Projet ISE - BadUSB

Partie 3 - Exfiltration de mots de passe

Introduction

Ce projet a pour objectif d'explorer les concepts derrière les attaques par "*rubber ducky*" (aussi connues sur le nom d'attaques *BadUSB*) et en apprenant à utiliser un Raspberry Pico 2 pour concevoir une clé USB malveillante. Les *rubber duckies* sont des dispositifs qui émulent un clavier USB pour injecter des commandes rapidement sur une machine cible. Leur popularité provient de leur simplicité et de leur efficacité pour réaliser des attaques telles que l'exfiltration de données, l'installation de logiciels malveillants, ou la création de portes dérobées.

> Ce projet a été rédigé à des fins pédagogiques. Il n'a pas pour vocation à être utilisé "sur le terrain". Pour rappel l'article L.323-1 du code pénal prévoit que "*Le fait d'accéder ou de se maintenir, frauduleusement, dans tout ou partie d'un système de traitement automatisé de données est puni de trois ans d'emprisonnement et de 100 000 € d'amende.*"

Le projet est divisé en 3 parties. Dans cette partie, vous allez concevoir une attaque complète utilisant votre Raspberry Pico 2 configuré en Rubber Ducky, afin d'exfiltrer des données sensibles (mots de passe stockés dans Firefox) vers un serveur externe.

1 Récupération des mots de passe du navigateur

Dans la partie 1, vous avez manipulé le l'outil `firefox_decrypt` via un notebook. Ce format est pratique pour une découverte pédagogique, mais n'est pas adapté à l'exécution dans le cadre d'une attaque. Une version en script classique est mise à disposition au lien suivant : https://ddealmei.github.io/assets/3b64d11ff0ac5da6a262b3490949ec7f6/firefox_decrypt.py

▶ **Question 1.** Vérifiez que vous êtes bien capable d'exécuter ce script pour retrouver les mots des passe du navigateur.

▶ **Question 2.** Comment pouvez-vous, depuis un terminal, télécharger ce script et l'exécuter ?

▶ **Question 3.** Le script va afficher les mots de passe dans le terminal. On voudrait les enregistrer dans un fichier. Comment pouvez-vous rediriger la sortie du script vers un fichier ?

▶ **Question 4.** Les scripts rubber ducky exécute les lignes les unes après les autres, sans forcément attendre la fin d'une commande lancée. Comment pouvez-vous vous assurer que le téléchargement du script sera terminé avant de tenter de l'exécuter ?

2 Exfiltration des données

Le site `webhook.site` permet de tester simplement des requêtes HTTP et visualiser leur contenu. Pour les besoins du TP, et pour des raisons évidentes de sécurité, il n'est pas sûr d'utiliser ce site pour des données sensibles. Pour cette raison, un site similaire a été créé et hébergé sur le réseau interne de l'université par l'enseignant responsable du projet.

Vous pouvez accéder à ce site via l'URL suivante : `ise2.istic.univ-rennes1.fr`. Au premier accès, votre navigateur vous avertira d'un risque de sécurité. Ce comportement est attendu, et normal au vu de la configuration particulière mise en place. Vous pouvez cliquer sur le bouton "Avancé" et "Accepter les risques et continuer" pour accéder au site.

> N'utilisez **pas** le site webhook.site pour ce TP. Utilisez le site mis à disposition, qui assure que les données ne seront pas traitées par qui que ce soit. Les données seront systématiquement effacées à 1h, toutes les nuits, pour éviter toute fuite.

Ce site s'utilise de la manière suivante :

1. Créez-vous un compte et connectez-vous.
2. Envoyez des requêtes à l'URL `ise2.istic.univ-rennes1.fr/hook/<username>` (vous verrez dans la suite comment effectuer ces requêtes)
3. Consultez les requêtes reçues sur la page de votre compte dans le navigateur `ise2.istic.univ-rennes1.fr/<username>`

> **Note:** Les requêtes reçues sont actualisées automatiquement. Vous seuls pouvez consulter les requêtes envoyées à votre compte, mais n'importe qui peut en envoyer.

Pour la suite du TP, on suppose que vous avez ouvert votre page de compte sur votre navigateur.

▶ **Question 5.** La commande `curl` permet (entre autre) d'envoyer des requêtes HTTP depuis le terminal. Au vu de la configuration du serveur, vous aurez besoin d'utiliser l'option `--insecure` pour éviter les soucis de vérification de certificats.

▶ **5.a.** Quel type de requête HTTP permet d'envoyer des données ?

▶ **5.b.** Comment envoyer de telles requêtes depuis le terminal en utilisant `curl` ?

▶ **Question 6.** Testez l'envoie du texte "Hello Wolrd" (ou tout autre message plus original) au serveur, avec `curl`. En cas de succès, vous devriez observer la requête apparaître dans le bandeau de gauche sur votre navigateur. Observez son contenu en cliquant dessus.

▶ **Question 7.** Au lieu d'envoyer du texte via la ligne de commande, on souhaite envoyer un fichier. Quelle option de `curl` permet d'envoyer le contenu d'un fichier dans la requête ?

▶ **Question 8.** Testez manuellement chaque étape : téléchargement, exécution du script **firefox_decrypt.py**, génération du fichier de mots de passe, et envoi du fichier via `curl`. Vérifiez que les données apparaissent correctement sur la page web du serveur.

3 Automatisation complète via Rubber Ducky

Vous êtes maintenant prêts à automatiser entièrement ce processus avec votre Rubber Ducky.

▶ **Question 1.** Rédigez un script complet au format DuckyScript qui réalise les opérations suivantes

1. Ouvre un terminal automatiquement.
2. Télécharge le script `firefox_decrypt.py`.
3. Exécute ce script pour extraire les mots de passe Firefox.
4. Sauvegarde le résultat dans un fichier.
5. Exfiltre ce fichier vers votre URL dédiée en utilisant `curl`.

Testez votre script sur votre PC, puis vérifiez sur Webhook.site que tout fonctionne comme prévu. Joignez votre script final à votre rendu.
Portez attention à l'enchaînement des commandes : si la carte passe au traitement de la ligne suivante du script avant la fin de l'exécution d'une commande, l'attaque pourrait ne pas fonctionner.

▶ **Question 2.** Ce script est relativement simple. On pourrait facilement l'étoffer en récupérant d'autres informations (mots de passe Wi-Fi, données d'autres navigateurs, historique, ...). En revanche, un DuckyScript trop long multiplie les chances qu'il échoue ou qu'il soit détecté. Comment pourriez-vous faire en sorte de minimiser le nombre de lignes à taper pour le script pour effectuer une attaque plus complexe ?