Cas possible

web

Cas possible	Outils & automatisation recommandés	Commande / usage rapide	Note / astuces
SQL Injection (SQLi)	sqlmap, Burp Suite (Intruder/Scanner), nuclei (templates SQLi), gf (patterns)	sqlmap -u "https://t/?id=1" --batch --threads=5 --dbs	Vérifier d'abord existence d'erreur ; --batch pour automation. Intrusif → timebox.
Cross-Site Scripting (XSS)	Burp Suite + Intruder, **DalFox**, **XSStrike**, nuclei (XSS templates), payload lists (XSS payloads)	dalfox url "https://t/?q=param" -b	Testez reflected/DOM/stored; encodez payloads; utiliser CSP/encode checks.
Local File Inclusion (LFI)	ffuf (fuzz file=), nuclei (LFI templates), Burp extensions, gobuster	ffuf -w lfi-wordlist.txt -u "https://t/?page=FUZZ"	Tester php://filter/ et wrappers; attention à file read noise.
Remote File Inclusion (RFI)	ffuf, Burp, httpx for probe	ffuf -w rfi-wordlist.txt -u "https://t/?page=FUZZ"	RFI peut permettre SSRF/RCE selon contexte; désactiver si hors scope.
Server-Side Request Forgery (SSRF)	httpx (probe), **ssrfmap**, nuclei (SSRF templates), wfuzz	ssrfmap -u "https://t/?url=http://127.0.0.1:8000" or use httpx to detect open endpoints	Tester metadata (AWS 169.254.169.254) ;

Cas possible	Outils & automatisation recommandés	Commande / usage rapide	Note / astuces
			prudence sur scanning interne.
Command injection / RCE	Burp Intruder, commix (test), payload lists, nuclei (RCE templates)	commix --url="http://t/?v=1" --data="v=1" --batch	Très intrusif — n'utiliser que sur cibles autorisées ; privilégier tests manuels contrôlés.
Insecure Direct Object Reference (IDOR) / Broken Access Control	Scripts curl/Burp automation, ffuf for endpoints, gobuster, nuclei (ACL templates)	`curl -s -H "Cookie: session=..." "https://t/user/1234"	jq`	
File upload vuln (upload bypass / webshell)	Burp (file upload), wfuzz, ffuf, detection scripts, nuclei	Use Burp to capture upload, then test payloads; wfuzz -c -w ext-list.txt -u "https://t/upload"	Tester double extensions, content-type bypass, null byte tricks. Évite upload sur services publics.
XML External Entity (XXE)	nuclei (XXE templates), XXEinjector patterns, Burp	nuclei -t xxe-templates/ -u https://t	Peut mener à file read/SSRF ; payloads DTD à tester.
Insecure deserialization	ysoserial (Java), ysoserial.net (dotnet), nuclei templates, Burp	Générer payloads avec ysoserial puis upload/test	Danger RCE ; outils puissants -> usage restreint.
CSRF (Cross-Site Request Forgery)	Burp, manual review (presence of CSRF tokens),	Inspecter forms for missing token; use Burp to craft forged POST	Vérifier tokens, SameSite cookie, referer checks.

Cas possible	Outils & automatisation recommandés	Commande / usage rapide	Note / astuces
	nuclei (CSRF checks)		
Open redirect	ffuf / wfuzz, Burp, nuclei	ffuf -w openredir.txt -u "https://t/?next=FUZZ"	Paramètres next, redirect, url souvent vulnérables.
JWT / Authentication issues	jwt_tool, jwt_tool.py, Burp, jwt_cracker, hashcat (brute secrets)	jwt_tool -t token -i or python3 jwt_tool.py --decode token	Tester alg=none, HMAC→RSA confusion, brute secret small.
Header / CORS misconfigurations	httpx, curl, Burp, nuclei (CORS templates)	httpx -u https://t -status-code -headers	Access-Control-Allow-Origin: * + creds is dangerous.
Business Logic Flaws	Manual + automation scripts, Burp, testcases generator	Script flows: create->modify->transfer with curl (simulate user)	Souvent unique au challenge → automatisation légère aide.
Directory Traversal	ffuf, gobuster, nuclei	ffuf -w ../traversal-list.txt -u "https://t/?path=FUZZ"	../../../../etc/passwd, php wrappers, URL-encoding variants.
Header injection / HOST header attacks	curl, Burp, httpx	curl -H "Host: evil" https://t	Tester virtual host routing, password reset links.

Forensic

Cas forensics	Outils & automatisation recommandés	Commande / usage rapide	Note / astuce
Identification type / quick triage	file, strings, xxd, binwalk -E, trid	`file sample.bin; strings sample.bin | head`	
Métadonnées (EXIF / doc props)	exiftool (images/docs), pdfinfo	exiftool image.jpg; pdfinfo file.pdf	Cherche UserComment, Producer, Creator, champs personnalisés.
Carving / extraction d'artefacts	binwalk -e, foremost, scalpel, photorec	binwalk -e sample.img; foremost -i sample.img -o outdir	Utiliser plusieurs outils car chacun récupère différemment.
Recherche de signatures / YARA	yara, yara-rules, bulk_extractor	yara -r rules.yar sample_dir/ ; bulk_extractor -o out sample.img	Ecrire règles YARA simples pour flag patterns.
Stéganographie images (LSB, palettes, bitplanes)	zsteg, stegsolve.jar, steghide, stegseek, outguess	zsteg image.png; java -jar StegSolve.jar	Tester canaux R/G/B, bitplanes, palettes, alpha.
Audio stego / spectrogramme	sox, audacity, scipy, spektral, sonic visualiser	sox hidden.wav -n spectrogram -o spec.png	Regarder spectrogramme (Audacity View→Spectrogram).
Embedded archives	grep -aob $'\x50\x4b\x03\x04'	grep -aob $'\x50\x4b\x03\x04' file.bin → dd if=file.bin	Rechercher signatures PK\x03\x04, 7z, Rar!.

Cas forensics	Outils & automatisation recommandés	Commande / usage rapide	Note / astuce
(PNG+ZIP, JPG+tar)	file, dd to extract, binwalk -e	of=embedded.zip bs=1 skip=OFFSET	
PDF forensics (JS, attachments)	pdfid, peepdf, pdf-parser.py	pdfid file.pdf; pdf-parser.py -a file.pdf	Cherche JavaScript, /OpenAction, Embedded files.
Memory forensics (RAM images)	volatility3, volatility (v2), rekall, bulk_extractor	vol -f memory.raw windows.pslist (vol plugin)	Récupérer process list, open sockets, creds, dlls.
PCAP / Network forensics	tshark, wireshark, zeek (Bro), scapy, networkminer	tshark -r capture.pcap -q -z conv,tcp ; zeek -r capture.pcap	Extraire files HTTP/SMB (tshark -Y "http" -T fields ...).
Recover files from disk image (FS forensics)	sleuthkit (fls, icat), pytsk3, tsk_recover, autopsy	fls -r image.dd ; icat image.dd inode > file	Mappe partition, timeline, récupérer inodes.
Registry / Windows artifacts	reglookup, python-registry, regripper	rip.exe -r SYSTEM -f JSON (example)	Extraire autostart, runkeys, LastWrite times, MRU.
Malware static analysis (binaire)	strings, radare2, ghidra, pefile (Python), rabin2	rabin2 -I sample.exe ; `strings sample.exe	grep -i url`
Malware dynamic analysis (sandbox)	Cuckoo, Any.Run, Hybrid Analysis, VirusTotal	Soumettre hash à VirusTotal, ou exécuter dans sandbox isolée	Ne jamais exécuter sur machine non-isolée.
Timeline / metadata correlation	plaso/log2timeline, mactime, timesketch	log2timeline.py plaso.dump image.dd ; importer dans Timesketch	Construire timeline unifiée (fichiers, registry, logs).
Password cracking / hashes from dumps	john, hashcat	john --wordlist=rockyou.txt hashfile	Parfois password hashes apparaissent dans dumps.

Cas forensics	Outils & automatisation recommandés	Commande / usage rapide	Note / astuce
QR / barcode extraction	zbarimg, zxing	zbarimg image.png	Utile pour challenges contenant QR masks.
Stego automation (mass stego checks)	scripts zsteg + stegsolve + strings pipeline, stegseek brute	Script: run zsteg, run stegsolve, run steghide info, run strings	Automatiser tests de plusieurs fichiers en parallèle.

Commandes & snippets prêts (copier-coller)

Identification & triage

```
file sample.bin
strings sample.bin | head -n 200
xxd -g 1 sample.bin | head
trid sample.bin
```

Binwalk extraction

```
binwalk -e sample.bin        # extract embedded files
binwalk -Me sample.bin       # recursive extraction (with -M)
```

Foremost / Scalpel carving

```
foremost -i sample.img -o carved_foremost
scalpel sample.img -o carved_scalpel
```

EXIF / PDF analysis

```
exiftool image.jpg
pdfinfo doc.pdf
pdfid.py doc.pdf
pdf-parser.py -a doc.pdf
```

Stego quick (PNG)

```
zsteg image.png
java -jar StegSolve.jar        # GUI explore bitplanes
steghide info image.jpg        # check embedded
steghide extract -sf image.jpg -xf out.txt -p PASSWORD
```

Audio spectrogram

```
sox hidden.wav -n spectrogram -o spec.png
python3 - <<'PY'
from scipy.io import wavfile
sr,data=wavfile.read('hidden.wav')
# build simple spectrogram with matplotlib (omitted here)
PY
```

PCAP quick analysis

```
# list endpoints and counts
tshark -r capture.pcap -q -z conv,tcp
# extract HTTP objects
tshark -r capture.pcap -Y http -T fields -e http.file_data | strings > http_files.bin
# zeek
zeek -r capture.pcap
```

Memory forensics (Volatility3 example)

```
# install: pip install volatility3
vol -f memory.raw windows.pslist.Plugins
vol -f memory.raw windows.malfind.Malfind
# volatility3 usage differs; voir docs
```

Carving embedded zip from arbitrary file

```
grep -aob $'\x50\x4b\x03\x04' suspect.bin
# offset=12345
dd if=suspect.bin of=embedded.zip bs=1 skip=12345
unzip embedded.zip -d extracted
```

YARA scanning

```
yara -r myrules.yar samples/
```

Reseau

Tableau : cas réseau → outils & commandes rapides

Cas réseau / objectif	Outils & automatisation recommandés	Commande / usage rapide	Note / astuce
Découverte hôte / ports (large)	masscan (very fast), nmap (détaillé), naabu (ProjectDiscovery)	masscan -p1-65535 10.0.0.0/24 --rate 10000 -oL masscan.out	masscan pour discovery → nmap pour fingerprinting. Respecte le scope.
Scan de services & fingerprint	nmap + NSE scripts (-sV -sC -A), httpx	nmap -sC -sV -p- -T4 -oA nmap_all 10.0.0.5	Utilise -Pn si hôte behind firewall.
Enum ports rapide	naabu, tcping, nping	naabu -host target.com -p 1-65535 -o naabu_out.txt	Naabu + httpx pour passer à web triage.
SNMP / NetBIOS / SMB enum	snmpwalk, smbclient, enum4linux, rpcclient, smbmap	enum4linux -a 10.0.0.5 ; smbclient -L //10.0.0.5 -U anonymous	SMB often reveals shares/creds.
SMB / Lateral movement helpers	impacket suite (smbclient, smbrelay, secretsdump), crackmapexec	python3 -m impacket.examples.secretsdump LOCAL	Impacket excellent pour exploitation/relays.
NetBIOS / LLMNR / WPAD poisoning	Responder, Inveigh	responder -I eth0 -r -d -w	Très utile en lab/CTF LAN; attention scope.

Cas réseau / objectif	Outils & automatisation recommandés	Commande / usage rapide	Note / astuce
DNS enumeration / transfer / zone	dnsenum, dnsrecon, dig, dnsx	dig axfr @ns1.example.com example.com	Use dnsx for bulk DNS probing.
Traffic capture & analysis	tcpdump, tshark, Wireshark, capinfos, tshark	tcpdump -i eth0 -w capture.pcap ; tshark -r capture.pcap -q -z conv,tcp	Capture with filters to limit noise.
PCAP analysis / IDS	Zeek (Bro), Suricata, NetworkMiner, pcapkit	zeek -r capture.pcap ; suricata -r capture.pcap -l out/	Zeek produces logs (http.log, conn.log) for quick triage.
Packet crafting / scanning	scapy (Python), hping3, nping	hping3 -S -p 80 target.com ; scapy interactive scripts	Scapy for custom probes and exploit payloads.
TCP/UDP pivoting / port forwarding	socat, ssh -L/-R/-D, chisel, sshuttle	socat TCP-LISTEN:8080,fork TCP:10.0.0.5:80	Use SSH tunnels or chisel for reverse tunnels.
ARP spoofing / MITM	bettercap, ettercap, arp-scan	bettercap -I eth0 and enable net.probe modules	MITM only on allowed networks.
Service brute / credential stuffing	hydra, medusa, crackmapexec	hydra -L users.txt -P pass.txt ssh://10.0.0.5	Use rate limits; avoid locking accounts.
Wireless (Wi-Fi) basics	aircrack-ng suite, iw, airmon-ng, airodump-ng	airmon-ng start wlan0 ; airodump-ng wlan0mon	Only in scope; requires adapter supporting monitor mode.
IPv6 / DHCP / NDP attacks	ndisc6, mitm6	mitm6 -i eth0	Useful for Windows AD pivoting in internal labs.
Vulnerability scanning (network CVE)	nmap NSE, nessus (commercial), nuclei with specific templates	nmap --script vuln -sV target	Quick vulnerability triage.

Cas réseau / objectif	Outils & automatisation recommandés	Commande / usage rapide	Note / astuce
Credential harvesting / sniffing	Responder, mitmproxy (HTTP), Wireshark	tcpdump -i eth0 -w sniff.pcap ; responder -I eth0	Collector logs then parse for NTLM hashes / credentials.
Active directory enumeration	bloodhound + neo4j, impacket tools, ldapsearch	Use SharpHound in Windows, import into BloodHound	Requires AD access; powerful for lateral movement.
DNS/HTTP/SMTP abuse (exfil/SSRF triage)	curl, httpx, smtp-user-enum, swaks	curl -v http://target/endpoint ; swaks --to user@target	Check for mechanisms that can be abused (uploads, redirects).

Commandes & snippets prêts (copier-coller)

1) Discovery rapid (masscan → nmap)

```
# masscan discovery (adjust rate & IP range)
sudo masscan 10.0.0.0/24 -p1-65535 --rate 10000 -oL masscan.out
# parse masscan → run nmap on discovered IPs
cat masscan.out | grep -oE '([0-9]+\.){3}[0-9]+' | sort -u > hosts.txt
nmap -sC -sV -T4 -iL hosts.txt -oA nmap_discovery
```

2) Quick web/HTTP probe

```
subfinder -d target.com -silent | httpx -silent -status-code -title -o alive.txt
nuclei -l alive.txt -t ~/tools/nuclei-templates/ -o nuclei_net.txt
```

3) Capture traffic (filters)

```
# capture only HTTP and DNS
sudo tcpdump -i eth0 -w capture.pcap 'tcp port 80 or udp port 53'
# live view of HTTP hosts
tshark -r capture.pcap -Y http -T fields -e http.host | sort -u
```

4) Zeek / Bro quick analysis

```
# run zeek on pcap
zeek -r capture.pcap
# check http logs
```

```
cat http.log | awk '{print $1,$2,$3}'
```

5) SMB enum (impacket)

```
# list shares anonymously
smbclient -L //10.0.0.5 -N
# use impacket for secretsdump / psexec
python3 /usr/share/doc/python3-impacket/examples/smbclient.py 10.0.0.5 -username user -password pass
```

6) ARP scan & responder

```
# list hosts in LAN
sudo arp-scan --localnet
# run Responder (LLMNR/NBT-NS/MDNS poisoning)
sudo responder -I eth0 -rdw
```

7) Packet crafting with scapy (example DNS probe)

```
from scapy.all import *
q=IP(dst="10.0.0.5")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="example.com"))
resp=sr1(q,timeout=2)
print(resp.summary())
```

8) Simple port forward with socat

```
# forward local 8080 to remote 10.0.0.5:80
socat TCP-LISTEN:8080,fork TCP:10.0.0.5:80
```

Exploitation binaire / Reverse-engineering

1) Cas / vulnérabilités courantes (CE QU'ON RENCONTRE)

- **Buffer overflow (stack)** — écriture au-delà du buffer → overwrite RET.
- **Heap overflow / Use-after-free / House of X** — corruption metadata → arbitrary write/alloc hijack.
- **Format string** (printf vuln) — leak / write arbitrary memory via %x/%n.
- **Integer overflow / underflow** — sizes mal calculés → malloc too small / OOB.
- **Return-oriented Programming (ROP)** / ret2libc — bypass NX.
- **ASLR / PIE / NX / Canary protections** — défenses courantes à contourner.
- **Information leak** (puts/printf leak, GOT/PLT) — récupérer libc/stack addr.
- **Format/endianness / type confusion** — parsing bugs.
- **Race conditions / TOCTOU** — file replace, symlink attacks.
- **File format fuzzing / parsing bugs** — crash → exploitation (fuzzers).
- **Symbol stripping & obfuscation / anti-debugging** — reverse challenges.

2) Outils indispensables (installés sur Kali ou à ajouter)

CRUCIAL: pwntools, gdb, pwndbg/**gef**, ropper/ROPgadget, checksec, radare2, ghidra, objdump, readelf, strings, ltrace, strace, file, gcc-multilib, qemu-user-static, qemu-system-*, gdb-multiarch, AFL/AFLplusplus, angr (symbolic), valgrind, ASAN/UBSAN build flags, one_gadget (libc gadgets), heap-exploitation tools (libc-database).

3) Commandes / vérifications rapides (triage initial)

```
file challenge              # arch / type
checksec --file=challenge   # NX/Canary/PIE/RELRO
readelf -h challenge
readelf -s challenge | grep GOT
objdump -d challenge | sed -n '1,200p'
strings challenge | less
ldd challenge               # libs used (if ELF x86_64)
```

Vérifier ASLR (local test) :

```
# 0 = disabled, 1 = enabled
cat /proc/sys/kernel/randomize_va_space
# temporaire disable (local only):
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Créer pattern pour trouver offset :

```
from pwn import *
print(cyclic(300))   # generate
# find offset
print(cyclic_find(0x61616162))  # value from crash
```

MISC

Cas MISC → outils & automatisation recommandés

Cas possible (misc)	Outils & automation recommandés	Commande / usage rapide	Note / astuce
Parsing massif de fichiers (logs / CSV / JSON)	Python (pandas, jq), rg (ripgrep), awk, csvkit	rg -n "CTF{" . -S ; `jq '.[]	select(.user=="alice")' file.json`
Extraire flags patternisés	rg, grep -P, Python re	rg -o 'CTF\{[^}]{1,200}\}' -S	Utilise regex robustes ; cherche variantes (FLAG{}, flag:, etc.).
Parsing binaire / custom format	Python (struct, construct), hexdump, xxd, bpython	`xxd -g 1 file.bin	head; script Pythonstruct.unpack`
API / Automation HTTP (bulk queries)	httpx, curl, httpie, Python requests, jq	httpx -silent -mc 200 -i urls.txt -o out.csv	Rate limit + concurrent workers via -threads.
Bulk URL extraction / normalization	gau, waybackurls, xsser	`echo domain	gau > urls.txt`
Transformations multi-couche / decodings	**CyberChef**, Python (binascii, base64), xxd	CyberChef recipe or python3 -c "import base64; print(base64.b64decode('...'))"	Keep CyberChef recipes saved for reuse.

Cas possible (misc)	Outils & automation recommandés	Commande / usage rapide	Note / astuce
Regex generation / complex extraction	ChatGPT / Copilot (assist), regex101, Python	Use regex101 to test; rg -P to apply	Save final regex into scripts.
Automatisation d'énigmes (puzzles, math)	Python (sympy), small LLM prompts	python solve_puzzle.py	Use ChatGPT for idea generation, but verify logic.
Document / office forensics	exiftool, strings, oletools (oleid, olevba)	olevba suspicious.doc	olevba extracts VBA macros quickly.
Simple web API exploitation (rate-limited)	curl + small Python loop + retries (tenacity)	Python script with exponential backoff	Respect rate limits; cache responses locally.
OSINT / quick lookup	gh (GitHub CLI), whois, crt.sh web, google dorks	gh search code --query 'CTF{'	Automate GitHub searches with token (watch rate limits).
Small binaries / scripts analysis (.pyc, .js, powershell)	uncompyle6, decompilers, jsbeautifier, python -m dis	uncompyle6 file.pyc -o out/	Often small scripts hide flags in obf code.
Time-saving templates (bulk work)	Makefiles / scripts, parallel, entr	`cat urls.txt	parallel -j50 httpx -silent -mc 200 {}`
Puzzle stego / encoding chains	CyberChef, small python toolchain, zsteg	Build pipeline: hex→base64→xor→rot	Save cookbook recipes for repeated patterns.

Commandes & snippets prêts (copier-coller)

1) Extraction rapide de flags (recursively)

```
# recherche dans tout le repo / dossiers
rg -n --hidden --no-ignore 'CTF\{[^}]+\}|FLAG\{[^}]+\}|flag\{[^}]+\}' -S || true
# fallback grep
grep -RIn --binary-files=text -E 'CTF\{[^}]+\}' . || true
```

2) Normaliser & filtrer JSON massifs

```
# lister tous les "message" contenant flag
jq -r '..|.message? // empty' big.json | rg 'CTF\{'
# extraire objets avec key "flag"
jq -c '.[] | select(.flag != null)' big.json > flagged_objects.json
```

3) Pipeline URLs → filtre alive → check flag patterns

```
cat urls.txt | httpx -silent -status-code -o alive.txt
cat alive.txt | parallel -j50 'curl -s {} | rg -o "CTF\{[^}]+\}" || true' > possible_flags.txt
```

4) Décodage multi-couche simple (bash + python)

```
# ex: hex -> base64 -> rot13 pipeline (pseudo)
echo '48656c6c6f' | xxd -r -p | base64 --decode | tr 'A-Za-z' 'N-ZA-Mn-za-m'
# python flexible
python3 - <<'PY'
import sys,base64,binascii
data = bytes.fromhex('48656c6c6f')
print(base64.b64encode(data))
PY
```

5) Bulk file triage (run many tools)

```
mkdir -p out && for f in suspicious/*; do
  echo "=== $f ===" > out/$(basename $f).txt
  file "$f" >> out/$(basename $f).txt
  strings "$f" | rg -n 'CTF\{|FLAG\{' >> out/$(basename $f).txt || true
  exiftool "$f" >> out/$(basename $f).txt 2>/dev/null || true
done
```

6) Small Python skeleton to brute XOR key (single-byte)

```
# xor_bruteforce.py
from collections import Counter
ct = bytes.fromhex(open('ct.hex').read().strip())
for k in range(1,256):
    pt = bytes(b ^ k for b in ct)
    if b'CTF{' in pt:
        print(k, pt)
        break
```

crypto

Tableau : cas crypto → outils & automatisation recommandés

Cas / objectif crypto	Outils & automation recommandés	Commande / usage rapide	Note / astuce
Identifier type de hash	hashid, hash-identifier, detect_hash.py	hashid 5f4dcc3b5aa765d61d8327deb882cf99	Commence toujours par identifier le format (MD5/SHA1/NTLM/ bcrypt).
Cracking hashes (dictionnaire / rules)	hashcat (GPU), john (CPU), hashcat-utils	hashcat -m 0 -a 0 hashes.txt rockyou.txt	Choisir le bon mode (-m) selon le type. Utilise rules pour mutations.
Brute-force / masks	hashcat masks, john --incremental	hashcat -a 3 -m 0 hash.txt ?l?l?l?l?d?d	Masques réduisent l'espace (ex: ?l = lowercase).
Salted hashes / PBKDF2 / bcrypt / Argon2	hashcat (modes PBKDF2/bcrypt/argon2), john	hashcat -m 3200 bcrypt_hashes.txt rockyou.txt	Ces algos sont lents — use rules + good candidates.
Rainbow / precomputed lookup	rcrack, rtgen (rare en CTF)	peu utilisé si salt présent	Pré-calcul rare ; salt les rend inutiles.

Cas / objectif crypto	Outils & automation recommandés	Commande / usage rapide	Note / astuce
Length-extension attack	hashpumpy, scripts Python, CyberChef (partial)	hashpumpy <hash> <orig> <append> <keylen>	Fonctionne pour MD5/SHA1/SHA256 (Merkle–Damgård) sur `hash(secret
HMAC forging / verification issues	Python hmac, test alg=none sur JWT	python3 -c "import hmac,hashlib; print(hmac.new(b'k',b'm',hashlib.sha256).hexdigest())"	HMAC correctement fait est résistant length-extension.
RSA — small e / common modulus / shared prime / gcd	RsaCtfTool, rsatool, msieve, yafu, sage	python3 RsaCtfTool.py --publickey pub.pem --uncipher	Scénarios: Hastad, common modulus, gcd(n1,n2)>1, Wiener's attack (small d).
RSA — padding oracle / Bleichenbacher	scripts spécifiques, rsatool, PoC d'attaque	tests manuels / scripts (vérifier oracle)	Très spécifique ; attention scope.
Diffie-Hellman weak params (small p)	sage, pohlig-hellman scripts, msieve	Pohlig-Hellman si p-1 factorisé	Si p petit/factored, DL solvable.
OTP / stream reuse (one-time pad reuse)	Python scripts (XOR of ciphertexts), xor-tool	python3 xor_reuse.py ct1.bin ct2.bin	Reuse of keystream → keystream = ct1 ^ pt1 => recover other pt.
Repeating-key XOR / Vigenère	cyberchef, quipqiup, Python bruteforce (Kasiski / IC)	python3 vigenere_break.py cipher.txt	Guess key length via IC, then single-byte XOR per column.

Cas / objectif crypto	Outils & automation recommandés	Commande / usage rapide	Note / astuce
Single-byte XOR	cyberchef, Python brute 0..255 (score by english freq)	snippet below (python)	Fast; score candidates by chi-square or frequency.
Hash length extension detection / tools	hashpumpy, detect by service behavior	hashpumpy.hashpump(...)	Test various key lengths.
Collision attacks (MD5/SHA1)	Precomputed collisions tools, fastcoll (MD5), published collisions	rarement in small CTFs; useful for forgery	Requires special inputs; practical on MD5/SHA1 for crafted collisions.
Key derivation / PBKDF2 params inspection	hashcat --show -m knowledge / bcrypt libs	Inspect storage format salt$hash	If many iterations/cost high, cracking slow.

Outils indispensables (rappel rapide)

- **hashcat**, **john** (cracking) — CRUCIAL
- **RsaCtfTool**, **rsatool**, **hashpumpy**, **hashid/hash-identifier**
- **msieve**, **yafu**, **gmpy2**, **sage** (maths)
- **pycryptodome**, **cryptography**, **gmpy2**, **sympy** (Python libs)
- **CyberChef** (online), **quipqiup** (substitution), **Online hash lookup** (CrackStation)
- **rockyou**, **SecLists** (wordlists)

Commandes & snippets prêts (copier-coller)
Identifier hash
hash-identifier hash.txt
hashid 5f4dcc3b5aa765d61d8327deb882cf99

hashcat examples

```
# MD5 dictionary
hashcat -m 0 -a 0 hashes.txt ~/ctf/wordlists/rockyou.txt --status -O

# bcrypt (mode 3200)
hashcat -m 3200 -a 0 bcrypt_hashes.txt ~/ctf/wordlists/rockyou.txt

# mask brute (4 lower + 2 digits)
hashcat -a 3 -m 0 hash.txt ?l?l?l?l?d?d
```

john examples

```
# format auto
john --wordlist=~/ctf/wordlists/rockyou.txt hashes.txt

# rockyou with rules
john --wordlist=rockyou.txt --rules hashes.txt
```

Single-byte XOR brute (Python)

```
ct = bytes.fromhex(open('ct.hex').read().strip())
import string
def score(s):
    # simple english freq
    return sum([s.count(c) for c in b' etaoinshrdlu'])
for k in range(256):
    pt = bytes([b ^ k for b in ct])
    if b'CTF{' in pt or score(pt) > 10:
        print(k, pt[:200])
```

Repeating-key XOR / Vigenère (Python skeleton)

```
# find key length via IC, then single-byte XOR per column (omitted here)
```

Length-extension (hashpumpy)

```
pip install hashpumpy
hashpumpy 5d41402abc4b2a76b9719d911017c592 "hello" "&admin=true" 8
# or in python
import hashpumpy
new_hash, new_msg = hashpumpy.hashpump(orig_hash, orig_msg, append, key_length)
```

RSA quick checks (gcd / factor share)

```
# gcd check between n's
python3 - <<'PY'
from math import gcd
n1=...; n2=...
print(gcd(n1,n2))
PY
# RsaCtfTool usage (see repo)
python3 RsaCtfTool.py --publickey pub.pem --uncipher
```

RSA small-e / Hastad (use RsaCtfTool or CRT scripts)

```
# Hastad: if same plaintext, small e and multiple ns -> use CRT then introot
```

osint

Tableau : cas OSINT → outils & automatisation recommandés

Cas OSINT / objectif	Outils & automation recommandés	Commande / usage rapide	Note / astuce
Découverte de sous-domaines	subfinder, amass, assetfinder, sublist3r, crt.sh (web)	subfinder -d example.com -silent > subs.txt	Start par crt.sh + subfinder then amass enum for depth.
Récupérer URLs historiques / endpoints	waybackurls, gau (getallurls), Wayback Machine web	`echo example.com	waybackurls > urls.txt`
Recherche de commits & secrets sur GitHub	GitHub search, gh CLI, gitleaks, truffleHog, gitrob	gh search code "password repo:org/repo" ; gitleaks detect -s .	Auth GitHub (token) améliore rate limits.
Certificats TLS → sous-domaines	**crt.sh** (web), certspotter, censys, crt.sh API	https://crt.sh/?q=%25.example.com (web)	Copiez la colonne des noms commums

Cas OSINT / objectif	Outils & automation recommandés	Commande / usage rapide	Note / astuce
			; attention aux wildcard entries.
IP / service discovery	**Shodan**, **Censys**, amass output → shodan host	shodan host 1.2.3.4 (API key needed)	API keys nécessaires ; script + cache les résultats.
Email harvesting / patterns	theHarvester, Hunter.io, mailmap, scripts regex	theHarvester -d example.com -b all	Hunter a limites gratuites ; combiné avec guessing patterns.
Username enumeration (réseaux sociaux)	sherlock, socialscan, namechk	python3 sherlock.py username	Très rapide pour trouver présences publiques.
Whois / DNS history / records	whois, dig, dnsx, dnsrecon	dig +short example.com ; whois example.com	Vérifie les contacts, emails, registrar.
Paste sites / leaks	pastebin search, DeHashed, IntelX, psh scripts	Recherche via API / web — attention paywall	Beaucoup de fuites passent par paste sites; use API.
OSINT media / images (reverse image)	Google Images, TinEye, saucenao	Uploader image → reverse search	Utile pour retrouver source d'une image publiée.
Monitoring / watch (certs, subdomains)	certstream (streaming), crt.sh monitoring, SecurityTrails	Script: subscribe to CertStream and grep domain	Permet détecter nouveaux certificats en temps réel.
Leaked credentials /	HaveIBeenPwned, DeHashed, LeakSearch	API queries (key req.)	Respecte TOS; ne pas brute

Cas OSINT / objectif	Outils & automation recommandés	Commande / usage rapide	Note / astuce
paste monitoring			force sur credentials leak.
Recon as-a-service (agrégateurs)	**Recon-ng**, **SpiderFoot**, **Maltego**	recon-ng modules pour github/crt.sh/whois	Très utile pour pipelines GUI/auto.
Phone / people search / geolocation	pipl (payant), Google, social platforms	Google Dork + social search	Légalité à vérifier selon pays/rules CTF.

Commandes & snippets prêts (copier-coller)

Sous-domaines — combiner sources

```
# crt.sh (web) -> download via curl (quick trick to fetch page then parse)
curl -s "https://crt.sh/?q=%25.example.com&output=json" | jq -r '.[].name_value' | sed 's/\*\.//g' | sort -u > subs_crt.txt

# subfinder
subfinder -d example.com -silent -o subs_subfinder.txt

# amass enum (passive)
amass enum -d example.com -o subs_amass.txt

# aggregate unique
cat subs_crt.txt subs_subfinder.txt subs_amass.txt | sort -u > subs_all.txt
```

URLs historiques & alive filtering

```
# wayback + gau -> unique -> probe
echo example.com | waybackurls | sort -u > urls_wayback.txt
gau example.com | sort -u > urls_gau.txt
cat urls_wayback.txt urls_gau.txt | sort -u > urls_all.txt
# filter alive
cat urls_all.txt | httpx -silent -status-code -o urls_alive.txt
```

GitHub quick search via gh (CLI)

```
# requires gh auth login
gh search code "example.com password" --limit 50 --json path,repository | jq -r '.[] |
.repository.fullName + ":" + .path'
```

Git secret scan (local repo clone)

```
# clone then run gitleaks
git clone https://github.com/target/repo.git
gitleaks detect -s repo -r repo/gitleaks_report.json
# trufflehog
trufflehog git https://github.com/target/repo.git --json > truffle.json
```

Shodan / Censys (API)

```
# shodan (needs API key set via env SHODAN_API_KEY)
shodan host 1.2.3.4
shodan search "hostname:example.com" --fields ip_str,port,org,hostnames
```

Whois & DNS (dig + dnsx)

```
whois example.com > whois_example.txt
dig +short NS example.com
# dnsx: resolve and retrieve records quickly
cat subs_all.txt | dnsx -a -resp -o dnsx_out.txt
```

Username enumeration (Sherlock)

```
git clone https://github.com/sherlock-project/sherlock.git
python3 sherlock/sherlock.py username -o sherlock_results/
```

Reverse image (command-line helper)

```
# use tineye api or google reverse via browser; or use image-match libs locally
# download image then manual upload to tineye/google images (no simple CLI free)
```

Ressources

https://github.com/Adamkadaban/CTFs

https://github.com/w181496/Web-CTF-Cheatsheet

https://uppusaikiran.github.io/hacking/Capture-the-Flag-CheatSheet/

Global Offset Table (GOT) - CTF Handbook

https://github.com/Adamkadaban/LearnPwn

https://github.com/lyudaio/cheatsheets/blob/main/security/tools/hydra.md

https://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats

https://akhil.sh/tutorials/cheatsheets/sql_injection_cs/

https://hackviser.com/tactics/pentesting

Katana

https://www.youtube.com/watch?v=0tLwVxkqBfk

https://ctf-katana.readthedocs.io/en/latest/

https://github.com/JohnHammond/katana