Tinycolor supply chain attack

Introduction

This document provides a security audit for the **TinyColor NPM supply chain attack** and evaluates whether the current project was impacted. The audit identifies potential vulnerabilities, verifies dependency integrity, and recommends mitigation measures.

Executive summary

In September 2025, the `tinycolor` package and its dependent ecosystem were compromised:

- Malicious code was injected into `tinycolor`, affecting multiple downstream packages and projects.
- The compromise allowed attackers to execute arbitrary scripts during installation, potentially exfiltrating environment variables, API tokens, and sensitive project data.
- The attack propagated through transitive dependencies, highlighting the risks of deep dependency chains.

(Source: Socket.dev report)

The audit aimed to:

1. Determine whether any project dependencies (direct or transitive) were affected by the TinyColor compromise.
2. Verify the integrity of all project dependencies.
3. Recommend protective measures to prevent future supply chain attacks.

Affected packages and versions

The following npm packages and versions were confirmed as affected:

- `angulartics2@14.1.2`
- `@ctrl/deluge@7.2.2`
- `@ctrl/golang-template@1.4.3`
- `@ctrl/magnet-link@4.0.4`
- `@ctrl/ngx-codemirror@7.0.2`
- `@ctrl/ngx-csv@6.0.2`
- `@ctrl/ngx-emoji-mart@9.2.2`
- `@ctrl/ngx-rightclick@4.0.2`
- `@ctrl/qbittorrent@9.7.2`
- `@ctrl/react-adsense@2.0.2`

- `@ctrl/shared-torrent@6.3.2`
- `@ctrl/tinycolor@4.1.1` , `@4.1.2`
- `@ctrl/torrent-file@4.1.2`
- `@ctrl/transmission@7.3.1`
- `@ctrl/ts-base32@4.0.2`
- `encounter-playground@0.0.5`
- `json-rules-engine-simplified@0.2.4` , `0.2.1`
- `koa2-swagger-ui@5.11.2` , 5.11.1
- `@nativescript-community/gesturehandler@2.0.35`
- `@nativescript-community/sentry 4.6.43`
- `@nativescript-community/text@1.6.13`
- `@nativescript-community/ui-collectionview@6.0.6`
- `@nativescript-community/ui-drawer@0.1.30`
- `@nativescript-community/ui-image@4.5.6`
- `@nativescript-community/ui-material-bottomsheet@7.2.72`
- `@nativescript-community/ui-material-core@7.2.76`
- `@nativescript-community/ui-material-core-tabs@7.2.76`
- `ngx-color@10.0.2`
- `ngx-toastr@19.0.2`
- `ngx-trend@8.0.1`
- `react-complaint-image@0.0.35`
- `react-jsonschema-form-conditionals@0.3.21`
- `react-jsonschema-form-extras@1.0.4`
- `rxnt-authentication@0.0.6`
- `rxnt-healthchecks-nestjs@1.0.5`
- `rxnt-kue@1.0.7`
- `swc-plugin-component-annotate@1.9.2`
- `ts-gaussian@3.0.6`

Our project scan (Results)

A custom **Python audit script** was developed to automate detection of compromised dependencies.

The script:

1. Accepts a list of compromised packages and versions (from the Socket.dev report).
2. Recursively scans both `package.json` and `package-lock.json` .
3. Identifies any matches or related dependencies.
4. Outputs results in a structured JSON report (`audit_report.json`).

Scan Summary

Item	Description
Total packages scanned	41
Compromised packages found	**0**
Affected components	None detected
Date of scan	12-10-2025
Tool used	Custom Python audit script

Impact assessment

Based on the results of the automated dependency audit, no direct or transitive references to the compromised NPM packages associated with the *@ctrl/tinycolor* incidents were found in the project's `package.json` or `package-lock.json` files.

As a result:

- The project remains unaffected by the TinyColor supply chain incident.
- No malicious code execution, data exfiltration, or dependency tampering was observed.
- The project's dependency integrity is intact

Since no vulnerable packages were found, the overall impact is assessed as **none (no compromise detected)**.

Protective Recommendations

Dependency Management

- Pin exact dependency versions in `package.json` to prevent automatic updates.
- Conduct regular audits using `npm audit` and third-party tools (e.g., Socket.dev, Snyk, Aikido Security).
- Monitor dependency trees with `npm ls --all`.

Continuous Monitoring

- Enable GitHub Dependabot or similar services to flag vulnerable dependencies.
- Subscribe to NPM security advisories for real-time alerts.
- Include CI/CD security scans to block builds with vulnerable dependencies.

Code Integrity & Provenance

- Verify checksums and signatures for internal and third-party packages.
- Enforce multi-factor authentication (MFA) and scoped tokens for NPM accounts.
- Use private mirrors or artifact repositories (Verdaccio, Artifactory) for trusted dependencies.

Response & Recovery

- Maintain a documented incident response plan for supply chain compromises.
- Archive previous versions of dependencies for rapid rollback.
- Record security incidents and lessons learned.

Audit Script

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

Usage:
  python npm_supplychain_audit.py --compromised compromised.txt \
      --project /path/to/project \
      --out report.json

compromised.txt lines: package or package@version
Example:
  chalk
  @ctrl/tinycolor@4.1.1
"""

import argparse
import json
import os
import sys
from collections import defaultdict, deque

def load_compromised(path):
    items = {}
    with open(path, 'r', encoding='utf-8') as f:
        for line in f:
            s = line.strip()
            if not s or s.startswith('#'):
                continue
            if '@' in s and not s.startswith('@'):
                # simple name@version
                name, ver = s.rsplit('@', 1)
                items[name] = items.get(name, set())
                items[name].add(ver)
            elif s.startswith('@') and s.count('@') >= 2:
                # scoped package like @ctrl/tinycolor@4.1.1
                # split from right
                name, ver = s.rsplit('@', 1)
                items[name] = items.get(name, set())
                items[name].add(ver)
            else:
                # name only
                items[s] = None  # None => any version
    return items  # dict: name -> None (any) or set(versions)

def load_json_if_exists(path):
    if not os.path.exists(path):
        return None
    with open(path, 'r', encoding='utf-8') as f:
        return json.load(f)

def check_package_json(pkg_json, compromised):
    results = []
    if not pkg_json:
        return results
    deps = {}
    for sec in ('dependencies', 'devDependencies', 'optionalDependencies',
'peerDependencies'):
        if sec in pkg_json:
            deps.update(pkg_json[sec])
    for name, spec in deps.items():
        if name in compromised:
            vers = compromised[name]
            if vers is None:
                results.append({'package': name, 'matched': 'any-version',
'declared_version_spec': spec})
            else:
                # can't determine matching version from package.json spec;
warn
                results.append({'package': name, 'matched': 'some-versions-
specified', 'declared_version_spec': spec, 'compromised_versions':
list(vers)})
    return results

def normalize_name(n):
    return n

def scan_lockfile(lock_json, compromised):
    """
    Supports package-lock v1 and v2 (node_modules-like tree in lockfile).
    Returns list of matches with paths (list of dependency names from root).
    """
    matches = []

    # Node structure differences:
    # - lockfile v1: lock_json['dependencies'] is a mapping of name->
{version, dependencies}
    # - lockfile v2: lock_json['packages'] with keys like "" or
"node_modules/pkg" and lock_json['dependencies']
    if 'packages' in lock_json:
        # lockfile v2+
        # Build a graph mapping package path -> its dependencies (names with
versions)
        packages = lock_json.get('packages', {})
        # We'll traverse starting from "" (root)
        # Build a mapping from (package name, version) to its children
occurrences via 'dependencies' fields in lock_json['dependencies']
        # But simpler: use lock_json['dependencies'] entries which map name
-> {version, requires}
        deps_root = lock_json.get('dependencies', {})
        # We'll BFS through dependencies using 'dependencies' structure to
resolve subtree.
        # Helper to record path
        def bfs():
            q = deque()
            # start with top-level dependencies
            for name in deps_root:
                q.append( (name, deps_root[name], [name]) )
            while q:
                name, meta, path = q.popleft()
                version = meta.get('version')
                # check match
                if name in compromised:
                    vers = compromised[name]
                    if vers is None or (version and version in vers):
                        matches.append({'package': name, 'version': version,
'path': list(path)})
                # push children
                requires = meta.get('requires') or {}
                for child in requires:
                    child_meta = deps_root.get(child) or
lock_json.get('dependencies', {}).get(child)
                    if child_meta:
                        q.append( (child, child_meta, path + [child]) )
        bfs()
    elif 'dependencies' in lock_json:
        # lockfile v1-ish
        top = lock_json.get('dependencies', {})
        def recurse(node, path):
            for name, meta in node.items():
                version = meta.get('version')
                if name in compromised:
                    vers = compromised[name]
                    if vers is None or (version and version in vers):
                        matches.append({'package': name, 'version': version,
'path': list(path + [name])})
                child_deps = meta.get('dependencies') or {}
                if child_deps:
                    recurse(child_deps, path + [name])
        recurse(top, [])
    else:
        # unknown structure: try to inspect for a 'dependencies' anywhere
        # fallback: search entire JSON for objects that look like {version:
"..."}
        def walk(obj, path):
            if isinstance(obj, dict):
                if 'version' in obj and isinstance(path, list) and
len(path)>0:
                    name = path[-1]
                    if name in compromised:
                        version = obj.get('version')
                        vers = compromised[name]
                        if vers is None or (version and version in vers):
                            matches.append({'package': name, 'version':
version, 'path': list(path)})
                for k,v in obj.items():
                    walk(v, path + [k])
            elif isinstance(obj, list):
                for i, v in enumerate(obj):
                    walk(v, path + [str(i)])
        walk(lock_json, [])
    return matches

def make_json_safe(obj):
        if isinstance(obj, set):
            return list(obj)
        if isinstance(obj, dict):
            return {k: make_json_safe(v) for k, v in obj.items()}
        if isinstance(obj, list):
            return [make_json_safe(v) for v in obj]
        return obj

def main():
    parser = argparse.ArgumentParser(description="Scan npm project for
compromised packages (direct & transitive)")
    parser.add_argument('--compromised', '-c', required=True, help='file
with compromised package names (name or name@version per line)')
    parser.add_argument('--project', '-p', default='.', help='path to
project root (containing package.json/package-lock.json)')
    parser.add_argument('--out', '-o', default='audit_report.json',
help='output JSON report path')
    args = parser.parse_args()

    compromised = load_compromised(args.compromised)
    project = args.project

    pkg_json = load_json_if_exists(os.path.join(project, 'package.json'))
    lock_json = load_json_if_exists(os.path.join(project, 'package-
lock.json'))

    report = {
        'project_path': os.path.abspath(project),
        'package_json_present': pkg_json is not None,
        'package_lock_present': lock_json is not None,
        'compromised_input': compromised,
        'direct_matches': [],
        'lockfile_matches': []
    }

    if pkg_json:
        report['direct_matches'] = check_package_json(pkg_json, compromised)

    if lock_json:
        report['lockfile_matches'] = scan_lockfile(lock_json, compromised)

    # Human summary
    summary = []
    if report['direct_matches']:
        summary.append(f"Direct references found:
{len(report['direct_matches'])}")
    if report['lockfile_matches']:
        summary.append(f"Transitive/lockfile matches found:
{len(report['lockfile_matches'])}")
    if not summary:
        summary_text = "No matches found in package.json or package-
lock.json."
    else:
        summary_text = "; ".join(summary)

    report['summary'] = summary_text


    with open(args.out, 'w', encoding='utf-8') as f:
        json.dump(make_json_safe(report), f, indent=2)
    print("Audit complete.")
    print(summary_text)
    print(f"Full JSON report written to: {args.out}")

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

Audit Report Output

```
{

  "project_path": "C:\\Users\\user\\OneDrive\\Desktop\\Web
Development\\Projects\\clone\\Codexa",

  "package_json_present": true,
  "package_lock_present": true,

  "compromised_input": {

    "backslash": null,

    "chalk-template": null,

    "supports-hyperlinks": null,

    "has-ansi": null,

    "simple-swizzle": null,

    "color-string": null,

    "error-ex": null,

    "color-name": null,

    "is-arrayish": null,

    "slice-ansi": null,

    "color-convert": null,

    "wrap-ansi": null,

    "ansi-regex": null,

    "supports-color": null,

    "strip-ansi": null,

    "chalk": null,

    "debug": null,

    "ansi-styles": null,

    "angulartics2": [

      "14.1.2"

    ],

    "@ctrl/deluge": [

      "7.2.2"
    ],
    "@ctrl/golang-template": [
      "1.4.3"
    ],
    "@ctrl/magnet-link": [
      "4.0.4"
    ],
    "@ctrl/ngx-codemirror": [
      "7.0.2"
    ],
    "@ctrl/ngx-csv": [
      "6.0.2"
    ],
    "@ctrl/ngx-emoji-mart": [
      "9.2.2"
    ],
    "@ctrl/ngx-rightclick": [
      "4.0.2"
    ],
    "@ctrl/qbittorrent": [
      "9.7.2"
    ],
    "@ctrl/react-adsense": [
      "2.0.2"
    ],
    "@ctrl/shared-torrent": [
      "6.3.2"
    ],
    "@ctrl/tinycolor": [
      "4.1.1",
      "4.1.2"
    ],
    "@ctrl/torrent-file": [
      "4.1.2"
    ],
    "@ctrl/transmission": [
      "7.3.1"
    ],
    "@ctrl/ts-base32": [
      "4.0.2"
    ],
    "encounter-playground": [
      "0.0.5"
    ],
    "json-rules-engine-simplified": [
      "0.2.1",
      "0.2.4"
    ],
    "koa2-swagger-ui": [
      "5.11.2",
      "5.11.1"

    ],

    "@nativescript-community/gesturehandler": [

      "2.0.35"

    ],

    "@nativescript-community/sentry 4.6.43": null,

    "@nativescript-community/text": [

      "1.6.13"

    ],

    "@nativescript-community/ui-collectionview": [

      "6.0.6"

    ],

    "@nativescript-community/ui-drawer": [

      "0.1.30"

    ],

    "@nativescript-community/ui-image": [

      "4.5.6"

    ],

    "@nativescript-community/ui-material-bottomsheet": [

      "7.2.72"

    ],

    "@nativescript-community/ui-material-core": [

      "7.2.76"

    ],

    "@nativescript-community/ui-material-core-tabs": [
    "7.2.76"

  ],

  "ngx-color": [

    "10.0.2"

  ],

  "ngx-toastr": [

    "19.0.2"

  ],

  "ngx-trend": [

    "8.0.1"

  ],

  "react-complaint-image": [

    "0.0.35"

  ],

  "react-jsonschema-form-conditionals": [

    "0.3.21"

  ],

  "react-jsonschema-form-extras": [

    "1.0.4"

  ],

  "rxnt-authentication": [

    "0.0.6"

  ],

  "rxnt-healthchecks-nestjs": [

    "1.0.5"
    ],
    "rxnt-kue": [
      "1.0.7"
    ],
    "swc-plugin-component-annotate": [
      "1.9.2"
    ],
    "ts-gaussian": [
      "3.0.6"
    ]
  },
  "direct_matches": [],
  "lockfile_matches": [],
  "summary": "No matches found in package.json or package-lock.json."
}
```