License compliance with AI-assisted coding

KNOW YOUR FRANKIE

Julian Coccia, CTO
“AI’s ban, like witchcraft in the Middle Ages, is fueled by fear”
Software Development Landscape

- Increasing adoption of Open Source
- Governments demanding SBOMs
- Proliferation of AI-assisted coding

"Third-party software integration is today almost involuntary"
“Developers who don’t use AI are being eclipsed by those who do”
Source Code vs Human language

Limited set of instructions
Precise, unambiguous, and deterministic
Higher chance of AI generating existing material

Thousands of words and expressions
Ambiguous and context-dependent
Lower chance of AI generating existing material

“We enforce software licenses with laws made for human expression”
“Computers learn from existing copyrighted code, just like humans”
Would courts relax towards AI generated code?

"As judges remain human, Copyright Law and existing jurisprudence will hold sway"
Risks of unknown Software Composition

- License Compliance
- Trade Compliance (Export Control)
- Security Vulnerabilities
- Code quality
- Component health (obsolescence)
- Quantum-unsafe encryption
- Other risks

“You can’t comply with/protect what you can’t see”
The Gray Zone

Verbatim Copy

The Gray Zone

Mere inspiration
Here's an implementation of the Winnowing algorithm in C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFFER_RATE 4
#define GRAM_SIZE 30
#define WINDOW_SIZE 64

typedef unsigned int uint32_t;

uint32_t calc_crc32c(const char *data, size_t length) {
    static uint32_t table[256];
    static uint32_t init = 0;
    if (!init) {
        init = 1;
    }
    // Additional code here...
}
```
Verbatim Copy: Example

```c
uint32_t add_hash(uint32_t hash, uint32_t line, uint32_t *hashes, uint32_t *counter)
{
    /* Consecutive repeating hashes are ignored */
    if (hash == last) {
        /* Hashing the hash will result in a better balanced resulting data */
        /* as it will counter the winnowing effect which selects the "minimum" */
        /* hash in each window */
        hashes[*counter] = calc_crc32c((char *)hash, 4);
        line[*counter] = line;
        last = hash;
        (*counter)++;
    }
    return last;
}

int main()
{
    char text[] = "This is an example text for testing the Winnowing algorithm."
    winnow(text, strlen(text));
    return 0;
}
```

Note that this implementation uses a sliding window of size 64, a gram size of 30, and a buffer rate of 4. The `calc_crc32c` function is used to calculate the hash of each gram. The `winnow` function iterates over the text, calculating the hash of each gram and adding it to the hash array.
“Identifying undeclared components is key to a complete SBOM”
Introducing SCANOSS

• Automated license compliance validation tool
• Integration with CI/CD pipelines
• Entirely Open Source
• Largest database of known Open Source
• Available for free: OSSKB.ORG
Software Composition Analysis

• Accurate, standardized detection of code plagiarism
• Adopted by Open Source communities
• Adopted by SCA suppliers
• Validated by European courts
Plagiarism check by detecting known OSS
Presence in public repositories

$ pip3 install scanoss
$ scanoss-py scan mycode/

$ npm install -g scanoss
$ scanoss-js scan mycode/

https://github.com/scanoss
First Multi-platform Auditing App

https://github.com/scanoss/audit-workbench
Bottom line

• Embrace AI-assisted development

• Validate your software composition with SCANOSS

• Foundational to all angles of Open Source risk
Thank you!

Julian Coccia
julian.coccia@scanoss.com
https://scanoss.com