Utilisation des outils de bug bounty en TI interne

P1, plz sir

switch @steakoverflow, *2023*

self

@swuitch
Team France 2019
https://0xswitch.fr

acceis

{ pentest interne / externe / AD, IoT, redteam, PASSI, CESTI, 8.6 }

pq ?

- bugbounty majoritairement scope web à travers internet
- pléthore d'outil et techniques "bug bounty" bien développés
- axé sur l'énumération DNS et services HTTP
- outils qui se *pipent* entre eux pour automatiser les tâches redondantes

- partie souvent délaissée lors d'un TI interne
- surface d'attaque supérieure dans un réseau d'entreprise
- pas besoin d'être exhaustif et pointu, on veut du **sang** 🩸
- ces outils peuvent nous aider sur certains **quicks wins**

Enumeration DNS

Cas d'usage

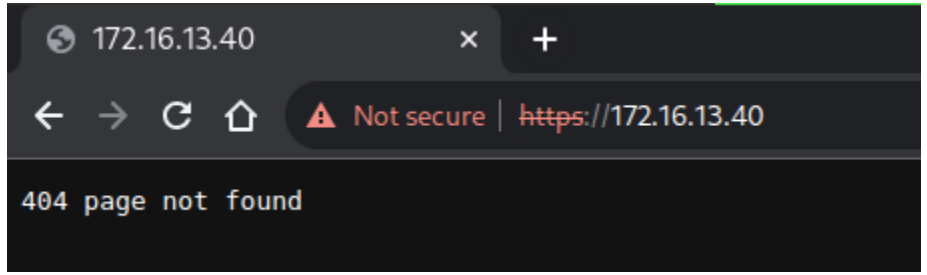
accès initial

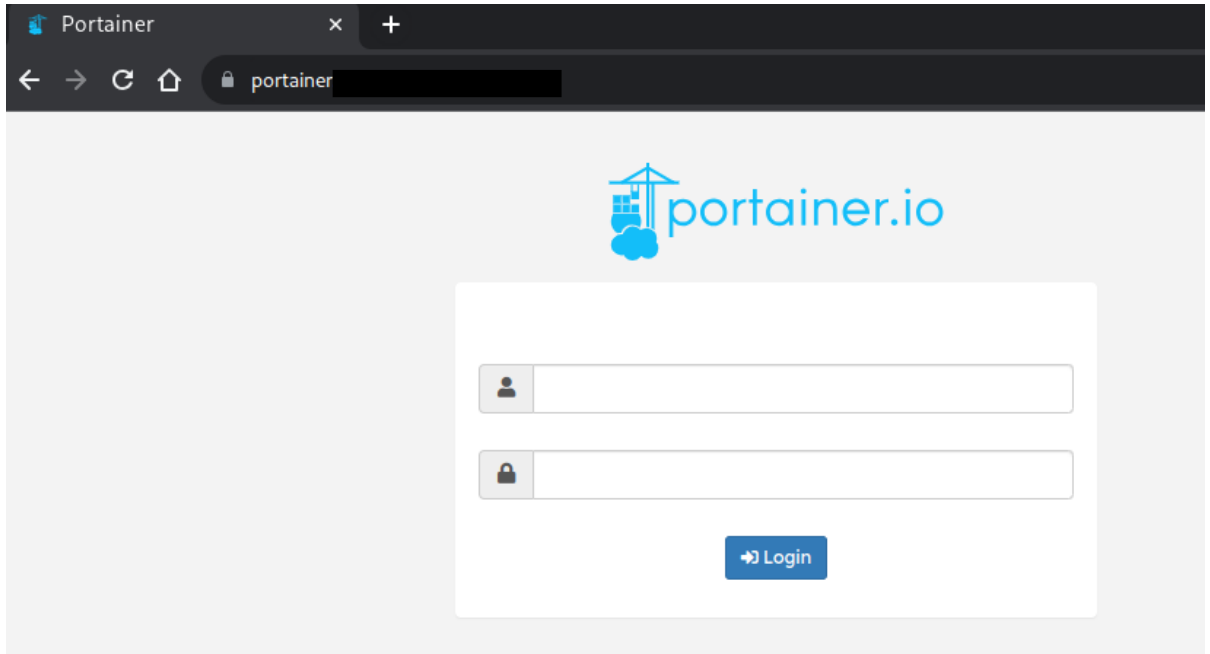
- nmap retourne des services web mais pas de vhost
  192.168.1.2:8080
  192.168.1.3:443
- pas de transfert de zone / adidns
- parfois vhosts dans certificats HTTPS

accès standard

- comptes peu privilégiés
- pas de liste des services internes
- liste des machines via LDAP

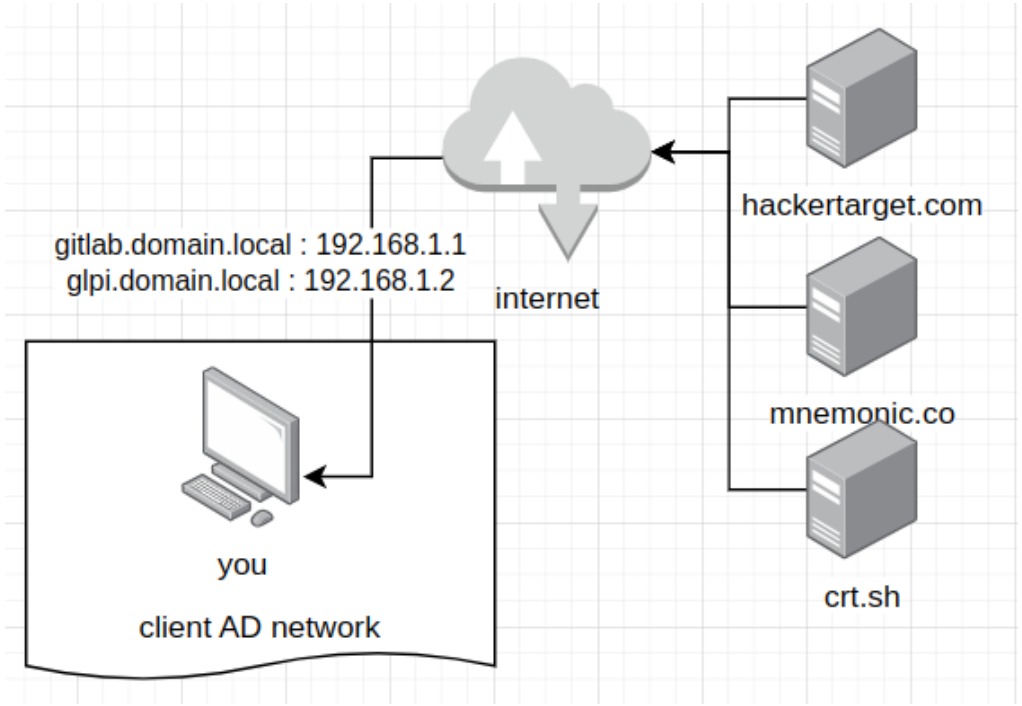
🏳 on passe à coté de services intéressants

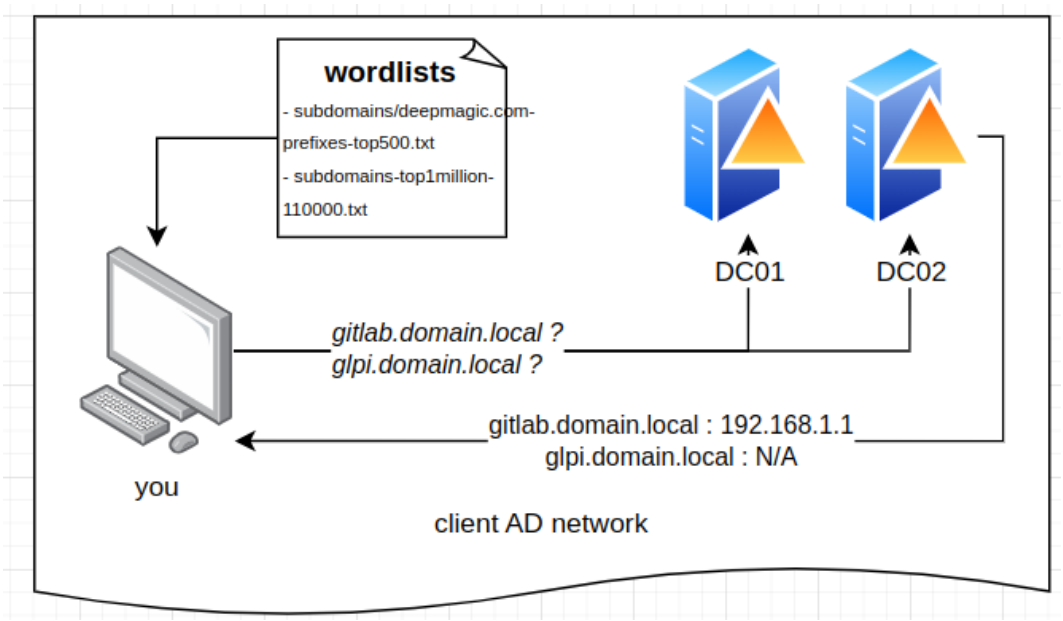




Énumération DNS

- **énumération passive**
- **énumération active**





Énumération passive

- rapide, quelques requêtes HTTP
- sous-domaines plus rare (shiny)
- plusieurs sources à maintenir + cycle de vie ***court***
- clés d'API / rate limiting
- résultats à parser (json,csv,txt) / dédupliquer
- résultats à valider en actif
- sous-domaines n'existant plus
- peu de résultats (car domaine pas sur internet)
- outils qui se recoupe beaucoup

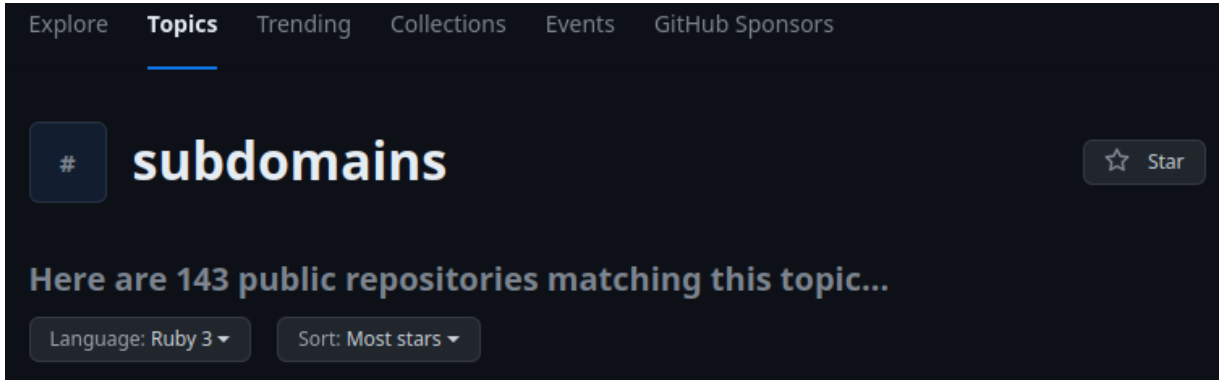
Énumération passive

APIs à exploiter

- https://api.hackertarget.com/reverseiplookup/?q={{target}}
- https://api.mnemonic.no/pdns/v3/{{target}}|
- https://tls.bufferover.run/dns?q={{target}}
- https://crt.sh/?q=%.{{target}}&output=json
- https://api.riskiq.net/pt/v2/dns/passive
- https://urlscan.io/api/v1/search/?q={{target}}
- Sonarr DB (rapid7)
- Chaos (pdiscovery)
- DNSdumpster.com
- subdomainfinder.c99.nl
- Censys
- Binaryedge
- Commoncrawl
- Alienvault
- Virustotal
- Web archive
- Securitytrails.com

outils

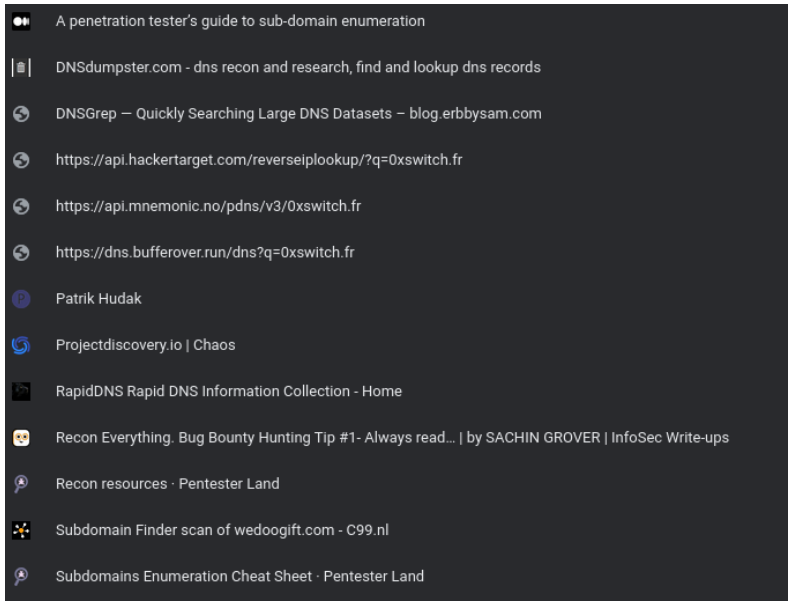
- Sudomy (API)
- Subfinder (API)
- Github-domains (github API)
- Asset-finder (API)
- Gauplus / getallurls / gau (API)
- amass
- ...



Énumération passive

`subfinder` can be used right after the installation, however the following services require configuring API keys to work:

BeVigil, BinaryEdge, BufferOver, C99, Censys, CertSpotter, Chaos, Chinaz, DnsDB, Fofa, FullHunt, GitHub, Intelx, PassiveTotal, quake, Robtex, SecurityTrails, Shodan, ThreatBook, VirusTotal, WhoisXML , ZoomEye API, dnsrepo, Hunter, Facebook



Sources

Please feel free to issue pull requests with new sources! :)

Implemented

- crt.sh
- certspotter
- hackertarget
- threatcrowd
- wayback machine
- dns.bufferover.run
- facebook
  - Needs `FB_APP_ID` and `FB_APP_SECRET` environment
  - You need to be careful with your app's rate limits
- virustotal
  - Needs `VT_API_KEY` environment variable set (https:/
- findsubdomains
  - Needs `SPYSE_API_TOKEN` environment variable set (t page, and you also get "25 unlimited requests") — (h

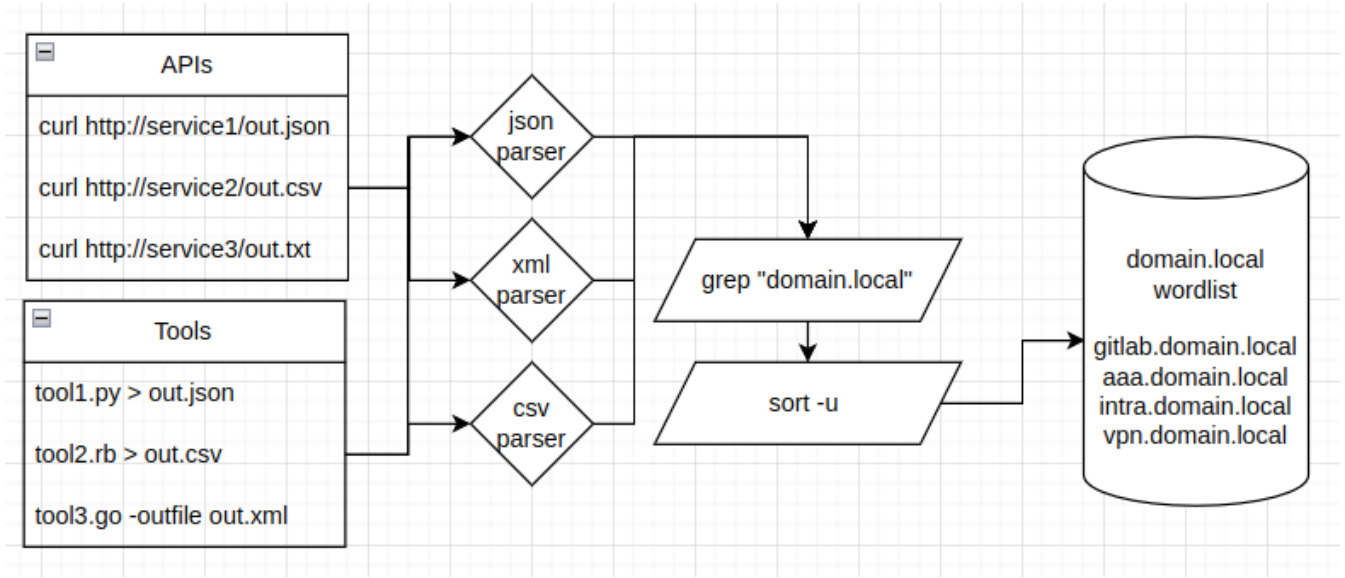
getallurls (gau) fetches known URLs from AlienVault's Open Threat Exchange, the Wayback Machine, Common Crawl, and URLScan for any given domain. Inspired by Tomnomnom's waybackurls.

these well-curated 22 third-party sites:

```
https://censys.io
https://developer.shodan.io
https://dns.bufferover.run
https://index.commoncrawl.org
https://riddler.io
https://api.certspotter.com
https://api.hackertarget.com
https://api.threatminer.org
https://community.riskiq.com
https://crt.sh
https://dnsdumpster.com
https://docs.binaryedge.io
https://securitytrails.com
https://graph.facebook.com
https://otx.alienvault.com
https://rapiddns.io
https://spyse.com
https://urlscan.io
https://www.dnsdb.info
https://www.virustotal.com
https://threatcrowd.org
https://web.archive.org
```

Énumération passive

TLDR :



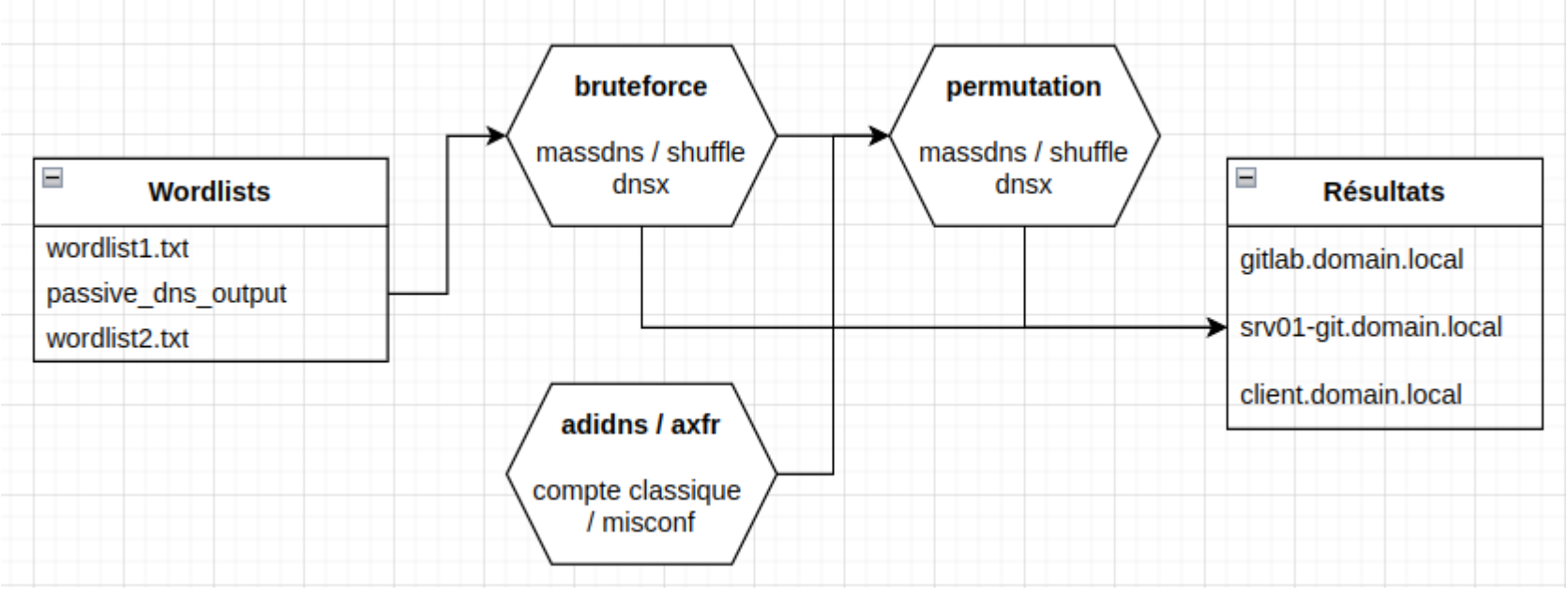

```
  steps:
    # subdomains enumeration
    fetch_subdomains_passive:
      config:
        dir: '{{enumeration_dir}}'
        force: false
        sequential: false
      actions:
#         - enumeration.dnsrecon.passive:
#           - outfile: '{{dir}}/dnsrecon-default.txt'
#           - outfile_json: '{{dir}}/dnsrecon-default.json'
#           - cmd: '§dnsrecon.passive§ > {{outfile}}'
        - enumeration.subfinder.default:
          - outfile: '{{dir}}/subfinder-default.txt'
          - cmd: '§subfinder.default§ -o {{ outfile }}'
        - enumeration.github-domains.default:
          - outfile: '{{dir}}/githubdomains-default.txt'
          - cmd: '§github-domains.default§ > {{ outfile }}'
        #- enumeration.amass.default:
        #  - outfile: '{{dir}}/amass-default.txt'
        #  - cmd: '§amass.passive-domains§ -o {{outfile}}'
        - enumeration.sudomy.default:
          - outdir: '{{dir}}/sudomy'
          - outfile: '{{outdir}}/Sudomy-Output/{{target}}/subdomain.txt'
          - cmd: 'cd /tools/sudomy && /tools/sudomy/§sudomy.no-probe§ -o {{outdir}}'
        - enumeration.assetfinder.default:
          - outfile: '{{dir}}/assetfinder-default.txt'
          - cmd: '§assetfinder.default§ > {{ outfile }}'
      # - enumeration.sublist3r.default: sublist3r.default
      # - enumeration.crobat.default: crobat.subdomains
        - enumeration.hackertarget.default:
          - outfile: '{{dir}}/hackertarget.txt'
          - cmd: 'curl -s https://api.hackertarget.com/reverseiplookup/?q={{target}} | grep -v "API count exceeded" > {{ outfile
}} || true'
        - enumeration.mnemonic.default:
          - outfile: '{{dir}}/mnemonic.txt'
          - cmd: 'curl -s https://api.mnemonic.no/pdns/v3/{{target}}| jq ".data[].query" -r > {{ outfile }}'
        - enumeration.bufferover.default:
          - outfile: '{{dir}}/bufferover.txt'
          - cmd: 'curl -s https://tls.bufferover.run/dns\?q\={{target}} -H "x-api-key: $({{hacks_dir}}/commands/get_api_keys.py
{{api_keys_path}} bufferoverun)" |  jq -r ".Results[]" | cut -d "," -f 5 > {{ outfile }}'
        - enumeration.crt.default:
          - outfile: '{{dir}}/crt.txt'
          - cmd: 'curl -s "https://crt.sh/?q=%.{{target}}&output=json" | jq ".[].name_value" -r | sort -u > {{ outfile }}'
        - enumeration.gauplus.default:
          - outfile: '{{dir}}/gauplus.txt'
          - cmd: 'gauplus -t {{threads}} -random-agent -subs {{target}} | unfurl -u domains | anew {{ outfile }}'
        - enumeration.oneforall.default:
          - outfile: '{{dir}}/oneforall.txt'
          - cmd: 'python /tools/oneforall/oneforall.py --target {{target}} --brute False --alive False --dns False --req False
run --path /tmp/oneforall.csv ; cat /tmp/oneforall.csv | cut -d "," -f 6 > {{ outfile }}'
        - enumeration.riskiq.all:
          - outfile: '{{dir}}/riskiq-all.txt'
          - cmd: "curl -s -u $( {{hacks_dir}}/commands/get_api_keys.py {{api_keys_path}} riskiq | tr '\\n' ':' | sed 's/.$//')
https://api.riskiq.net/pt/v2/dns/passive -XGET -H 'Content-Type: application/json' --data '{\"query\": \"{{target}}\"}' | jq -r '.
results[].value' > {{ outfile }} "
        - enumeration.urlscan.default:
          - outfile: '{{dir}}/urlscan-all.txt'
          - cmd: "curl -s https://urlscan.io/api/v1/search/?q={{target}} | jq '.results[].task.domain' -r | sort -u  > {{ outfile
}}"
```

Énumération active

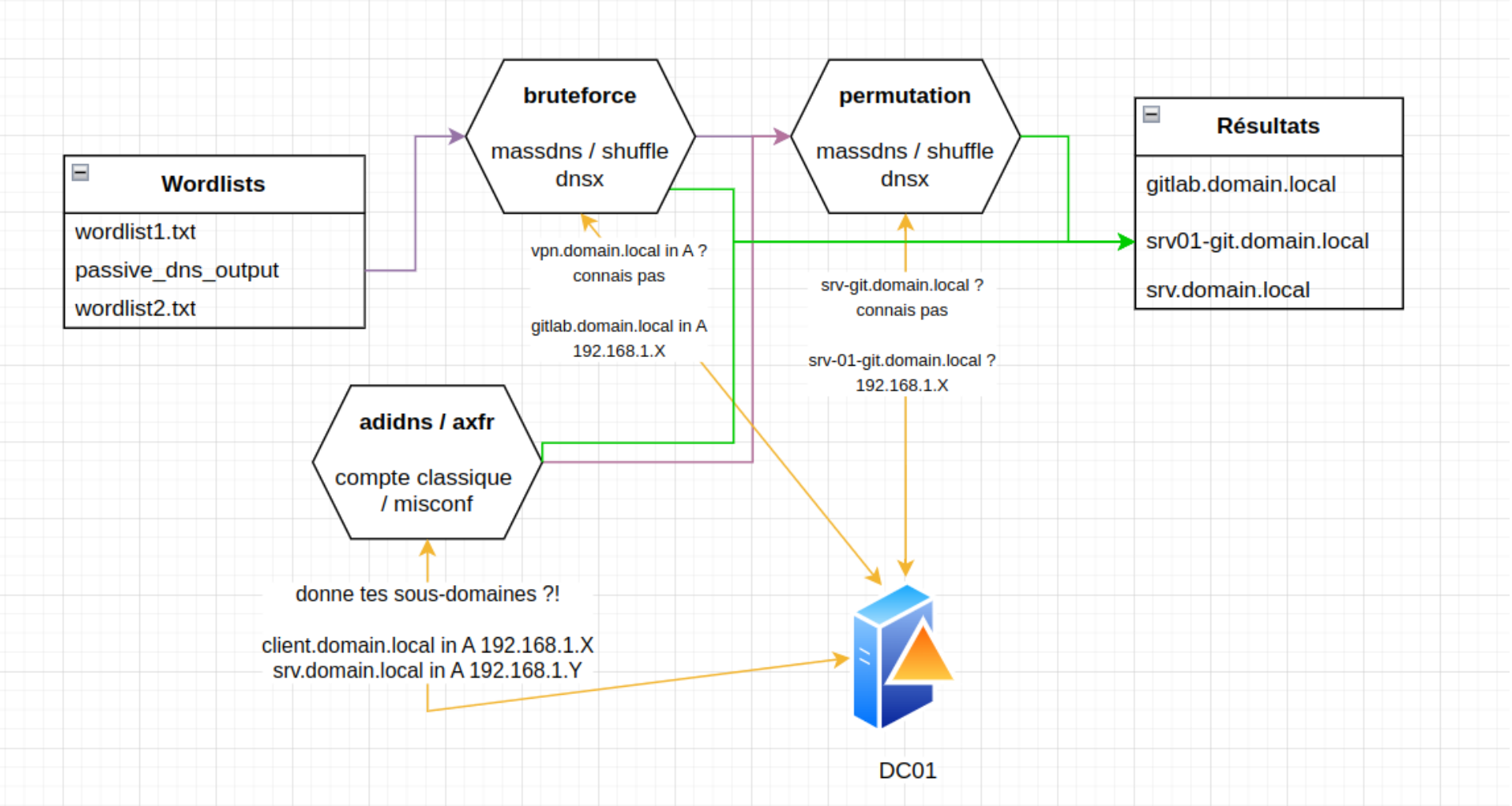
- moins de faux positif
- simple, efficace, pas cher (pas de clé API)
- résultats exploitables rapidement
- domaines les plus connus / génériques
- long
- charge réseau / machine conséquente
- dépend du nombre de DC (résolveur DNS)
- pas *vraiment* discret
- wordlists existantes pas dingues (wordlist spécifique interne à maintenir ?)

Énumération active

- bruteforce
- permutation
- transfert de zone / adidns



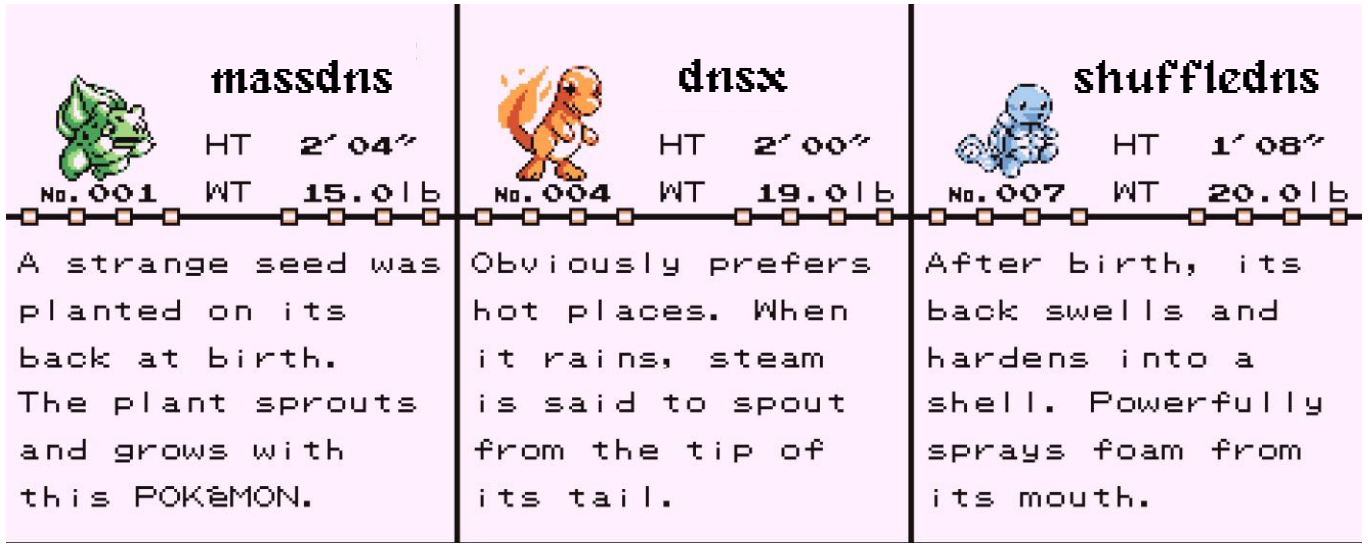
Énumération active



Énumération active

bruteforce

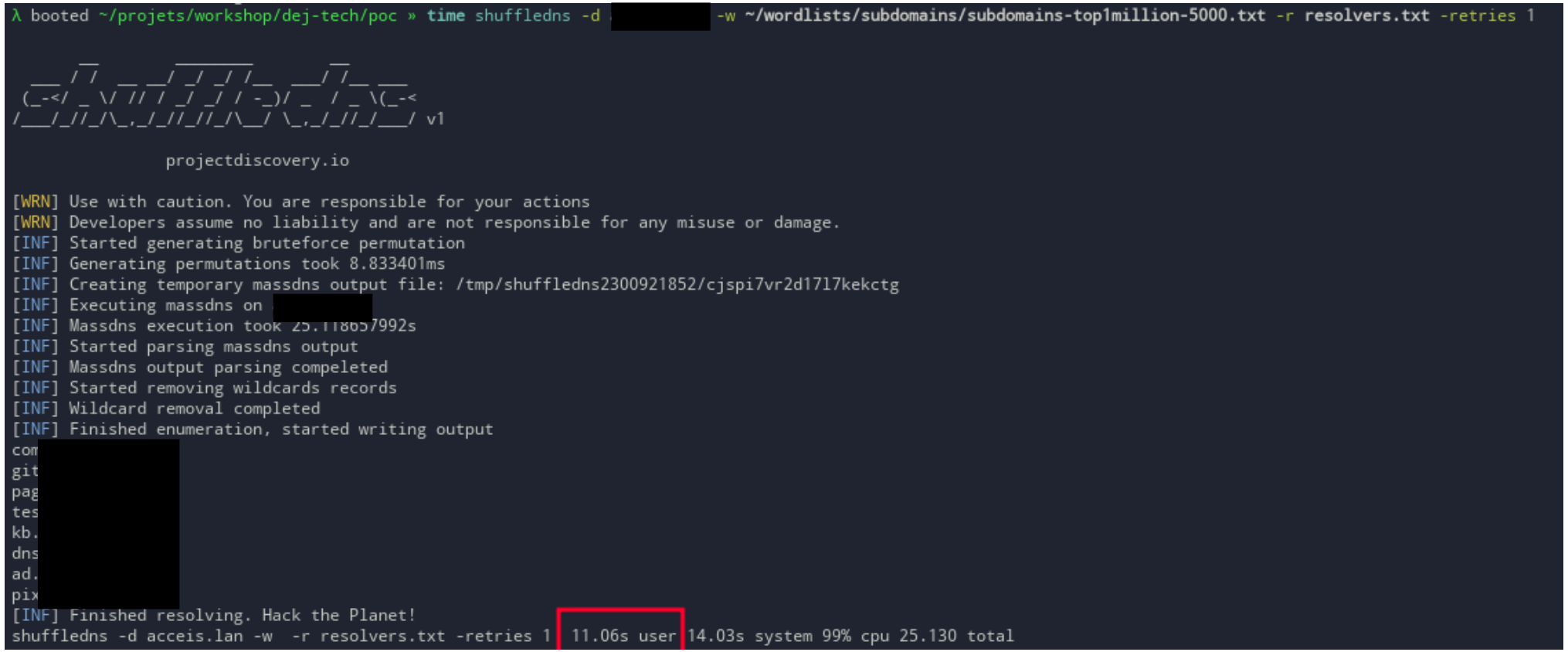
- attention au nombre requêtes ⟶ utiliser plusieurs résolveurs (DC)
- les listes DNS actuelles pas faites pour sous-domaines **internes**
- utiliser listes courtes et connues (git. / gitlab. / glpi. / intra.)



Énumération active

pdiscovery - shuffledns

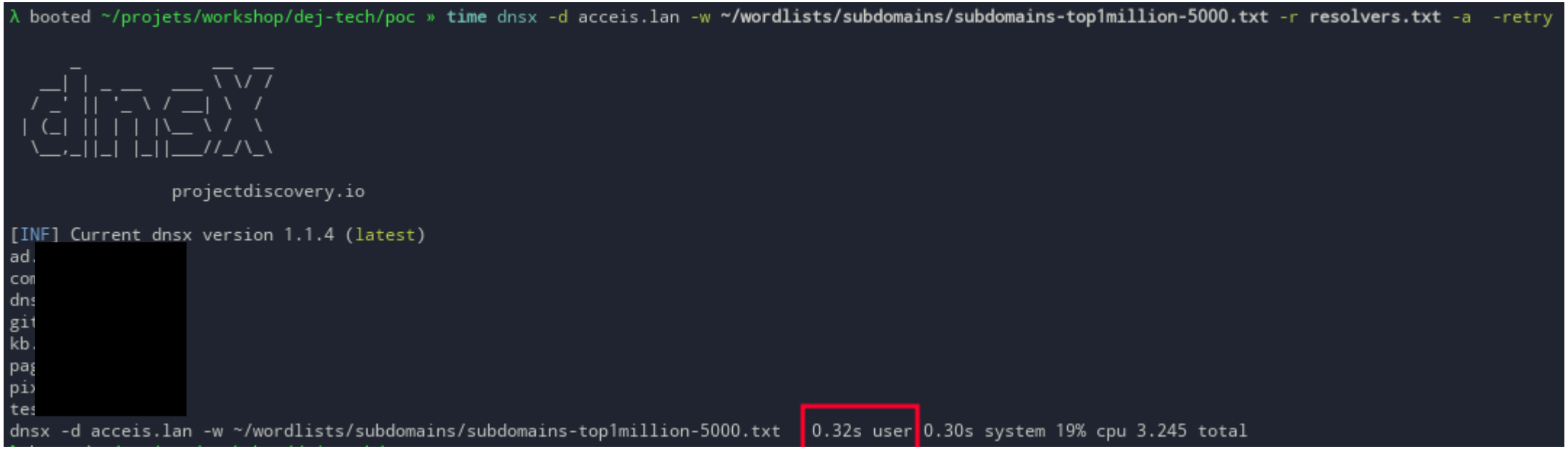
- simple massdns wrapper



Énumération active

pdiscovery - dnsx

- toolkit DNS complet
- DNS resolution / bruteforce
- A, AAA, CNAME, PTR etc



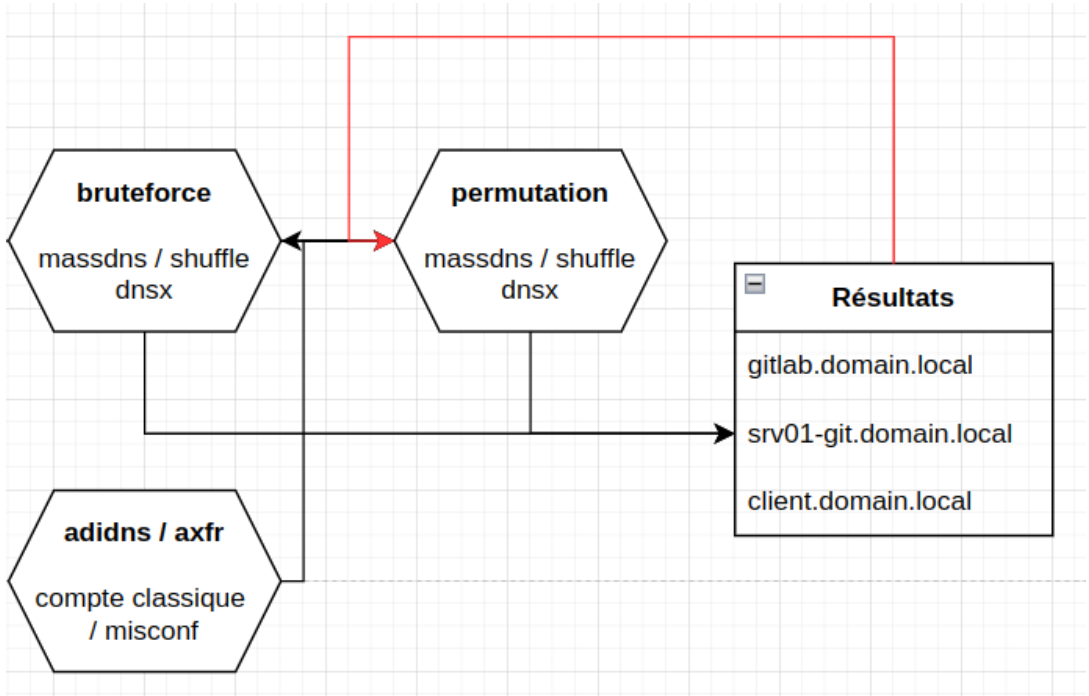
Énumération active

permutations

- srv.domain.local
- srv01.domain.local
- srv-01.domain.local

beaucoup d'outils

- https://github.com/subfinder/goaltdns
- https://github.com/Josue87/gotator
- https://github.com/cramppet/regulator
- https://github.com/projectdiscovery/alterx
- https://github.com/bp0lr/dmut
- https://github.com/ProjectAnte/dnsgen



overkill

- plusieurs algorithmes dont *ranking*
- beaucoup de test pour peu de résultats
- a tester quand déjà beaucoup de domaine ont été identifiés

Énumération active

transfert de zone

- exploite la réplication des entrées d'un maître vers ses esclaves
- dépend d'une mauvaise configuration
- arrive parfois en TI interne

```
λ booted ~ » dig axfr poc.local @192.168.1.5

; <<>> DiG 9.18.19 <<>> axfr poc.local @192.168.1.5
;; global options: +cmd
poc.local.              60      IN      SOA     poc.local. root.poc.local. 20160505 10800 3600 604800 3600
poc.local.              60      IN      NS      ns.poc.local.
flag.poc.local.         60      IN      TXT     "thegame"
gitlab.poc.local.       60      IN      A       192.168.1.17
glpi.poc.local.         60      IN      A       192.168.1.13
intervention.poc.local. 60      IN      A       10.0.0.1
ns.poc.local.           60      IN      A       192.168.0.15
poc.local.              60      IN      SOA     poc.local. root.poc.local. 20160505 10800 3600 604800 3600
;; Query time: 0 msec
;; SERVER: 192.168.1.5#53(192.168.1.5) (TCP)
;; WHEN: Fri Oct 06 17:00:10 CEST 2023
;; XFR size: 8 records (messages 1, bytes 274)
```

Énumération active

adidns

- Active Directory-Integrated DNS
- contient la liste des entrées DNS du domaine
- https://dirkjanm.io/getting-in-the-zone-dumping-active-directory-dns-with-adidnsdump/
- nécessite simplement un compte du domaine
- liste des serveurs et workstation
- rarement de liste de service mais permet d'accéder à un service via nom de machine vs IP

```
$ adidnsdump -u <domain>\\<user> --forest <dc>

$ ldapsearch -H ldap://<dc> -D <domain>\\<users> -w <password> -b
"CN=MicrosoftDNS,DC=DomainDnsZones,DC=<dn>,DC=<dn>" "(name=*)" name | grep 'name:'
```

Enumeration web

Énumération web

pourquoi

- ne pas se focus sur Active Directory uniquement
- peut aider à obtenir des comptes du domaine

- beaucoup, *beaucoup*, d'applicatif utilisés
  - intranet, ticketing, LMS, blogs, ...
  - interfaces des équipements (switch, caméra, imprimante, ...)
  - interfaces d'administration (ESXi, HP iLO, fortinet, QNAP, ...)
  - besoins métiers (gitlab, jenkins, ERP, Sonarqube, ...)
  - une partie n'est même pas connues des équipes du SI

- pas exposés sur internet donc (faux) sentiment de protection
  - mot de passe par défaut
  - version pas à jour voir obsolète
  - pas de configuration de sécurité (directory listing, phpinfo, metrics activées)

- avec un SSO / connexion LDAP
  - vulns authentifiées deviennent possibles via comptes du domaine
  - MITM pour extraire comptes de connexion AD si pas de chiffrement

Énumération web

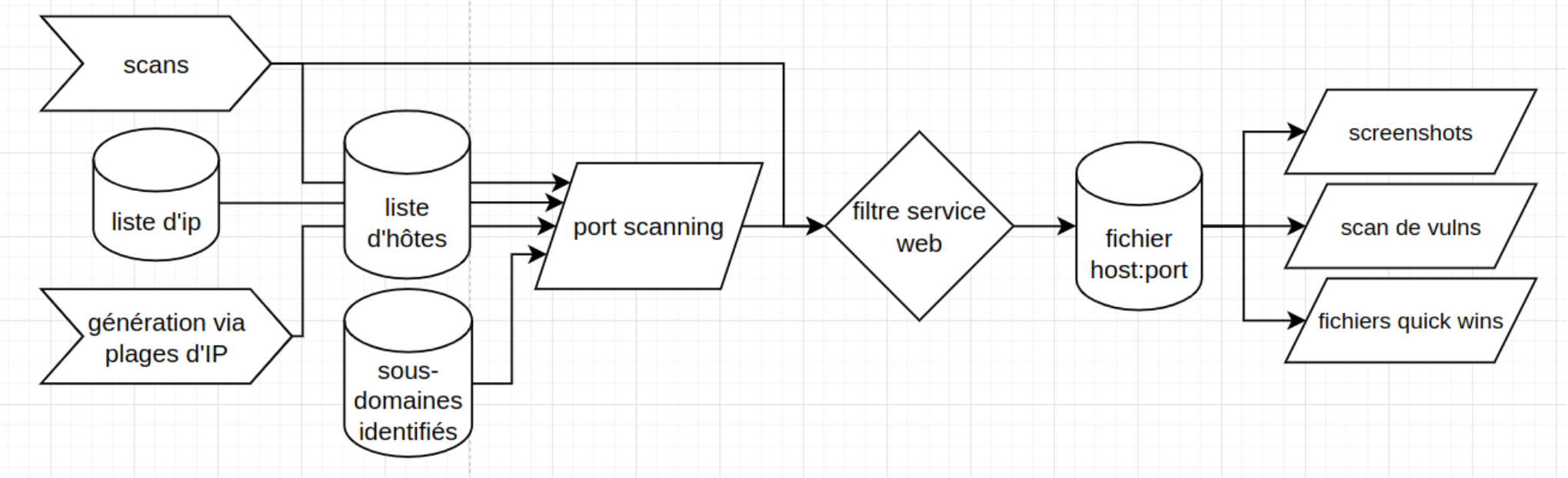
quick wins

- **arbitrary file read** ⟶ lecture du fichier de configuration avec identifiants AD
- **dir listing** ⟶ sauvegardes serveurs web, fichiers keepass, fichiers métiers sensibles
- **mot de passe par défaut** ⟶ accès caméra, switch (changement des VLANs), imprimante (identifiants AD)
- **pas à jour** ⟶ EXSi RCE et bypass d'auth, zabbix bypass d'auth,
  RCE sur machine windows ⟶ dump SAM, LSA, LSASS et password spraying

Énumération web

méthodologie

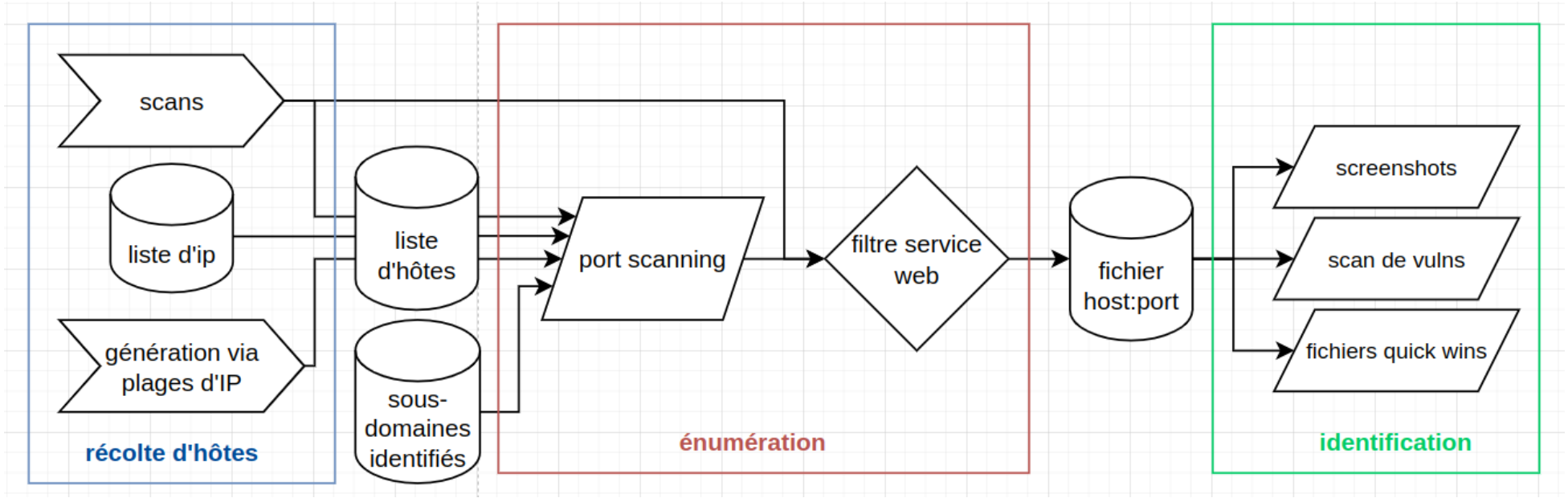
- génération des hosts possibles
- identification ports ouverts et filtrage services web
- identification
  - produits / interfaces
  - vulnérabilités
  - fichiers sensibles



Énumération web

méthodologie

- génération des hosts possibles
- identification ports ouverts et filtrage services web
- identification
  - produits / interfaces
  - vulnérabilités
  - fichiers sensibles



Récolte d'hôtes

connus ou découverts

- IP fournies par le client
- scans réseaux
- wireshark
- génération des plages d’IP

```
nmap -sL -n 192.168.1.0/24 | awk '/Nmap scan report/{print $NF}'

# 192.168.1.0
# ...
# 192.168.1.255
```

Énumération

récupération des cibles

- énumération des services (ports ouverts)
- filtre des services intéressant (services HTTP)

deux possibilités

- via un scan de l'ensemble des ports TCP + filtrage des services HTTP
- via un scan HTTP sur un nombre de port défini

comment savoir si c'est un service HTTP ?

- le numéro de port ne fait pas fois
- détection de nmap pas *folle* (pas de mention si HTTP)

Énumération

httprobe / httpx

- domaines / IP en input
- choix des ports à tester
- liste de services HTTP en sorti

```
λ booted /tmp » time generate_ip_list 192.168.1.0/24 | httprobe
http://192.168.1.5
https://192.168.1.5
https://192.168.1.119
http://192.168.1.254
https://192.168.1.254
generate_ip_list 192.168.1.0/24  0.03s user 0.01s system 110% cpu 0.031 total
httprobe  0.09s user 0.20s system 0% cpu 1:19.07 total
```

```
λ 130 booted /tmp » time generate_ip_list 192.168.1.0/24 | httpx-pd -duc -silent
https://192.168.1.119
https://192.168.1.254
https://192.168.1.5
generate_ip_list 192.168.1.0/24  0.02s user 0.01s system 112% cpu 0.034 total
httpx-pd -duc -silent  0.81s user 0.38s system 1% cpu 1:08.70 total
```

projectdiscovery

httpx

httpx is a fast and multi-purpose HTTP toolkit that allows running multiple probes using the retryablehttp library.

projectdiscovery.io

5,8k stars 706 forks

tomnomnom

httprobe

Take a list of domains and probe for working HTTP and HTTPS servers

2,6k stars 486 forks

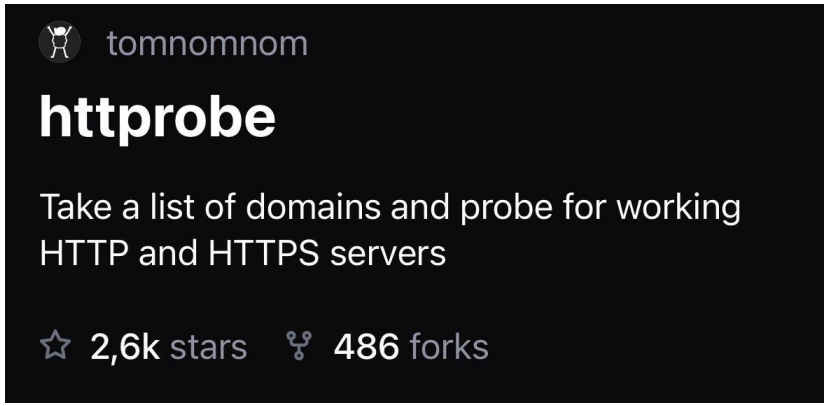
Énumération

httprobe

- 80 et 443 par défaut
- -p http:8000 –p https:8000 –p http:9443 –p https:9443 🤧
- ne fait que ça mais le fait bien !

protip alias :

```
httprobe_ports () {
        httprobe -t 2000 -c 40 \
        -p http:80 -p https:443 \
        -p http:8443 -p https:8443 \
        -p http:8080 -p https:8080 \
        -p http:9000 -p https:9000 \
        -p http:4000 -p https:4000 \
        -p http:5000 -p https:5000 \
        -p http:8000 -p https:8000 \
        -p http:9443 -p https:9443 \
}
```



Énumération

httpx-pd

- 80 et 443 par défaut
- -p http:80,8000,https:443,9443 🥹
- fait également plein d'autres choses :
  - parsing réponse HTTP (taille, code, location, etc)
  - screenshot
  - favicon hash
  - headers / banner

```
λ hackbox /tmp » cat http.txt | httpx -sc -location -favicon -title -server -tech-detect

    __    __  __       _  __
   / /_  / /_/ /_____ | |/ /
  / __ \/ __/ __/ __ \|   /
 / / / / /_/ /_/ /_/ /   |
/_/ /_/\__/\__/ .___/_/|_|
             /_/

                projectdiscovery.io

[INF] Current httpx version v1.3.5 (latest)
http://192.168.1.5:5357 [501] [] [Error response] [BaseHTTP/0.6 Python/3.7.3] [Python:3.7.3]
http://192.168.1.5:9091 [301] [/transmission/web/] [] [Transmission]
http://192.168.1.5:8201 [200] [] [Backup] [Tiny WebServer] [-678545852]
http://192.168.1.5:1184 [412] [] [] [UPnP/1.0 DLNADOC/1.50 Platinum/1.0.5.13]
https://192.168.1.5:25565 [404] [] [] []
http://192.168.1.5:32469 [404] [] [] [UPnP/1.0 DLNADOC/1.50 Platinum/1.0.5.13]
https://192.168.1.5:25565 [404] [] [] []
https://192.168.1.5 [404] [] [] []
https://192.168.1.5 [404] [] [] []
https://192.168.1.5:32400 [200] [] [] [] [-895890586]
http://192.168.1.5:1111 [200] [] [Document] [SimpleHTTP/0.6 Python/3.7.3] [Bootstrap:5.3.2,Python:3.7.3,SimpleHTTP:0.6]
http://192.168.1.5:8000 [200] [] [openmediavault control panel - [redacted]] [nginx] [Nginx] [-693082538]
https://192.168.1.5:32400 [200] [] [] [] [-895890586]
https://192.168.1.5:8384 [401] [] [] [] [Basic]
https://192.168.1.5:8384 [401] [] [] [] [Basic]
http://192.168.1.5:8200 [200] [] [MiniDLNA 1.2.1] [Debian DLNADOC/1.50 UPnP/1.0 MiniDLNA/1.2.1] [Debian]
λ hackbox /tmp » _
```

projectdiscovery

httpx

httpx is a fast and multi-purpose HTTP toolkit that allows running multiple probes using the retryablehttp library.

projectdiscovery.io

☆ 5,8k stars 706 forks

Énumération

scan de l'ensemble des ports
scan via nmap ou équivalent (masscan)

```
1 echo "192.168.1.1 192.168.1-10.0/24 10.10.0.0/16" | tr "" "\n" > hosts.txt
2
3 sudo nmap -sS -p 80,443,8000,8080,9000,9001,4000,5000,8443,9443,9200,(...) \
4 -iL host.txt \
5 -oA out.nmap
```

résultat : liste d'hôtes up + liste de ports ouvert
problème : lesquels sont des services HTTP ?

prendre la liste de tous les ports ouverts et donner à manger à httprobe / httpx

```
1 gnmap_hosts.py out.nmap.gnmap > ip_up.txt
2 # 192.168.1.x
3 # 192.168.1.y
4
5 gnmap_ports.py out.nmap.gnmap > seen_ports.txt
6 # 80
7 # 443
8
9  cat ip_up.txt | httprobe $(cat seen_ports | port_to_httprobe.py) | tee http.txt
```

permet d'exploiter des scans TCP déjà réalisés et de ne pas oublier de ports

Énumération

```
λ 130 booted /tmp » echo 192.168.1.5 | httprobe $(gnmap_all_open_ports 192.168.1.5.gnmap | port_to_httprobe.py) | remove_duplicate_http  | tee http.txt
http://192.168.1.5:32400
http://192.168.1.5:1111
http://192.168.1.5
http://192.168.1.5:25565
http://192.168.1.5:1184
http://192.168.1.5:32469
https://192.168.1.5
http://192.168.1.5:5357
https://192.168.1.5:25565
http://192.168.1.5:8200
https://192.168.1.5:32400
http://192.168.1.5:9091
http://192.168.1.5:8201
http://192.168.1.5:8000
https://192.168.1.5:8384
http://192.168.1.5:8384
```

Énumération

scan sur un nombre de port définis

- nos hôtes en entrées
- spécifier la liste des ports à vérifier en HTTP et HTTPS

```
1 echo "10.10.10.10" > hosts.txt
2 nmap -sL -n 10.10.0.0/16 | awk '/Nmap scan report/{print $NF} | anew hosts.txt
3
4 cat hosts.txt | httprobe -p http:80 -p http:443 -p http:8443 -p http:8080 -p http:9000 -p http:4000 -p
  http:5000 -p http:8000 -p http:9443 -p https:80 -p https:443 -p https:8443 (...) | tee http.txt
5
```

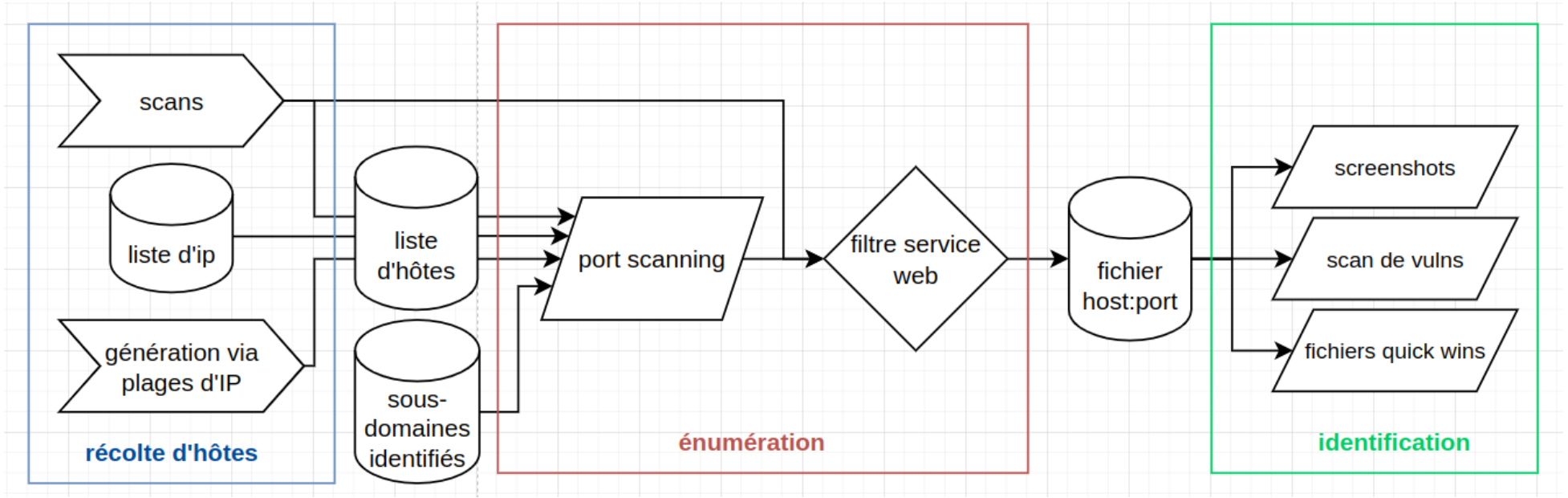
- on obtient également une liste de services HTTP en sorti
- un seul scan de chaque hôte est réalisé

- plus lent (pas d'optimisation de scans hormis threading)
- limité à une liste de port (on ne trouvera pas les services sur des ports exotiques)

Énumération résultat

fichier contenant protocole://host:port

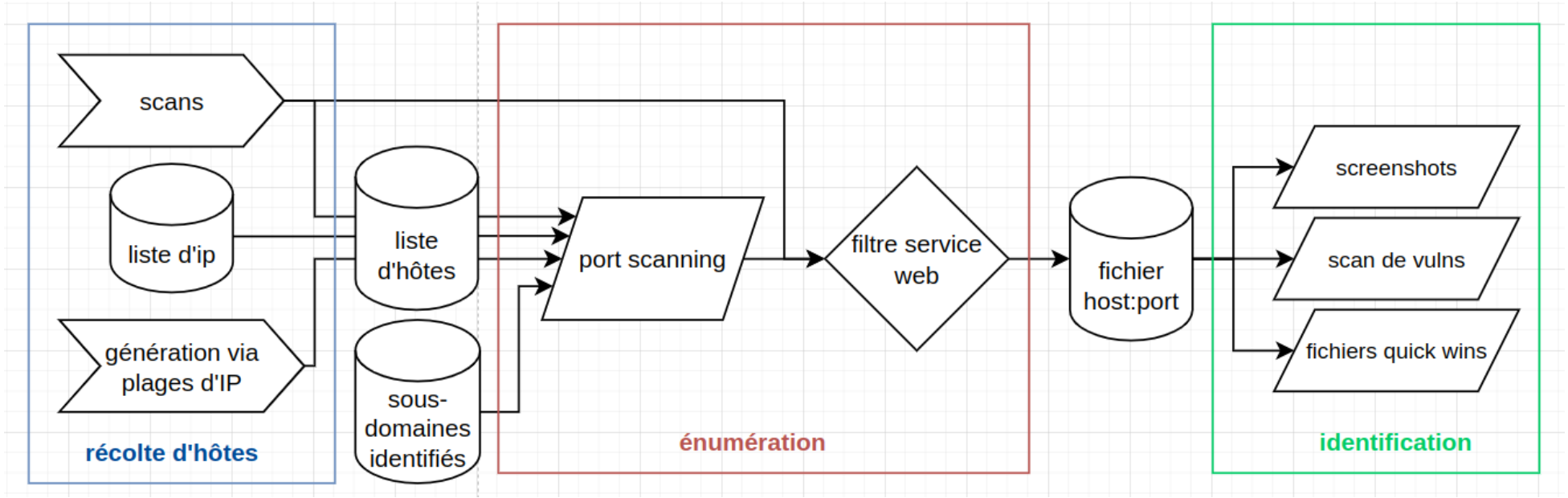
- uniquement des services web
- aucune idée de ce qu'il y a derrière
- certains sont pas accessibles (HTTP 401 / 403)
- fichier exploitables par d'autres outils



Identification

plusieurs tâches

- quick wins via des screenshots
- scan de vulnérabilité (creds par défaut, CVE, panel d'administration)
- fichiers quick wins (install.php, db.sql, etc)
- récolte d'informations supplémentaire httpx



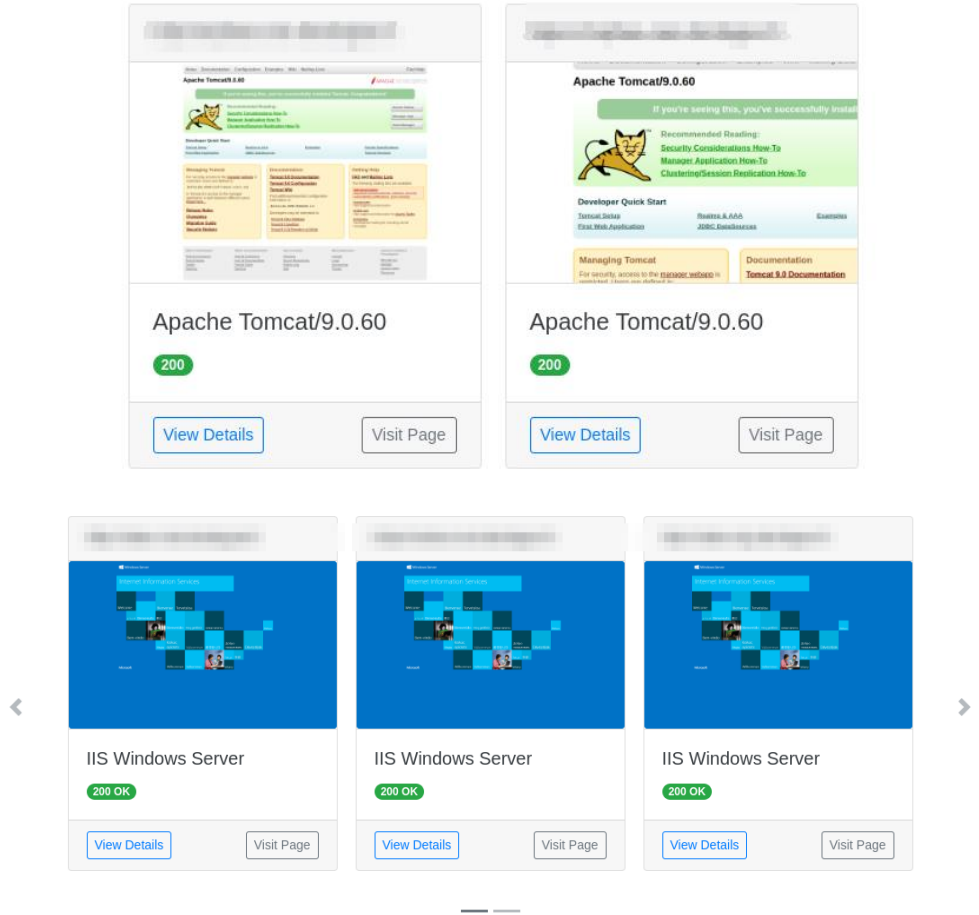
screenshots

pourquoi

- centaines de services accessibles
- certains sont reconnaissables via un coup d'œil
  - phpinfo
  - mire d'authentification
  - directory listing
- quick wins for the win

```
λ booted ~/projets/workshop/bb-interne » cat http.txt
https://10.10.11.2:443
http://10.10.11.2:80
https://10.10.11.2:9080
http://10.10.11.5:1947
http://10.10.11.10:443
https://10.10.11.10:443
http://10.10.11.15:15443
https://10.10.11.15:15443
http://10.10.11.15:30028
http://10.10.11.15:30029
http://10.10.11.15:443
https://10.10.11.15:443
http://10.10.11.15:5357
http://10.10.11.15:80
http://10.10.11.15:8443
https://10.10.11.15:8443
https://10.10.11.90:443
https://10.10.11.98:443
http://10.10.11.98:80
http://10.10.11.99:5985
λ booted ~/projets/workshop/bb-interne »
```

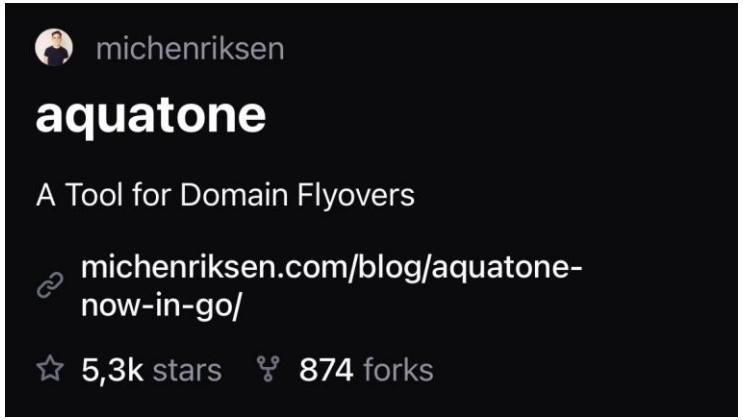
VS



screenshots

aquatone

"Aquatone is a tool for visual inspection of websites across a large amount of hosts and is convenient for quickly gaining an overview of HTTP-based attack surface."

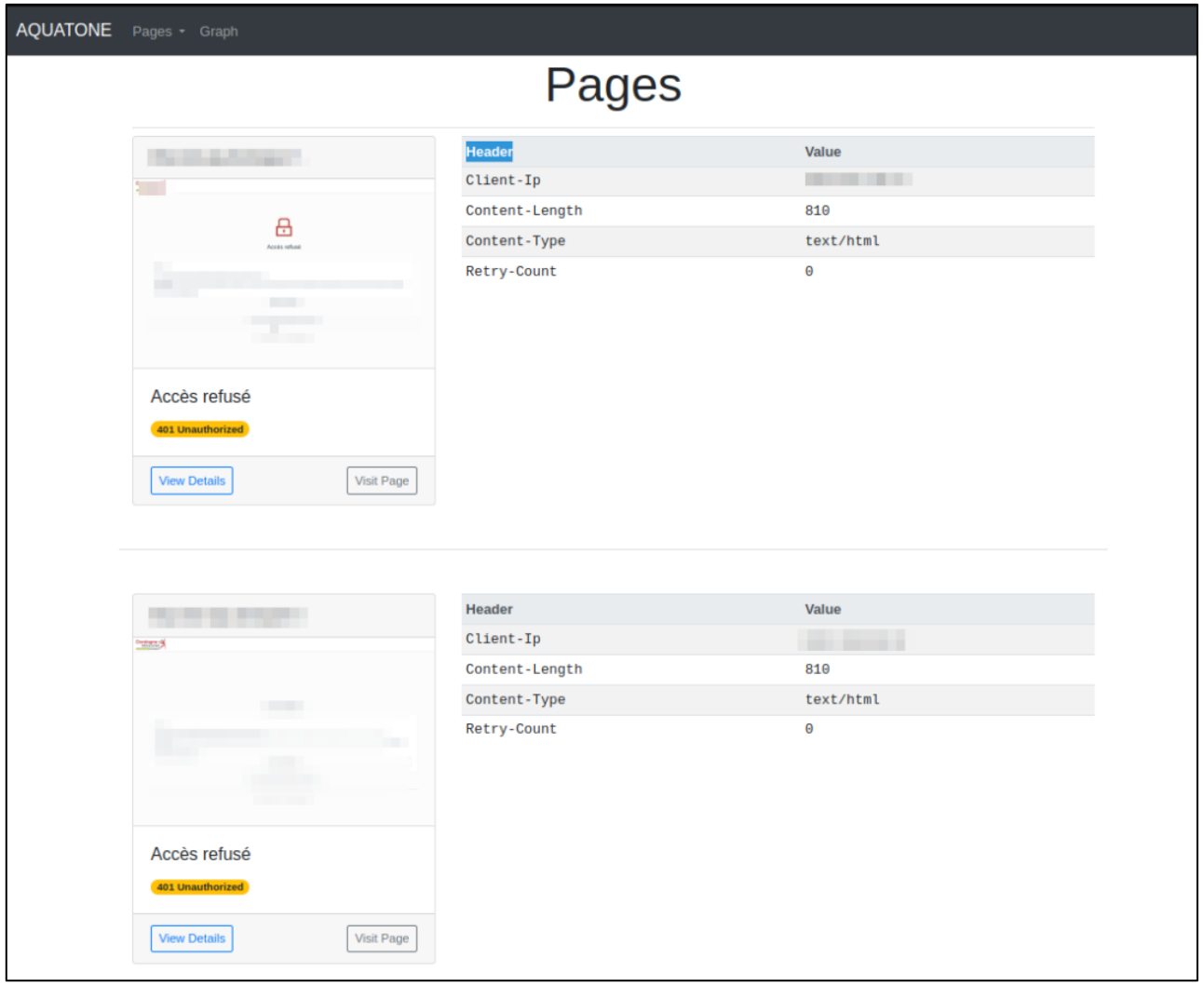


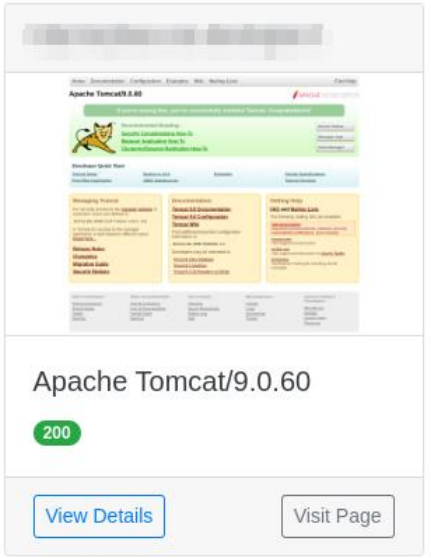

```
cat http.txt | aquatone -chrome-path /usr/bin/google-chrome-stable
```

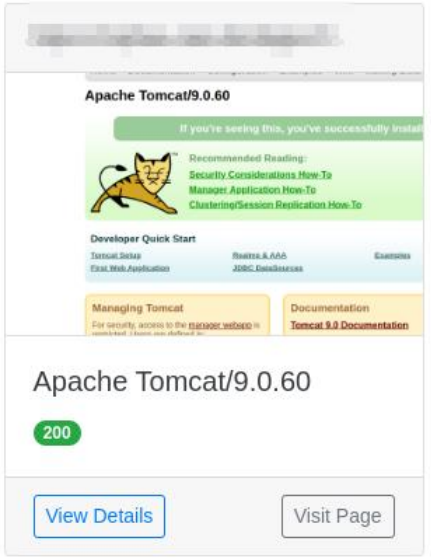
- crée un rapport HTML permettant de visualiser par
  - hôtes
  - similarité
  - page unique

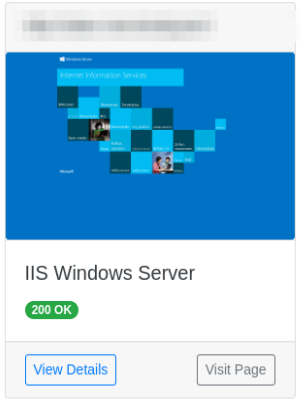
- peut prendre un scan nmap en entrée (-nmap)
- /!\ ne prend pas en compte host:port /!\
- liste de port à définir
- plus maintenu mais marche et fait le taff

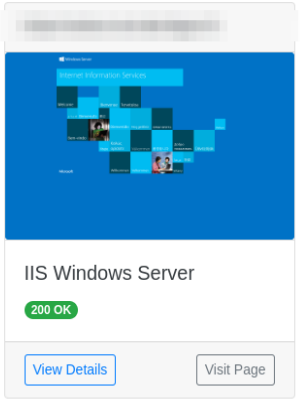
screenshots

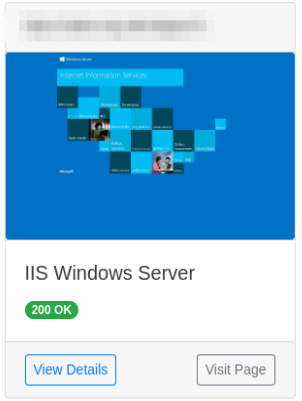












screenshots

sensepost

gowitness

🔍 gowitness - a golang, web screenshot utility using Chrome Headless

☆ 2,4k stars 321 forks

gowitness

"gowitness is a website screenshot utility written in Golang, that uses Chrome Headless to generate screenshots of web interfaces using the command line, with a handy report viewer to process results"

scan via

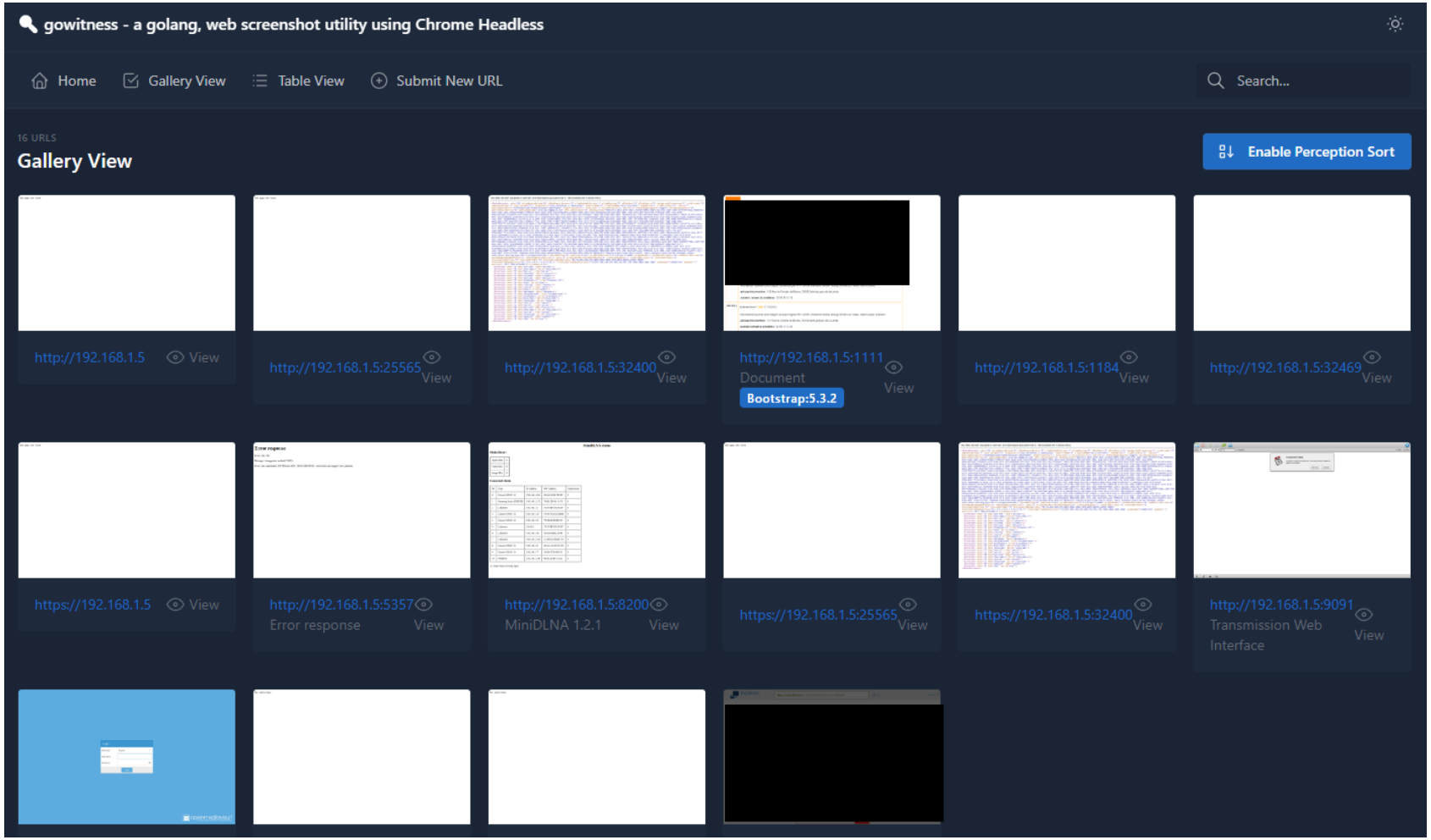
- fichiers
- CIDR avec liste de ports
- ajout d'URL en live

consultation

- consultation web de la db sqlite
- tri des pages identifiées

```
λ 130 booted /tmp » echo 192.168.1.5 | httprobe $(gnmap_all_open_ports 192.168.1.5.gnmap | port_to_httprobe.py) | remove_duplicate_http  | tee http.txt
http://192.168.1.5:32400
http://192.168.1.5:1111
http://192.168.1.5
http://192.168.1.5:25565
http://192.168.1.5:1184
http://192.168.1.5:32469
https://192.168.1.5
http://192.168.1.5:5357
https://192.168.1.5:25565
http://192.168.1.5:8200
https://192.168.1.5:32400
http://192.168.1.5:9091
http://192.168.1.5:8201
http://192.168.1.5:8000
https://192.168.1.5:8384
http://192.168.1.5:8384
λ 1 booted /tmp » gowitness file -f http.txt
06 Oct 2023 16:08:51 WRN preflight result statuscode=404 title= url=http://192.168.1.5:25565
06 Oct 2023 16:08:51 WRN preflight result statuscode=404 title= url=http://192.168.1.5
06 Oct 2023 16:08:51 INF preflight result statuscode=200 title= url=http://192.168.1.5:32400
06 Oct 2023 16:08:51 INF preflight result statuscode=200 title=Document url=http://192.168.1.5:1111
06 Oct 2023 16:08:54 WRN preflight result statuscode=412 title= url=http://192.168.1.5:1184
06 Oct 2023 16:08:54 WRN preflight result statuscode=404 title= url=http://192.168.1.5:32469
06 Oct 2023 16:08:54 WRN preflight result statuscode=404 title= url=https://192.168.1.5
06 Oct 2023 16:08:55 WRN preflight result statuscode=501 title="Error response" url=http://192.168.1.5:5357
06 Oct 2023 16:08:58 WRN preflight result statuscode=404 title= url=https://192.168.1.5:25565
06 Oct 2023 16:08:58 INF preflight result statuscode=200 title="MiniDLNA 1.2.1" url=http://192.168.1.5:8200
06 Oct 2023 16:08:58 INF preflight result statuscode=200 title= url=https://192.168.1.5:32400
06 Oct 2023 16:08:58 INF preflight result statuscode=200 title="Transmission Web Interface" url=http://192.168.1.5:9091
06 Oct 2023 16:09:01 INF preflight result statuscode=200 title=Backup url=http://192.168.1.5:8201
06 Oct 2023 16:09:01 INF preflight result statuscode=200 title="openmediavault control panel - nop.whiterabbit" url=http://192.168.1.5:8000
06 Oct 2023 16:09:01 WRN preflight result statuscode=401 title= url=https://192.168.1.5:8384
06 Oct 2023 16:09:01 WRN preflight result statuscode=401 title= url=http://192.168.1.5:8384
06 Oct 2023 16:09:05 INF processing complete
```

screenshots

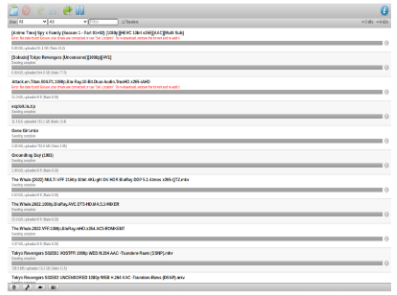


screenshots

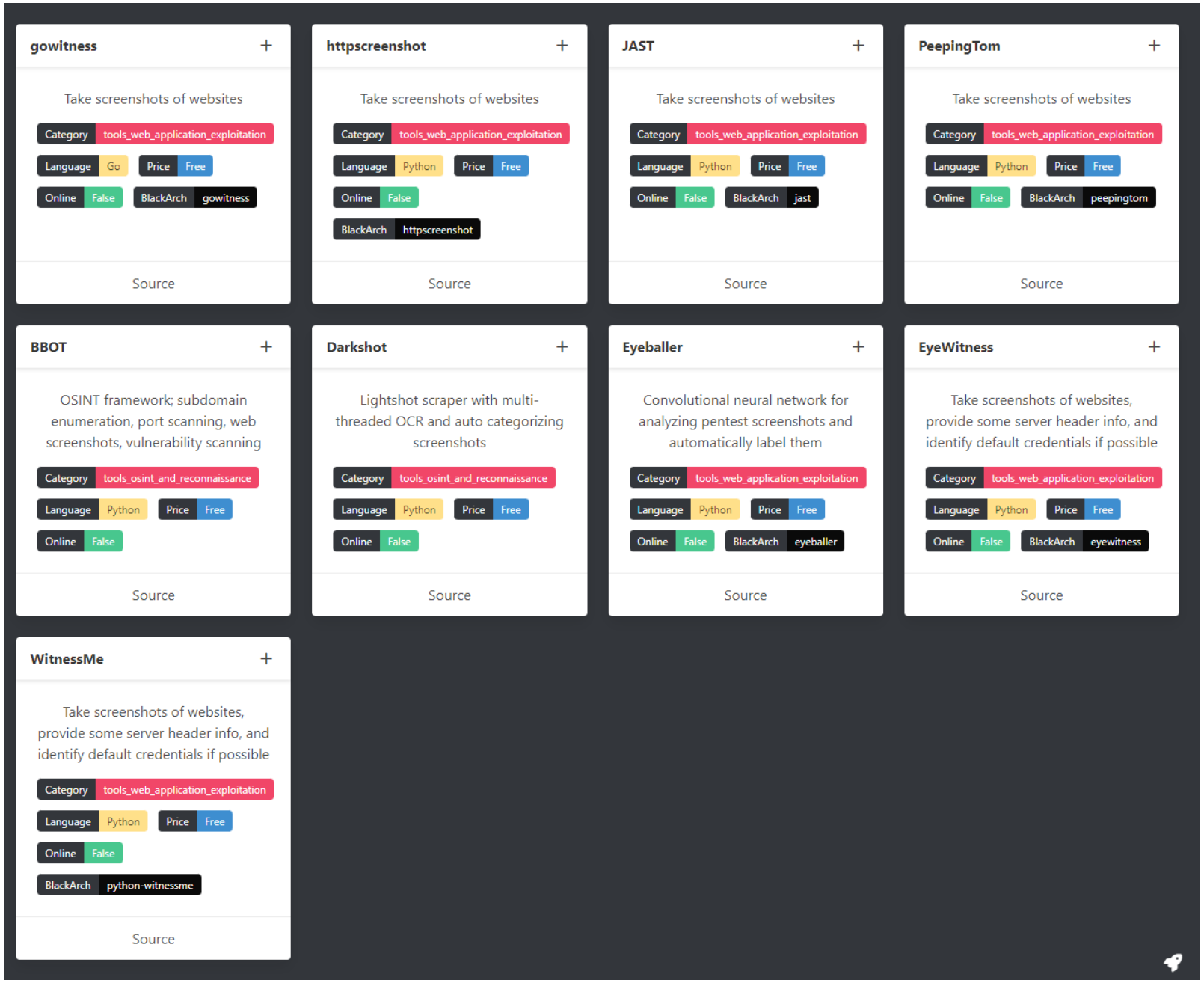
httpx

- une option permet de faire des screenshot mais pas de rapports
- rapide mais basique

```
λ booted /tmp/z » cat ../http.txt | httpx-pd --screenshot

    __    __  __       _  __
   / /_  / /_/ /_____ | |/ /
  / __ \/ __/ __/ __ \|   /
 / / / / /_/ /_/ /_/ /   |
/_/ /_/\__/\__/ .___/_/|_|
             /_/

                projectdiscovery.io

[INF] Current httpx version v1.3.5 (latest)
http://192.168.1.5:9091
http://192.168.1.5:1111
http://192.168.1.5:8000
https://192.168.1.5:25565
http://192.168.1.5:5357
https://192.168.1.5
https://192.168.1.5:25565
https://192.168.1.5
https://192.168.1.5:32400
https://192.168.1.5:32400
https://192.168.1.5:8384
https://192.168.1.5:8384
http://192.168.1.5:8201
http://192.168.1.5:32469
http://192.168.1.5:1184
http://192.168.1.5:8200
λ booted /tmp/z » _
```

Response Info	Screenshot
Host: http://192.168.1.5:9091	
Host: http://192.168.1.5:1111	
Host: http://192.168.1.5:8000	

#collaboration commerciale



plus d'outils
Rawsec inventory https://inventory.raw.pm/

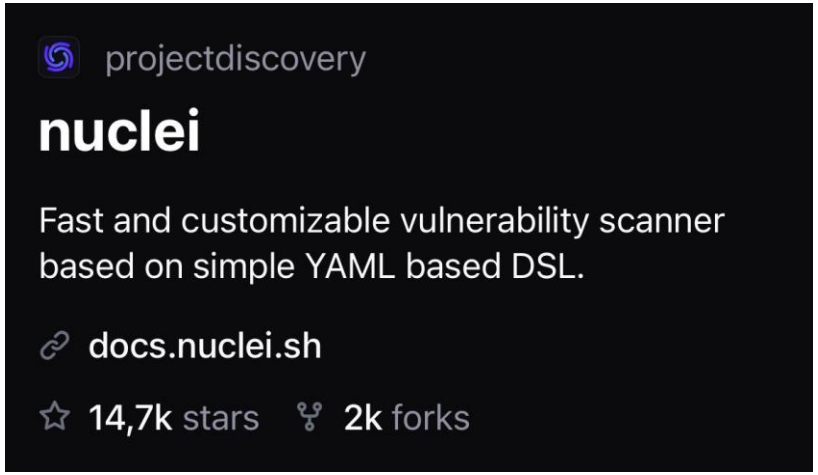
scans de vulns 🔪

comment

- des centaines de services HTTP
- on veut des quick wins
- gros scanner style Nessus overkill

nuclei

- scanner de vuln léger basé sur des templates crées par la communauté
- prend proto://host:port en entrée
- les templates sont catégorisées par type et sévérité
  - panel
  - cve
  - defaut-login
  - config
  - ...
- requêtes *clusterisées* (permet d'optimiser le nombre de requête)
- *très* souvent mises à jour (v3 le 19/10/2023)



scans de vulns

nuclei how to

- en interne on s'en **tape** des headers de sécurité, on veut du **sang** 🩸
- sévérité **high** et **critical**
- CVE, panel exposé, creds par défaut
- combinaison permettant de bien viser sa cible
- peut utiliser wapalizer, attention leak d'infos client

```
cat http.txt | nuclei -s high,critical
cat http.txt | nuclei -tags panel,config,cve,default-config,defaut-login
cat http.txt | nuclei -t cves/2023/
```

scans de vulns

mettre meilleurs screens

fichiers quick wins

pourquoi

- impossible de ffuf l'ensemble des apps web et surtout inutile
- on veut que des fichiers quick wins
- utiliser une liste de fichier très spécifique et courte
- on s'en tape de robots.txt alors que .env et .git 😉


```
meg ~/hacks/conf/meg_paths.txt x -d 0
```

- création d'un index avec hash des réponse + status HTTP
- on grep sur 200

```
λ booted /tmp » cat out/index | grep "200 OK"
out/localhost/399b66bcbadae728d2d61f04e9f83bbe2b07d9c8 http://localhost:8080/.bashrc (200 OK)
out/localhost/70cede75658267f9d4895d375c6e9f370c067ee2 http://localhost:8080/.profile (200 OK)
out/localhost/57a0870e714d511bee5d8cdd42f963d0d3854905 http://localhost:8080/.bash_history (200 OK)
```

- pas mal de faux positifs (serveurs qui renvoient toujours 200)

Questions ?