultats de moteurs de recherche, qui donnent un accès à l'information sans cliquer sur les liens, entraînerait une baisse de la fréquentation des sites web. Cette collecte massive de données en ligne pose également la question de la licéité et des obligations de l'entité qui collecte ces données lorsqu'elles incluent des données à caractère personnel. La CNIL rappelle d'ailleurs les obligations du responsable de traitement dans le cadre du développement d'un système d'IA impliquant le traitement de données personnelles dans [ses fiches pratiques dédiées](#).

Les éditeurs de ces sites internet disposent de moyens techniques pour encadrer cette pratique : ils peuvent choisir d'autoriser ou non l'accès à leurs données, en mettant en place des mécanismes permettant de bloquer les robots, ou en définissant explicitement les conditions d'accès via des fichiers spécifiques et consultés par les programmes informatiques qui procèdent à la collecte de leurs données.

Pourquoi encadrer les pratiques de moissonnage ?

Des risques pour le bon fonctionnement des sites, ...

Les risques associés au moissonnage, surtout massif, sont nombreux. Non surveillées, ces pratiques représentent un danger technique pour l'hébergement de sites internet. Le moissonnage entraîne en effet une augmentation de requêtes effectuées sur un même serveur, ce qui peut conduire à des surcharges, voire des plantages de serveurs ayant des conséquences similaires à des attaques en déni de service (DDoS). [La saturation récente du site de l'entreprise Triplegangers causée par les robots moissonneurs d'OpenAI en est un exemple.](#)

... pour la propriété intellectuelle, ...

Le moissonnage peut également porter atteinte à la propriété intellectuelle. Dans une note présentant un état des lieux [des mécanismes d'opposition, le PEReN](#) (service public qui accompagne les pouvoirs publics dans la régulation des plateformes numériques et de l'IA par la mise à disposition de compétences techniques) expose le problème des droits d'auteur en donnant l'exemple d'une sanction visant Google en 2024 pour avoir entraîné des modèles d'IA générative sur les données de sites et d'agences de presse sans les en avertir et sans fournir de mécanisme d'opposition à l'utilisation des contenus par l'IA de Google sans impacter l'affichage dans le moteur de recherche. Le PEReN discute plus généralement de la question du partage de valeur que pose la collecte de données publiquement accessibles en ligne, entre les développeurs de modèles de fondation et les propriétaires des sites internet dont les données sont utilisées pour l'entraînement.

... et pour la protection des données personnelles des personnes concernées

D'autre part, le moissonnage entraîne souvent la collecte de données personnelles. Il faut alors se conformer au RGPD et s'assurer que ce traitement de données est licite. La CNIL reconnaît par exemple la base légale de l'intérêt légitime (article 6.1.f du RGPD) comme valable pour collecter des données publiquement accessibles en ligne à des fins de développement de systèmes d'IA, sous réserve de la mise en œuvre de garanties fortes détaillées dans la [fiche pratique IA dédiée](#).

Nous proposons ainsi une série de trois articles, visant à :

- revenir sur les pratiques existantes de collecte de données accessibles en ligne ;

- détailler les méthodes déclaratives qu'un éditeur de site internet peut mettre en œuvre pour les encadrer ;
- explorer les mesures qui permettent de bloquer complètement l'accès au site internet lorsque l'utilisateur est détecté comme étant un robot moissonneur.

Faut-il moduler son opposition ?

Des méthodes faillibles, inadaptées aux données personnelles

Associer protocoles déclaratifs et méthodes bloquantes limite au maximum le risque de moissonnage non souhaité d'un site internet. Les premiers reposent sur des standards déjà existants et sur des initiatives conçues spécialement pour les robots moissonneurs. La question de leur respect systématique se pose néanmoins. Les mesures bloquantes permettent plus généralement un contrôle du trafic malveillant sur un site web, incluant la détection de robots moissonneurs qui ne respecteraient pas les protocoles déclaratifs. Si, de facto, elles permettent d'empêcher à un robot de moissonnage détecté de parcourir le site, elles peuvent nuire à l'expérience utilisateur, être intrusives et restent malgré tout faillibles.

D'autre part, ces méthodes ne sont pas conçues spécifiquement pour assurer l'exercice du droit d'opposition au sens du RGPD, lorsque cette collecte concerne des données à caractère personnel. Les méthodes déclaratives se limitant à des autorisations par page et/ou par type de contenu, une méthode d'opposition granulaire à l'échelle de chaque personne concernée reste à construire.

Moduler son opposition en fonction de la finalité du robot

Pour finir, il est intéressant de se demander quels types robots moissonneurs bloquer, car il en existe plusieurs sortes aux fonctions différentes. Dans sa [note sur les mécanismes d'opposition](#), le PEReN propose de classifier les différents robots utilisés par des LLMs en quatre catégories : les « AI data scrapers », qui collectent des données à des fins d'entraînement de modèles de fondation, les « AI search crawlers » ou « AI assistants » qui se rendent sur internet pour alimenter les réponses en contenu à jour pour les utilisateurs des agents conversationnels et les « undocumented AI Agents », des robots moissonneurs non identifiés dont la finalité de collecte n'est pas connue.

Les agents conversationnels, bien que sujets à des hallucinations, s'imposent comme une [source d'information majeure](#) et parfois exclusive. Ils implémentent désormais presque systématiquement des fonctionnalités de recherche web (par l'intermédiaire des « AI search crawlers », ou « AI assistants ») pour avoir des réponses fiables et d'actualité, qui créditent donc les sites internet utilisés pour les générer. L'utilisateur peut se rendre sur les liens fournis pour vérifier les sources et approfondir ses recherches. Si cette pratique n'entraîne pas un clic systématique de l'utilisateur, elle en donne au moins la possibilité. Il est donc valable de

questionner l'opposition de l'accès par ces robots au contenu d'un site internet, qui ne pourrait alors plus être utilisé comme référence dans une réponse d'un agent conversationnel.

La question est d'autant plus d'actualité que l'émergence d'IA agentiques implique que les modèles de fondation peuvent désormais effectuer des actions autonomes sur un navigateur virtuel à la demande d'un utilisateur. Bloquer ces robots, c'est bloquer l'utilisateur humain à l'origine de la requête.

Des alternatives à l'opposition au moissonnage

L'éditeur d'un site internet peut donc souhaiter rendre son contenu accessible à ces robots moissonneurs pour apparaître dans les sources des réponses fournies par les agents conversationnels aux utilisateurs finaux. Avec cette volonté émerge le concept de SAIO (*Search AI Optimization*), l'équivalent du SEO (*Search Engine Optimization*) appliqué au référencement conçu pour rendre son site internet visible et exploitable par les robots. Il est possible également de noter l'émergence de protocoles tels que « LLMs.txt », un fichier texte qui s'insère dans le code source du site internet et qui permet d'y écrire des informations facilement lisibles par un agent conversationnel pour lui donner directement accès au contenu d'intérêt sur le site internet. Des entreprises telles que Parallel AI ou Perplexity se spécialisent dans la recherche optimisée par IA, ce qui témoigne de l'importance que prend cet usage.

Enfin, il est possible d'établir des contrats avec les entreprises qui procèdent à la collecte de données en ligne. Le PEReN donne notamment l'exemple des partenariats entre OpenAI et Le Monde, Google et Associated Press, Perplexity Humanoid (Ebra) et Mistral et l'AFP. Reddit met également à disposition une API qui donne accès aux données publiques du forum en ligne mises en forme et nettoyées. Ces partenariats permettent de tirer une rémunération du moissonnage et participent à un rééquilibrage des bénéfices tirés par les deux parties de la collecte.

Crawling, scraping, TDM de quoi parle-t-on ?

Avant d'aborder les moyens permettant aux éditeurs de sites internet de s'opposer à la collecte de leurs données, il convient d'abord de définir les pratiques mises en œuvre par les moteurs de recherche et les entreprises développant des modèles d'intelligence artificielle générative. En effet, plusieurs notions sont mobilisées dans ce contexte : *crawling*, *scraping* et TDM (*Text and Data Mining*). Bien qu'elles désignent des pratiques similaires, leur périmètre exact varie selon les usages et les interprétations. Il est donc essentiel de clarifier la signification de chacun de ces termes afin de poser un cadre précis à la discussion.

Web crawling



Image générée par IA (Imagen, Google)

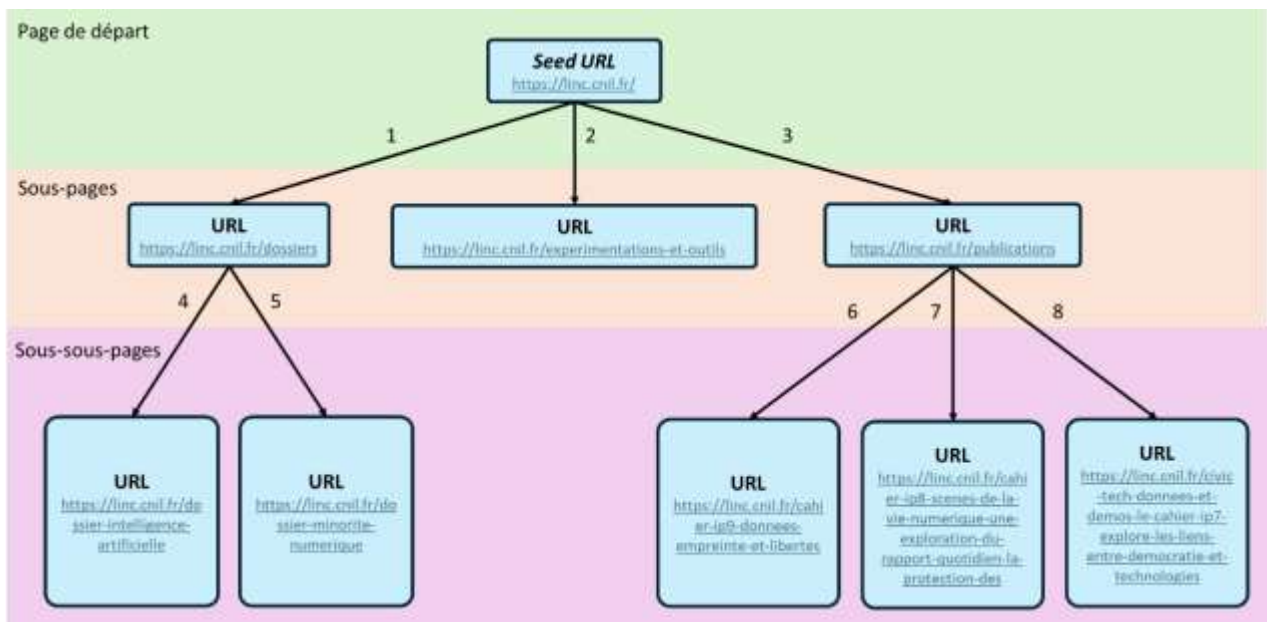
Le **web crawling**, ou « exploration du web », désigne le processus d'exploration automatique des pages web accessibles sur Internet. Il débute à partir d'un ensemble initial d'URLs, appelées *seed URLs*. Ces points de départ permettent de lancer l'exploration du web. Le reste des pages disponibles en ligne, dont [le nombre se chiffre en plusieurs milliards](#), est alors inconnu à ce stade. L'objectif du *crawling* est précisément de découvrir les pages internet restantes, en suivant les liens hypertextes des pages visitées. Le résultat de cette opération de collecte est appelé un *crawl*.

Prenons l'exemple de CommonCrawl, une organisation à but non lucratif spécialisée dans le *crawling*. Elle publie chaque mois le résultat de ses explorations, en open data. Comme point de départ, CommonCrawl utilise des URLs fournies par des tiers, d'autres extraites de « plans de sites » (ou *sitemaps* en anglais, un protocole qui permet aux fournisseurs de sites internet de donner des informations lisibles par une machine sur les pages qu'elle peut explorer), [des](#)

noms de domaine majeurs identifiés à partir de graphes web qu'elle construit, ou encore des échantillons aléatoires d'URLs provenant de crawls antérieurs. Ces *seed* URLs se comptent alors en centaines de millions. Ensuite, l'organisation procède par parcours en largeur (*Breadth-First Search*), avec une profondeur maximale fixée autour de 4 à 5 « sauts », permettant la découverte de nouvelles URLs. Ces méthodes leur permettent d'explorer autour de 2 milliards de pages web par mois, portant à 20 milliards le nombre total d'URLs uniques collectées entre juillet 2024 et juin 2025, par exemple. Chaque *crawl* permet de découvrir de nouvelles URLs tout en incluant inévitablement des duplicatas d'URLs présentes dans les crawls précédents. Ces doublons permettent toutefois de **mettre à jour les pages déjà connues**, ce qui constitue l'un des grands défis du *web crawling*.

Cette exploration s'apparente à un parcours de graphe dirigé. En effet, les sites web sont interconnectés par des liens internes, qui permettent la navigation entre les différentes pages d'un même site, et des liens externes, qui renvoient vers d'autres sites. Par exemple, un comparateur d'hôtels peut regrouper les offres disponibles dans une région et sur une période donnée, puis rediriger l'utilisateur vers les sites des hôtels pour finaliser la réservation. À l'inverse, les pages de **Wikipédia** sont riches en liens internes facilitant la navigation d'un article à l'autre au sein du site.

A partir d'URLs données, le *web crawling* permet donc la découverte progressive et récursive de liens. Cette exploration s'accompagne de la collecte des URLs visitées, des titres des pages web, de leur description et leur contenu HTML. Les bases de données constituées à l'issue du *crawling* atteignent des tailles de l'ordre du PB (Petabyte ou Petaoctet, équivalent à 1000 Teraoctets).



Exemple de découverte de liens en utilisant comme *seed* URL la page d'accueil du site du LINC. La numérotation indique l'ordre de découverte des pages en respectant l'algorithme de parcours BFS.

A l'origine, le *web crawling* est une des étapes clés du développement de moteurs de recherche. Une fois les URLs parcourues et le contenu des pages associées collecté, une phase **d'indexation** permet de référencer et classer les pages web entre elles en examinant leur contenu HTML afin de fournir des résultats de recherche pertinents et personnalisés à l'utilisateur de ce moteur de recherche. A titre d'exemple, [50 milliards de pages web étaient indexées par le moteur de recherche Google en janvier 2025](#).

Web scraping

Le *web scraping*, ou « moissonnage » en français, concerne quant à lui l'extraction d'informations présentes sur des pages internet **déjà connues**. C'est la principale distinction avec le *web crawling*. A partir d'URLs identifiées, le *web scraping* consiste à analyser le **contenu HTML** de chaque page afin d'en extraire les données pertinentes, puis à structurer ces données pour les rendre exploitables.

Le *web scraping* offre une large gamme d'usages, parmi lesquels figure la veille tarifaire, qui consiste à comparer automatiquement les prix pratiqués par une entreprise avec ceux de ses concurrents. Cela permet de mener une surveillance concurrentielle ou d'analyser les tendances tarifaires sur un marché donné. Cette technique facilite également l'agrégation automatisée de données, utile dans de nombreux contextes : analyses financières, constitution de corpus journalistiques, ou encore recherches académiques. Dans ces derniers cas, le *scraping* permet de rassembler un volume important d'informations pour mener des enquêtes, étayer des analyses, ou établir un état de l'art aussi exhaustif que possible.

Les principes du *web crawling* et de *web scraping* peuvent donc être résumés ainsi : le premier consiste à collecter les **URLs** de pages web accessibles en ligne, tandis que le second vise à extraire et structurer les informations pertinentes contenues dans ces pages



Image générée par IA (Imagen, Google)

Les robots de *scraping* et de *crawling*

Le *crawling* et le *scraping* sont deux pratiques complémentaires qui permettent la collecte automatisée de données à grande échelle. Cette automatisation repose sur l'utilisation de robots (appelés *crawlers* et *scrapers*, respectivement, ou plus généralement *spiders* ou parfois *wanderers*), c'est-à-dire des programmes informatiques conçus pour explorer systématiquement des pages web et en extraire les informations pertinentes. Mis en œuvre à grande échelle, ces robots sont généralement déployés sur plusieurs machines afin de répartir la charge et d'accélérer le processus de collecte. Ils sont hébergés dans des régions géographiques proches des serveurs des sites ciblés, dans le but de [réduire la latence et d'optimiser l'efficacité de la collecte](#).

Leur mise en œuvre à l'échelle de l'internet soulève donc de nombreux défis techniques, algorithmiques et d'infrastructure, en raison des milliards de pages à explorer, du besoin d'exhaustivité, et du volume massif de données généré. Parmi les enjeux majeurs, il est possible de citer :

- La **répartition intelligente des robots** entre les sites, afin d'éviter les risques de surcharge des serveurs ;
- La **mise à jour régulière** des pages collectées, avec des questions de fréquence et de priorisation pour garantir l'intégrité et l'exactitude des données ;
- La **déduplication** des informations collectées, incluant la **détection des quasi-duplicatas** (*near duplicates*) à l'aide d'algorithmes tels que **SimHash**, qui permettent d'identifier des pages très similaires mais non parfaitement identiques (par exemple, des variations contenant des publicités différentes) ;
- Et enfin, le **nettoyage** des données récoltées, condition indispensable pour les rendre exploitables dans des analyses ou applications en aval.

Exemple de *scraping*

Le moissonnage s'effectue en repérant les balises HTML d'intérêt par leur type (paragraphes <p>, sections <div>, etc.) ou les classes et identifiants CSS associés (champs « id » et « class »). Sur Python, nous pouvons par exemple utiliser la bibliothèque **requests** pour télécharger le contenu .html d'une page web, puis la bibliothèque **bs4** pour en extraire le contenu d'intérêt.

La figure ci-dessous montre comment il est possible d'automatiser la collecte de données sur une page web, en utilisant soit les balises HTML, soit les noms d'identifiants CSS :

```

theatre.html
1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4 <meta charset="UTF-8">
5 <title>Théâtre Classique et Héritage Grec</title>
6 </head>
7 <body>
8
9 <h1>Les Fondements du Théâtre Classique</h1>
10
11 <p id="comedie">
12 La comédie classique met en scène des situations quotidiennes,
13 souvent tournées vers la critique sociale et les travers humains.
14 Elle se distingue par une structure rigoureuse (règles des trois unités,
15 bienséance) et une fin généralement heureuse.
16 </p>
17
18 <p id="tragedie">
19 La tragédie classique repose sur des personnages nobles confrontés à
20 des conflits moraux ou fatals. Ils ne peuvent échapper à leur destin.
21 Elle cherche à susciter la terreur et la pitié tout en respectant
22 les règles des trois unités et de bienséance
23 </p>
24
25 <p id="grecs">
26 Les auteurs classiques s'inspirent largement du théâtre grec,
27 notamment des œuvres de Sophocle, Euripide et Aristophane,
28 reprenant l'idée de catharsis et de structure dramatique équilibrée.
29 </p>
30
31 </body>
32 </html>
33
scraping.py
25 from bs4 import BeautifulSoup
26
27 # Parser le contenu HTML
28 soup = BeautifulSoup(html_content, 'html.parser')
29
30 # Récupération de tous les paragraphes
31 paragraphes = soup.find_all('p')
32 with open('paragraphes.txt', 'w', encoding='utf-8') as f:
33     for p in paragraphes:
34         f.write(p.get_text(strip=True) + '\n')
35
36 # Récupération du paragraphe avec id="tragedie"
37 paragraphe_tragedie = soup.find('p', id='tragedie')
38 if paragraphe_tragedie:
39     with open('tragedie.txt', 'w', encoding='utf-8') as f:
40         f.write(paragraphe_tragedie.get_text(strip=True))
41

```

Les Fondements du Théâtre Classique

La comédie classique met en scène des situations quotidiennes, souvent tournées vers la critique sociale et les travers humains. Elle se distingue par une structure rigoureuse (règles des trois unités, bienséance) et une fin généralement heureuse.

La tragédie classique repose sur des personnages nobles confrontés à des conflits moraux ou fatals. Ils ne peuvent échapper à leur destin. Elle cherche à susciter la terreur et la pitié tout en respectant les règles des trois unités et de bienséance

Les auteurs classiques s'inspirent largement du théâtre grec, notamment des œuvres de Sophocle, Euripide et Aristophane, reprenant l'idée de catharsis et de structure dramatique équilibrée.

Sur cet exemple est collecté d'abord le texte contenu dans tous les paragraphes de la page web, puis uniquement le paragraphe dont l'identifiant CSS est id="tragedie"

TDM (Text and Data mining)

Pour finir, le terme de TDM (*Text and Data Mining* ou minage de texte et de données) regroupe l'ensemble des pratiques permettant l'analyse d'un grand nombre de données sous toutes ses formes : texte, image, vidéo, audio, etc. Le concept de TDM désigne donc plus largement toutes les étapes qui permettent de tirer des informations à partir de données obtenues en trop grande quantité pour être traitées par des méthodes traditionnelles. Les pratiques de TDM comprennent donc la collecte et le traitement de ces données (nettoyage, extraction de caractéristiques et représentation mathématique, etc.), ainsi que l'application à ces données d'algorithmes et de traitements analytiques pour en extraire de l'information (clustering, arbres de décision, SVMs, réseaux de neurones, modèles de langages, etc.) ou pour leur structuration. Le moissonnage constituerait donc en ce sens une étape du TDM dans le contexte de développement d'un modèle de fondation. Cependant, la terminologie TDM est parfois utilisée pour désigner la phase initiale de collecte de données et peut donc être confondue avec le *web scraping* et/ou le *web crawling*. Par exemple, le protocole TDMRep

étudié plus loin désigne un moyen de contrôler la collecte de données publiquement accessibles par les robots sur les sites web.

Application à la création de grandes bases de données pour entraîner des LLMs

La maîtrise du crawling et du scraping permettent notamment la création de grandes bases de données de qualité pour l'entraînement de modèles de fondation, et elle a un [impact majeur sur les performances finales du modèle développé](#).

Ces techniques combinées permettent de procéder à une collecte de masse, aussi exhaustive, diversifiée et de qualité que possible. Le *crawling* permet la découverte des pages web, qui sont ensuite *scrapées* pour en extraire le contenu de valeur et l'utiliser pour entraîner des modèles d'IA.

Leur usage simultané implique en général de désigner les deux pratiques par un de ces termes. Dans la suite, lorsque la distinction n'est pas nécessaire, le terme de « moissonnage » sera utilisé pour désigner le *scraping* et le *crawling*, et le terme « robot » pour désigner tout algorithme de collecte d'informations publiquement accessibles en ligne.

S'opposer par des méthodes déclaratives

Il existe plusieurs manières d'empêcher le moissonnage, qui peuvent se cumuler et qui fonctionnent différemment. Une première manière de le faire est d'avoir recours à des méthodes déclaratives, c'est-à-dire qu'elles n'empêchent pas l'accès du robot au site, mais elles permettent de lister les droits d'accès qui lui sont donnés de manière non contraignante. La méthode la plus simple est d'inclure des mentions dans les CGUs du site. Elles permettent une opposition de principe à la collecte des données du site internet, mais elle est difficilement exploitable par les robots moissonneurs. En effet, cela suppose de collecter les CGUs de chaque domaine moissonné, d'en parcourir le contenu et d'en extraire les règles entourant l'accès au reste du site web. Une telle pratique est sujette à erreur car non normalisée, et passe difficilement à l'échelle. Elle doit donc être complétée par des méthodes qui sont interprétables par une machine.

Le protocole RFC 9309, ou [« robots.txt »](#)

Le *Robots Exclusion Protocol* ([RFC 9309](#)) est un protocole datant de [1994](#) permettant à un éditeur de site de définir des autorisations et des interdictions d'accès aux robots moissonneurs. A l'origine, il servait à encadrer le *crawling* à des fins d'indexation pour le développement de moteurs de recherche. Il prend la forme d'un fichier texte « robots.txt » qui est ajouté au code source d'un site internet. Il doit être accessible publiquement en ajoutant « /robots.txt » à l'URL racine d'un site. L'usage des fichiers « robots.txt » s'étend désormais au contrôle des accès des robots moissonneurs qui collectent des données accessibles en ligne pour entraîner des modèles d'IA.

La configuration du fichier « robots.txt » se fait en deux étapes. Tout d'abord, est désigné le robot auquel sont dictés les droits d'accès, grâce au champ « **User-agent** ». Ce champ correspond au nom du robot, qui est en général fourni par son développeur et accessible sur son site internet (à titre d'exemple, voir le nom des robots d'Anthropic ci-dessous). Il est également possible de trouver des bases de données qui rassemblent les [noms de robots connus](#). Le champ « **User-agent** » peut aussi être complété par un astérisque (*). Les règles concernent alors **tous les robots**. Ensuite, pour chaque champ « **User-agent** », il reste à faire la liste des droits d'accès, à l'aide des champs « **Allow** » pour une autorisation, et « **Disallow** » pour une interdiction. Il est possible d'affiner les autorisations par :

- Page ou répertoire de pages web,
- Par type de fichier (à partir de son extension)

```

User-agent: *
# CSS, JS, Images
Allow: /core/*.css?
Allow: /core/*.js$
Allow: /core/*.js?
Allow: /core/*.gif
Allow: /core/*.jpg
Allow: /core/*.jpeg
Allow: /core/*.png
Allow: /core/*.svg

# Directories
Disallow: /core/
Disallow: /profiles/

Disallow: /fr/core/
Disallow: /fr/profiles/

Disallow: /en/core/
Disallow: /en/profiles/

# Files
Disallow: /README.md
Disallow: /composer/Metapackage/README.txt
Disallow: /composer/Plugin/ProjectMessage/README.md
    
```

Extrait du fichier « robots.txt » du [site du LINC](#)

Bot	Use	What happens when you disable it
ClaudeBot	ClaudeBot helps enhance the utility and safety of our generative AI models by collecting web content that could potentially contribute to their training.	When a site restricts ClaudeBot access, it signals that the site's future materials should be excluded from our AI model training datasets.
Claude-User	Claude-User supports Claude AI users. When individuals ask questions to Claude, it may access websites using a Claude-User agent.	Claude-User allows site owners to control which sites can be accessed through these user-initiated requests. Disabling Claude-User on your site prevents our system from retrieving your content in response to a user query, which may reduce your site's visibility for user-directed web search.
Claude-SearchBot	Claude-SearchBot navigates the web to improve search result quality for users. It analyzes online content specifically to enhance the relevance and accuracy of search responses.	Disabling Claude-SearchBot on your site prevents our system from indexing your content for search optimization, which may reduce your site's visibility and accuracy in user search results.

Tableau des [robots moissonneurs d'Anthropic](#) et leurs usages

Les tags <meta>

Il est également possible de s'opposer par l'insertion de tags <meta> dans l'en-tête du code HTML d'une page web afin que les robots n'indexent pas le contenu d'une page donnée (argument « NOINDEX ») et qu'il ne scanne pas les liens de la page (argument « NOFOLLOW »).

```
<html>
<head>
<title>...</title>
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
</head>
```

Exemple de l'utilisation de la balise HTML <meta> pour s'opposer au moissonnage

Cette méthode a le désavantage de s'appliquer sans distinction entre les robots. Des paramètres plus fins des autorisations peuvent être souhaités dans certains cas.

Des alternatives émergentes, spécifiques au moissonnage

Le principal défaut du fichier « robots.txt » réside dans l'absence de catégorisation fine des robots.

En effet, les règles d'autorisation ou d'interdiction s'appliquent soit à l'ensemble des robots via le champ « **User-agent** : * », soit à un certain nombre d'agents qui doivent être nommés précisément dans le fichier. Il n'est donc pas possible de distinguer différentes classes de robots, en séparant ceux utilisés pour l'indexation par des moteurs de recherche de ceux qui servent à la collecte de données en vue d'entraîner des modèles de fondation.

Ainsi, des solutions alternatives émergent, pour compléter le protocole « robots.txt » ou pour donner des droits d'accès spécifiques au moissonnage pour l'entraînement de modèles d'IA.

Le protocole « ai.txt »

Le protocole « ai.txt », proposé par [l'entreprise Spawning](#), reprend la structure des fichiers « robots.txt ». Il se place au même endroit dans le code source d'une page web et est accessible publiquement en ajoutant « /ai.txt » à la racine du site internet.

Cependant, il s'adresse spécifiquement aux robots qui moissonnent les pages internet à des fins d'entraînement de modèles d'IA. Plutôt qu'un fonctionnement par page web, le fichier

« ai.txt » permet d'interdire la collecte par type de fichier (texte, image, audio, vidéo, code). Un générateur de fichiers « ai.txt » est disponible sur le site internet de [Spawning](#).

```
# Spawning AI
# Prevent datasets from using the following file types

User-Agent: *
Disallow: *.txt
Disallow: *.pdf
Disallow: *.doc
Disallow: *.docx
Disallow: *.odt
Disallow: *.rtf
Disallow: *.tex
Disallow: *.wks
Disallow: *.wpd
Disallow: *.wps
Disallow: *.html
Disallow: *.bmp
Disallow: *.gif
Disallow: *.ico
Disallow: *.jpeg
Disallow: *.jpg
```

Exemple de fichier "ai.txt" généré sur le site internet de Spawning.

Le protocole TDMRep

Enfin, le protocole communautaire [TDMRep](#) (*TDM Reservation Protocol*) du *World Wide Web Consortium* (W3C), est une norme non officielle. Deux champs régissent ce protocole :

- *tdm-reservation* : ce champ est booléen. Il prend la valeur 0 lorsque l'éditeur du site internet autorise le moissonnage de ses données. Lorsqu'il prend la valeur 1, les droits sont réservés, et il faut alors se conformer à la politique de moissonnage du site internet.
- *tdm-policy* : ce champ **optionnel** permet d'indiquer un lien vers la politique de moissonnage du site internet. Elle peut être au format texte au sein d'un fichier HTML (lisible par un humain) ou prendre la forme d'un dictionnaire au format JSON, lisible par une machine. Ce paramètre est optionnel. S'il n'est pas spécifié et que le champ *tdm-reservation* vaut 1, cela signifie que la collecte est *a priori* interdite.

Le protocole TDMRep permet le respect des directives de trois manières différentes :

1. Par l'intégration d'un fichier « **tdmrep.json** » dans le code source du site internet, qui doit être accessible publiquement en ajoutant « /tdmrep.json » à l'URL racine du site internet. Ce fichier prend la forme d'une liste de dictionnaire JSON. Pour chacun d'entre eux, trois attributs doivent figurer :
 - a. *location* : ce dernier précise le répertoire du site internet (page ou ensemble de pages web) concerné par l'application des droits
 - b. *tdm-reservation*,
 - c. *tdm-policy* (optionnel).

```
[
  {
    "location": "/directory-a/",
    "tdm-reservation": 1
  },
  {
    "location": "/directory-b/html/",
    "tdm-reservation": 1,
    "tdm-policy": "https://provider.com/policies/policy.json"
  },
  {
    "location": "/directory-b/images/*.jpg",
    "tdm-reservation": 0
  }
]
```

Exemple type d'un fichier "tdmrep.json" tiré de la [documentation du protocole](#).

2. Par l'inclusion de champs dans les en-têtes de réponse HTTP. Lorsque le contenu d'une page web est envoyée à un utilisateur le protocole propose d'intégrer, dans les en-têtes http qui complètent le contenu du site internet, les paramètres *tdm-reservation* et *tdm-policy* pour préciser les droits de collecte de chaque requête effectuée par le client.
 - a. **Note** : la documentation du protocole précise que c'est la méthode à favoriser car elle est déjà intégrée dans [l'API Spawning](#). Elle permet la création de bases de données par la collecte de données en ligne dans le respect des protocoles d'opposition, dont celui-ci.

```
HTTP/1.1 200 OK
Date: Wed, 14 Jul 2021 12:07:48 GMT
Content-type: text/html
tdm-reservation: 1
tdm-policy: https://provider.com/policies/policy.json
```

Exemple d'insertion des champs "tdm-reservation" et "tdm-policy" dans l'en-tête HTTP des réponses du serveur envoyées au client. Tiré de la [documentation du protocole](#).

3. Comme pour les fichiers « robots.txt », il est possible d'inclure ces champs directement **dans le code HTML** ou **dans des fichiers EPUB** (abréviation de « *Electronic PUblication* », format de fichier standardisé conçu pour la diffusion d'e-books) en utilisant la balise <meta>.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="tdm-reservation" content="1">
    <meta name="tdm-policy" content="https://provider.com/policies/policy.json">
    <title>Document title</title>
  </head>
  <body>
    ...
    <!-- body content -->
    ...
  </body>
</html>

```

```

<package prefix="tdm: http://www.w3.org/ns/tdmrep#" ...>
  <metadata ...>
    <dc:title>Document title</dc:title>
    <meta property="tdm:reservation">1</meta>
    <meta property="tdm:policy">https://provider.com/policies/policy.json</meta>
  </metadata>
</package>

```

Exemple d'usage des balises `<meta>` dans des fichiers HTML d'abord, puis EPUB, pour appliquer le protocole TDMRep.

Discussion sur la portée de ces méthodes

Une intégration inégale

Le protocole « robots.txt » reste le protocole le plus utilisé à ce jour pour encadrer les pratiques de robots moissonneurs. D'après Common Crawl, [30 millions des 37 millions de domaines présents dans leur crawl de Novembre/Décembre 2023 disposent d'un fichier « robots.txt »](#) (soit environ 81%). A l'inverse, par exemple, moins de 0,002% des sites internet ont intégré le protocole TDMRep en cumulant la présence des champs *tdm-reservation* et *tdm-policy* dans les en-têtes de requête HTTP et dans les métadonnées de sites internet. D'autant que l'adoption concernerait principalement des éditeurs de sites [provenant d'Europe de l'Ouest et particulièrement la France](#), avec 134 des 250 domaines les plus utilisés qui en seraient équipés. Cette tendance peut s'expliquer par le fait que cette norme a été proposée par l'ONG française EDRLab.

La tenue d'un fichier « robots.txt » reste donc la solution la plus largement répandue pour s'opposer au moissonnage d'un site internet. Elle est également celle qui est la plus susceptible d'être respectée par les robots moissonneurs. Néanmoins, il reste utile d'envisager l'implémentation cumulative d'autres méthodes dans la perspective d'un accroissement de leur notoriété, nécessairement progressive, mais qui pourrait faciliter à terme l'exercice des droits dans le contexte du moissonnage.

Des mesures aux garanties limitées

Le défaut inhérent aux solutions déclaratives est qu'elles ne sont pas contraignantes. Si certaines ont une valeur légale qui rend illégitime la collecte sur le site internet donné, elles peuvent ne pas être respectées. La prise en compte de ces mesures relève de la confiance en les acteurs qui développent des systèmes d'IA, parfois affaiblie par le manque de transparence sur la constitution des bases de données d'entraînement des modèles, ainsi que par des manquements identifiés. En effet, d'après l'entreprise [Cloudflare](#) ou encore le [média Wired](#), Perplexity aurait par exemple collecté massivement des données en contournant volontairement les fichiers « robots.txt » des sites internet concernés. Cette perte de confiance des éditeurs de site web provoque un risque de fermeture d'internet, comme le décrit le [PEReN](#), par la mise en place de mesures qui limitent l'accès public à leur contenu.

Pour information, l'entreprise Spawning met à disposition la plateforme [Have I Been Trained](#) qui permet de déterminer si un site internet fait partie de bases de données d'entraînement connues.

L'expérience utilisateur reste intacte

L'avantage majeur des méthodes déclaratives est qu'elles n'ont aucun impact sur le parcours utilisateur humain, qui n'est pas modifié par la mise en place de ces mesures. Elles ne nécessitent pas non plus la collecte de données particulières sur l'utilisateur du site internet pour être fonctionnelles, contrairement à la mise en œuvre de solutions bloquantes décrites dans l'article suivant. Nous verrons également que certaines de ces méthodes entraînent un ralentissement de l'expérience utilisateur à qui l'on demande de prouver qu'il est bien un humain.

S'opposer en bloquant l'accès au site internet

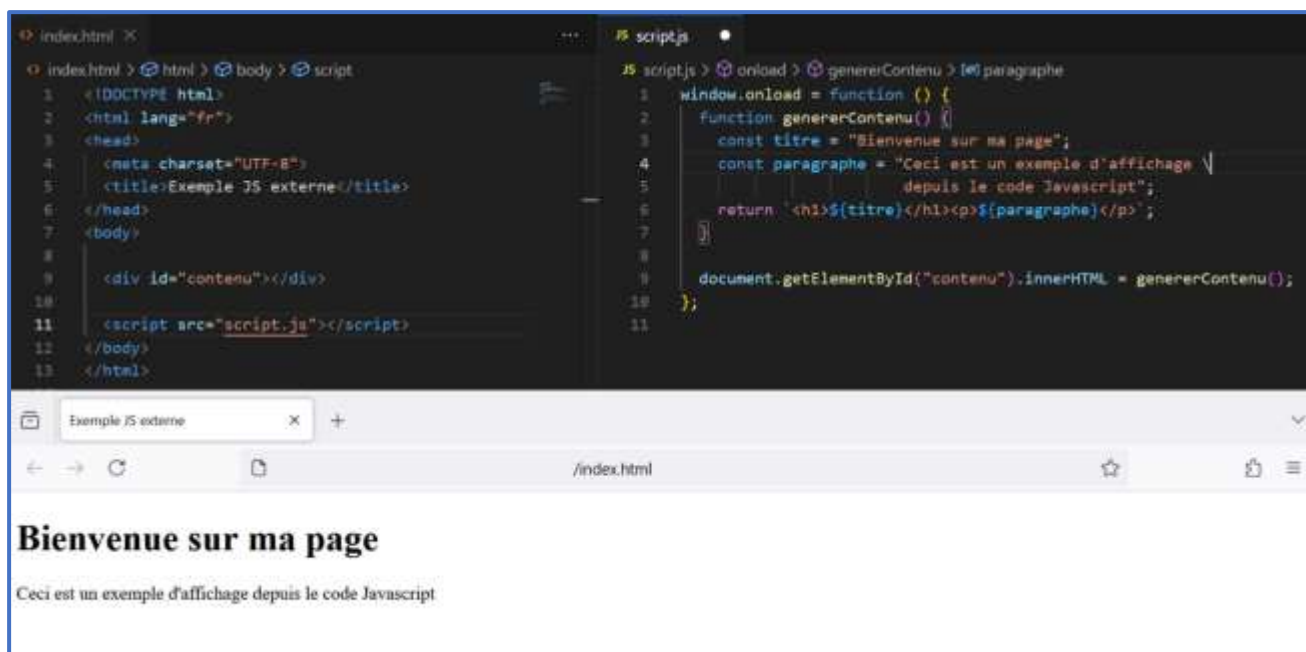
L'article précédent examinait comment s'opposer au moissonnage par des mentions dans les conditions générales d'utilisation ou par des protocoles qui permettent de déclarer des droits d'accès lisibles par une machine. Cependant, un éditeur de site internet peut souhaiter, en complément de ces mesures non bloquantes pour un robot moissonneur, mettre en place des mesures de détection et de blocage de ces robots, au travers de méthodes bloquantes.

Le *Dynamic Page Loading* pour développer le site internet

Il est possible de commencer par une méthode qui ne bloque pas l'accès au site internet aux robots, mais qui peut en cacher le contenu.

Pour développer un site internet, il est désormais très courant d'avoir recours au « Chargement dynamique de page » (ou *Dynamic Page Loading* en anglais). Ce paradigme permet de modifier le contenu d'un site web sans avoir à en recharger systématiquement la page. Pour ce faire, le contenu HTML à afficher est contenu dans des fonctions Javascript qui sont activées par des actions utilisateur (appui sur un bouton, entrée de texte dans des zones dédiées, ouverture d'un menu déroulant, etc.), ou au chargement d'une page lors de l'accès au site internet. Ainsi, la majorité du code HTML est en fait contenu dans les fichiers Javascript du site internet.

Les méthodes de moissonnage les plus simples (et plus particulièrement de *scraping*) se font par le téléchargement des fichiers .html sans exécution du code Javascript, ce qui empêche l'accès aux contenus HTML présent dans les fichiers .js. La commande *curl* utilisable dans un terminal, ou la bibliothèque python *requests* fonctionnent ainsi.



Cet exemple simple illustre comment le contenu HTML du site internet peut s'intégrer dans des fonctions Javascript. Ce contenu n'est alors pas récupérable par une collecte naïve des balises HTML du fichier .html d'un site internet, dont il est absent.

Une mesure à portée limitée

Développer un site internet dynamique présente des avantages pour la gestion de son contenu, la personnalisation et donc d'expérience utilisateur et peut avoir un effet indirect sur le moissonnage des données du site internet. En revanche, utiliser ce paradigme uniquement pour nuire aux robots moissonneurs aura une portée limitée. En effet, de nombreuses librairies de programmation telles que *Selenium* permettent de simuler un navigateur internet et peuvent donc exécuter du code JavaScript. Lorsqu'une fonction Javascript est déclenchée par l'interaction de l'utilisateur avec le site internet, le contenu HTML présent dans les fonctions JavaScript est alors chargé sur la page internet (le fichier .html est modifié en conséquence) et peut donc être collecté.

Mettre en place des restrictions d'accès

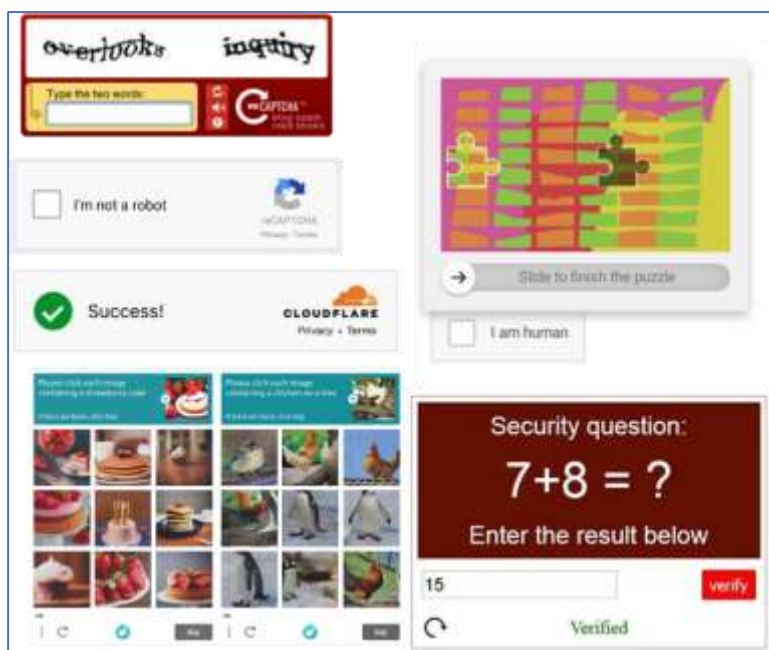
Il est également possible de limiter l'accès à certaines pages en imposant la connexion à un compte utilisateur, avec des méthodes d'authentification. Cette approche a été choisie notamment par [404media](#) après avoir découvert que le contenu de leurs articles était massivement moissonné. Une alternative préférée par d'autres médias est de protéger les pages web par des *paywalls* (littéralement « murs payants »), qui nécessitent de souscrire à un abonnement ou de payer pour accéder à leur contenu.

Ces techniques sont rarement implémentées dans le but initial d'empêcher le moissonnage d'un site internet et peuvent avoir des conséquences sur l'accès à son contenu, dans le cas où l'utilisateur serait obligé et créer un compte et/ou de payer pour accéder à une partie du site internet. Il est toutefois utile de relever l'impact que ces fonctionnalités ont sur l'accès au site internet pour un robot moissonneur.

Utiliser des CAPTCHAS

CAPTCHA est l'acronyme de *Completely Automated Public Turing test to tell Computers and Humans Apart* (qui pourrait être traduit par « un test public complètement automatisé permettant de distinguer les humains des machines »). Lorsqu'un site internet en est équipé, un CAPTCHA prend la forme d'une fenêtre qui apparaît lorsqu'un utilisateur tente d'y accéder. Son rôle est de déterminer si l'utilisateur qui tente de se connecter à un site internet donné est humain, ou s'il s'agit d'une machine. La vérification passe par la résolution d'énigmes qui peuvent prendre plusieurs formes (voir des exemples de CAPTCHAS ci-dessous) :

- Une analyse d'**images** : trouver les images répondant à une consigne donnée, résoudre un puzzle, orienter un objet 3D dans la bonne direction, ...
- La lecture d'un **texte** déformé (suite de chiffres, de lettres, de mots, etc.) qu'il faut écrire,
- La transcription d'un échantillon audio (mot, suite de chiffres, etc.),
- La résolution d'**énigmes mathématiques** : des questions mathématiques simples sont posées en langage naturel, et elles doivent être résolues,
- Etc.



Exemples de CAPTCHAS qu'on peut rencontrer sur internet. Il y en a beaucoup d'autres, qui peuvent prendre des formes variées.

Ces tâches, réalisables par des humains, le sont plus difficilement par une machine, et peuvent en empêcher l'accès au site internet.

L'impact sur l'expérience utilisateur

Le désavantage majeur des CAPTCHAs est leur impact sur l'expérience utilisateur humaine. En effet, leur implémentation systématisée implique pour un utilisateur de devoir résoudre ces énigmes à chaque fois qu'il visite un nouveau site internet. [D'après le site internet BuiltWith](#), près d'un tiers des sites internet parmi le top le million des plus visités sont équipés d'une technologie CAPTCHA. [Une étude de Stanford](#) montrait en 2010 que la résolution de CAPTCHAs visuels prenait en moyenne 9,8 secondes et 28,4 secondes pour un CAPTCHA audio.

Des alternatives moins chronophages, mais plus questionnables du point de vue de la protection des données personnelles, existent, pour lesquelles la résolution d'énigme est soit complètement abandonnée, soit déclenchée en cas de comportement suspect. Lors du premier accès d'un utilisateur sur un site internet, la technologie CAPTCHA déclenche un suivi de son comportement (fréquence des requêtes, temps passé par page et type de pages visitées, etc.). Les statistiques de l'utilisateur sont ensuite transmises à l'éditeur de site qui peut choisir de le bloquer s'il détecte une activité anormale (c'est ainsi que fonctionne [reCAPTCHA v3](#) de Google). Dans le cas de la version 2 de reCAPTCHA, l'analyse du comportement utilisateur peut déclencher automatiquement l'apparition d'une énigme à résoudre s'il est probable que l'utilisateur soit un robot. D'autres concurrents comme [hCAPTCHA](#) ou [Cloudflare Turnstile](#) proposent des solutions similaires.

Il s'agit en fait de combiner les CAPTCHAs traditionnels avec d'autres méthodes de détection de robots, qui se rapprochent de celles utilisées par des CDNs (voir plus loin). En effet, un éditeur de site peut mettre en place ce **suivi des actions utilisateur** et déterminer si elles sont inhabituelles ou anormales (fréquence et quantité de requêtes très élevées, par exemple). Dès lors, il peut choisir de bloquer l'adresse IP concernée pour lui empêcher l'accès au site.

Une méthode loin d'être infallible

Certes, les CAPTCHAs sont un rempart au moissonnage, mais en plus de poser des questions d'accessibilité aux sites internet qui en sont équipés, ils peuvent être contournés. Les modèles d'IA à usage général peuvent résoudre des CAPTCHAs de plus en plus facilement. Un exemple visuel est la [nouvelle IA agentique d'OpenAI](#) qui est capable de remplir les CAPTCHAs rencontrés seule lorsqu'elle répond à une requête de l'utilisateur. Il est également possible d'avoir recours à des [fermes à CAPTCHAs](#), qui permettent la location de main d'œuvre à faible coût vers laquelle sont redirigés tous les CAPTCHAs rencontrés pas des robots moissonneurs. Ils sont ensuite résolus manuellement, puis le robot poursuit sa navigation sur le site internet souhaité.

D'autre part, lorsque l'apparition d'un CAPTCHA n'est pas systématique, mais déclenchée par un comportement suspicieux, il suffit de faire en sorte que le robot soit toujours considéré par le site internet comme un nouvel utilisateur, avec des méthodes de rotation d'adresse IP, par exemple. Le CAPTCHA n'est alors jamais déclenché.

Examiner les requêtes HTTP du client

Une source d'information très riche pour identifier des robots moissonneurs est l'examen des requêtes HTTP envoyées par le client (l'utilisateur naviguant sur Internet) au serveur web qui héberge un site internet. Ces derniers communiquent en suivant des protocoles standardisés, organisés en plusieurs couches : applicative (protocole HTTP), de sécurité (protocole TLS), de transport (protocoles TCP ou QUIC), etc. En examinant les en-têtes des requêtes HTTP et les propriétés du navigateur utilisé, il est possible de vérifier si l'utilisateur est un robot moissonneur, par le calcul des « empreintes » des requêtes à chaque couche du protocole et en les comparant avec des empreintes connues. On parle de *fingerprinting*.

Une **empreinte** est une chaîne de caractères encodée qui résume un ensemble caractéristique de paramètres observés lors d'une communication réseau. Elle permet d'identifier la machine à l'origine d'une requête et de déduire certaines informations sur ses attributs techniques. L'analyse d'empreintes permet par exemple de détecter l'utilisation d'un VPN ou de programmes informatiques pour se connecter au site internet (Python, Javascript). Ces informations sont des indices qui peuvent indiquer que le client qui se connecte est un robot.

Une fois l'empreinte d'un client déterminée à une couche donnée du protocole de communication réseau, elle est comparée à des empreintes connues. Si elle fait partie d'empreintes considérées comme autorisées à accéder au site internet, la connexion peut s'établir. Sinon, l'adresse IP est bloquée. Il est également utile de **comparer les empreintes entre les couches du protocole HTTP**, pour vérifier la cohérence des informations entre les couches. Des informations incohérentes entre elles peuvent être un indice de la présence d'un robot, qui a tenté de simuler le comportement d'un utilisateur humain.

Ainsi, la [bibliothèque JA4+](#) est une bibliothèque open source qui dispose de plusieurs méthodes permettant de déterminer les empreintes TCP, TLS et HTTP d'un client. A chaque couche, les données obtenues sont triées, concaténées, hachées puis tronquées, afin d'obtenir une empreinte de taille identique pour chaque client.

Pour en savoir plus sur le *fingerprinting*, vous pouvez consulter [notre article dédié](#).

Utiliser des pare-feux ou des CDNs

Plutôt que d'implémenter et de maintenir ces solutions « à la main », il est possible de faire appel à des pare-feux, des CDNs (*Content Delivery Network*), qui sont des outils complémentaires permettant de filtrer et de réguler le trafic réseau. Ils implémentent en général des outils de détection de robots. La bibliothèque JA4+ fournit par exemple une [liste de services](#) qui implémentent ses empreintes (dont Wireshark, outil utilisé dans cet article pour visualiser des requêtes HTTP ; vous en avez donc peut-être repéré des empreintes sur certaines images explicatives).

[Cloudflare](#), par exemple, utilise les empreintes JA4 pour détecter des robots et explique dans un article que l'empreinte TLS est particulièrement efficace et robuste. A noter que l'entreprise explore également des alternatives, comme l'implémentation de **labyrinthes de pages web** : plutôt que de bloquer systématiquement tous les robots, il peut suffire de les détecter, puis d'ajouter des liens invisibles qui les enferment dans une suite de pages infinie, les rendant inefficaces.

Le [PEReN donne quelques compléments](#) d'informations au sujet des pare-feux et des CDNs dans leur note dédiée.

Des méthodes imparfaites et parfois intrusives

Dans le cas de robots moissonneurs qui respectent les directives déclaratives présentées dans la section précédente et qui sont identifiés avec un nom publiquement accessible, le recours à des mesures bloquantes n'est en principe pas nécessaire. Elles peuvent cependant les compléter, car les robots moissonneurs seront aisément identifiés et pourront donc être bloqués. Elles sont aussi une garantie supplémentaire pour les éditeurs de site qui n'ont pas de moyen de vérifier que les directives des fichiers normatifs sont respectées.

Toutefois, elles permettent surtout le blocage des robots moissonneurs qui s'affranchissent des règles de moissonnage renseignées par les éditeurs de sites internet qui pourraient souhaiter collecter le contenu disponible en passant inaperçu, par simulation des navigateurs existants et d'un comportement humain crédible. D'autre part, les méthodes présentées servent plus généralement à la détection de robots malveillants et peuvent donc améliorer la sécurité globale. Toute méthode reste cependant faillible, et il existe une multitude de contenus en ligne expliquant comment contourner les méthodes de blocage des robots, en imitant toujours plus fidèlement des utilisateurs humains.

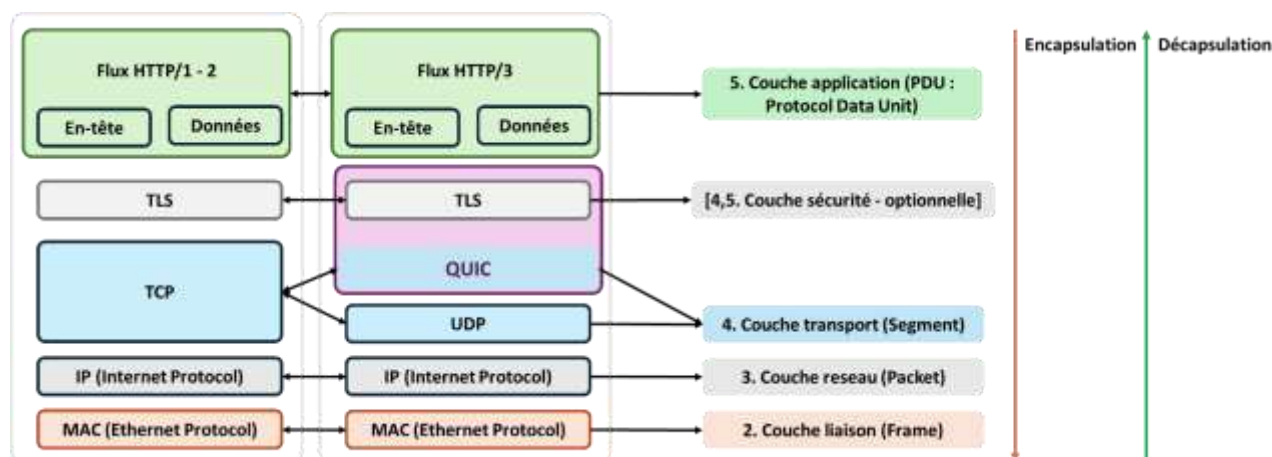
D'autre part, comme évoqué plus haut, ces méthodes bloquantes (CAPTCHAs, paywalls, authentification) peuvent nuire à l'expérience utilisateur en rendant moins direct l'accès au contenu du site.

Article bonus : Le « *fingerprinting* »

Le terme de *fingerprinting* (ou « empreinte ») désigne une technique permettant de caractériser le client qui émet une requête à un serveur à l'aide d'une chaîne de caractères unique. Une empreinte permet en effet d'identifier la machine à l'origine d'une requête et de déduire certaines informations sur ses attributs techniques : navigateur utilisé, adresse IP, utilisation ou non d'un VPN, etc. Cette technique est très utile pour filtrer le trafic web d'un site internet et bloquer les robots qui pourraient s'y connecter, à des fins de moissonnage ou pour conduire des attaques de type DDOS, par exemple.

Cet article commencera par un rappel sur les protocoles de communication web, avant d'expliquer quelles informations peuvent servir à l'identification du client qui accède à un site internet à partir de l'analyse de chaque couche de ces protocoles et de leurs empreintes respectives.

Rappel sur les protocoles de communication web



Rappel des protocoles de communication utilisés sur le World Wide Web Certains sites internet utilisent les trois versions des protocoles HTTP, d'où l'intérêt de les étudier.

Les protocoles de communication Web

Lors de l'envoi d'une requête, un émetteur encapsule ses données progressivement en respectant les protocoles de chaque couche. Pour commencer, les données de l'émetteur sont divisées en unités de taille fixe, appelées « Protocol Data Units » ou PDUs. A chaque couche, un en-tête est ajouté, précisant les modalités de mise en œuvre du protocole, ainsi que les informations d'adressage qui permettent d'orienter la requête dans la bonne direction pour

qu'elle atteigne son destinataire. Ainsi, le PDU est encapsulé par des informations des couches :

- **Application** : l'en-tête contient le nom de domaine auquel l'utilisateur veut se connecter,
- **Transport** : l'en-tête mentionne le port de l'application émettrice de la requête (un navigateur, par exemple) et le port de destination du récepteur, pour former un « segment » de données ;
- **Sécurité** (le cas échéant) : à cette étape a lieu le chiffrement du PDU envoyé par le client,
- **Réseau** : les adresses IP source et destination sont ajoutées pour former un « paquet » de données,
- **Liaison** : adresses MAC source et destination complètent et forment la requête complète, appelée alors « frame ».

Une requête prend la forme d'une suite de nombres hexadécimaux (une fois déchiffrée), que le récepteur décapsule ensuite en sens inverse.

Avant de pouvoir émettre ces requêtes, un client et un serveur doivent **établir une connexion**, qui advient au niveau des **couches transport et sécurité**. Les premiers échanges ne contiennent pas encore de données et se limitent à des segments permettant au client et au serveur de s'identifier et mettre en œuvre leur communication. C'est à ce moment qu'il est possible pour le serveur de récolter des informations sur le client, et donc de détecter s'il s'agit d'un robot ou non. Une fois la connexion établie, les en-têtes **de la couche application** sont également utiles à cette fin.

HTTP/2 et HTTP/3, quelles différences ?

[Jusqu'à la version 2 du protocole HTTP, publiée en 2015](#), les protocoles de communication web reposent sur **une architecture TCP/IP**. Le client et le serveur établissent une connexion initiale via la couche transport TCP avant d'échanger des données. Les requêtes doivent être envoyées et reçues dans l'ordre, avec un accusé de réception garantissant la bonne livraison ou, en cas d'erreur, une demande par le récepteur du renvoi des données corrompues. Il s'agit de protocole orienté connexion. Ces protocoles sont également utilisés dans des applications comme l'envoi d'e-mails. Ils offrent une meilleure fiabilité, mais au prix d'une latence accrue.

HTTP/3, proposé en 2018, se distingue par l'utilisation du **protocole QUIC, reposant sur UDP** comme couche de transport. Le protocole UDP ne garantit ni l'ordre de réception des paquets ni leur accusé de réception au niveau du protocole lui-même. Ce protocole n'est pas fiable malgré le gain en latence. La [couche QUIC](#) permet de fiabiliser les requêtes et de s'assurer qu'elles sont toutes transmises sans erreur, dans le bon ordre. Des mesures de contrôle de congestion et de gestion des paquets perdus plus efficaces et une identification plus rapide entre client et serveur permettent de combiner une connexion fiable et plus rapide que pour l'architecture TCP/IP.

Dans la suite sont détaillées les informations qui peuvent être extraites à chaque couche du protocole de communication HTTP. La partie suivante détaille comment ces informations peuvent être utilisées pour détecter des robots moissonneurs.

Informations de la couche transport TCP

Pour les protocoles HTTP reposant sur la structure TCP/IP, l'identification entre le client et le serveur se fait par une poignée de main TCP à trois voies ([TCP three-way handshake](#)) : le client envoie un premier paquet SYN (pour *synchronized*) au serveur, auquel ce dernier répond par un paquet SYN-ACK (*synchronize, acknowledge*). Ils échangent également leurs paramètres de communication souhaités. Enfin, le client envoie un accusé de réception ACK au serveur (pour *acknowledge*). La connexion est établie.

Dès le premier échange d'informations avec le client et l'envoi du paquet SYN, il est possible de **connaître son adresse IP** puis d'effectuer alors un premier contrôle de la [réputation de l'adresse IP](#) au moyen de bases de données dédiées. Il est également possible de mettre en place une **liste noire** d'adresses IP bloquées et de vérifier que la tentative de connexion ne provient pas d'une adresse déjà référencée dans cette liste. D'autres techniques comme le blocage d'adresses IP à l'origine de requêtes anormalement rapides et nombreuses peuvent être mises en place dès que cette dernière est connue.

Il est également possible d'analyser le contenu du segment TCP de la requête SYN pour caractériser le client grâce aux champs :

- *Window Size* : c'est la taille maximale de données que peut accepter le receveur en mémoire cache avant d'envoyer un accusé de réception (ACK) à l'émetteur,
- *Maximum Segment Size (MSS)* : c'est la taille maximale des données contenues dans la couche HTTP d'une unique requête pouvant être envoyé par le client,
- *Window scale* : c'est un facteur multiplicateur qui permet d'augmenter la taille de la fenêtre en la multipliant par ce facteur,
- Etc.

```

> Frame 28: 60 bytes on wire (528 bits), 60 bytes captured (528 bits) on interface (Device:WIF_79AC98F-80CF-4248-9F94-00072884D775), 16 B
Ethernet II, Src: Intel_4e:00:00 (ac:84:77:aa:00:00), Dst: Legosendras_08:0f:2c (08:00:ad:08:0f:2c)
Internet Protocol Version 4, Src: 192.168.1.11, Dst: 23.192.237.211
  0200 ..... = Version: 4
  .... 0200 = Header Length: 20 bytes (3)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 52
  Identification: 0x0ad5 (34924)
  0200 ..... = Flags: 0x0, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: TCP (6)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.1.11
  Destination Address: 23.192.237.211
[Stream index: 4]
Transmission Control Protocol, Src Port: 55487, Dst Port: 443, Seq: 8, Len: 0
  Source Port: 55487
  Destination Port: 443
  [Stream index: 7]
  [Stream Packet Number: 1]
  [Conversation completeness: Complete, WITH_DATA (32)]
  [TCP Segment Len: 0]
  Sequence Number: 8 (relative sequence number)
  Sequence Number (raw): 143029959
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  0200 ..... = Header length: 32 bytes (8)
  0200 ..... = Flags: 0x001 (SYN)
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0xc777 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (12 bytes), Maximum segment size, No-Operation (NOP), window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
    TCP Option - Maximum segment size: 1460 bytes
    TCP Option - No-Operation (NOP)
    TCP Option - window scale: 8 (multiply by 256)
    TCP Option - No-Operation (NOP)
    TCP Option - No-Operation (NOP)
    TCP Option - SACK permitted
  [Timestamps]

```

Exemple d'un paquet SYN envoyé par le client, contenant son adresse IP ainsi que des options qui le caractérisent.

L'agrégation de ces informations permet [d'identifier le système d'exploitation ou l'appareil du client, ainsi que l'usage éventuel de proxies intermédiaires, de VPNs, etc.](#)

Mise en place du protocole de sécurité

Le protocole **TLS** (*Transport Layer Security*) assure la confidentialité et l'intégrité des données échangées sur le réseau. Dans le cas d'un protocole de type TCP/IP, la poignée de main TCP est suivie par une poignée de main TLS, qui permet la sécurisation des échanges. Le client envoie une première requête non chiffrée *Client Hello* informant le serveur de la version TLS la plus récente dont il dispose (entre TLSv1.2 et TLSv1.3), des algorithmes de chiffrement qu'il peut mettre en œuvre (soit l'ensemble des algorithmes que le client et le serveur peuvent utiliser pour établir la connexion sécurisée : échange de clés, chiffrement des données, etc.). Les communications suivantes permettent notamment d'échanger les clés de chiffrement avant l'échange de données entre les deux machines. En fonction de la version du protocole de sécurité ([TLSv1.2](#) et [TLSv1.3](#)), le nombre de messages échangés en clair varie.

La première requête du client au serveur pour établir le protocole de chiffrement n'étant pas chiffrée. Elle permet donc d'établir pour un client donné :

- Les algorithmes de chiffrements (*ciphers*) pris en charge ainsi que leur nombre,
- Les valeurs des autres champs « Extension » et leur nombre.

L'agrégation de ces informations permet notamment de caractériser la bibliothèque TLS, spécifique à l'application utilisée pour se connecter. Ainsi, il est très probable que des programmes écrits en Python aient la même empreinte TLS, et des programmes spécifiques tels que des clients VPN, un client qui utilise Windows auront respectivement un profil TLS unique.

The screenshot shows a detailed view of a TLS v1.3 Client Hello handshake. Key fields highlighted include:

- Handshake Type:** Client Hello (1)
- Version:** TLS 1.3 (0x0303)
- Session ID Length:** 32
- Session ID:** c0f7d0d8b9946a209430f0c5591623e8fe3c1249a8979da9c451c07e6965
- Cipher Suites Length:** 32
- Cipher Suites (16 suites):** A list of supported cipher suites including TLS_AES_128_GCM_SHA256, TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256, etc.
- Compression Methods Length:** 1
- Extensions Length:** 1907
- Extensions:** A list of extensions including supported_versions (TLS 1.3, TLS 1.2), encrypted_client_hello, psk_key_exchange_modes, renegotiation_info, extended_master_secret, server_name (www.bing.com), and pre_shared_key.

Exemple de segment "Client Hello" lors d'une poignée de main TLS v1.3.

Poignée de main QUIC

Comme expliqué précédemment, le protocole HTTP/3 [remplace le protocole de transport TCP](#) par un protocole QUIC/UDP. Une des différences est le remplacement d'un processus à deux poignées de main (TCP puis TLS, pour établir la connexion et son chiffrement) par un processus à unique poignée de main. Elle combine identification du client et chiffrement de la connexion. Ainsi, alors que la poignée de main TCP, ainsi que le début de la poignée de main TLS sont en clair, le chiffrement intervient plus tôt lorsque le protocole QUIC/UDP est utilisé, rendant plus difficile l'extraction d'informations. Cependant, et comme précédemment, le premier message envoyé par le client au serveur reste envoyé en clair. La récolte d'informations est donc similaire à celle présentée plus haut pour les protocoles TCP/IP.

Informations obtenues dans les en-têtes HTTP

Une fois le client et le serveur identifiés et la connexion établie entre les deux, ils commencent à échanger des données, contenues dans la couche application de la requête. Dans le cas du web, le protocole utilisé est le protocole HTTP. Chaque *PDU* (Protocol Data Unit) contenue dans la requête est accompagnée d'un en-tête qui permet de déceler des informations à propos du client (voir un exemple ci-dessous).

```
GET /home.html HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0)
Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/testpage.html
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT
If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"
Cache-Control: max-age=0
```

Exemple des informations contenues dans la couche applicative du protocole HTTP/1.1

Les [champs de cet en-tête](#) peuvent être analysés pour détecter des robots :

- **User-Agent** : ce paramètre permet d'identifier l'application, le navigateur ou de manière général l'outil utilisé pour interagir avec le serveur. Ce paramètre peut permettre à lui seul d'identifier un robot, dont le nom peut figurer dans la valeur de ce champ,
- **Accept, Accept-Language, Accept-Encoding** : ces champs indiquent les types de fichier, les langues et encodages que le client peut traduire,
- **Connection** : ce paramètre précise si la connexion aux serveurs du site internet doit être maintenue une fois la requête traitée. La valeur **keep-alive** indique que oui.
- **If-Modified-Since** permet d'obtenir des informations sur le fuseau horaire du client et donc sur son emplacement,
- Etc.

Un robot peut être détecté par l'absence de champs traditionnellement renseignés par des navigateurs (un champ « **Accept-Language** » manquant est caractéristique d'un robot), des valeurs anormales ou en repérant le nom du robot dans le champ « User-Agent ». Certaines entreprises qui utilisent des robots moissonneurs rendent public la valeur de ce champ afin de permettre un blocage facile (de manière similaire au remplissage de champ « **User-agent** » dans les fichiers « robots.txt »).

USER AGENT	DESCRIPTION & DETAILS
OAI-SearchBot	<p>OAI-SearchBot is for search. OAI-SearchBot is used to link to and surface websites in search results in ChatGPT's search features. It is not used to crawl content to train OpenAI's generative AI foundation models. To help ensure your site appears in search results, we recommend allowing OAI-SearchBot in your site's robots.txt file and allowing requests from our published IP ranges below.</p> <p>Full user-agent string will contain ; OAI-SearchBot/1.0; +https://openai.com/searchbot</p> <p>Published IP addresses: https://openai.com/searchbot.json</p>
ChatGPT-User	<p>ChatGPT-User is for user actions in ChatGPT and Custom GPTs. When users ask ChatGPT or a CustomGPT a question, it may visit a web page with a ChatGPT-User agent. ChatGPT users may also interact with external applications via GPT Actions. ChatGPT-User is not used for crawling the web in an automatic fashion, nor to crawl content for generative AI training.</p> <p>Full user-agent string: Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko); compatible; ChatGPT-User/1.0; +https://openai.com/bot</p> <p>Published IP addresses: https://openai.com/chatgpt-user.json</p>
GPTBot	<p>GPTBot is used to make our generative AI foundation models more useful and safe. It is used to crawl content that may be used in training our generative AI foundation models. Disallowing GPTBot indicates a site's content should not be used in training generative AI foundation models.</p> <p>Full user-agent string: Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko); compatible; GPTBot/1.1; +https://openai.com/gptbot</p> <p>Published IP addresses: https://openai.com/gptbot.json</p>

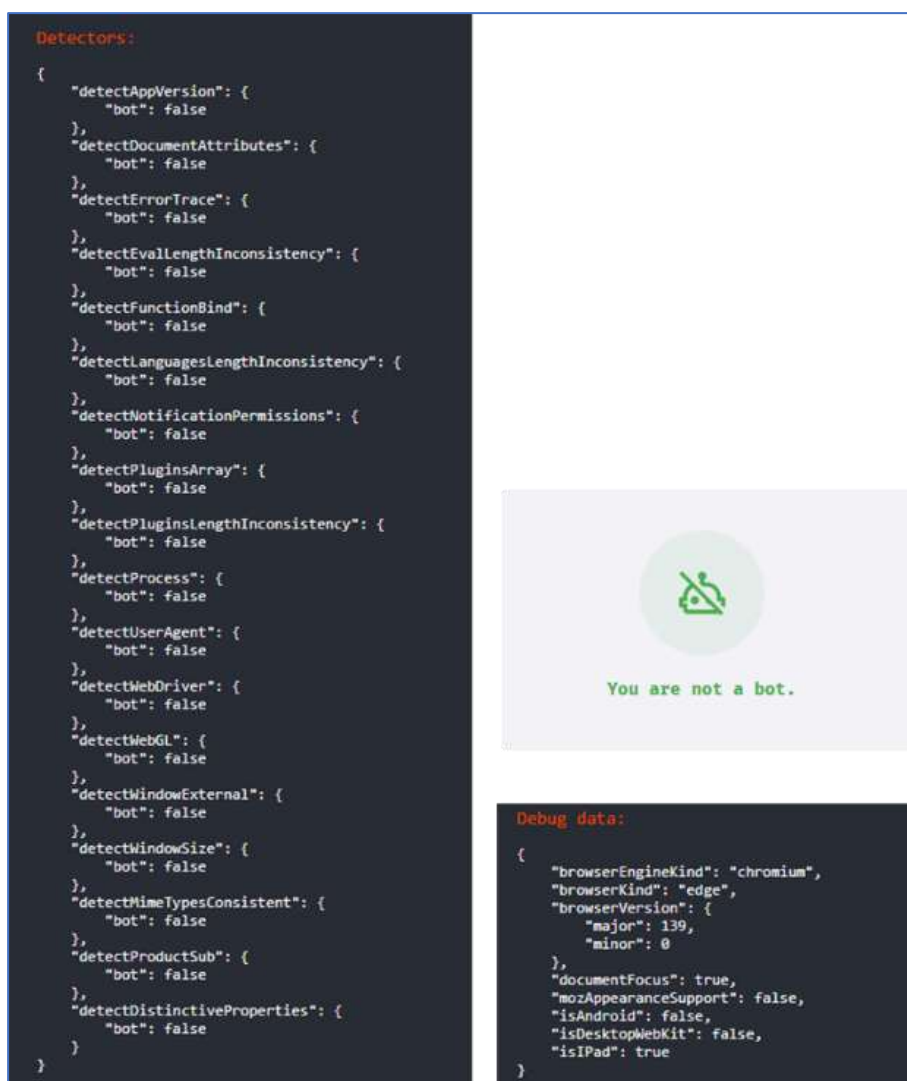
Exemple des champs "User-agent" des robots moissonneurs d'OpenAI. L'entreprise met également à disposition la liste de leurs adresses IP. Il est donc possible de filtrer ces robots par analyse de l'en-tête HTTP des requêtes et par l'adresse IP du client.

Identifier le navigateur par l'exécution côté client

Il est également possible, une fois la connexion établie et l'envoi par le serveur du code source de la page internet à laquelle le client souhaite accéder, d'obtenir des informations sur son navigateur. Le code Javascript de la page web peut être utilisé pour collecter les informations suivantes :

- Versions du système d'exploitation et du navigateur du client,
- Taille d'écran,
- Fuseau horaire,
- Polices installées,
- Technologie de rendu graphique (« WebGL ») utilisée,
- Les plug-ins ou extensions installées,
- etc.

Cette collecte plus intrusive d'informations à propos du client permet de l'identifier de manière presque unique. L'absence de plugins installés, des technologies de rendu graphique génériques simplifiées, ou l'absence de certains champs peuvent alerter sur la nature robotique de l'utilisateur. La bibliothèque FingerprintJS propose d'ailleurs un [outil](#) permettant de vérifier à partir de leurs critères si l'utilisateur qui se rend sur ce site est susceptible d'être un robot.



Exemple d'informations collectées par la bibliothèque FingerprintJS pour détecter un éventuel robot.

L'accessibilité des informations

Les informations extraites de l'examen des en-têtes TCP et TLS lors de l'établissement de la connexion entre le client et le serveur peuvent être collectées avant même le premier échange de données applicatives. Cela permet, par exemple, d'identifier et de bloquer un robot très tôt dans la communication. Par ailleurs, ces en-têtes étant échangés en clair, leur analyse ne nécessite pas d'être effectuée sur le serveur du site internet : elle peut aussi être assurée par des services tiers, comme les réseaux de diffusion de contenu (CDN) ou les pare-feux applicatifs fonctionnant en mode cloud, qui contribuent à surveiller, filtrer et réguler le trafic d'un site web.

A l'inverse, l'examen d'en-têtes HTTP et des informations sur le navigateur nécessite de pouvoir déchiffrer le contenu des requêtes et doivent donc s'effectuer sur le serveur qui héberge le site internet.

Utiliser ces informations : les empreintes

Afin de mettre à profit les informations qui peuvent être extraites lors de l'établissement de la connexion entre un client et un serveur, en examinant les en-têtes des requêtes HTTP, et les propriétés du navigateur, il est utile d'avoir une approche systématisée. Une manière de le faire est d'avoir recours à des bibliothèques qui permettent de calculer des empreintes.

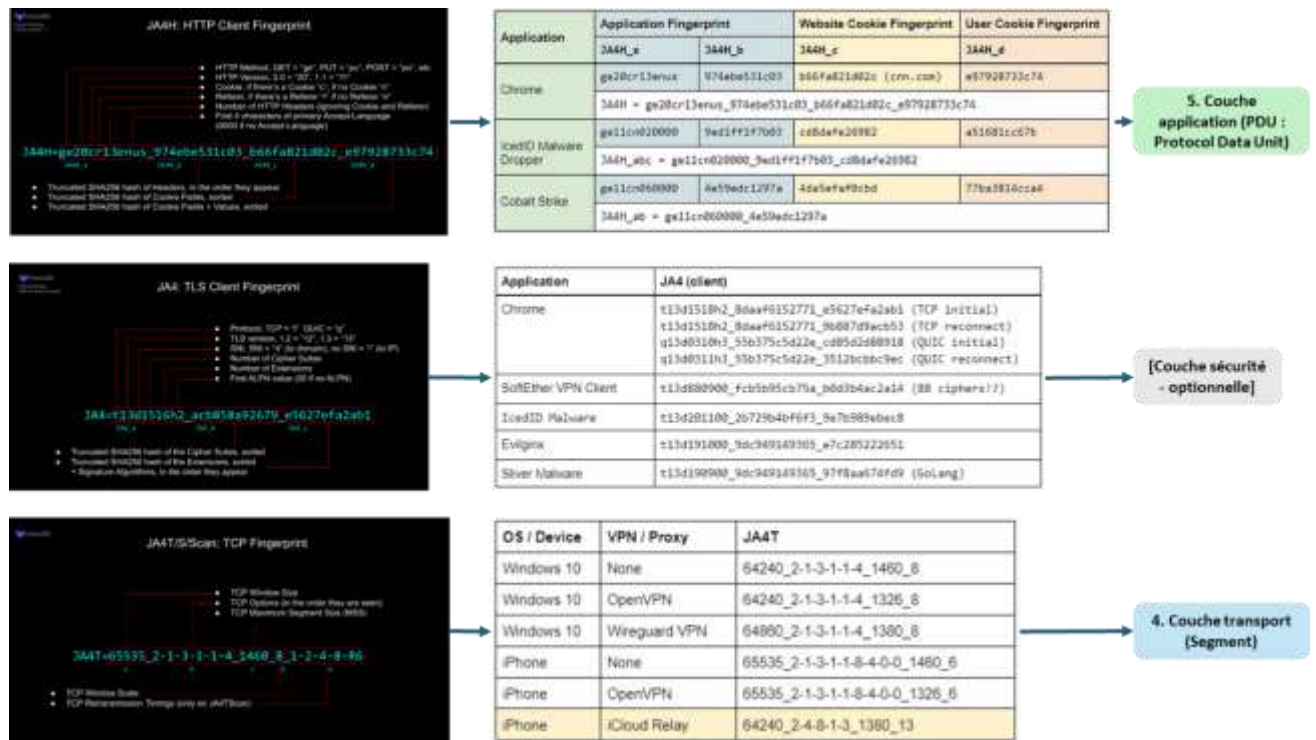
Une **empreinte** est une chaîne de caractères encodée qui résume un ensemble caractéristique de paramètres observés lors d'une communication réseau. Elle permet d'identifier la machine à l'origine d'une requête et de déduire certaines informations sur ses attributs techniques.

Implémentation des bibliothèques d'empreinte [TCP](#), [TLS](#) et [HTTP](#)

La [bibliothèque JA4+](#) est une bibliothèque open source qui dispose de plusieurs méthodes permettant de déterminer les empreintes TCP, TLS et HTTP d'un client. A chaque couche, les données obtenues sont triées, concaténées, hachées puis tronquées, afin d'obtenir une empreinte de taille identique pour tous les clients.

Les informations obtenues à une couche donnée peuvent être enrichies par des informations provenant d'autres couches. Ainsi, l'empreinte TLS, en plus des informations sur la version du protocole du chiffrement, le nombre de suites de chiffrements et d'extensions du client, est complétée par les champs suivants :

- **SNI (*Server Name Identification*)** : le nom de l'hôte auquel il essaye de se connecter,
- **ALPN (*Application-Layer Protocol Negotiation*)** : le protocole utilisé par pour communiquer au niveau de la couche applicative (en pratique, une version du protocole HTTP),
- Si le protocole transport est le protocole TCP ou QUIC.



Description des empreintes de la bibliothèque JA4+ et exemples de clients correspondants.

Ces empreintes s'utilisent ainsi : une fois l'empreinte d'un client déterminée à une couche donnée, elle est comparée à des empreintes connues. Si elle fait partie d'empreintes considérées comme autorisées à accéder au site internet, la connexion peut s'établir. Sinon, l'adresse IP est bloquée. **La bibliothèque JA4+ met pour cela à disposition une [base de données](#)** qui, pour chaque type de client (caractérisé par l'application utilisée, la bibliothèque TLS, le type d'appareil, le système d'exploitation, l'argument « User-Agent » et l'autorité de certification utilisée), fournit les empreintes que sa bibliothèque met à disposition.

L'analyse d'empreintes permet par exemple de détecter l'utilisation d'un VPN ou d'une bibliothèque TLS d'un langage de programmation utilisé pour faire du moissonnage (Python, JavaScript). Ces informations sont des indices qui peuvent indiquer que le client qui se connecte est un robot. Il est également utile de **comparer les empreintes entre elles**, pour vérifier la cohérence des informations entre les couches. En effet, certaines empreintes permettent d'obtenir des informations similaires à propos du client, et dénicher des incohérences entre elles peut être un indice de la présence d'un robot, qui a tenté de simuler un utilisateur humain. Un exemple de présentation de la bibliothèque est donné dans l'article du LINC [S'opposer au moissonnage en bloquant l'accès au site internet](#), ainsi qu'un examen plus fin d'empreintes qui permet de déceler l'usage d'un VPN, et donc potentiellement d'un robot.

```

{
  "application": "Chrome 31.0 ",
  "library": null,
  "device": null,
  "os": "Windows 8 .1",
  "user_agent_string": null,
  "certificate_authority": null,
  "observation_count": 1,
  "verified": true,
  "notes": "Windows 8.1 on chrome 31.0",
  "ja4_fingerprint": "t12d1310sp_0571d07754c8_736b2aled4d3",
  "ja4_fingerprint_string": "t12d1310sp_002f,0032,0033,0035,0039,009c,009e,c009,c00a,c013,c014,c02b,c02f_0005",
  "ja4s_fingerprint": null,
  "ja4h_fingerprint": "gallnn1lenus_fad376c0528a_000000000000_000000000000",
  "ja4x_fingerprint": null,
  "ja4t_fingerprint": null,
  "ja4ts_fingerprint": null,
  "ja4tscan_fingerprint": null
}

```

Exemple d'élément de la base de données pour le navigateur Chrome, version 31.0

Implémentation des bibliothèques d'empreinte navigateur

Concernant l'implémentation d'empreintes navigateur, il existe la bibliothèque [FingerprintJS](#). Elle met à disposition un module spécifique dédié à la détection de robots à partir de son [empreinte navigateur](#).

```

{
  "version": "4.6.2",
  "visitorId": "8b72fee6491c3039da8bada414b388fa", ⓘ
  "confidence": {
    "score": 0.6, ⓘ
  },
  "components": { ... },
}

```

Exemple d'identifiant du client (champ « visitorId » calculé à l'aide de la bibliothèque FingerprintJS, ainsi que la fourniture d'un score de confiance).

Conclusion

Le *fingerprinting* constitue donc un moyen efficace d'identifier le client qui tente de se connecter à un serveur. Bien qu'aucune méthode ne soit totalement infaillible, l'analyse et la comparaison des empreintes issues des différentes couches du protocole HTTP — à la fois avec des empreintes reconnues comme fiables et entre elles afin d'en vérifier la cohérence — permettent de réduire au maximum les accès non autorisés à un site web.

Cette approche est notamment utile pour bloquer les robots moissonneurs (« scrapers ») utilisés pour collecter des données accessibles publiquement en ligne, notamment dans le but d'entraîner des modèles d'intelligence artificielle. Pour en savoir plus, vous pouvez consulter notre série d'articles dédiés.