ARTRAGIS
SITUPHEN
STALONE
GUSTAVI

Zeste de savoir

On a comblé des failles de sécurité sur ZDS

24 octobre 2021

Table des matières

Introduction 1
1. Comment fonctionne ZDS 2
2. La découverte des failles 3
2.1. Liste des failles 3
2.2. Quelques détails 4
3. Communiquer la faille et l'étudier 5
3.1. Les canaux de communication 5
3.2. Suivi de la part des développeurs 6
3.3. Et si on n'était pas les seuls touchés? 7
4. Résoudre les failles 9
4.1. Les solutions retenues 9
5. Découvrir les failles: un métier? 10

Introduction

Ah les vacances, quelle belle période! Repos, rattrapage de la pile de livres, découvrir des nouvelles choses, faire des photos…

son de notification discord

> gustavi : Salut Artragis, j'ai une faille de sécurité sur la génération des PDF.
> artragis : Merci de ton signalement c'est quel genre de faille.
> gustavi : une injection de code, je peux réussir à afficher les `/etc/passwd`.
> artragis : Argl, ça semble critique.
> gustavi : L'impact est critique mais comme il faut un contenu validé pour les PDF l'importance est relative.
> artragis : Ah mais non, on peut faire des PDF sur les billets du coup pas de validation.
>
> *Discussion le jour des congés*[1] *footnote:1*

Malheureusement ZDS (mais pas seulement, on vous en dira plus plus tard) a bien été touché par une faille de sécurité. Le but de cet article est de vous présenter comment elle a pu être découverte, déclarée et comment on comble une faille de sécurité dans *la vraie vie*™.

1. [2]footnote:1 c'est vraiment le cas

1. Comment fonctionne ZDS

> La publication, c'est simple, c'est la partie la plus compliquée de ZDS

Sur ZDS on essaie de vous fournir un ensemble de fonctionnalités intéressantes lors de la rédaction puis de la publication d'un contenu (tutoriel/article/billet).

Pour cela nous nous appuyons tant sur des logiciels que nous avons nous-mêmes créés que sur des logiciels tiers éprouvés et performants.

Lors de l'appui sur le bouton «valider», votre validateur favori va enclencher une *pile logicielle* de quatre composants principaux:

— une base de données qui stockera un lien vers la version du contenu qui a été validé, ainsi que la date de validation;
— ZMD, notre logiciel capable de comprendre Markdown pour en faire du HTML (pour le Web ou pour ePUB) ou du LaTeX;
— un logiciel en tâche de fond appelé **publication-watchdog** et capable de comprendre qu'un contenu vient d'être ajouté à la BDD;
— une «boîte magique» qui contient tout ce qui est nécessaire pour que **lualatex** transforme votre contenu en un beau PDF aux couleurs de ZDS.

Nous avons dû nous organiser ainsi car générer un PDF, un ZIP ou un ePUB ça prend du temps et que nous n'avons qu'une minute au total pour vous répondre.

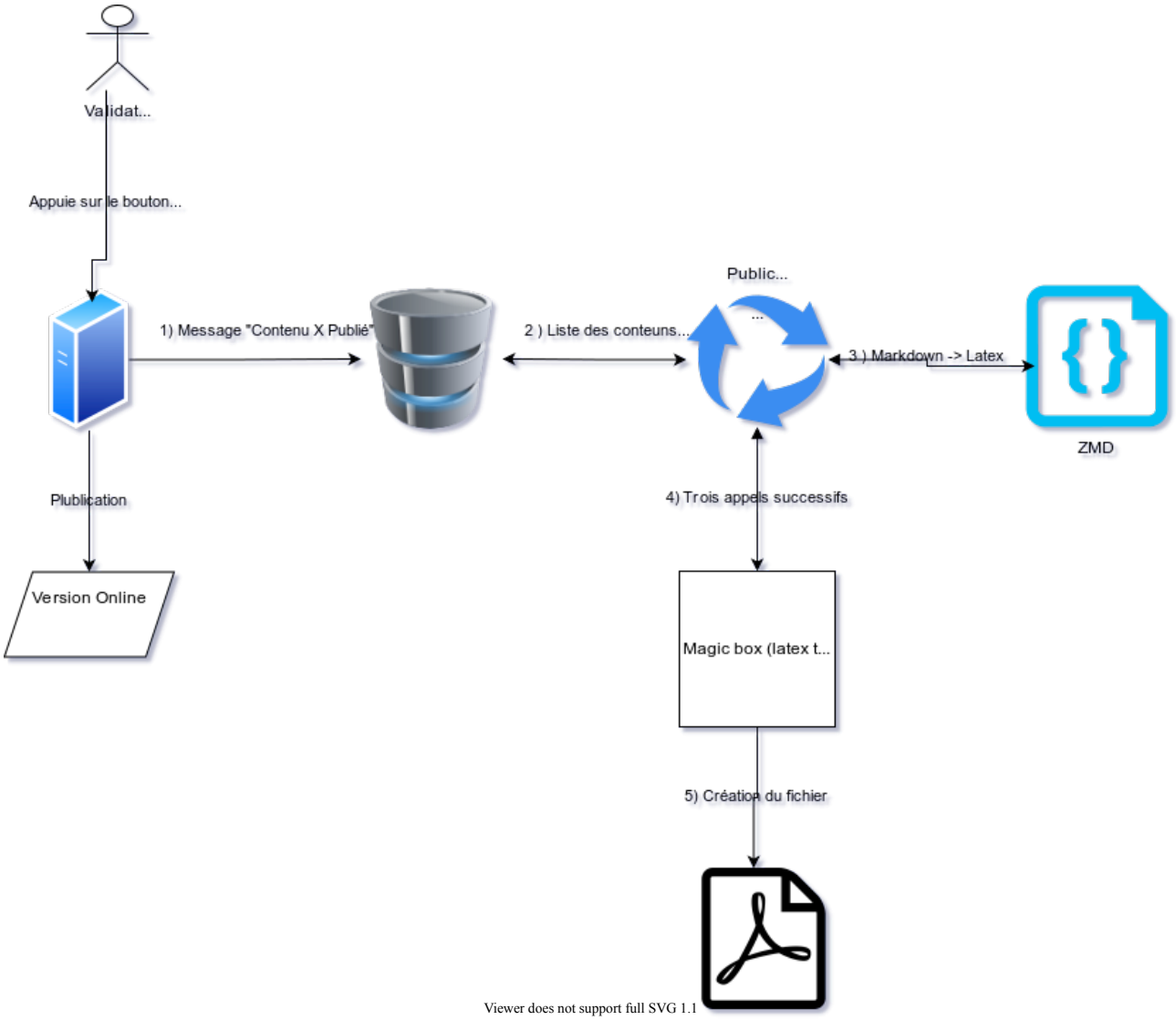

Figure: Lorsqu'un validateur appuie sur le bouton «valider», la BDD est prévenue que le contenu est publié, le **watchdog** s'en rend compte et demande à ZMD de générer le LaTeX puis appelle les outils pour créer le PDF.

De ce fait lorsque le validateur clique sur «valider», le serveur ne fait que produire la version Web et prévient le **watchdog** qu'il lui faudra faire les exports. Ainsi, pour lui tout est «rapide» [3]footnote:1 mais en tâche de fond, le **watchdog** va créer les exports. C'est pour ça que lorsqu'un contenu est publié, au moment de sa publication il n'y a pas de lien de téléchargement possible: le **watchdog** n'a pas eu le temps de terminer son travail.

2. La découverte des failles

2.1. Liste des failles

Trêve de suspense, voici les quatre failles découvertes par @gustavi:

1. [4]footnote:1 Entendons-nous: «rapide» signifie ici: la page travaille un temps acceptable puis le site est disponible de manière fluide. On considère qu'un validateur acceptera d'attendre une quinzaine de secondes au maximum, c'est long pour une page web normal, mais la publication n'est pas une page web normale 🍊 .

— il était possible de générer un PDF avec des blocs de code qui, ensuite, exécute des commandes Bash. Par exemple, cela peut permettre d'afficher des données confidentielles du serveur telles que le contenu du fichier `/etc/passwd` ou le fichier de configuration du site;
— il était possible d'intégrer des images se situant n'importe où sur le serveur, via un chemin absolu ou en utilisant des chemins relatifs, ces images n'étaient interprétées que dans le cas des PDF;
— il était possible de faire des requêtes sur l'infrastructure interne du site. C'est une faille très technique, elle peut permettre de faire fuiter des données sur notre infrastructure;
— il était possible d'insérer des objets JSFiddle sur les messages du forum, ce qui génère une faille XSS.

2.2. Quelques détails

Une des failles sort du lot car elle ne permet que de cartographier l'infrastructure, elle est très technique et fera sûrement l'objet d'un billet pour elle seule.

Pour les autres, elles relèvent toutes d'une classe de vulnérabilités appelées **injections de code**.

L'idée de ces vulnérabilités est d'injecter un code malveillant qui sera exécuté par le serveur ou le navigateur du visiteur. Les effets potentiels de ce code sont multiples: donner le contrôle au pirate, faire fuiter des données confidentielles, générer un déni de service…

Sur ZDS, les auteurs vous ont déjà présentées deux types de failles:

— les injections SQL ;
— les failles XSS .

Je vous laisse lire le second tutoriel pour que vous compreniez pourquoi la possibilité de mettre du JSFiddle sur le forum (à contrario des tutoriels) est une faille XSS.

Pour l'injection de commandes dans les PDF, le plus simple est de vous montrer le code qu'a envoyé @gustavi sur son environnement de test:

```
```

Une fois ce code envoyé dans un billet, lorsque le PDF était produit, le LaTeX généré était celui-ci:

```
\begin{CodeBlock}{text}

\immediate\write18{cat /etc/passwd > outputrce}
\input{outputrce}

\begin{CodeBlock}{text}
```

La commande LaTeX `\immediate\write18{cat /etc/passwd > outputrce}` permet d'envoyer dans le PDF le contenu de `outputrce`, et ça tombe bien, on avait exécuté la commande `cat /etc/passwd` qui avait copié le contenu du fichier `/etc/passwd` dans outputrce. Voilà, on a fait fuité des données.

POC

Ce petit code fourni par @gustavi est appelé POC, c'est-à-dire proof of concept: il permet de prouver que la faille théorique découverte est exploitable.

3. Communiquer la faille et l'étudier

3.1. Les canaux de communication

Lorsqu'une personne détecte une faille sur un site internet ou un outil il faut prévenir les développeurs de la manière la plus confidentielle possible afin de ne pas révéler la faille à tout le monde.

Sauf si le projet le dit explicitement, cela signifie qu'il ne faut pas passer par le bug tracker (chez nous c'est GitHub) mais par un canal parallèle auquel seules quelques personnes ont accès.

Le plus souvent vous trouverez l'information dans la catégorie «contact» des logiciels que vous utilisez.

Sur ZDS, on vous propose deux possibilités:

— nous contacter sur `technique@zestedesavoir.com`: seuls notre sysadmin (@Situphen), le président (@AmauryPi) et le mainteneur (@artragis) du projet y ont accès;
— contacter directement le mainteneur (@artragis) en message privé sur discord.

La déclaration se fait du mieux que vous pouvez, tout le monde n'est pas à l'aise avec les longs pavés écrits et c'est pourquoi il n'y a pas à s'inquiéter sur la forme. Cela dit, si vous voulez un guide pour rassembler toutes les informations utiles, voici le guide que @gustavi à utilisé:

```
1 - **Vulnérabilité **: type de vulnérabilité - par exemple
    "*Injection de code/commande*",
2                       si vous n'avez pas la connaissance technique
                            pour nommer la faille, ce n'est pas grave.
3 - **Éléments affecté s**: Quelles sont les conséquences sur
    l'application ?
4                            Est-ce un simple déni de service ou
                                l'accès à des données sensibles ?
5 - **Scénario d'attaque **: Quelles sont les conditions pour
    réussir à exploiter la vulnérabilité ?
6                            Faut-il un compte sur la plateforme ?
                                Avec des privilèges ?
7 - **Risque **: Listes des risques possibles: exemple
```

```
    Détails des risques : lecture, écriture et injection de
        commandes sur le système, vol de la base de données,
        altération du site, etc.
- **Remédiation **: Si vous avez une idée de comment corriger la
    faille, sinon les développeurs analyseront, c'est ce qu'il
    s'est passé ici.
- **Mesures à prendre pour vérifier si une compromission à déjà eu
    lieu** : Quelles manœuvre opérer pour voir si quelqu'un en
    profite vraiment ? Par exemple "chercher que tel mot est
    utilisé dans les contenus".
- **Mesures à prendre en attendant un correctif **: exemple:
    "désactiver la génération du PDF"
```

Ce modèle est là pour faciliter la vie de tout le monde car il organise le retour du découvreur et présente toutes les informations disponibles.

3.2. Suivi de la part des développeurs

Une fois que la faille est déclarée, les développeurs vont donc tenter de trouver comment la corriger. Mais en attendant cela, il faudra mettre en place les éléments qui permettent d'éviter que la faille ne soit utilisée. Si on ne peut pas empêcher la faille d'être utilisée il faut tout faire pour en limiter les impacts, c'est ce qu'on appelle **la mitigation**[5]footnote:1.

3.2.1. Désactiver la fonctionnalité vulnérable

En informatique, souvent, ce genre de situation génère la désactivation pure et simple d'une fonctionnalité. Chez nous, il a fallu désactiver la génération des PDF sur les billets (les tutoriels sont validés, donc pas de soucis).

Dans le meilleur des cas, il s'agit simplement de modifier une configuration. Mais dans notre cas, malheureusement, rien n'avait été fait en ce sens. Il a donc fallu aller sur le serveur pour modifier à chaud le code pour retirer la génération des PDF des billets. Cette modification n'a pas été versionnée mais a bien été mise en place le mercredi 5 août sur les serveurs de production et de bêta test.

3.2.2. Analyser la cause racine

Toutes les failles détectées faisaient intervenir ZMarkdown. La question qui se posait était donc:

? Était-ce ZMarkdown qui avait une faille ou notre communication avec ce service qui était mauvaise?

1. [6]footnote:1 La préfecture du Nord vous explique tout ça en détail si vous avez envie d'approfondir.

Vous l'aurez peut être deviné, loi de Murphy oblige, les deux briques (ZMarkdown et la communication) étaient touchées. Voici un petit tableau récapitulatif:

Faille	Source
XSS: Insérer JSFiddle dans le forum	Faille dans la communication
Injection de commande: Bloc de code qui se coupe et permet d'exécuter des scripts Bash	Bug dans rebber (Markdown -> LaTeX)
Découverte du serveur: Insérer des images non servies par ZDS	Faille dans remark-downlad-images, ZMarkdown et dans la communication

TABLE 3.2. – Les trois failles corrigées

3.3. Et si on n'était pas les seuls touchés ?

3.3.1. Déclarer la faille au public

ZMarkdown n'est pas utilisée que par Zeste de Savoir, il faut donc prévenir tous les utilisateurs qu'il y a potentiellement un souci. Pour cela, GitHub nous fournit un outil: les **security advisory** [7]footnote:2. Il s'agit d'un formulaire à remplir qui permettra à GitHub de prévenir tous les mainteneurs de projets qui dépendent de rebber ou de remark-download-images que s'ils n'utilisent pas la dernière version du logiciel, ils sont vulnérables.

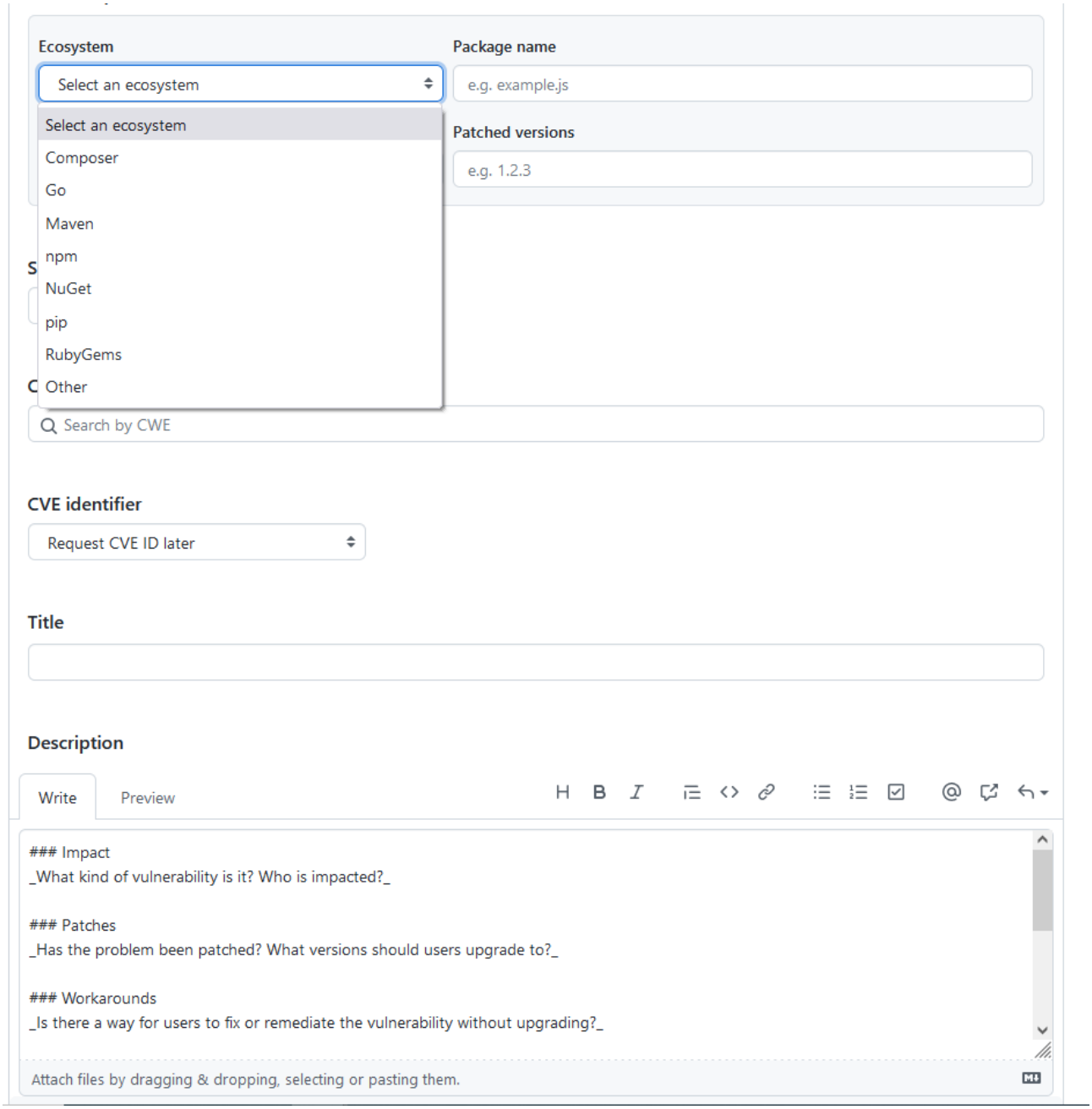

FIGURE 3.1. – Créer une **security advisory** sur GitHub: entrer le type de projet (npm pour JavaScript, composer pour PHP, pip pour Python…), son nom, la version affectée et corrigée, ainsi que le type de faille grâce au CWE.

3.3.2. D'autres projets avec une faille similaire ?

La faille d'injection de commande à cause des blocs de code est une faille qui était complexe à colmater. Il nous fallait de ce fait voir comment d'autres développeurs qui avaient fait face aux mêmes problèmes que nous s'en étaient sortis.
Nous avons donc tenté d'analyser le code de pandoc , un outil codé en Haskell qui permet de convertir plusieurs types de langages en PDF, ePUB, Word ou ODT.
À peine l'analyse avait-elle commencé que @gustavi s'est rendu compte que pandoc souffrait de la même faille que nous. Elle a été rapidement corrigée par son auteur et ladite correction nous

a inspiré le code final pour ZMarkdown.

4. Résoudre les failles

4.1. Les solutions retenues

4.1.1. Résolution de la faille critique

La première vulnérabilité remontée était la faille *Remote Code Execution*. Cette faille permettait d'injecter n'importe quelle commande LaTeX via un bloc de code. La solution proposée pour sa résolution est globalement simple: on souhaite empêcher la fermeture d'un bloc de code dans ledit bloc. Il suffit donc de retirer du contenu du bloc de code toute commande `\end{CodeBlock}`. Cette solution est espérée temporaire, en cela qu'elle empêche la coloration d'un élément de LaTeX.
La première version de cette correction retirait simplement la commande comme précisé ci-avant, mais il est en fait possible en LaTeX d'écrire `\end {CodeBlock}`. Une seconde itération du correctif a donc été réalisée afin de prendre en compte les cas contenant des espaces.

4.1.2. Correction de la faille sur le téléchargement des images

Lors du processus de génération des PDF, on souhaite que toutes les images liées au contenu se trouvent dans un même dossier. Pour cela, pendant la conversion du Markdown en LaTeX, le plugin `remark-download-images`, utilisé par ZMarkdown, télécharge les images incluses dans le contenu. Ce plugin possède deux comportements distincts, en fonction de si l'image est située localement sur notre serveur ou embarquée via un lien distant:

— si l'image est locale, alors on effectue des transformations prédéfinies sur le chemin de l'image avant de tenter de la télécharger. Outre une interdiction par défaut des chemins contenant `../`, ces transformations sont définies par un tableau de remplacements à effectuer, via le paramètre `local_url_to_local_path`;
— si l'image est distante, une requête est effectuée par récupérer l'image, et on s'assure qu'elle ne contienne pas de bêtises avant de la stocker (envoi de faux MIME, SVG invalides, etc.).

À la fin du processus, il faut considérer que toutes les images du contenu sont dans un dossier unique, portant un identifiant aléatoire généré au préalable. En tout cas, c'est ce qu'il se passe quand tout se déroule comme prévu, car il y a parfois des problèmes lors du téléchargement, auquel cas on peut provoquer une ou plusieurs de ces actions:

— lever une erreur non-bloquante (*warning*), c'est le cas le plus fréquent;
— lever une erreur bloquante de publication, qui empêche purement et simplement la génération. Ces erreurs sont désormais rarissimes;
— remplacer l'image problématique par une fausse image, qui consiste en un simple pixel noir.

Maintenant que le processus est clarifié, notre faille LFI était causée par deux éléments: d'abord, le paramètre `local_url_to_local_path`, mentionné plus haut, n'était pas utilisé, ce qui ne déclenchait pas la copie de l'image comme souhaité, on pouvait donc inclure l'image `/home/img.png` par exemple. Déclarer ce paramètre, en lui donnant par exemple la valeur `['/', '/opt/zds/']`, remplacerait tous les slashs en début de chemin par `/opt/zds/`. Ce

2. [8]footnote:2 Notons que depuis octobre 2021, la base de données des security advisory de GitHub est utilisée par la commande `npm audit` pour détecter les dépendances vulnérables. (Source : NextInpact)

paramètre permet de déclencher la copie de l'image; cette partie du correctif concerne donc purement `zds-site` à priori.
Par ailleurs, même en déclarant ce paramètre, un comportement problématique était observé: la copie de l'image `/opt/zds/home/img.png` étant impossible (l'image n'existe pas), le serveur ZMarkdown levait une erreur *non-bloquante*, et ne remplaçait pas le lien de l'image. On se retrouvait donc avec une inclusion directe de l'image d'origine (et non de l'image copiée, celle-ci n'ayant pas pu l'être) dans le code LaTeX. Il est probablement possible de résoudre ce problème en évitant la possibilité pour LaTeX d'accéder aux images extérieures à son dossier de compilation, mais préférant éviter de dépendre uniquement de ce genre de correctif, nous avons opté pour un remplacement dans ce cas de l'image par la fausse image mentionnée ci-dessus.

4.1.3. Vulnérabilité SSRF

La dernière faille est plus complexe à corriger, en cela qu'elle constitue en partie un comportement souhaité: on veut que le moteur de Markdown soit en mesure d'effectuer des requêtes extérieures pour récupérer des images. Il n'est donc pas envisageable d'interdire toute requête externe, cela n'aurait pas de sens. On considère alors une solution de compromis: les requêtes étant effectuées de manière très ponctuelles lors de la publication du PDF, il est très improbable qu'elle puissent être utilisées pour réaliser une attaque par déni de service. Le risque posé est donc accepté en ce qui concerne les informations accessibles publiquement.
La faille contient toutefois un autre volet: les informations non-publiques. Par exemple, l'accès à une image exposée sur le réseau local. Ce volet constitue lui un risque inutile, et il est donc important de le corriger. C'est ce sur quoi se focalise le correctif proposé: il empêche toute adresse IP privée d'être utilisée pour les images contenues dans un PDF.
Imaginons qu'on laisse le correctif comme tel: il ne sera alors plus possible de faire télécharger `http://192.168.2.1/img.png`, mais un pirate malin pourrait tenter une attaque réfléchie. Le principe serait par exemple de créer un nom de domaine `centquatrevingtdouze.fr` dont la résolution donnerait `192.168.2.1`. Une fois cela fait, il suffirait d'introduire l'image à l'adresse `http://centquatrevingtdouze.fr/img.png` pour aller chercher l'image, ce qui échapperait à notre système de détection.
Cet exemple montre qu'il est important de considérer également la résolution DNS d'un domaine lors de l'interdiction des IPs locale. On rajoute donc au correctif un système tel que:

- si l'URL de l'image est une adresse IP, on vérifie qu'elle soit bien en dehors de la plage des IP privées, et donc interdites, sans quoi on retourne une erreur;
- lorsque l'URL ne correspond pas à une IP, on résout le domaine, et on vérifie que l'IP obtenue soit bien en dehors de la plage des IP privées.

5. Découvrir les failles : un métier ?

?

> Bonjour @gustavi, découvrir des failles de sécurité d'un logiciel fait partie de ton métier? Quel est-il? En quoi consiste-t-il exactement?

Bonjour, c'est exact je travaille dans une entreprise spécialisée dans les tests d'intrusion en informatique et mon métier consiste à trouver des failles de sécurité.
Aujourd'hui l'informatique est devenu un élément vital pour les entreprises et de nombreux business ne reposent que sur son bon fonctionnement. On le voit d'ailleurs très régulièrement dans les médias: un système à l'arrêt, que ce soit suite à une attaque ou un problème technique,

peut avoir des conséquences financières voire humaines très importantes. Pour s'assurer de la bonne protection de leur systèmes, des entreprises font vérifier leurs systèmes par des auditeurs spécialisées qu'on appelle *pentesters*.
C'est un métier assez diversifié car chaque système informatique est différent: dans un même réseau on peut trouver un serveur Web Ubuntu, un Active Directory et un AS/400 . Trois systèmes d'exploitation qui impliquent 3 approches différentes et donc une connaissance précise de la sécurité de chacun.

? Un audit, ça se passe comment?

Comme je viens de le dire, il existe une multitude d'audits différents. Voici une liste de types audits que j'ai pu rencontrer directement ou indirectement lors de mes missions:

— site Web;
— application mobile;
— configuration d'un système d'exploitation (Windows, GNU/Linux, UNIX, AS400);
— configuration d'un logiciel (SGBD, serveur Web, LDAP, etc);
— Wi-Fi;
— code source;
— réseau interne;
— intrusion physique;
— architecture;
— fonctionnel;
— et procédures.

Je vais m'arrêter là mais vous voyez l'idée: chaque élément d'un système informatique est une brique à part avec ses particularités et sa propre sécurité. Comme il est quand même complexe de toucher à tout chacun se spécialise dans des domaines de prédilection. De mon côté c'est l'audit de code source et les systèmes Linux/UNIX.
Un audit classique se déroule généralement sur plusieurs jours et comprend un périmètre d'attaque précis. La première étape est la reconnaissance: l'idée est de trouver un maximum d'éléments exposés (par exemple des URL dans le cas d'un site web ou des machines et les ports ouverts dans le cas d'un réseau) afin d'avoir une meilleure vision du périmètre et de cibler en priorité les éléments qui semblent être les plus sensibles. La seconde partie de très loin la plus longue consiste à tenter différentes attaques sur les éléments précédemment trouvés. Si jamais on réussit à trouver une vulnérabilité on peut dans certains cas l'exploiter afin d'augmenter la surface du système audité. La dernière partie d'un audit est la moins marrante pour nous: la rédaction d'un rapport expliquant les vulnérabilités trouvées et donnant des indications de correction.
Je rappelle que tout ça est fait dans un cadre légal et contractualisé entre mon employeur et le client qui commande un audit. Le fait de s'introduire dans un système informatique sans l'accord du propriétaire est passible de 5 ans d'emprisonnement et 300 000 € d'amende (Articles 323–1 à 323–8 du code pénal). Rajoutez 2 ans et doublez l'amende si c'est un service de l'État.

? Qu'est-ce qui t'a donné l'idée qu'on pouvait faire cette injection de code? Est-ce parce que ZDS et ZMD sont open-source?

En toute franchise je n'ai pas souvenir de comment l'idée m'est venue mais elle était là déjà depuis quelques semaines: m'injecter dans des balises spéciales de ZMarkdown. J'avais fait une

première tentative dans les balises de lien il y a plusieurs mois sans grand succès. J'ai profité de quelques jours de congés pour creuser un peu le sujet.
Je passe donc un peu de temps à regarder toutes les possibilités de balises, la balise de code me semble une bonne candidate à contenir des failles (puisqu'on y met du code). Un petit test sur la bêta confirme mes soupçons: le code me semble injectable. Le temps d'installer une instance de Zeste de Savoir sur une machine virtuelle locale et trois petites heures plus tard j'avais un exploit fonctionnel qui permettait d'exfiltrer n'importe quel fichier du serveur (note: l'exploit n'a pas été lancé en production ni en bêta de mon côté, aucune donnée n'a été divulguée). J'ai ensuite directement alerté @artragis qui a déployé un *hotfix* sur la bêta et la production afin de bloquer l'export PDF.
Le fait que Zeste de Savoir soit open-source m'a effectivement facilité les choses:

— les sources du code étant accessibles il est facile d'aller voir le fonctionnement sous le capot sans devoir faire des suppositions sur ce qui se passe réellement;
— il est possible d'installer facilement une instance de l'outil pour procéder à des tests locaux sans impacter la production;
— la communauté qui est derrière l'outil est presque toujours bienveillante et réactive, ce qui est un réel plus.

Cette transparence est clairement une force de l'open-source d'un point de vue sécurité.
Je remercie encore l'équipe technique de Zeste de Savoir qui a été très réactive (34 minutes entre mon premier message et la désactivation de l'export PDF en production).

? Dans ton expérience quel est la faille qui t'a le plus marqué? Quelques conseils pour s'en prémunir?

Pour des raisons de confidentialité je ne peux pas rentrer dans les détails mais j'ai quelques expériences marquantes que je peux vous raconter.
La première était lors d'une de mes premières missions d'audit de code source. Il s'agissait d'un très gros morceau de code en PHP et après quelques jours je suis tombé sur un fichier très étrange: tout était encodé en base64. Je le décode et là je découvre une boîte à outils une peu spéciale: des dizaines de fonctionnalités qui ressemblent à des outils d'administration systèmes, de quoi exécuter des requêtes SQL ou scanner le réseau. Vous l'aurez peut-être compris, il s'agissait en fait d'un *webshell* déposé par un attaquant lors d'une précédente attaque. Le client a tout de suite été prévenu et la mission s'est un peu transformée en *forensic*. Par chance la charge n'avait pas été utilisée au cours des dernières années, plus de peur que de mal.
Une seconde mission assez importante pour moi s'est déroulée sur un périmètre assez large: quelques dizaines de sites web et autant de serveurs. Nous avions carte blanche dans un temps limité pour trouver le maximum de vulnérabilités. Ma première grosse mission qui peut se rapprocher d'un *red team* et nous n'avons pas démérité: en 10 jours nous avons réussi à compromettre la moitié des serveurs en tant que `root` et l'autre moitié en tant qu'utilisateur normal. Le scénario d'attaque complet faisait apparaître 6 vulnérabilités différentes utilisées les unes à la suite des autres ce qui en a fait une superbe expérience.
Si je devais donner un conseil pour réduire drastiquement le risque d'une compromission en 2 points clés: les mises à jour et la gestion des mots de passe[9]footnote:1. Tenez-vous informés des mises à jour et appliquez les correctifs le plus rapidement possible en particulier sur les périmètres exposés sur Internet. Pour les mots de passe c'est très simple: utilisez des mots de passe aléatoires stockés dans un coffre fort de passe et utilisez autant que possible l'authentification à facteurs multiples. L'ANSSI vient d'ailleurs de mettre à jour son guide sur les mots de passe .

? Faut-il faire des études particulières pour faire ton métier?

Pour moi la voie royale c'est une école d'ingénieur ou un master avec une spécialisation en sécurité informatique dans la fin du cursus. De nombreux établissements supérieurs ont ouvert, ces dernières années, des places dans le domaine de la cybersécurité car le secteur peine à recruter.
En revanche je ne connais pas du tout les formations plus courtes sur 2 ou 3 ans (BTS, DUT et licences professionnelles) et donc ne peux pas me prononcer sur le sujet. Dans tous les cas il existe de nombreux autres métiers liés à la cybersécurité, en particulier dans la partie défensive.
Si ce métier ou la cybersécurité vous intéresse voici quelques ressources pour apprendre (par ordre croissant de difficulté):

— le MOOC de l'ANSSI accessible à **tout niveau** permet d'apporter les premières bases de la sécurité (et devrait être fait par toute personne tapant sur un clavier !);
— TryHackMe un site d'apprentissage en ligne assez bien fait et très accessible aux débutants;
— Root Me un site de challenges pour apprendre la sécurité informatique dans de nombreuses catégories;
— Hack The Box un site pour apprendre dans des conditions un peu plus réelles (il faut réussir à compromettre une machine);
— CryptoHack un site de challenges spécialisés dans la cryptographie.

Vous pouvez également retrouver plusieurs contenus sur Zeste de Savoir avec le tag sécurité !

1. [10]footnote:1 On ne vous encouragera jamais assez à utiliser un gestionnaire de mots de passe . D'ailleurs l'ANSSI a mis à jour ses conseils pour générer des bons mots de passe .