Sécurité WEB

Attaques Web

- Identification des attaques Web
  - ✓ Incapacité de l'utilisateur à accéder aux services en ligne.
  - ✓ Redirection des URLs bien saisis vers des sites indésirables.
  - ✓ Diminution inhabituelle des performances du réseau.
  - ✓ Redémarrage fréquent du serveur.
  - ✓ Anomalies dans les fichiers journaux (log).

Attaques Web (suite)

- Types d'attaques Web
  - ✓ Cross-site scripting (XSS).
  - ✓ Cross-site request forgery (CSRF).
  - ✓ Injection SQL.
  - ✓ Injection de code.
  - ✓ Injection de commande.
  - ✓ Falsification de paramètre (parameter tampering).
  - ✓ Empoisonnement de cookie (cookie poisoning).
  - ✓ Buffer overflow.
  - ✓ Espionnage de cookie (cookie snooping).

Attaques Web (suite)

- Types d'attaques Web (suite)
  - ✓ Attaque de protocoles dans la DMZ.
  - ✓ Détournement d'authentification (authentication hijacking).
  - ✓ Falsification de journal (log tampering).
  - ✓ Interception cryptographique.
  - ✓ Interprétation d'URL.
  - ✓ Usurpation d'identité.

Authentification WEB

Authentification = prouver *qui* est l'utilisateur (différent de l'autorisation = ce qu'il a le droit de faire).

- Facteurs :
  - *Ce que je sais* (mot de passe, code PIN)
  - *Ce que je possède* (téléphone, clé de sécurité)
  - *Ce que je suis* (biométrie).
    → MFA/2FA combine ≥2 facteurs pour plus de sécurité.

Comment une appli web authentifie

- Saisie des identifiants (login + secret).
- Vérification côté serveur (hash du mot de passe) ou via un fournisseur d'identité (SSO).
- Création d'une session si OK :
  - Cookie de session (classique) : le serveur stocke l'état de session, le navigateur garde un identifiant opaque.
  - Jeton (JWT, PASETO…) : le serveur émet un jeton signé que le client renvoie ensuite (souvent dans un cookie).

Mécanismes d'authentification courants

- Formulaire + cookie de session (le plus répandu).
- Tokens stateless (JWT) : utiles en API/microservices, attention à la rotation et à la révocation.
- SSO fédéré :
  - OAuth 2.0 (délégation d'accès), OpenID Connect (couche identité sur OAuth → "qui est l'utilisateur ?").
  - SAML 2.0 (entreprises, intégrations anciennes).
- Sans mot de passe :
  - WebAuthn / Passkeys (FIDO2) : clé matérielle / authentificateur intégré (sécurisé, anti-phishing).
  - Magic link / OTP par email/SMS (pratique, sécurité variable).
- Biométrie (via WebAuthn), TOTP (appli Authenticator), Push (Duo, Microsoft Authenticator).

Authentification WEB

Stockage des mots de passe (côté serveur)

- Jamais en clair.
- Hash lents et salés : Argon2id (recommandé), sinon bcrypt/scrypt.
- Paramètres (coût mémoire/temps) calibrés pour votre machine.

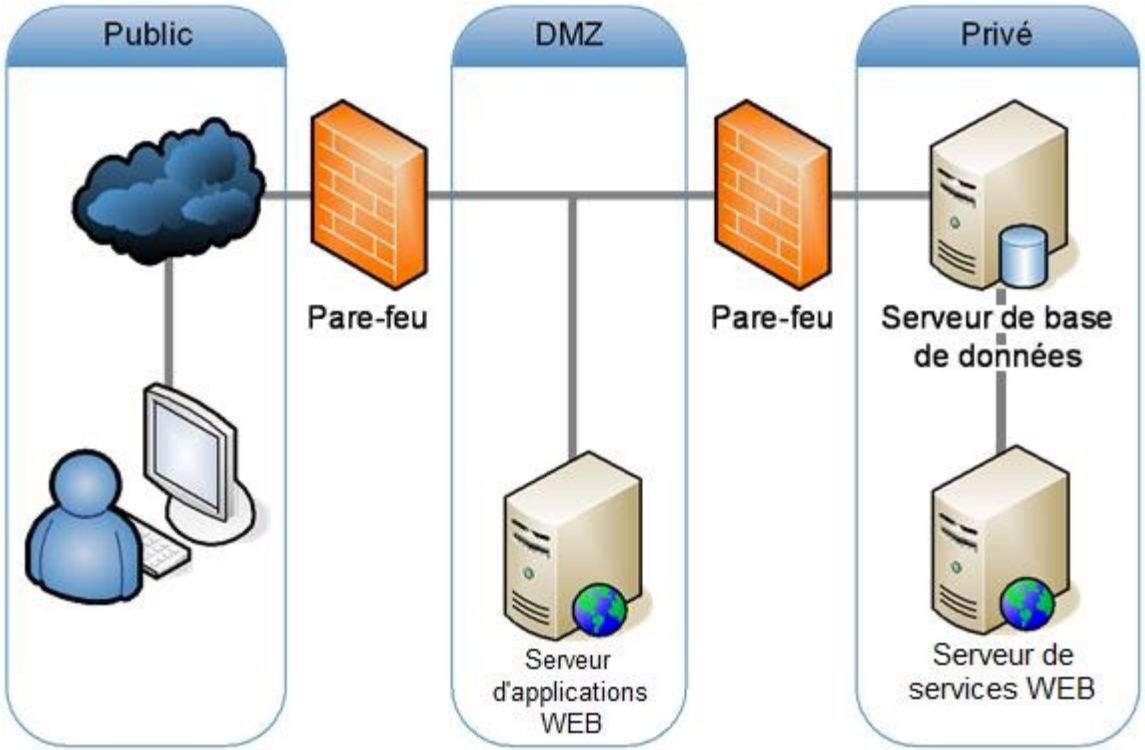
Sécurité des sessions

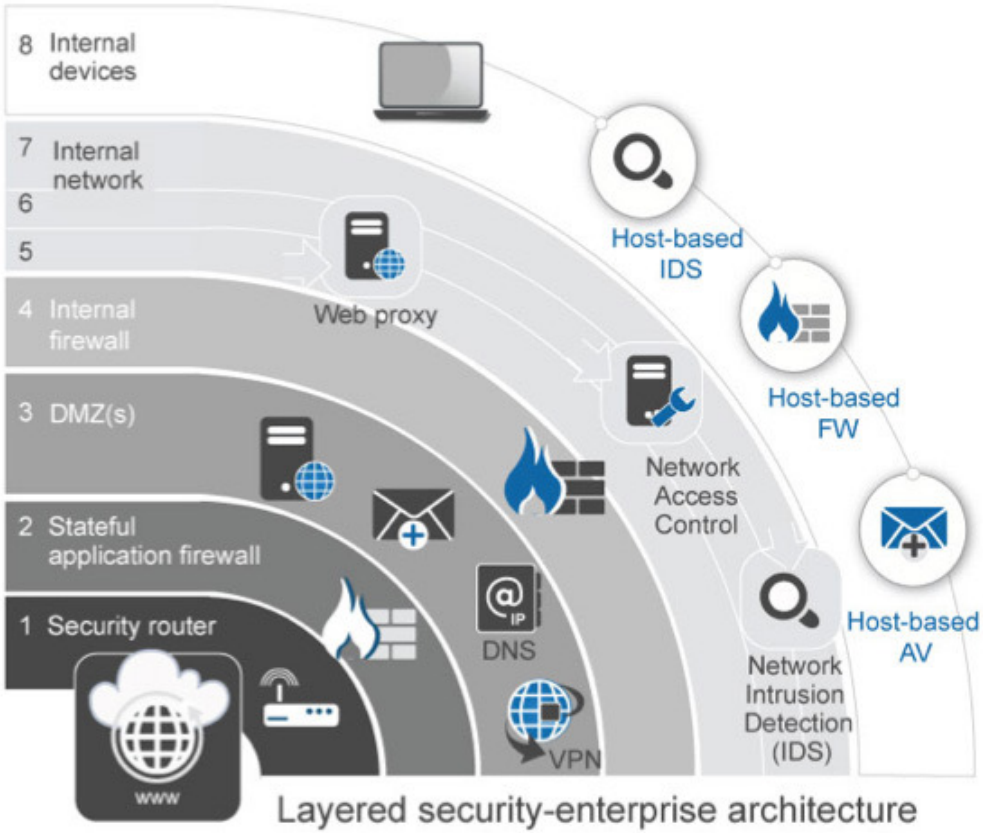
- Cookies HttpOnly; Secure; SameSite=Lax/Strict.
- Rotation de session après login / privilèges élevés.
- Expiration (inactivité, absolue), invalidation à la déconnexion.
- CSRF : token par session + SameSite + vérification d'origine.

Menaces et défenses

- Phishing → WebAuthn/passkeys, MFA, détection d'anomalies, éducation.
- Bruteforce/credential stuffing → limites de tentatives, CAPTCHA raisonné, HIBP/mot de passe compromis, 2FA.
- XSS → encodage à la sortie, CSP, cookies HttpOnly.
- Vol de session → Secure/HttpOnly, rotation, détection IP/UA, SameSite.
- Rejeu/élevations → anti-replay sur tokens, validation d'audience/portée/exp, PFS côté TLS.

Pare-Feu - Zone DMZ





Sécurité par profondeur

Cookie de session

- Un cookie de session est un petit jeton que le serveur web envoie au navigateur pour identifier la session d'un utilisateur pendant sa visite. Il contient un identifiant opaque (ex. sid=abC123...), pas vos données personnelles ; les infos réelles de session (qui vous êtes, vos droits, panier, etc.) sont stockées côté serveur et reliées à cet identifiant.
- Caractéristiques
  - Durée de vie : il existe jusqu'à la fermeture du navigateur (ou jusqu'à expiration courte si le serveur la fixe).
  - Contenu : typiquement seulement un ID (ex. sessionid=…).
- But : éviter de vous redemander de vous connecter à chaque page et reconnaître vos requêtes successives comme venant du même utilisateur.

Vol de Cookie de session

- Le « vol de cookie » consiste à obtenir illicitement le cookie de session d'un utilisateur pour usurper sa session (accéder à son compte sans mot de passe). Le serveur croit parler au vrai user, car le cookie sert d'étiquette d'identité.
- Principaux vecteurs
  - XSS (Cross-Site Scripting) : du JS injecté peut lire/expédier les cookies non HttpOnly.
  - Malware / extensions malveillantes : accès aux fichiers de cookies du navigateur.
  - Réseau non chiffré (HTTP, Wi-Fi piégé) : interception si le cookie n'est pas Secure/HTTPS.
  - Phishing / ingénierie sociale : l'utilisateur divulgue son cookie (ou un token) à un faux site.
  - Mauvaise configuration : cookies sur un domaine trop large, flags absents, sessions qui n'expirent pas.

Empoisonnement de cookie

- Empoisonnement de cookie
  - ❑ L'attaquant modifie le contenu du cookie pour voler les informations personnelles de l'utilisateur ou frauder sur un site web pour lequel l'utilisateur s'authentifie avec confiance.
  - ❑ Exemple de cookie

```
GET /bigstore/buy.asp?checkout=yes HTTP/1.0
Host: www.onshopline.com
Accept: */*
Referrer: http://wwwonshopline.com/showprods.asp
Cookie: SESSIONID=5435761ASDD23SA2321; Basket Size=6;
    Item1=2189; Item2=3331; Item3=9462; Total Price=14982;
```

- Ceci indique l'identifiant unique de session de l'utilisateur et les articles que l'utilisateur désire acheter, leurs prix respectifs et le prix total.

Attaque XSS

- Une XSS survient quand une application réinsère des données contrôlées par l'utilisateur dans une page sans encodage/échappement adapté. Le navigateur les interprète alors comme du code (JavaScript), ce qui permet à un attaquant de :
  - agir au nom de la victime (ses cookies/session/droits) ;
  - altérer l'interface (phishing in-page, formulaires piégés) ;
  - exfiltrer des données visibles par l'utilisateur.
- Trois familles : Réfléchi (réponse immédiate), Stocké (donnée persiste en BD), DOM-Based (injection côté client via JS qui manipule mal le DOM).

XSS - Attaque

- Fonctionnement (pas à pas)
  - Entrée non fiable : l'appli lit une valeur fournie par l'utilisateur (paramètre d'URL, champ de formulaire, commentaire…).
  - Réinsertion : cette valeur est renvoyée dans la page sans encodage (ex. dans du HTML ou un attribut).
  - Interprétation : le navigateur exécute ce contenu comme du code (car il n'a pas été traité comme du texte).
  - Impact : le script s'exécute dans le contexte du site (mêmes cookies/permissions) → actions au nom de l'utilisateur, collecte d'infos, etc.
  - Remarque : l'attaque réussit parce que la page ne distingue pas correctement "données" et "code".

XSS – attaque exemple

Exemple vulnérable (réfléchi)

- Serveur (template côté serveur) :
    - <!-- PROBLÈME : réinsère la saisie brute -->
    - Résultats pour : {{ user_input }}
  - Ici, si user_input contient du contenu interprétable, il sera exécuté par le navigateur.
- Côté client (DOM-Based XSS typique) :
    - // PROBLÈME : insère du contenu non fiable dans innerHTML
    - const params = new URLSearchParams(location.search);
    - document.getElementById('q').innerHTML = params.get('q') || ’’;
  - PROBLÈME :
    - {{ user_input }} n’est pas échappé (selon le moteur de template).
    - innerHTML interprète les balises et gestionnaires d’événements.

XSS – attaque exemple (suite)

- Correction problème → Échapper

- « Échapper » (ou *encoder*) en contexte XSS signifie transformer les caractères qui pourraient être interprétés comme du code (HTML/JS) afin qu'ils soient affichés comme du texte.
  But : empêcher qu'une donnée contrôlée par l'utilisateur devienne du JavaScript exécutable quand elle est réinsérée dans la page.
  - Sans échappement : <script>alert(1)</script> est exécuté.
  - Avec échappement : <script>alert(1)</script> est affiché littéralement.
  - Échapper & < > " ' / → & < > " ' /

XSS – attaque exemple (suite)

- Correction problème → Échapper

- Serveur : échapper selon le contexte HTML - La plupart des moteurs ont un auto-escape ; sinon, utilisez une fonction d'échappe.
    - <!-- Sécurittaire : échappement HTML -->
    - Résultats pour : {{ user_input | escape }} ment.)

- Client : insérer comme texte, pas comme HTML
    - const params = new URLSearchParams(location.search);
    - document.getElementById('q').textContent = params.get('q') || '';

Exemple attaque Web CSRF

Un site malveillant pousse un utilisateur déjà connecté à un autre site à envoyer une requête à son insu. Comme la requête part du navigateur de la victime, ses cookies de session sont envoyés automatiquement, et le site cible pense que l'action est légitime.
Conditions pour que ça marche

- La victime est connectée au site cible (cookie actif).
- Le site cible accepte des actions sensibles sans preuve d'origine (pas de token CSRF, pas de vérif d'Origin/Referer).
- Le navigateur envoie automatiquement les cookies du site cible lors de la requête déclenchée depuis le site malveillant.

Exemple attaque Web CSRF

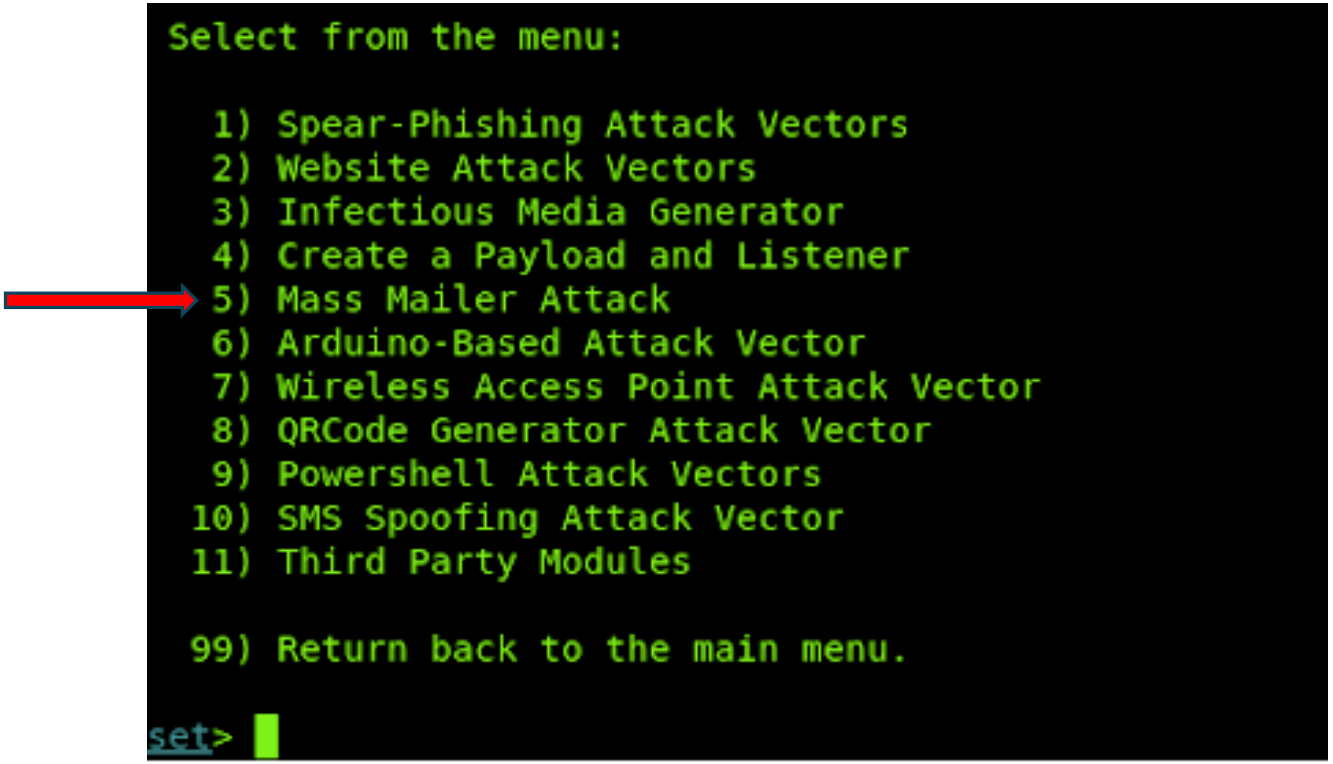
- Cross-site Request Forgery (CSRF)
  - ❑ L'attaquant force la victime à soumettre au serveur Web victime un formulaire préparé par l'attaquant.
  - ❑ Le formulaire soumis par l'attaquant permet de forger une requête.
  - ❑ Le serveur Web accepte les données du formulaire car provenant d'un utilisateur de confiance.
  - ❑ Utilisation d'expressions régulières pour détecter l'attaque

Hameçonnage courriels (‘sending phishing e-mails’) : **setoolkit**

(1)

Sélectionner 1 ‘social-engineering attacks’

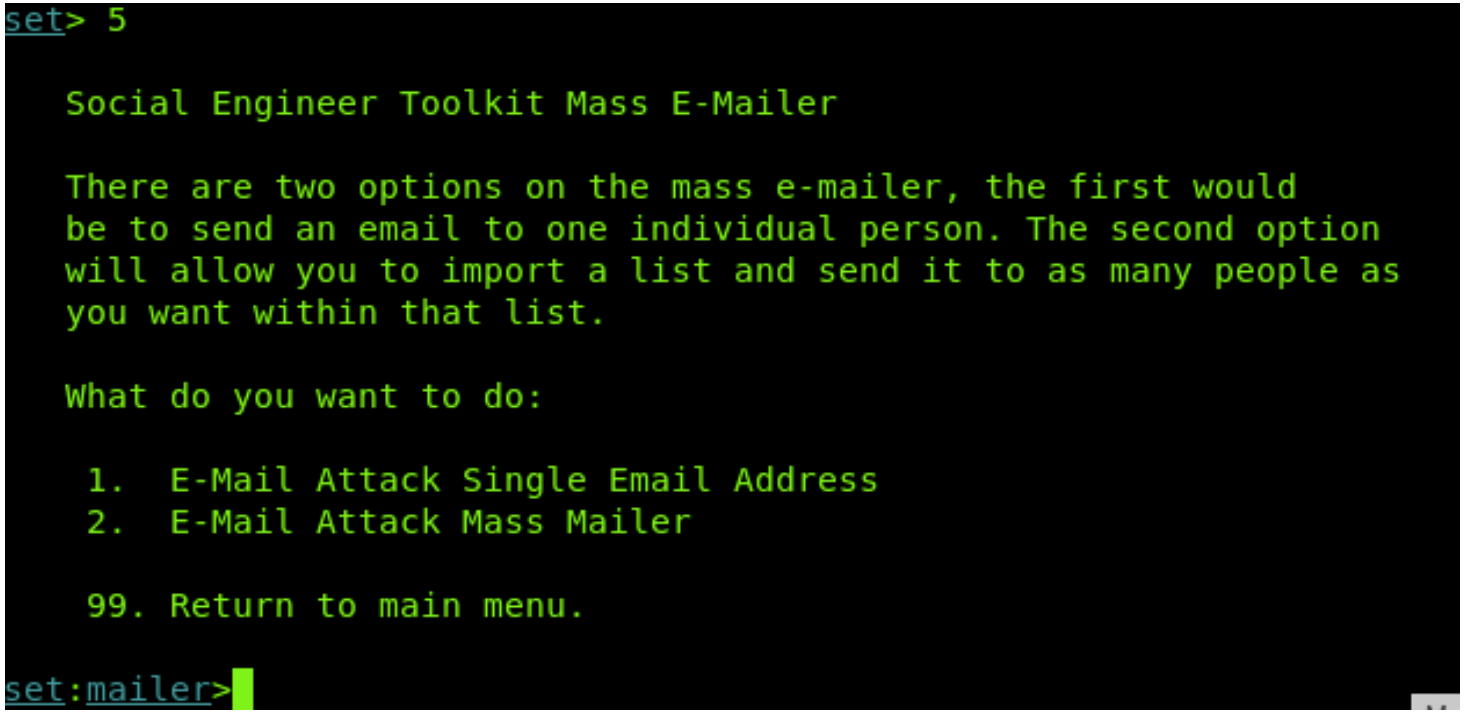
Hameçonnage courriels (‘sending phishing e-mails’) : **setoolkit**



Sélectionner 5 ‘Mass Mailer Attack’

(2)

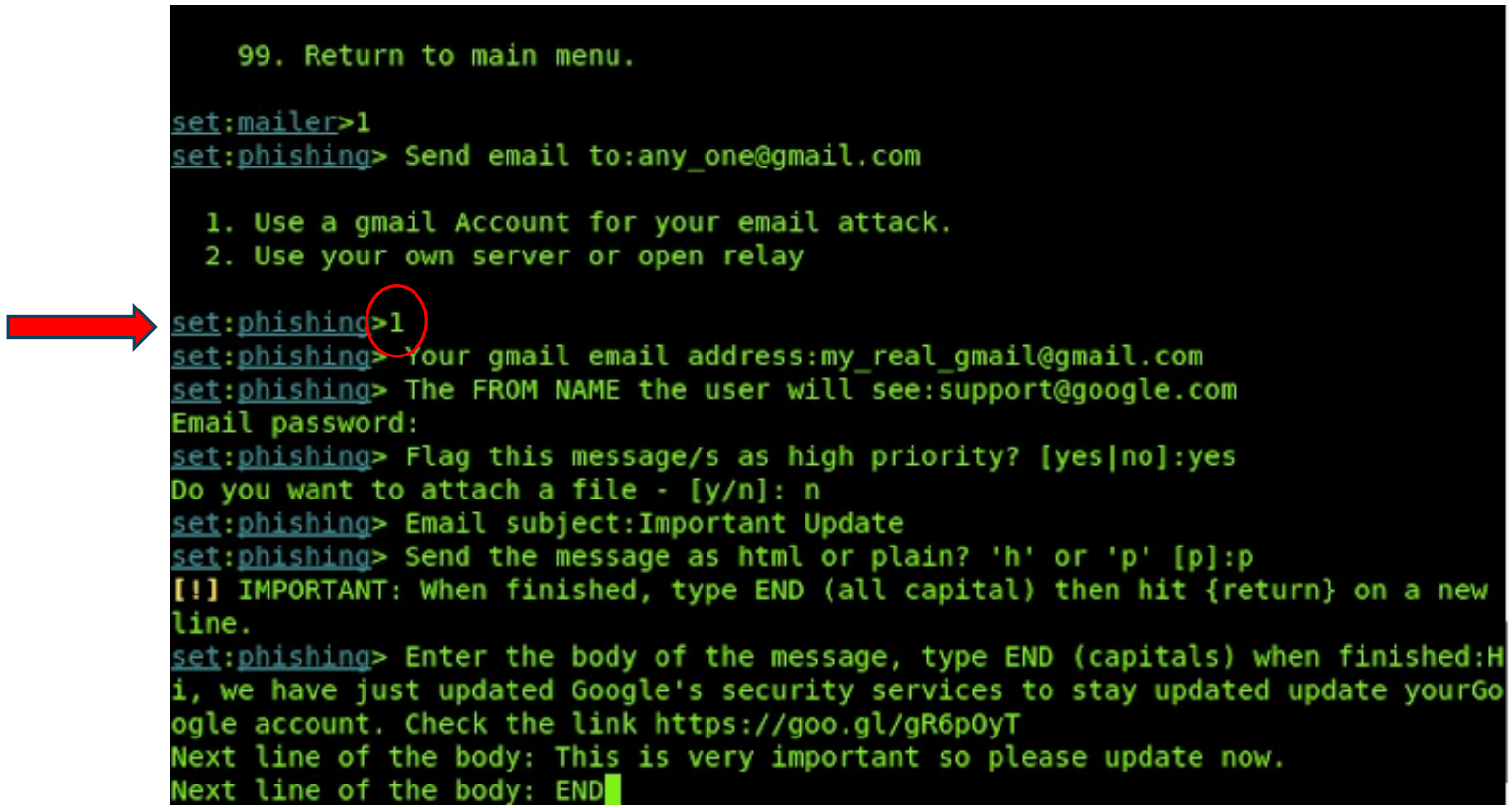
Hameçonnage courriels (‘sending phishing e-mails’) : **setoolkit**



(3)

Sélectionner 1 ‘Email attack Single Email Address’’

Hameçonnage courriels (‘sending phishing e-mails’) : **setoolkit**



(4)