**Task 7**

**Objectives**

- List all extensions installed in the browser
- Analyze permissions requested by each extension
- Check developer credibility and extension reputation
- Flag suspicious, unused, or unnecessary extensions
- Remove harmful/unknown extensions
- Improve security hygiene by reducing attack surface

**Files Included**

README.md

task7_report.md

extension_risk_chart.png

permission_risk_chart.png

**Outcome**

A set of suspicious extensions were identified and removed. Browser security, stability, and performance were improved.

**TASK 7 — Full Report**

## 1. Introduction

Browser extensions improve productivity but also increase risk. Extensions with excessive permissions, unverified developers, or odd behavior can leak data, inject ads, or perform unauthorized actions.
This task documents a full review of installed extensions and the removal of suspicious ones.

## 2. Methodology

A systematic approach was used:

**Step 1 — Open Extension Manager**

- Chrome: chrome://extensions/

- Edge: edge://extensions/
- Firefox: about:addons

**Step 2 — List Each Extension**

For every extension:

- Name
- Developer/publisher
- Permission set
- Version
- Install source

**Step 3 — Evaluate Risk Indicators**

An extension was considered **suspicious** if ANY of the following applied:

- Unknown or unverifiable developer
- Poor reputation or low install count
- Recently flagged by users in reviews
- Requests invasive permissions like:
    - “Read and change all your data on all websites”
    - “Access browser tabs”
    - “Manage downloads”
    - “Communicate with cooperating native applications”
- Installed unintentionally
- No longer used

**Step 4 — Remove or Disable**

Extensions identified as **risky**, **unused**, or **unnecessary** were removed.

**Step 5 — Validate**

After removal:

- Browser restarted
- Checked loading speed
- Verified no unexpected pop-ups or redirects

## 3. Findings

**Installed Extensions (Before Review)**

| Extension Name | Status | Reason |
|---|---|---|
| Grammarly | Safe | Trusted publisher, low permissions |
| Adblock Plus | Safe | Well-known, transparent permissions |
| Quick PDF Converter | Suspicious | Requested full data access |
| Tab Manager Pro | Suspicious | Poor reviews, unnecessary access |
| XYZ Productivity | Unnecessary | Installed unintentionally |
| Google Docs Offline | Safe | From Google, required for use |

**Suspicious Extensions Identified**

**Tab Manager Pro**

- Invasive permissions (read/change all site data)
- Bad user reviews
- Unclear purpose / redundant functionality

**XYZ Productivity**

- Installed without user awareness
- Reached external URLs from background script

**Extensions Removed**

- Tab Manager Pro
- XYZ Productivity

**Reasons for Removal**

- Unnecessary permissions
- Potential privacy risks
- Untrusted publishers
- No functional value

## 5. Recommendations

### Security

- Only install extensions after reviewing developer credibility
- Avoid extensions requiring full site-wide access unless absolutely necessary
- Update extensions frequently
- Disable auto-updates for unknown publishers
- Re-review installed extensions once a month

### Privacy

- Prefer extensions with transparent data policies
- Avoid extensions that "store data on third-party servers"
- Monitor browser behavior after adding a new extension

### Performance

- Keep only essential extensions
- Remove slow or outdated add-ons
- Monitor CPU usage of background scripts

**list_extensions_chrome_firefox.py**

```
#!/usr/bin/env python3
"""
list_extensions_chrome_firefox.py

Reads local Chrome and Firefox extension folders (user profile) and extracts manifest.json information.
Generates `extensions.csv` with columns:
  browser, id, name, version, install_path, permissions (comma-separated)

Notes:
- This script only reads local files. It will not contact any remote servers.
- Run it as the same user that runs the target browser so it can access profile folders.
- Tested on Linux paths; Windows/macOS locations included below.
"""

import json
import csv
import os
from pathlib import Path
import platform

# Adjust these profile paths if needed
def chrome_extensions_paths():
    sys = platform.system()
    if sys == "Linux":
        # default Chrome & Chromium
        home = Path.home()
        return [
            home / ".config" / "google-chrome" / "Default" / "Extensions",
            home / ".config" / "chromium" / "Default" / "Extensions",
        ]
    elif sys == "Darwin":
        home = Path.home()
        return [home / "Library" / "Application Support" / "Google" / "Chrome" / "Default" /
"Extensions"]
    elif sys == "Windows":
        local = os.environ.get("LOCALAPPDATA")
        return [Path(local) / "Google" / "Chrome" / "User Data" / "Default" / "Extensions"]
    return []

def firefox_extensions_paths():
    sys = platform.system()
    if sys == "Linux":
        home = Path.home()
        # multiple profiles; we will search profile folders
        firefox_dir = home / ".mozilla" / "firefox"
    elif sys == "Darwin":
        home = Path.home()
        firefox_dir = home / "Library" / "Application Support" / "Firefox" / "Profiles"
    elif sys == "Windows":
        appdata = os.environ.get("APPDATA")
        firefox_dir = Path(appdata) / "Mozilla" / "Firefox" / "Profiles"
    else:
        firefox_dir = None
```

```
  paths = []
  if firefox_dir and firefox_dir.exists():
    for p in firefox_dir.glob("*"):
      # extension folders stored differently; check extensions.json and extensions
      if p.is_dir():
        paths.append(p)
  return paths


def read_chrome_extensions(out_rows):
  for base in chrome_extensions_paths():
    if not base or not base.exists():
      continue
    for ext_id_dir in base.iterdir():
      if not ext_id_dir.is_dir():
        continue
      # Chrome stores versions as subdirs inside extension id folder.
      for version_dir in ext_id_dir.iterdir():
        manifest = version_dir / "manifest.json"
        if manifest.exists():
          try:
            m = json.loads(manifest.read_text(encoding="utf-8"))
            name = m.get("name", "<unknown>")
            # name may be localized: check default_locale -> _locales set; we won't
resolve localization here.
            version = m.get("version", "")
            perms = m.get("permissions", []) + m.get("optional_permissions", [])
            perms = [str(p) for p in perms]
            out_rows.append({
                "browser": "chrome",
                "id": ext_id_dir.name,
                "name": name,
                "version": version,
                "install_path": str(version_dir),
                "permissions": ",".join(perms)
            })
        except Exception as e:
            # skip malformed manifests
            pass


def read_firefox_extensions(out_rows):
    # Firefox typically stores installed add-ons metadata in extensions.json under profile
folder
    for profile in firefox_extensions_paths():
        extensions_json = profile / "extensions.json"
        if extensions_json.exists():
            try:
                data = json.loads(extensions_json.read_text(encoding="utf-8"))
                addons = data.get("addons", [])
                for a in addons:
                    # only list installed ones
                    install_status = a.get("active", False)
                    # skip disabled if you want; we include all
                    name = a.get("defaultLocale", {}).get("name") or a.get("name") or a.get("id")
                    version = a.get("version", "")
                    ext_id = a.get("id") or a.get("guid") or a.get("internalUUID", "")
                    path = a.get("path", "")
```

```
          # Firefox permissions are defined in manifest if present (but not always
exposed in extensions.json)
          perms = []
          # try to load manifest from path if it looks like a directory
          if path:
            p = Path(path)
            if p.exists():
              man = p / "manifest.json"
              if man.exists():
                try:
                  mm = json.loads(man.read_text(encoding="utf-8"))
                  perms = mm.get("permissions", []) + mm.get("optional_permissions",
[])
                except:
                  pass
          out_rows.append({
            "browser": "firefox",
            "id": ext_id,
            "name": name,
            "version": version,
            "install_path": path,
            "permissions": ",".join(perms)
          })
      except Exception as e:
        continue


def main():
  out_rows = []
  read_chrome_extensions(out_rows)
```

```
    read_firefox_extensions(out_rows)


    if not out_rows:
        print("No extensions found using the default paths. Update paths in the script to
match custom profiles.")
    csv_path = Path("extensions.csv")
    with csv_path.open("w", newline='', encoding="utf-8") as fh:
        writer = csv.DictWriter(fh,
fieldnames=["browser","id","name","version","install_path","permissions"])
        writer.writeheader()
        for r in out_rows:
            writer.writerow(r)
    print("Wrote", csv_path)


if __name__ == "__main__":
    main()
```

**2) classify_permissions.py**

```
#!/usr/bin/env python3
"""
classify_permissions.py


Small helper: given a permissions list, returns a risk score and human category.
Used by generate_markdown_report.py to label permissions as
Low/Medium/High/Critical.
"""
from typing import List


CRITICAL_PERMS = set([
    "<all_urls>", "cookies", "history", "webRequest", "webRequestBlocking", "clipboardRead",
    "clipboardWrite", "nativeMessaging", "proxy", "management", "unlimitedStorage", "fileSystem"
])

HIGH_PERMS = set([
    "tabs", "activeTab", "storage", "bookmarks", "downloads", "notifications"
])

MEDIUM_PERMS = set([
    "geolocation", "background", "identity", "contextMenus", "cookies"
])

def score_permissions(perms: List[str]):
    perms_set = set(perms)
    score = 0
    reasons = []
    for p in perms_set:
        if p in CRITICAL_PERMS or p == "<all_urls>":
            score += 50
            reasons.append(("critical", p))
        elif p in HIGH_PERMS:
            score += 25
            reasons.append(("high", p))
        elif p in MEDIUM_PERMS:
            score += 10
            reasons.append(("medium", p))
        else:
```

```
        # heuristics: host patterns like "*://*/*" are critical
        if isinstance(p, str) and ("*" in p or p.startswith("http") or p.startswith("https")):
            score += 30
            reasons.append(("host", p))
        else:
            score += 1
            reasons.append(("low", p))
    # clamp
    if score >= 60:
        cat = "Critical"
    elif score >= 30:
        cat = "High"
    elif score >= 10:
        cat = "Medium"
    else:
        cat = "Low"
    return {"score": score, "category": cat, "reasons": reasons}


# quick CLI test
if __name__ == "__main__":
    import sys, json
    perms = json.loads(sys.argv[1]) if len(sys.argv) > 1 else ["tabs","<all_urls>"]
    print(score_permissions(perms))
```