RagnarNet OS — Dossier Technique ULTRA DÉTAILLÉ (FR)

Version du 2025-08-15 16:12

Aperçu & objectifs

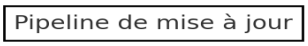
Ce document décrit l’OS RagnarNet dans sa globalité : architecture, flux, sécurité, configuration, base de données et code. Chaque module est détaillé ligne par ligne avec annotations de sécurité, plus 12 schémas explicatifs.

Architecture des modules

Flux de démarrage

Couches de sécurité

Pipeline de mise à jour



Flux d'authentification

Intégrité des fichiers

Intégrité des fichiers

Modèle de privilèges

Modèle de privilèges

Gestion des erreurs

Gestion des erreurs

Flux de configuration

Navigation UI

Schéma users.db

Séquence d’update

Module : update.lua

Rôle détecté : Installateur / mise à jour, Manipulation fichiers, Affichage terminal/UI, Redémarrage système, Exécution de scripts

Fonctions déclarées	BXOR, download, extractCodeVer, fileHash, fnv1a, println, readAll, w
Dépendances (require/shell.run/dofile)	pastebin get

API	Méthodes utilisées
fs	delete, exists, isDir, open
os	reboot
http	—
shell	run
term	setTextColor

Code & explication ligne par ligne

Ligne 1 :
```
-- update.lua : Installation / MAJ complète RagnarNet (met à jour manifest & version)
```
→ Commentaire du développeur.

Ligne 3 : `local function println(c, msg)`
→ Instruction Lua générique.

Ligne 4 : `  if term and colors and c then term.setTextColor(c) end`
→ Interaction avec l'affichage terminal (UI).

Ligne 5 : `  print(msg)`
→ Instruction Lua générique.

Ligne 6 : `  if term and colors then term.setTextColor(colors.white) end`
→ Interaction avec l'affichage terminal (UI).

Ligne 7 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 9 : `-- Télécharge via pastebin`
→ Commentaire du développeur.

Ligne 10 : `local function download(id, dest)`
→ Instruction Lua générique.

Ligne 11 : `  if fs.exists(dest) then fs.delete(dest) end`
→ Suppression de fichier (danger : perte de données).
■■ Suppression de fichier — protéger par confirmation/whitelist.

Ligne 12 : `  return shell.run("pastebin get " .. id .. " " .. dest)`
→ Exécution d'un autre script via le shell.
■■ Exécution de script — éviter entrées non fiables.

Ligne 13 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 15 : `local function readAll(p)`
→ Instruction Lua générique.

Ligne 16 : `  if not fs.exists(p) or fs.isDir(p) then return "" end`
→ Instruction Lua générique.

Ligne 17 : `  local f = fs.open(p, "r"); local s = f.readAll() or ""; f.close(); return s`
→ Déclaration de variable locale « f » et initialisation à « fs.open(p, "r"); local s = f.readAll() or ""; f.close(); return s ».

Ligne 18 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 20 : `-- BXOR portable + FNV1a`
→ Commentaire du développeur.

Ligne 21 : `local function BXOR(a, b)`
→ Instruction Lua générique.

Ligne 22 : `  if bit and bit.bxor then return bit.bxor(a, b) end`
→ Instruction Lua générique.

Ligne 23 : `if bit32 and bit32.bxor then return bit32.bxor(a, b) end`
→ Instruction Lua générique.

Ligne 24 : `local r, v = 0, 1`
→ Instruction Lua générique.

Ligne 25 : `while a > 0 or b > 0 do`
→ Boucle WHILE : répète tant que « a > 0 or b > 0 » est vrai.

Ligne 26 : `local A, B = a % 2, b % 2`
→ Instruction Lua générique.

Ligne 27 : `if (A + B) % 2 == 1 then r = r + v end`
→ Instruction Lua générique.

Ligne 28 : `a = math.floor(a / 2); b = math.floor(b / 2); v = v * 2`
→ Affecte « math.floor(a / 2); b = math.floor(b / 2); v = v * 2 » à la variable globale « a ».

Ligne 29 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 30 : `return r`
→ Retourne une valeur au code appelant « r ».

Ligne 31 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 32 : `local function fnv1a(s)`
→ Instruction Lua générique.

Ligne 33 : `local h = 2166136261`
→ Déclaration de variable locale « h » et initialisation à « 2166136261 ».

Ligne 34 : `for i = 1, #s do`
→ Boucle FOR : itère sur « i = 1, #s ».

Ligne 35 : `h = BXOR(h, s:byte(i))`
→ Affecte « BXOR(h, s:byte(i)) » à la variable globale « h ».

Ligne 36 : `h = (h * 16777619) % 4294967296`
→ Affecte « (h * 16777619) % 4294967296 » à la variable globale « h ».

Ligne 37 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 38 : `return tostring(h)`
→ Retourne une valeur au code appelant « tostring(h) ».

Ligne 39 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 40 : `local function fileHash(path) return fnv1a(readAll(path)) end`
→ Instruction Lua générique.

Ligne 41 :
`local function extractCodeVer(txt) return txt:match('CODE_VER%s*=%s*"%s*([^"]-)%s*"') end`
→ Instruction Lua générique.

Ligne 43 : `-- IDs OFFICIELS (on n'utilise plus ceux de config.lua)`
→ Commentaire du développeur.

Ligne 44 : `local files = {`
→ Déclaration de variable locale « files » et initialisation à « { ».

Ligne 45 : `{ id = "m7wpD8wF", name = "startup.lua" },`
→ Instruction Lua générique.

Ligne 46 : `{ id = "DWHJU4bC", name = "ui.lua" },`
→ Instruction Lua générique.

Ligne 47 : `{ id = "jK7srvyY", name = "config.lua" },`
→ Instruction Lua générique.

Ligne 48 : `{ id = "gNHAVd7D", name = "update.lua" },`
→ Instruction Lua générique.

Ligne 49 : `}`
→ Instruction Lua générique.

Ligne 51 : `println(colors.cyan, "=== RagnarNet Installer ===")`
→ Instruction Lua générique.

Ligne 53 : `-- 1) Téléchargements`
→ Commentaire du développeur.

Ligne 54 : `for _, f in ipairs(files) do`
→ Boucle FOR : itère sur « _, f in ipairs(files) ».

Ligne 55 : `println(colors.lightBlue, "Telechargement de "..f.name.." ...")`
→ Instruction Lua générique.

Ligne 56 : `local ok = download(f.id, f.name)`
→ Déclaration de variable locale « ok » et initialisation à « download(f.id, f.name) ».

Ligne 57 :
`if not ok then println(colors.red, "Echec de telechargement: "..f.name); return end`
→ Instruction Lua générique.

Ligne 58 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 60 : `-- 2) Heuristique anti-sabotage pour le startup`
→ Commentaire du développeur.

Ligne 61 : `local sTxt = readAll("startup.lua")`
→ Déclaration de variable locale « sTxt » et initialisation à « readAll("startup.lua") ».

Ligne 62 : `local ok_ver  = sTxt:match('local%s+CODE_VER%s*=%s*"7%.1%.0"')`
→ Déclaration de variable locale « ok_ver » et initialisation à « sTxt:match('local%s+CODE_VER%s*=%s*"7%.1%.0"') ».

Ligne 63 : `local ok_db   = sTxt:match('usersDB%s*=%s*"users%.db"')`
→ Déclaration de variable locale « ok_db » et initialisation à « sTxt:match('usersDB%s*=%s*"users%.db"') ».

Ligne 64 : `if not (ok_ver and ok_db) then`
→ Condition IF : exécute le bloc si « not (ok_ver and ok_db) » est vrai.

Ligne 65 : `println(colors.red, "Startup invalide (signature heuristique). Annulation.")`
→ Instruction Lua générique.

Ligne 66 : `return`
→ Retourne une valeur au code appelant « ».

Ligne 67 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 69 : `-- 3) Manifest & expected version`
→ Commentaire du développeur.

Ligne 70 : `local cfg = {}`
→ Déclaration de variable locale « cfg » et initialisation à « {} ».

Ligne 71 : `local ver  = extractCodeVer(sTxt) or "7.1.0"`
→ Déclaration de variable locale « ver » et initialisation à « extractCodeVer(sTxt) or "7.1.0" ».

Ligne 73 : `cfg.expectedStartupVersion = ver`
→ Instruction Lua générique.

Ligne 74 : `cfg.autoSeal = true`
→ Instruction Lua générique.

Ligne 75 : `cfg.tamperAction, cfg.outdatedAction = "error","error"`
→ Instruction Lua générique.

Ligne 76 : `cfg.askUpdateAtBoot = true`
→ Instruction Lua générique.

Ligne 77 : `cfg.key="RAGNAR123456789KEYULTRA2025"; cfg.protocol="ragnarnet"`
→ Instruction Lua générique.

Ligne 78 : `cfg.adminUser="ragnar"; cfg.adminCode="2013.2013"`
→ Instruction Lua générique.

Ligne 79 : `cfg.spamLimit=5; cfg.maxMessageLength=200; cfg.spamResetTime=300`
→ Instruction Lua générique.

Ligne 80 : `cfg.pepper="RAG-PEPPER-2025"; cfg.pwdHashRounds=512`
→ Instruction Lua générique.

Ligne 81 : `cfg.updateURL_startup="m7wpD8wF"; cfg.updateURL_ui="DWHJU4bC"`
→ Instruction Lua générique.

Ligne 82 : `cfg.updateURL_config="jK7srvyY"; cfg.updateURL_update="gNHAVd7D"`
→ Instruction Lua générique.

Ligne 83 : `cfg.errorCodeTamper=163; cfg.errorCodeOutdated=279`
→ Instruction Lua générique.

Ligne 84 : `cfg.manifest = {`
→ Instruction Lua générique.

Ligne 85 : `["startup.lua"] = fileHash("startup.lua"),`
→ Instruction Lua générique.

Ligne 86 : `["ui.lua"] = fileHash("ui.lua"),`
→ Instruction Lua générique.

Ligne 87 : `["update.lua"] = fileHash("update.lua"),`
→ Instruction Lua générique.

Ligne 88 : `}`
→ Instruction Lua générique.

Ligne 89 : `local function writeConfigTable(tbl)`
→ Instruction Lua générique.

Ligne 90 : `local ser = textutils.serialize(tbl)`
→ Déclaration de variable locale « ser » et initialisation à « textutils.serialize(tbl) ».

Ligne 91 : `local f = fs.open("config.lua", "w"); f.write("return " .. ser); f.close()`
→ Déclaration de variable locale « f » et initialisation à « fs.open("config.lua", "w"); f.write("return " .. ser); f.close() ».
■■ Écriture — vérifier chemins & droits.

Ligne 92 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 93 : `writeConfigTable(cfg)`
→ Instruction Lua générique.

Ligne 95 : `println(colors.lime, "Installation terminee. Redemarrage dans 3 secondes...")`
→ Instruction Lua générique.

Ligne 96 : `sleep(3)`
→ Instruction Lua générique.

Ligne 97 : `os.reboot()`
→ Sécurité/maintenance : déclenche un redémarrage du système.
■■ Action critique — limiter aux cas sûrs/logués.

Module : startup.lua

Rôle détecté : Point d’entrée du système, Manipulation fichiers, Affichage terminal/UI, Exécution de scripts

Fonctions déclarées	BXOR, addMessage, askUpdateAtBoot, fileHash, fnv1a, fnvRounds,
Dépendances (require/shell.run/dofile)	config, update.lua

API	Méthodes utilisées
fs	exists, isDir, open
os	clock, epoch, pullEvent, time
http	—
shell	run
term	clear, getSize, setBackgroundColor, setCursorPos, setTextColor

Code & explication ligne par ligne

Ligne 1 : `-- startup.lua : RagnarNet OS principal v7.1.0 (restauré + durci)`
→ Commentaire du développeur.

Ligne 3 : `----------------------------------------------------------------`
→ Commentaire du développeur.

Ligne 4 : `-- >>> SECURE PREAMBLE (sans hash, anti-suppression d'appel) <<<`
→ Commentaire du développeur.

Ligne 5 : `----------------------------------------------------------------`
→ Commentaire du développeur.

Ligne 6 : `do`
→ Instruction Lua générique.

Ligne 7 : `-- 1) Fichiers essentiels présents + non vides + marqueurs de structure`
→ Commentaire du développeur.

Ligne 8 : `local essentiels = {`
→ Déclaration de variable locale « essentiels » et initialisation à « { ».

Ligne 9 : `{ "ui.lua", "return", "drawUI" }, -- doit être un module qui 'return ' un tableau + avoir drawUI`
→ Instruction Lua générique.

Ligne 10 : `{ "config.lua", "return", "{" }, -- doit 'return {'`
→ Instruction Lua générique.

Ligne 11 : `{ "users.db", nil, nil }, -- peut être vide au 1er boot, ma is doit exister`
→ Instruction Lua générique.

Ligne 12 : `}`
→ Instruction Lua générique.

Ligne 14 : `local function readAll(p)`
→ Instruction Lua générique.

Ligne 15 :
```
    if not fs or not fs.exists or not fs.exists(p) or fs.isDir(p) then return "" end
```
→ Instruction Lua générique.

Ligne 16 : `local f = fs.open(p, "r"); local s = f.readAll() or ""; f.close(); return s`
→ Déclaration de variable locale « f » et initialisation à « fs.open(p, "r"); local s = f.readAll() or ""; f.close(); return s ».

Ligne 17 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 19 : `for _, spec in ipairs(essentiels) do`
→ Boucle FOR : itère sur « _, spec in ipairs(essentiels) ».

Ligne 20 : `local path, m1, m2 = spec[1], spec[2], spec[3]`
→ Instruction Lua générique.

Ligne 21 : `if not fs or not fs.exists or not fs.exists(path) then`
→ Condition IF : exécute le bloc si « not fs or not fs.exists or not fs.exists(path) » est vrai.

Ligne 22 : `error("[SECURITE] Fichier essentiel manquant : "..tostring(path))`
→ Instruction Lua générique.

Ligne 23 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 24 : `local data = readAll(path)`
→ Déclaration de variable locale « data » et initialisation à « readAll(path) ».

Ligne 25 : `if #data == 0 and path ~= "users.db" then`
→ Condition IF : exécute le bloc si « #data == 0 and path ~= "users.db" » est vrai.

Ligne 26 : `error("[SECURITE] Fichier essentiel vide : "..tostring(path))`
→ Instruction Lua générique.

Ligne 27 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 28 : `if m1 and not data:find(m1, 1, true) then`
→ Condition IF : exécute le bloc si « m1 and not data:find(m1, 1, true) » est vrai.

Ligne 29 :
```
error("[SECURITE] Structure invalide dans "..path.." (marqueur "..m1.." absent)")
```
→ Instruction Lua générique.

Ligne 30 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 31 : `if m2 and not data:find(m2, 1, true) then`
→ Condition IF : exécute le bloc si « m2 and not data:find(m2, 1, true) » est vrai.

Ligne 32 :
```
error("[SECURITE] Structure invalide dans "..path.." (marqueur "..m2.." absent)")
```
→ Instruction Lua générique.

Ligne 33 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 34 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 35 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 36 : `-- >>> FIN PREAMBLE <<<`
→ Commentaire du développeur.

Ligne 38 : `local CODE_VER = "7.1.0"`
→ Déclaration de variable locale « CODE_VER » et initialisation à « "7.1.0" ».

Ligne 39 : `local cfg = require("config")`
→ Déclaration de variable locale « cfg » et initialisation à « require("config") ».

Ligne 41 : `--------------------------------`
→ Commentaire du développeur.

Ligne 42 : `-- Garde-fous config forcés`
→ Commentaire du développeur.

Ligne 43 : `--------------------------------`
→ Commentaire du développeur.

Ligne 44 :
```
cfg.spamLimit        = math.max(1,  math.min(50,   tonumber(cfg.spamLimit or 5)))
```
→ Instruction Lua générique.

Ligne 45 :
```
cfg.maxMessageLength = math.max(10, math.min(500,  tonumber(cfg.maxMessageLength or 200)))
```
→ Instruction Lua générique.

Ligne 46 :
```
cfg.pwdHashRounds    = math.max(128,math.min(4096, tonumber(cfg.pwdHashRounds or 512)))
```
→ Instruction Lua générique.

Ligne 47 : `if cfg.strict_mode == false then cfg.strict_mode = true end`
→ Instruction Lua générique.

Ligne 48 : `-- Empêcher un contournement via actions trop ?douces?`
→ Commentaire du développeur.

Ligne 49 : `cfg.tamperAction   = (cfg.tamperAction   == "halt" or cfg.tamperAction   == "er ror") and cfg.tamperAction   or "error"`
→ Instruction Lua générique.

Ligne 50 : `cfg.outdatedAction = (cfg.outdatedAction == "halt" or cfg.outdatedAction == "error") and cfg.outdatedAction or "error"`
→ Instruction Lua générique.

Ligne 52 : `----------------`
→ Commentaire du développeur.

Ligne 53 : `-- Utils`
→ Commentaire du développeur.

Ligne 54 : `----------------`
→ Commentaire du développeur.

Ligne 55 : `local function BXOR(a, b)`
→ Instruction Lua générique.

Ligne 56 : `  if bit and bit.bxor then return bit.bxor(a, b) end`
→ Instruction Lua générique.

Ligne 57 : `  if bit32 and bit32.bxor then return bit32.bxor(a, b) end`
→ Instruction Lua générique.

Ligne 58 : `  local r, v = 0, 1`
→ Instruction Lua générique.

Ligne 59 : `  while a > 0 or b > 0 do`
→ Boucle WHILE : répète tant que « a > 0 or b > 0 » est vrai.

Ligne 60 : `    local A, B = a % 2, b % 2`
→ Instruction Lua générique.

Ligne 61 : `    if (A + B) % 2 == 1 then r = r + v end`
→ Instruction Lua générique.

Ligne 62 : `    a = math.floor(a / 2); b = math.floor(b / 2); v = v * 2`
→ Affecte « math.floor(a / 2); b = math.floor(b / 2); v = v * 2 » à la variable globale « a ».

Ligne 63 : `  end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 64 : `  return r`
→ Retourne une valeur au code appelant « r ».

Ligne 65 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 67 : `local function readAll(p)`
→ Instruction Lua générique.

Ligne 68 : `  if not fs.exists(p) or fs.isDir(p) then return "" end`
→ Instruction Lua générique.

Ligne 69 : `  local f = fs.open(p, "r"); local s = f.readAll() or ""; f.close(); return s`
→ Déclaration de variable locale « f » et initialisation à « fs.open(p, "r"); local s = f.readAll() or ""; f.close(); return s ».

Ligne 70 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 72 : `local function fnv1a(s)`
→ Instruction Lua générique.

Ligne 73 : `  local h = 2166136261`
→ Déclaration de variable locale « h » et initialisation à « 2166136261 ».

Ligne 74 : `  for i = 1, #s do`
→ Boucle FOR : itère sur « i = 1, #s ».

Ligne 75 : `    h = BXOR(h, s:byte(i))`
→ Affecte « BXOR(h, s:byte(i)) » à la variable globale « h ».

Ligne 76 : `    h = (h * 16777619) % 4294967296`
→ Affecte « (h * 16777619) % 4294967296 » à la variable globale « h ».

Ligne 77 : `  end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 78 : `  return tostring(h)`
→ Retourne une valeur au code appelant « tostring(h) ».

Ligne 79 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 80 : `local function fileHash(path) return fnv1a(readAll(path)) end`
→ Instruction Lua générique.

Ligne 82 : `local function writeConfigTable(tbl)`
→ Instruction Lua générique.

Ligne 83 : `local ser = textutils.serialize(tbl)`
→ Déclaration de variable locale « ser » et initialisation à « textutils.serialize(tbl) ».

Ligne 84 : `local f = fs.open("config.lua", "w"); f.write("return " .. ser); f.close()`
→ Déclaration de variable locale « f » et initialisation à « fs.open("config.lua", "w"); f.write("return " .. ser); f.close() ».
■■ Écriture — vérifier chemins & droits.

Ligne 85 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 87 : `------------------------`
→ Commentaire du développeur.

Ligne 88 : `-- Erreurs propres`
→ Commentaire du développeur.

Ligne 89 : `------------------------`
→ Commentaire du développeur.

Ligne 90 : `local function showErrorAndExit(code, reason)`
→ Instruction Lua générique.

Ligne 91 : `term.setBackgroundColor(colors.black)`
→ Interaction avec l'affichage terminal (UI).

Ligne 92 : `term.setTextColor(colors.red)`
→ Interaction avec l'affichage terminal (UI).

Ligne 93 : `term.clear()`
→ Interaction avec l'affichage terminal (UI).

Ligne 94 : `term.setCursorPos(2,2)`
→ Interaction avec l'affichage terminal (UI).

Ligne 95 : `print("[ERREUR " .. tostring(code) .. "] RagnarNet")`
→ Instruction Lua générique.

Ligne 96 : `term.setTextColor(colors.white)`
→ Interaction avec l'affichage terminal (UI).

Ligne 97 : `print(reason or "Erreur de securite")`
→ Instruction Lua générique.

Ligne 98 :
```
print("\nLe programme s'arrete. Lance 'update' ou utilise la disquette de recovery.")
```
→ Instruction Lua générique.

Ligne 99 : `sleep(2.5)`
→ Instruction Lua générique.

Ligne 100 : `error("ERR_"..tostring(code), 0)`
→ Instruction Lua générique.

Ligne 101 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 103 : `local function handleBreach(kind, reason, action)`
→ Instruction Lua générique.

Ligne 104 : `action = action or "error"`
→ Affecte « action or "error" » à la variable globale « action ».

Ligne 105 : `local code = 199`
→ Déclaration de variable locale « code » et initialisation à « 199 ».

Ligne 106 : `if kind == "tamper"   then code = (cfg.errorCodeTamper or 163) end`
→ Instruction Lua générique.

Ligne 107 : `if kind == "outdated" then code = (cfg.errorCodeOutdated or 279) end`
→ Instruction Lua générique.

Ligne 108 : `if action == "error" then showErrorAndExit(code, reason)`
→ Instruction Lua générique.

Ligne 109 : `elseif action == "halt" then error(reason or "Security halt", 0)`
→ Instruction Lua générique.

Ligne 110 : `else showErrorAndExit(199, reason or "Security error") end`
→ Instruction Lua générique.

Ligne 111 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 113 : `------------------------------------------------`
→ Commentaire du développeur.

Ligne 114 : `-- Intégrité (avec anti-reseal et auto-run)`
→ Commentaire du développeur.

Ligne 115 : `----------------------------------------------------`
→ Commentaire du développeur.

Ligne 116 : `local function integrityCheck()`
→ Instruction Lua générique.

Ligne 117 : `  -- Anti-reseal : un seul ?seal? autorisé à l?installation`
→ Commentaire du développeur.

Ligne 118 : `  local sealedFlag = ".sealed"`
→ Déclaration de variable locale « sealedFlag » et initialisation à « ".sealed" ».

Ligne 119 : `  local firstBoot = not fs.exists(sealedFlag)`
→ Déclaration de variable locale « firstBoot » et initialisation à « not fs.exists(sealedFlag) ».

Ligne 121 : `  -- 1) Version attendue (si définie dans config)`
→ Commentaire du développeur.

Ligne 122 :
```
  if cfg.expectedStartupVersion and cfg.expectedStartupVersion ~= CODE_VER then
```
→ Condition IF : exécute le bloc si « cfg.expectedStartupVersion and cfg.expectedStartupVersion ~= CODE_VER » est vrai.

Ligne 123 : `    handleBreach("outdated",`
→ Instruction Lua générique.

Ligne 124 : `      "Version trop ancienne: "..tostring(CODE_VER).." (attendue "..tostring(cfg.expectedStartupVersion)..")",`
→ Instruction Lua générique.

Ligne 125 : `      cfg.outdatedAction or cfg.tamperAction or "error")`
→ Instruction Lua générique.

Ligne 126 : `  end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 128 : `  -- 2) Cibles d?intégrité ? on n?inclut jamais config.lua`
→ Commentaire du développeur.

Ligne 129 : `  local targets = { "startup.lua", "ui.lua", "update.lua" }`
→ Déclaration de variable locale « targets » et initialisation à « { "startup.lua", "ui.lua", "update.lua" } ».

Ligne 131 : `  -- 3) Manifest`
→ Commentaire du développeur.

Ligne 132 : `  if not cfg.manifest then`
→ Condition IF : exécute le bloc si « not cfg.manifest » est vrai.

Ligne 133 : `    if cfg.autoSeal and firstBoot then`
→ Condition IF : exécute le bloc si « cfg.autoSeal and firstBoot » est vrai.

Ligne 134 : `      local newcfg = {}; for k,v in pairs(cfg) do newcfg[k] = v end`
→ Déclaration de variable locale « newcfg » et initialisation à « {}; for k,v in pairs(cfg) do newcfg[k] = v end ».

Ligne 135 : `      newcfg.manifest = {}`
→ Instruction Lua générique.

Ligne 136 : `      for _, p in ipairs(targets) do newcfg.manifest[p] = fileHash(p) end`
→ Instruction Lua générique.

Ligne 137 : `      writeConfigTable(newcfg)`
→ Instruction Lua générique.

Ligne 138 : `      local f = fs.open(sealedFlag, "w"); f.write("ok"); f.close()`
→ Déclaration de variable locale « f » et initialisation à « fs.open(sealedFlag, "w"); f.write("ok"); f.close() ».
■■ Écriture — vérifier chemins & droits.

Ligne 139 : `      term.setTextColor(colors.lime); print("[Integrite] Scellage initial OK (manifest ecrit)."); term.setTextColor(colors.white)`
→ Interaction avec l'affichage terminal (UI).

Ligne 140 : `    else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 141 :
```
      -- Si manifest absent mais pas ?vraie? installation : tentative de reseal -> breach
```
→ Commentaire du développeur.

Ligne 142 : `      local msg = firstBoot and "Manifest absent et autoSeal=false (installation corrompue)."`
→ Déclaration de variable locale « msg » et initialisation à « firstBoot and "Manifest absent et autoSeal=false (installation corrompue)." ».

Ligne 143 :
```
                                    or  "Manifest absent (tentative de re-scellage interdite)."
```
→ Instruction Lua générique.

Ligne 144 : `handleBreach("tamper", msg, cfg.tamperAction or "error")`
→ Instruction Lua générique.

Ligne 145 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 146 : `else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 147 : `for _, p in ipairs(targets) do`
→ Boucle FOR : itère sur « _, p in ipairs(targets) ».

Ligne 148 : `local exp, act = cfg.manifest[p], fileHash(p)`
→ Instruction Lua générique.

Ligne 149 : `if not exp or exp ~= act then`
→ Condition IF : exécute le bloc si « not exp or exp ~= act » est vrai.

Ligne 150 :
```
        handleBreach("tamper", "Integrite rompue sur: "..p, cfg.tamperAction or "error")
```
→ Instruction Lua générique.

Ligne 151 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 152 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 153 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 154 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 156 : `-- AUTO-RUN : vérifie l?intégrité même si l?appel plus bas est supprimé`
→ Commentaire du développeur.

Ligne 157 : `do`
→ Instruction Lua générique.

Ligne 158 : `local ok, err = pcall(integrityCheck)`
→ Appel protégé (pcall) pour capturer les erreurs.

Ligne 159 : `if not ok then`
→ Condition IF : exécute le bloc si « not ok » est vrai.

Ligne 160 : `handleBreach("tamper", "Echec verif integrite: "..tostring(err), cfg.tampe rAction or "error")`
→ Instruction Lua générique.

Ligne 161 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 162 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 164 : `--------------------------------------------------`
→ Commentaire du développeur.

Ligne 165 : `-- MAJ au démarrage (prompt unique)`
→ Commentaire du développeur.

Ligne 166 : `--------------------------------------------------`
→ Commentaire du développeur.

Ligne 167 : `local _askedUpdateOnce = false`
→ Déclaration de variable locale « _askedUpdateOnce » et initialisation à « false ».

Ligne 168 : `local function askUpdateAtBoot()`
→ Instruction Lua générique.

Ligne 169 : `if _askedUpdateOnce then return end`
→ Instruction Lua générique.

Ligne 170 : `_askedUpdateOnce = true`
→ Affecte « true » à la variable globale « _askedUpdateOnce ».

Ligne 171 : `if cfg.askUpdateAtBoot == false then return end`
→ Instruction Lua générique.

Ligne 172 : `if not fs.exists("update.lua") then return end`
→ Instruction Lua générique.

Ligne 173 : `term.setTextColor(colors.cyan); print("\nFaire la mise a jour maintenant ? (o/n)"); term.setTextColor(colors.white)`

→ Interaction avec l'affichage terminal (UI).

Ligne 174 : `local a = read()`

→ Déclaration de variable locale « a » et initialisation à « read() ».

■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 175 : `if a and a:lower() == "o" then`

→ Condition IF : exécute le bloc si « a and a:lower() == "o" » est vrai.

Ligne 176 : `local dat = readAll("update.lua")`

→ Déclaration de variable locale « dat » et initialisation à « readAll("update.lua") ».

Ligne 177 : `if dat == "" then showErrorAndExit(503, "update.lua invalide ou vide") end`

→ Instruction Lua générique.

Ligne 178 : `shell.run("update.lua")`

→ Exécution d'un autre script via le shell.

■■ Exécution de script — éviter entrées non fiables.

Ligne 179 : `end`

→ Fin de bloc (fonction/condition/boucle).

Ligne 180 : `end`

→ Fin de bloc (fonction/condition/boucle).

Ligne 182 : `-- ====== Boot: intégrité -> prompt MAJ -> charge UI ======`

→ Commentaire du développeur.

Ligne 183 : `-- Même si quelqu?un supprime la ligne suivante, l?intégrité a déjà été vérifiée (auto-run ci-dessus).`

→ Commentaire du développeur.

Ligne 184 : `integrityCheck()`

→ Instruction Lua générique.

Ligne 185 : `askUpdateAtBoot()`

→ Instruction Lua générique.

Ligne 187 : `---------------------------------`

→ Commentaire du développeur.

Ligne 188 : `-- Chargement et vérif de l'UI`

→ Commentaire du développeur.

Ligne 189 : `---------------------------------`

→ Commentaire du développeur.

Ligne 190 : `local function loadUI()`

→ Instruction Lua générique.

Ligne 191 : `if package and package.loaded then package.loaded["ui"] = nil end`

→ Instruction Lua générique.

Ligne 192 : `local ok, mod = pcall(dofile, "ui.lua")`

→ Appel protégé (pcall) pour capturer les erreurs.

Ligne 193 : `if not ok then showErrorAndExit(501, "ui.lua: "..tostring(mod)) end`

→ Instruction Lua générique.

Ligne 194 : `if type(mod) ~= "table" or not mod.drawUI or not mod.showMessages then`

→ Condition IF : exécute le bloc si « type(mod) ~= "table" or not mod.drawUI or not mod.showMessages » est vrai.

Ligne 195 : `showErrorAndExit(502, "ui.lua invalide (fonctions manquantes)")`

→ Instruction Lua générique.

Ligne 196 : `end`

→ Fin de bloc (fonction/condition/boucle).

Ligne 197 : `return mod`

→ Retourne une valeur au code appelant « mod ».

Ligne 198 : `end`

→ Fin de bloc (fonction/condition/boucle).

Ligne 199 : `local ui = loadUI()`

→ Déclaration de variable locale « ui » et initialisation à « loadUI() ».

Ligne 201 : `-----------------------------`

→ Commentaire du développeur.

Ligne 202 : `-- App / runtime`

→ Commentaire du développeur.

Ligne 203 : `----------------------------`
→ Commentaire du développeur.

Ligne 204 : `local w, h     = term.getSize()`
→ Interaction avec l'affichage terminal (UI).

Ligne 205 : `local uiHeight = h - 6`
→ Déclaration de variable locale « uiHeight » et initialisation à « h - 6 ».

Ligne 206 : `local usersDB  = "users.db"`
→ Déclaration de variable locale « usersDB » et initialisation à « "users.db" ».

Ligne 208 : `-- Nettoyage d?artefacts connus`
→ Commentaire du développeur.

Ligne 209 :
`for _, f in ipairs({"HACKER.db"}) do if fs.exists(f) then pcall(fs.delete, f) end end`
→ Appel protégé (pcall) pour capturer les erreurs.

Ligne 211 :
`local messages, users, spamTracker, blacklist, banDuration = {}, {}, {}, {}, {}`
→ Instruction Lua générique.

Ligne 212 : `local username, isAdmin, lockdown = "?", false, false`
→ Instruction Lua générique.

Ligne 214 : `local function xorCrypt(msg, keyStr)`
→ Instruction Lua générique.

Ligne 215 : `local out = {}`
→ Déclaration de variable locale « out » et initialisation à « {} ».

Ligne 216 : `for i = 1, #msg do`
→ Boucle FOR : itère sur « i = 1, #msg ».

Ligne 217 : `local m = msg:byte(i)`
→ Déclaration de variable locale « m » et initialisation à « msg:byte(i) ».

Ligne 218 : `local k = keyStr:byte((i - 1) % #keyStr + 1)`
→ Déclaration de variable locale « k » et initialisation à « keyStr:byte((i - 1) % #keyStr + 1) ».

Ligne 219 : `out[i] = string.char(BXOR(m, k))`
→ Instruction Lua générique.

Ligne 220 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 221 : `return table.concat(out)`
→ Retourne une valeur au code appelant « table.concat(out) ».

Ligne 222 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 224 : `local function addMessage(from, text, adminFlag)`
→ Instruction Lua générique.

Ligne 225 : `table.insert(messages, { id = #messages + 1, from = from, text = text, admin = adminFlag or false })`
→ Instruction Lua générique.

Ligne 226 : `ui.drawUI(username, isAdmin, w, h, CODE_VER)`
→ Instruction Lua générique.

Ligne 227 : `ui.showMessages(messages, uiHeight, blacklist, cfg.adminUser)`
→ Instruction Lua générique.

Ligne 228 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 230 : `-- Users (hash + migration)`
→ Commentaire du développeur.

Ligne 231 : `local function loadUsers()`
→ Instruction Lua générique.

Ligne 232 : `if not fs.exists(usersDB) then return {} end`
→ Instruction Lua générique.

Ligne 233 :
`local f = fs.open(usersDB, "r"); local d = textutils.unserialize(f.readAll()); f.close()`
→ Déclaration de variable locale « f » et initialisation à « fs.open(usersDB, "r"); local d = textutils.unserialize(f.readAll()); f.close() ».

Ligne 234 : `return d or {}`
→ Retourne une valeur au code appelant « d or {} ».

Ligne 235 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 236 : `local function saveUsers(u)`
→ Instruction Lua générique.

Ligne 237 : `local f = fs.open(usersDB, "w"); f.write(textutils.serialize(u)); f.close()`
→ Déclaration de variable locale « f » et initialisation à « fs.open(usersDB, "w"); f.write(textutils.serialize(u)); f.close() ».
■■ Écriture — vérifier chemins & droits.

Ligne 238 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 239 : `local function randHex(n) local s={} for i=1,n do s[i]=string.format("%x",math.random(0,15)) end return table.concat(s) end`
→ Instruction Lua générique.

Ligne 240 : `local function fnvRounds(s) local r=tonumber(cfg.pwdHashRounds or 512) or 512; local h=s; for _=1,r do h=fnv1a(h) end; return h end`
→ Instruction Lua générique.

Ligne 241 : `local function hashPassword(pwd, salt) return fnvRounds(tostring(salt or "") .. tostring(pwd or "") .. tostring(cfg.pepper or "")) end`
→ Instruction Lua générique.

Ligne 242 : `local function verifyPassword(stored, input)`
→ Instruction Lua générique.

Ligne 243 : `if type(stored)=="string" then return stored==input, "legacy"`
→ Instruction Lua générique.

Ligne 244 : `elseif type(stored)=="table" and stored.salt and stored.hash then return stored.hash==hashPassword(input,stored.salt),"hashed" end`
→ Instruction Lua générique.

Ligne 245 : `return false,"unknown"`
→ Retourne une valeur au code appelant « false,"unknown" ».

Ligne 246 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 248 : `-- Modem`
→ Commentaire du développeur.

Ligne 249 : `for _, side in ipairs({"left","right","top","bottom","front","back"}) do`
→ Boucle FOR : itère sur « _, side in ipairs({"left","right","top","bottom","front","back"}) ».

Ligne 250 : `if peripheral.getType(side) == "modem" then rednet.open(side); break end`
→ Instruction Lua générique.

Ligne 251 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 253 : `-- Login`
→ Commentaire du développeur.

Ligne 254 : `users = loadUsers()`
→ Affecte « loadUsers() » à la variable globale « users ».

Ligne 255 : `math.randomseed(os.epoch and os.epoch("utc") or os.time() or os.clock())`
→ Instruction Lua générique.

Ligne 257 :
`term.setTextColor(colors.yellow) write("Pseudo > "); term.setTextColor(colors.white)`
→ Interaction avec l'affichage terminal (UI).

Ligne 258 : `username = read()`
→ Affecte « read() » à la variable globale « username ».
■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 260 : `if users[username] then`
→ Condition IF : exécute le bloc si « users[username] » est vrai.

Ligne 261 : `term.setTextColor(colors.yellow) write("Mot de passe > "); term.setTextColor(colors.white)`
→ Interaction avec l'affichage terminal (UI).

Ligne 262 : `local pwd = read("*")`
→ Déclaration de variable locale « pwd » et initialisation à « read("*") ».
■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 263 : `local ok, mode = verifyPassword(users[username], pwd)`
→ Instruction Lua générique.

Ligne 264 : `if not ok then print("Mot de passe incorrect.") return end`
→ Instruction Lua générique.

Ligne 265 : `if mode == "legacy" then`
→ Condition IF : exécute le bloc si « mode == "legacy" » est vrai.

Ligne 266 : `local salt = randHex(16)`
→ Déclaration de variable locale « salt » et initialisation à « randHex(16) ».

Ligne 267 : `users[username] = { salt = salt, hash = hashPassword(pwd, salt) }`
→ Instruction Lua générique.

Ligne 268 : `saveUsers(users)`
→ Instruction Lua générique.

Ligne 269 : `addMessage("SYSTEM", "Compte migre vers hachage.", false)`
→ Instruction Lua générique.

Ligne 270 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 271 : `else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 272 : `term.setTextColor(colors.yellow) write("Creer un mot de passe > "); term.set TextColor(colors.white)`
→ Interaction avec l'affichage terminal (UI).

Ligne 273 : `local pwd = read("*"); local salt = randHex(16)`
→ Déclaration de variable locale « pwd » et initialisation à « read("*"); local salt = randHex(16) ».
■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 274 : `users[username] = { salt = salt, hash = hashPassword(pwd, salt) }`
→ Instruction Lua générique.

Ligne 275 : `saveUsers(users)`
→ Instruction Lua générique.

Ligne 276 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 278 : `if username == cfg.adminUser then`
→ Condition IF : exécute le bloc si « username == cfg.adminUser » est vrai.

Ligne 279 : `write("Code Ragnar > ")`
→ Instruction Lua générique.

Ligne 280 :
```
if read() == cfg.adminCode then isAdmin = true else print("Code incorrect.") return end
```
→ Instruction Lua générique.
■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 281 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 283 : `-- UI initiale`
→ Commentaire du développeur.

Ligne 284 : `ui.drawUI(username, isAdmin, w, h, CODE_VER)`
→ Instruction Lua générique.

Ligne 285 : `ui.showMessages(messages, uiHeight, blacklist, cfg.adminUser)`
→ Instruction Lua générique.

Ligne 287 : `-- Watchdog runtime : re-vérifie l?essentiel régulièrement`
→ Commentaire du développeur.

Ligne 288 : `local function watchdog()`
→ Instruction Lua générique.

Ligne 289 : `while true do`
→ Boucle WHILE : répète tant que « true » est vrai.

Ligne 290 : `-- re-check fichiers essentiels`
→ Commentaire du développeur.

Ligne 291 : `local ok, err = pcall(function()`
→ Appel protégé (pcall) pour capturer les erreurs.

Ligne 292 : `local marks = {`
→ Déclaration de variable locale « marks » et initialisation à « { ».

Ligne 293 : `{ "ui.lua", "return", "drawUI" },`
→ Instruction Lua générique.

Ligne 294 : `{ "config.lua", "return", "{" },`
→ Instruction Lua générique.

Ligne 295 : `{ "users.db", nil, nil },`
→ Instruction Lua générique.

Ligne 296 : `}`
→ Instruction Lua générique.

Ligne 297 : `for _, spec in ipairs(marks) do`
→ Boucle FOR : itère sur « _, spec in ipairs(marks) ».

Ligne 298 : `local path, m1, m2 = spec[1], spec[2], spec[3]`
→ Instruction Lua générique.

Ligne 299 : `if not fs.exists(path) then error("Essentiel supprimé: "..path) end`
→ Instruction Lua générique.

Ligne 300 : `local data = readAll(path)`
→ Déclaration de variable locale « data » et initialisation à « readAll(path) ».

Ligne 301 :
`if #data == 0 and path ~= "users.db" then error("Essentiel vide: "..path) end`
→ Instruction Lua générique.

Ligne 302 : `if m1 and not data:find(m1, 1, true) then error("Structure invalide (manque "..m1..") dans "..path) end`
→ Instruction Lua générique.

Ligne 303 : `if m2 and not data:find(m2, 1, true) then error("Structure invalide (manque "..m2..") dans "..path) end`
→ Instruction Lua générique.

Ligne 304 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 305 : `end)`
→ Instruction Lua générique.

Ligne 306 : `if not ok then`
→ Condition IF : exécute le bloc si « not ok » est vrai.

Ligne 307 : `handleBreach("tamper", "Watchdog: "..tostring(err), "error")`
→ Instruction Lua générique.

Ligne 308 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 309 : `sleep(math.max(2, tonumber(cfg.watchdogDelay or 5)))`
→ Instruction Lua générique.

Ligne 310 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 311 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 313 : `-- Threads`
→ Commentaire du développeur.

Ligne 314 :
`local function spamReset() while true do sleep(cfg.spamResetTime) spamTracker = {} end end`
→ Instruction Lua générique.

Ligne 315 : `local function handleClick()`
→ Instruction Lua générique.

Ligne 316 : `while true do`
→ Boucle WHILE : répète tant que « true » est vrai.

Ligne 317 : `local _, _, x, y = os.pullEvent("mouse_click")`
→ Instruction Lua générique.

Ligne 318 : `if y == 1 and x >= w - 12 then`
→ Condition IF : exécute le bloc si « y == 1 and x >= w - 12 » est vrai.

Ligne 319 : `term.setTextColor(colors.red) print("\nConfirmer l'arret ? (o/n)")`
→ Interaction avec l'affichage terminal (UI).

Ligne 320 : `term.setTextColor(colors.white)`
→ Interaction avec l'affichage terminal (UI).

Ligne 321 : `if (read() or ""):lower() == "o" then error("Arret utilisateur", 0) end`
→ Instruction Lua générique.

■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 322 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 323 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 324 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 325 : `local function receiver()`
→ Instruction Lua générique.

Ligne 326 : `while true do`
→ Boucle WHILE : répète tant que « true » est vrai.

Ligne 327 : `local _, encrypted = rednet.receive(cfg.protocol)`
→ Instruction Lua générique.

Ligne 328 : `local raw = (encrypted and cfg.key) and (function(m,k)`
→ Déclaration de variable locale « raw » et initialisation à « (encrypted and cfg.key) and (function(m,k) ».

Ligne 329 : `local out, b = {}, nil`
→ Instruction Lua générique.

Ligne 330 : `for i = 1, #m do`
→ Boucle FOR : itère sur « i = 1, #m ».

Ligne 331 : `local mm = m:byte(i)`
→ Déclaration de variable locale « mm » et initialisation à « m:byte(i) ».

Ligne 332 : `local kk = k:byte((i - 1) % #k + 1)`
→ Déclaration de variable locale « kk » et initialisation à « k:byte((i - 1) % #k + 1) ».

Ligne 333 : `out[i] = string.char(BXOR(mm, kk))`
→ Instruction Lua générique.

Ligne 334 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 335 : `return table.concat(out)`
→ Retourne une valeur au code appelant « table.concat(out) ».

Ligne 336 : `end)(encrypted, cfg.key) or ""`
→ Instruction Lua générique.

Ligne 337 : `local from, text = raw:match("(.+):(.+)")`
→ Instruction Lua générique.

Ligne 338 : `if not from or not text then`
→ Condition IF : exécute le bloc si « not from or not text » est vrai.

Ligne 339 : `-- DoS fix : on ignore le paquet mal formé`
→ Commentaire du développeur.

Ligne 340 : `sleep(0)`
→ Instruction Lua générique.

Ligne 341 : `else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 342 : `if #text > cfg.maxMessageLength then`
→ Condition IF : exécute le bloc si « #text > cfg.maxMessageLength » est vrai.

Ligne 343 : `addMessage("SYSTEM","Message trop long de "..from)`
→ Instruction Lua générique.

Ligne 344 : `elseif blacklist[from] then`
→ Branche ELSEIF : alternative si « blacklist[from] » est vrai.

Ligne 345 : `-- ignore`
→ Commentaire du développeur.

Ligne 346 : `elseif from ~= username then`
→ Branche ELSEIF : alternative si « from ~= username » est vrai.

Ligne 347 : `spamTracker[from] = (spamTracker[from] or 0) + 1`
→ Instruction Lua générique.

Ligne 348 : `if spamTracker[from] > cfg.spamLimit and from ~= cfg.adminUser then`
→ Condition IF : exécute le bloc si « spamTracker[from] > cfg.spamLimit and from ~= cfg.adminUser » est vrai.

Ligne 349 : `banDuration[from] = (banDuration[from] or 7200) * 5`
→ Instruction Lua générique.

Ligne 350 : `blacklist[from] = true`
→ Instruction Lua générique.

Ligne 351 : `addMessage("SYSTEM", from.." banni pour spam")`
→ Instruction Lua générique.

Ligne 352 : `else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 353 : `addMessage(from, text, (from == cfg.adminUser))`
→ Instruction Lua générique.

Ligne 354 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 355 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 356 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 357 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 358 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 359 : `local function sender()`
→ Instruction Lua générique.

Ligne 360 : `while true do`
→ Boucle WHILE : répète tant que « true » est vrai.

Ligne 361 : `term.setCursorPos(2, h); term.setTextColor(colors.white); write("Vous > ")`
→ Interaction avec l'affichage terminal (UI).

Ligne 362 : `local input = read()`
→ Déclaration de variable locale « input » et initialisation à « read() ».
■■ Saisie — masquer mots de passe, filtrer injections.

Ligne 363 : `if input ~= "" and not lockdown then`
→ Condition IF : exécute le bloc si « input ~= "" and not lockdown » est vrai.

Ligne 364 : `if #input > cfg.maxMessageLength then`
→ Condition IF : exécute le bloc si « #input > cfg.maxMessageLength » est vrai.

Ligne 365 :
```
addMessage("SYSTEM","Message trop long (max "..cfg.maxMessageLength..")")
```
→ Instruction Lua générique.

Ligne 366 : `else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 367 : `local plain = username..":"..input`
→ Déclaration de variable locale « plain » et initialisation à « username..":"..input ».

Ligne 368 : `rednet.broadcast(xorCrypt(plain, cfg.key), cfg.protocol)`
→ Instruction Lua générique.

Ligne 369 : `addMessage(username, input, isAdmin)`
→ Instruction Lua générique.

Ligne 370 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 371 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 372 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 373 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 375 : `parallel.waitForAny(receiver, sender, handleClick, spamReset, watchdog)`
→ Instruction Lua générique.

Module : config.lua

Rôle détecté : Configuration globale

Fonctions déclarées	—
Dépendances (require/shell.run/dofile)	—

API	Méthodes utilisées
fs	—
os	—
http	—
shell	—
term	—

Code & explication ligne par ligne

Ligne 1 : `return {`

→ Retourne une valeur au code appelant « { ».

Ligne 2 : `updateURL_update = "gNHAVd7D",`

→ Affecte « "gNHAVd7D", » à la variable globale « updateURL_update ».

Ligne 3 : `spamLimit = 5,`

→ Affecte « 5, » à la variable globale « spamLimit ».

Ligne 4 : `errorCodeOutdated = 279,`

→ Affecte « 279, » à la variable globale « errorCodeOutdated ».

Ligne 5 : `maxMessageLength = 200,`

→ Affecte « 200, » à la variable globale « maxMessageLength ».

Ligne 6 : `adminUser = "ragnar",`

→ Affecte « "ragnar", » à la variable globale « adminUser ».

Ligne 7 : `updateURL_ui = "DWHJU4bC",`

→ Affecte « "DWHJU4bC", » à la variable globale « updateURL_ui ».

Ligne 8 : `manifest = {`

→ Affecte « { » à la variable globale « manifest ».

Ligne 9 : `[ "update.lua" ] = "723186416",`

→ Instruction Lua générique.

Ligne 10 : `[ "startup.lua" ] = "2620602876",`

→ Instruction Lua générique.

Ligne 11 : `[ "ui.lua" ] = "322429472",`

→ Instruction Lua générique.

Ligne 12 : `},`

→ Instruction Lua générique.

Ligne 13 : `pepper = "RAG-PEPPER-2025",`

→ Affecte « "RAG-PEPPER-2025", » à la variable globale « pepper ».

Ligne 14 : `key = "RAGNAR123456789KEYULTRA2025",`

→ Affecte « "RAGNAR123456789KEYULTRA2025", » à la variable globale « key ».

Ligne 15 : `expectedStartupVersion = "7.1.0",`

→ Affecte « "7.1.0", » à la variable globale « expectedStartupVersion ».

Ligne 16 : `tamperAction = "error",`

→ Affecte « "error", » à la variable globale « tamperAction ».

Ligne 17 : `pwdHashRounds = 512,`

→ Affecte « 512, » à la variable globale « pwdHashRounds ».

Ligne 18 : `autoSeal = true,`

→ Affecte « true, » à la variable globale « autoSeal ».

Ligne 19 : `askUpdateAtBoot = true,`

→ Affecte « true, » à la variable globale « askUpdateAtBoot ».

Ligne 20 : `errorCodeTamper = 163,`

→ Affecte « 163, » à la variable globale « errorCodeTamper ».

Ligne 21 : `updateURL_config = "jK7srvyY",`
→ Affecte « "jK7srvyY", » à la variable globale « updateURL_config ».

Ligne 22 : `updateURL_startup = "m7wpD8wF",`
→ Affecte « "m7wpD8wF", » à la variable globale « updateURL_startup ».

Ligne 23 : `protocol = "ragnarnet",`
→ Affecte « "ragnarnet", » à la variable globale « protocol ».

Ligne 24 : `adminCode = "2013.2013",`
→ Affecte « "2013.2013", » à la variable globale « adminCode ».

Ligne 25 : `outdatedAction = "error",`
→ Affecte « "error", » à la variable globale « outdatedAction ».

Ligne 26 : `spamResetTime = 300,`
→ Affecte « 300, » à la variable globale « spamResetTime ».

Ligne 27 : `}`
→ Instruction Lua générique.

Module : ui.lua

Rôle détecté : Interface utilisateur, Affichage terminal/UI

Fonctions déclarées	fill, footBar, titleBar, trunc, ui.drawUI, ui.showMessages
Dépendances (require/shell.run/dofile)	—

API	Méthodes utilisées
fs	—
os	—
http	—
shell	—
term	clear, clearLine, getSize, isColor, setBackgroundColor, setCursorPos, setTextColor, write

Code & explication ligne par ligne

Ligne 1 : `-- ui.lua : RagnarNet UI (adapté 7.2.x)`
→ Commentaire du développeur.

Ligne 2 : `-- Exporte: ui.drawUI(username, isAdmin, w, h, version)`
→ Commentaire du développeur.

Ligne 3 : `--          ui.showMessages(messages, uiHeight, blacklist, adminUser)`
→ Commentaire du développeur.

Ligne 5 : `local ui = {}`
→ Déclaration de variable locale « ui » et initialisation à « {} ».

Ligne 7 : `-- Couleurs sûres (fallback si écran non couleur)`
→ Commentaire du développeur.

Ligne 8 : `local HAS_COLOR = term.isColor and term.isColor()`
→ Déclaration de variable locale « HAS_COLOR » et initialisation à « term.isColor and term.isColor() ».

Ligne 9 : `local C_BG      = HAS_COLOR and colors.lightGray or colors.white`
→ Déclaration de variable locale « C_BG » et initialisation à « HAS_COLOR and colors.lightGray or colors.white ».

Ligne 10 : `local C_TITLE   = HAS_COLOR and colors.blue      or colors.black`
→ Déclaration de variable locale « C_TITLE » et initialisation à « HAS_COLOR and colors.blue or colors.black ».

Ligne 11 : `local C_TEXT    = HAS_COLOR and colors.white     or colors.black`
→ Déclaration de variable locale « C_TEXT » et initialisation à « HAS_COLOR and colors.white or colors.black ».

Ligne 12 : `local C_MUTE    = HAS_COLOR and colors.gray      or colors.black`
→ Déclaration de variable locale « C_MUTE » et initialisation à « HAS_COLOR and colors.gray or colors.black ».

Ligne 13 : `local C_WARN    = HAS_COLOR and colors.orange    or colors.black`
→ Déclaration de variable locale « C_WARN » et initialisation à « HAS_COLOR and colors.orange or colors.black ».

Ligne 14 : `local C_BAD     = HAS_COLOR and colors.red       or colors.black`
→ Déclaration de variable locale « C_BAD » et initialisation à « HAS_COLOR and colors.red or colors.black ».

Ligne 16 : `local function fill(x1,y1,x2,y2,c)`
→ Instruction Lua générique.

Ligne 17 : `  paintutils.drawFilledBox(x1,y1,x2,y2,c)`
→ Instruction Lua générique.

Ligne 18 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 20 : `local function titleBar(w, title)`
→ Instruction Lua générique.

Ligne 21 : `  paintutils.drawLine(1, 1, w, 1, C_TITLE)`
→ Instruction Lua générique.

Ligne 22 : `  term.setCursorPos(2,1)`
→ Interaction avec l’affichage terminal (UI).

Ligne 23 : `  term.setTextColor(C_TEXT)`
→ Interaction avec l’affichage terminal (UI).

Ligne 24 : `  term.write(title or "")`
→ Interaction avec l’affichage terminal (UI).

Ligne 25 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 27 : `local function footBar(w, h, text)`
→ Instruction Lua générique.

Ligne 28 : `paintutils.drawLine(1, h-2, w, h-2, C_MUTE)`
→ Instruction Lua générique.

Ligne 29 : `term.setCursorPos(2, h-1)`
→ Interaction avec l'affichage terminal (UI).

Ligne 30 : `term.setTextColor(C_MUTE)`
→ Interaction avec l'affichage terminal (UI).

Ligne 31 : `term.write(text or "")`
→ Interaction avec l'affichage terminal (UI).

Ligne 32 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 34 : `local function trunc(s, maxw)`
→ Instruction Lua générique.

Ligne 35 : `if #s <= maxw then return s end`
→ Instruction Lua générique.

Ligne 36 : `return s:sub(1, math.max(0, maxw-1)) .. "?"`
→ Retourne une valeur au code appelant « s:sub(1, math.max(0, maxw-1)) .. "?" ».

Ligne 37 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 39 : `-- Public: dessine l'UI statique (cadre, barres, bouton arrêter)`
→ Commentaire du développeur.

Ligne 40 : `function ui.drawUI(username, isAdmin, w, h, version)`
→ Déclare la fonction « ui.drawUI » avec paramètres « username, isAdmin, w, h, version ».

Ligne 41 : `term.setBackgroundColor(colors.black)`
→ Interaction avec l'affichage terminal (UI).

Ligne 42 : `term.clear()`
→ Interaction avec l'affichage terminal (UI).

Ligne 43 : `fill(1,1,w,h,C_BG)`
→ Instruction Lua générique.

Ligne 45 : `local head = " RagnarNet UI "`
→ Déclaration de variable locale « head » et initialisation à « " RagnarNet UI " ».

Ligne 46 : `if version then head = head .. "v"..tostring(version).." " end`
→ Instruction Lua générique.

Ligne 47 : `titleBar(w, head)`
→ Instruction Lua générique.

Ligne 49 : `-- bande inférieure (zone d'aide)`
→ Commentaire du développeur.

Ligne 50 : `local who = "Connecté en tant que " .. (username or "?") .. (isAdmin and " [ADMIN]" or "")`
→ Déclaration de variable locale « who » et initialisation à « "Connecté en tant que " .. (username or "?") .. (isAdmin and " [ADMIN]" or "") ».

Ligne 51 : `footBar(w, h, who)`
→ Instruction Lua générique.

Ligne 53 : `-- bouton arrêt (click: y==1, x>=w-12)`
→ Commentaire du développeur.

Ligne 54 : `term.setCursorPos(math.max(1, w-12), 1)`
→ Interaction avec l'affichage terminal (UI).

Ligne 55 : `term.setTextColor(C_BAD)`
→ Interaction avec l'affichage terminal (UI).

Ligne 56 : `term.write("[ARRETER]")`
→ Interaction avec l'affichage terminal (UI).

Ligne 57 : `end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 59 : `-- Public: affiche la liste des messages dans la zone centrale`
→ Commentaire du développeur.

Ligne 60 : `-- messages = { {id=1, from="u", text="...", admin=false}, ... }`
→ Commentaire du développeur.

Ligne 61 : `function ui.showMessages(messages, uiHeight, blacklist, adminUser)`
→ Déclare la fonction « ui.showMessages » avec paramètres « messages, uiHeight, blacklist, adminUser ».

Ligne 62 : `  local w, h = term.getSize()`
→ Interaction avec l'affichage terminal (UI).

Ligne 63 : `  -- on nettoie la zone centrale (lignes 2..h-3)`
→ Commentaire du développeur.

Ligne 64 : `  for y = 2, (h-3) do`
→ Boucle FOR : itère sur « y = 2, (h-3) ».

Ligne 65 : `    term.setCursorPos(2, y)`
→ Interaction avec l'affichage terminal (UI).

Ligne 66 : `    term.clearLine()`
→ Interaction avec l'affichage terminal (UI).

Ligne 67 : `  end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 69 : `  local maxWidth = math.max(1, w - 4) -- marge à gauche/droite`
→ Déclaration de variable locale « maxWidth » et initialisation à « math.max(1, w - 4) -- marge à gauche/droite ».

Ligne 70 : `  local start = math.max(1, #messages - uiHeight + 1)`
→ Déclaration de variable locale « start » et initialisation à « math.max(1, #messages - uiHeight + 1) ».

Ligne 72 : `  for i = start, #messages do`
→ Boucle FOR : itère sur « i = start, #messages ».

Ligne 73 : `    local m = messages[i]`
→ Déclaration de variable locale « m » et initialisation à « messages[i] ».

Ligne 74 : `    local line = (i - start) + 2 -- commence sous la barre titre`
→ Déclaration de variable locale « line » et initialisation à « (i - start) + 2 -- commence sous la barre titre ».

Ligne 76 : `    -- couleur par type`
→ Commentaire du développeur.

Ligne 77 : `    if m and m.admin then`
→ Condition IF : exécute le bloc si « m and m.admin » est vrai.

Ligne 78 : `      term.setTextColor(C_WARN)`
→ Interaction avec l'affichage terminal (UI).

Ligne 79 : `    elseif m and m.from and blacklist and blacklist[m.from] then`
→ Branche ELSEIF : alternative si « m and m.from and blacklist and blacklist[m.from] » est vrai.

Ligne 80 : `      term.setTextColor(C_BAD)`
→ Interaction avec l'affichage terminal (UI).

Ligne 81 : `    else`
→ Branche ELSE : cas par défaut si la condition est fausse.

Ligne 82 : `      term.setTextColor(C_TEXT)`
→ Interaction avec l'affichage terminal (UI).

Ligne 83 : `    end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 85 : `    local id   = tostring(m.id or i)`
→ Déclaration de variable locale « id » et initialisation à « tostring(m.id or i) ».

Ligne 86 : `    local from = tostring(m.from or "?")`
→ Déclaration de variable locale « from » et initialisation à « tostring(m.from or "?") ».

Ligne 87 : `    local txt  = tostring(m.text or "")`
→ Déclaration de variable locale « txt » et initialisation à « tostring(m.text or "") ».

Ligne 89 : `    local raw = "["..id.."] "..from..": "..txt`
→ Déclaration de variable locale « raw » et initialisation à « "["..id.."] "..from..": "..txt ».

Ligne 90 : `    local out = trunc(raw, maxWidth)`
→ Déclaration de variable locale « out » et initialisation à « trunc(raw, maxWidth) ».

Ligne 92 : `    term.setCursorPos(2, line)`
→ Interaction avec l'affichage terminal (UI).

Ligne 93 : `    term.write(out)`
→ Interaction avec l'affichage terminal (UI).

Ligne 94 : `  end`
→ Fin de bloc (fonction/condition/boucle).

Ligne 95 : `end`

→ Fin de bloc (fonction/condition/boucle).

Ligne 97 : `return ui`

→ Retourne une valeur au code appelant « ui ».

Base de données (users.db)

Impossible de lire la base : file is not a database

Recommandations de sécurité

• Signer ou hasher (SHA-256) les paquets téléchargés (http.*) et vérifier avant installation.
• Restreindre fs.delete et fs.open('w') aux chemins autorisés (whitelist).
• Journaliser (log) les actions critiques (suppression, reboot, update).
• Utiliser pcall/xpcall autour des I/O réseau et disque.
• Charger la configuration en lecture seule pour les modules non privilégiés.