**SOC Sensor LLM Concept**

**Initial Focus on Training Data**

[**tp@compris.com**](mailto:tp@compris.com)**, Nov. 8, 2023**

# Table of Contents

[1 Table of Contents 1](#_Toc150383124)

[2 Overview 1](#_Toc150383125)

[3 Proposed File Structure 1](#_Toc150383126)

[3.1 Standards 2](#_Toc150383127)

[3.2 File Sections 3](#_Toc150383128)

[3.2.1 Hierarchical\_Vulnerability\_Type\_Codes 4](#_Toc150383129)

[3.2.2 Hierarchical\_Exploit\_Type\_Codes 4](#_Toc150383130)

[4 Extensions and Alternatives 4](#_Toc150383131)

[4.1 Collecting and comparing older with newer Sources and Binaries to see how Vulnerabilities were fixed 4](#_Toc150383132)

[4.2 AI Agent Frameworks 4](#_Toc150383133)

[4.2.1 For Data Labeling / Basic Cybersecurity 4](#_Toc150383134)

[4.2.2 MetaGPT 4](#_Toc150383135)

[4.3 Buying Threat Intelligence 4](#_Toc150383136)

[5 Strategy 6](#_Toc150383137)

[5.1 Bunch of Opportunities 6](#_Toc150383138)

[5.2 Costs / Project Plan 6](#_Toc150383139)

[6 Appendix 6](#_Toc150383140)

[6.1 Job Ad: X86 Vulnerability and Exploit Analyst helping towards SOC Detection AIs 6](#_Toc150383141)

# Overview

With the free availability of LLMs like CodeLlama it is possible to fine-tune them (i.e. do incremental training) to teach them additional skills – and that could by hacking, vulnerability analysis, threat intelligence and SOC analyst skills. These could now be trained with tuples (in XML files) of vulnerability, exploits, detection patterns and new AI-based detection of malware/attacks and the vulnerabilities. This is a brand new exciting opportunity at the heart of a **multi-billion-dollar market** that is also the **fastest growing** and many institutions like the WEF have warned that **cybersecurity will be the no. 1 topic in the next years and they warn of a big cybersecurity attack as main threat to expect and to invest to avoid it**!

The results could also allow to bring out books and threat intelligence and of course create a SOC sensor or an entire SOC, e.g. leveraging open source like OpenSearch.

# Proposed File Structure

**Initial task:** Can you fill out this structure for one comprehensive example of a tuple?

I'm willing to pay $40 for each non-trivial tuple as described in this file for C/assembler language and each tuple for a different category of vulnerability for up to 3 tuples. Based on that we discuss further payment or automation tasks as described here.

The structure for machine learning must be maximally structured. Therefore this is a living document and we plan to norm everything. File sections are put under each other with one Latin1-coded text file for each tuple of vulnerabilities, exploits, etc.

We should try to **automate** as much as possible for the transfer of vulnerabilities, exploits, detection code, IoCs from web portals with web scrapers, IRPA (intelligent robotic process automation, i.e. automating the calling of (de)compilers, (dis)assemblers, debuggers), etc.

This task asks for strong types of tuples. If not, the model training never will fail.

**Steps for automating:**

1 - Specify the sources of databases (APIs, files, GitHub, etc.).

2- Analyze the collected data by categorizing it by type.

3- Analyze the exploit/vulnerabilities text and data.

4- Organize these analyses into tuples.

**Possible exploit sources:**

1. <https://www.exploit-db.com/>
2. <https://www.rapid7.com/db/modules/>
3. <https://github.com/rapid7/metasploit-framework>
4. <https://www.kernel-exploits.com/>
5. [www.ubuntu.com/usn/](http://www.ubuntu.com/usn/)
6. <https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/Linux-Linux-Kernel.html>
7. <https://www.cvedetails.com/>
8. <https://www.fireeye.com/blog/threat-research.html>
9. <https://www.hackread.com/sec/>
10. Open source virus scanners
11. Threat intelligence companies (against payment)

**Possible areas of content creation focus:**

1. Focusing on **buffer overflows** and other **memory corruption** in C/C++, JavaScript, etc., i.e. code injection possibilities
2. Focusing on **parser errors** of various parser types like lex/bison, i.e. code injection possibilities.
3. Focusing on parse errors around common multimedia formats like **JPG, MP4, MOV**: That allows to hack most messengers and mobile apps and **without having to click on a link**.
4. Focusing on vendors with many vulnerabilities like **Juniper, IBM, Big-IP, TP-Link, Barracuda, Ivanti, Apple, Citrix, Adobe, Acronis, Atlassian, Android, web browsers, most virus scanners, ZIP/rar programs, Huawei, printers and other old hardware, Consensys MetaMask and other crypto wallet systems**.
5. Maybe creating/buying collections of old malware. However, that often is hopelessly outdated in most regards.
6. **Leveraging vulnerability checkers** like PMD, FindBugs, etc. Where they find issues, looking for exploits.
7. **Creating / extending binary vulnerability checkers.**
8. **Generally leveraging the Cyber Grand Challenge tools with AI.**
9. **Systematically comparing older vulnerable binaries with their patched counterparts**, e.g. for Windows to nail down the changes (fixes) and possible new vulnerabilities (which are on purpose introduced to create back doors for intelligence agencies).
10. **FraudGPT:** Writing phishing mails, developing cracking tools and finding victims.

## Standards

The full source code is needed and old API functions like C’s strcpy are deprecated

It all should be self-contained, i.e. no external source references. Also the checks must be more detailed, minimally pseudo-code if not representable as RegEx

Hyphens in XML-names are forbidden by XML. Replace those with underscores.

**General rules:**

1. **Optimize the descriptions for LLM attention/focus:** String the core fields down to the absolute necessary parts and add comments to where exactly the vulnerability is and why exactly this attack works in a self-contained way, i.e. describing everything that is needed to detect the vulnerability, craft the attack and the SOC detection pattern and not more – except in additional fields like the description.
2. **Organize and think like a hacker:** What is relevant for them in finding vulnerabilities and exploiting them and guide the LLM to memorizing exactly these parts and have the same priorities. Hacker typically memorize strategies as well as knowledge clusters in all areas: Different programming languages, assembler and the resulting binary code and how this could be debugged and which variations could exist.
3. **The actual details of the exploit must be unraveled, i.**e. shell codes, targets, injected bytes, addresses used and why these?
4. We are building and continuously updating a **library of exploit utility code** that will only be referenced in the XML files to not confuse the LLMs.

**Source codes:**

1. **Indentation** is only done with tabs, not with spaces to have less characters.
2. Just the **relevant code/binary parts** are shown but with all declarations so that they could be made to compile when given a defined header and footer. I.e. includes (Java/Scala/C...) must be specified.
3. Source code has **minimal indentation**, i.e. it starts in column 0 to minimize characters.
4. All **source code shall be standardized** based on their pretty-printed form, e.g. for **assembler** all key words and opcodes shall be written in capitals.

## File Sections

Please use the following section tags in XML format – if multiple apply, e.g. multiple parameterizable parts/variants then they will all be listed just like another XML element:

1. **Textual overview for LLM attention:** <Overview>A buffer overflow was discovered in the GNU C Library's dynamic loader ld.so while processing the GLIBC\_TUNABLES environment variable. This issue could allow a local attacker to use maliciously crafted GLIBC\_TUNABLES environment variables when launching binaries with SUID permission to execute code with elevated privileges.</Overview>
2. **Vulnerabilities in source code**: <VULN> Language-Id: C, ASM, PY, JS, HEX of binary, JBC (Java Byte Code), Hierarchical\_Vulnerability\_Type\_Code... source/hex code </VULN>
3. **Patched source/binary**: <Patched> Language-Id: C, ASM, PY, JS, HEX of binary, JBC (Java Byte Code), Hierarchical\_Vulnerability\_Type\_Code... source/hex code </Patched> . This should be added for the various source code and binary versions so the LLM learns the differences between vulnerable and patched code / binaries.
4. **CVE Assessment**: <CVE\_Vector>Base Score: x.x Assessment code like CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H</CVE\_Vector>
5. **Short comments** are given in the syntax of the used programming language. The **actual vulnerability position** and the core **part of the exploit shall get comments**. “;” as comment till the end of the line for hex. E.g. use “//VULN:” or “//EXPL:” after the line or other comment-starting symbols replacing “//”.
6. **Shorter explanations**, e.g. relevant parts from a threat intelligence/malware analysis company writing a blog post/article: <EXPL>... Article in HTML, ideally in XHTML with external images ...</EXPL> We should not throw away any information but save all minimally as explanation.
7. **Longer explanations/background info**, e.g. less relevant parts (but still useful content) from a threat intelligence/malware analysis company writing a blog post/article: <EXPL2>... Article in HTML, ideally in XHTML with external images ...</EXPL2> We should not throw away any information but save all minimally as explanation.
8. **Characteristic Assembler-Binary-Mapping** to use inside the sections below:<ASM-Bin-Mapping>Asm-Statements separated by semicolons in one line “=>” Binary code in hex</ASM-Bin-Mapping>
9. **Synergistically combinable with vulnerability / exploit** to use inside the sections below and use these in strict pairs beginning with the CVE codes: <combine\_desc>Textual description</combine\_desc> and <combine\_CVE>CVE-code</combine\_CVE>
10. **Parameterizable part/variant**: <PAR>...</PAR>, e.g. jump/stack/heap parts or CPU registers used. **All parameterizable parts should be marked** as such so that all variants can be recognized by the AI.
11. **Web links of relevant sources / descriptions**: Put them all for each section into <Web\_src> ..URLs separated by “ ;”</Web\_src>
12. **Exploit**: <EXPL> Language-Id: C, ASM, PY, JS, HEX of binary, JBC (Java Byte Code), Hierarchical\_Exploit\_Type\_Code, ... source/hex code </EXPL>
13. **Memory/Stack/Heap/Network bytes** to expect and how to exploit them: TBD
14. **Exploit-Preparation**: <EXPL\_PREP> Language-Id: C, ASM, PY, JS, HEX of binary, JBC (Java Byte Code), Hierarchical\_Exploit\_Type\_Code ... source/hex code </EXPL\_PREP> That could be code to e.g. send the exploit **over a network, to a local driver and then e.g. a printer, (DMA) chip, device**.
15. **Exploit-Cleanup**: <EXPL\_CU> Language-Id: C, ASM, PY, JS, HEX of binary, JBC (Java Byte Code), ... Hierarchical\_Exploit\_Type\_Code source/hex code </EXPL\_CU>
16. **Indicator of Compromise (IoC)**: <IoC>...</IoC> The parts inside the IoC must also be standardized e.g. where they are to be found, in which file, memory location, etc.
17. **Detection pattern (extended virus scanner pattern with regular expressions, unique elements, etc**.: <PATT> Hierarchical\_Exploit\_Type\_Code ...</PATT> Using e.g. ClamAV, Yara, snort, tripwire, nmap, etc. as example and extending its patterns.
18. **Mitigation for admins**: <Mitigation>...</Mitigation>

All files should contain ideally the **vulnerability and exploit in a higher language like C or decompiled assembler** and also **in binary hex code** form.

### Hierarchical\_Vulnerability\_Type\_Codes

TBD, See CERT codes

### Hierarchical\_Exploit\_Type\_Codes

TBD, See CERT codes

# Extensions and Alternatives

## Collecting and comparing older with newer Sources and Binaries to see how Vulnerabilities were fixed

## AI Agent Frameworks

### For Data Labeling / Basic Cybersecurity

**Adala** – The First Open Source Data-Labeling Agent

We could adapt it and add cybersecurity knowledge, e.g. how to call a (dis)assembler:

<https://github.com/HumanSignal/Adala>

<https://venturebeat.com/ai/humansignal-launches-adala-open-source-framework-for-autonomous-data-labeling-agents/>

<https://thesequence.substack.com/p/guest-post-adala-the-first-open-source>

### MetaGPT

For various other tasks, e.g. decompiling, disassembling, creating simple sample programs, ...

## Buying Threat Intelligence

**Important Sources to monitor for Threat Intelligence:**

The following sources / newsgroups will be automatically monitored (every 6 hours) by a web scraper with immediate alerting functionality by e-mail:

1. <http://theregister.co.uk/>
2. <http://www.heise.de/security>
3. <http://cwe.mitre.org>
4. <http://cve.mitre.org/cve/cve.html>
5. <http://projects.webappsec.org>
6. <https://www.bsi.bund.de/ContentBSI/grundschutz/kataloge>
7. <http://www.hackerwiki.org/>
8. <https://www.wired.com/category/security/>
9. <https://www-945.ibm.com/support/fixcentral>
10. <http://seclists.org/fulldisclosure/>
11. <https://www.ansible.com/security>
12. <https://technet.microsoft.com/de-de/security/bulletins>
13. <https://technet.microsoft.com/de-de/security/advisories>
14. <http://googleprojectzero.blogspot.com/>
15. <https://www.exploit-db.com/>
16. <https://www.rapid7.com/db/modules/>
17. <https://github.com/rapid7/metasploit-framework>
18. <https://www.kernel-exploits.com/>
19. [www.ubuntu.com/usn/](http://www.ubuntu.com/usn/)
20. <https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/Linux-Linux-Kernel.html>
21. <https://www.cvedetails.com/>
22. <https://www.fireeye.com/blog/threat-research.html>
23. <https://www.hackread.com/sec/>
24. <http://9to5google.com/>
25. <http://www.phrack.org/issues/69/1.html>
26. <https://threatpost>.com/
27. <https://threatpost.com/tag/pastebin/>
28. <https://www.google.com/search?q=pastebin+security+news&ie=utf-8>
29. <https://securityintelligence.com/news/>
30. [https://www.helpnet**security**.com](https://www.helpnetsecurity.com)
31. [https://naked**security**.sophos.com](https://nakedsecurity.sophos.com)
32. <https://www.google.com/search?hl=en&gl=us&tbm=nws&authuser=0&q=exploit&oq=exploit>
33. <https://www.google.com/search?hl=en&gl=us&tbm=nws&authuser=0&q=vulnerability>
34. <https://www.google.com/search?hl=en&gl=us&tbm=nws&authuser=0&q=security+vulnerability>
35. <https://bits-please.blogspot.de/>
36. <http://blog.checkpoint.com/>
37. <https://blog.fortinet.com/>
38. <http://hackaday.com/blog/>
39. <https://www.perfect-privacy.com/german/blog/>
40. <http://t3n.de/news/>
41. <https://eprint.iacr.org/>
42. <https://media.defcon.org/>
43. <https://securelist.com/threats/detected-objects/>
44. <https://securelist.com/threats/vulnerabilities-and-hackers/>
45. <https://hpi-vdb.de/vulndb/>
46. <http://krebsonsecurity.com/>
47. <https://blog.ripstech.com/>
48. <https://blog.fefe.de/>
49. <https://www.kuketz-blog.de/>
50. <http://www.silicon.de/kategorie/security/>
51. <https://www.vusec.net/publications/>

**Important Tools/Toolkits/Patches/Apps:**

1. SELinux
2. AppArmor
3. Firejail <https://firejail.wordpress.com/>
4. Grsecurity patches

**Commercial Offerings for Threat Intelligence/Vulnerability Management**

The following commercial offerings (and more) exist with a functionality identical or similar to what is requested:

1. <https://www.virustotal.com/gui/intelligence-overview> Has videos at the bottom and a free trial
2. <http://www.flexerasoftware.com/enterprise/products/software-vulnerability-management/vulnerability-intelligence-manager/>
3. <http://www.tenable.com/de/securitycenter-continuous-view/?gclid=CJGHzcHvvc0CFdZsGwodBIoGHA>
4. <http://www.veracode.com/security/vulnerability-management>
5. <http://www.tripwire.com/>
6. <https://www.matrix42.com/de/produkte/unified-endpoint-management/>
7. <http://www.mcafee.com/us/products/security-scanner-for-databases.aspx>
8. <https://www.qualys.com/>
9. <https://www.rapid7.com/products/nexpose/>
10. <http://www.tenable.com/products/nessus-vulnerability-scanner>

# Strategy

## Bunch of Opportunities

This is a grandiose combination of opportunities:

1. Vulnerability scanner for various source codes and binaries
2. Fix and workaround proposing for source codes and binaries
3. 0day exploit and PoC (proof of concept) exploit creation system which can be sold at high prices to intelligence agencies, companies and for bug bounties. If these all fail, we could also use this ourselves for hacking (if a survival case).
4. New antivirus and SOC security patterns and next-level AI-based detection mechanisms.
5. ...

... and all this with the next level of AI!

## Costs / Project Plan

# Appendix

## Job Ad: X86 Vulnerability and Exploit Analyst helping towards SOC Detection AIs

As a cybersecurity and AI company, we want to develop a new next-generation SOC (security operations center) enterprise security sensor, i.e. a component which recognizes hacking and malware with AI. For that we need deep learning training data, in particular we’re looking for help with developing

1. Pairs of vulnerable x86 assembly/C code and corresponding exploit assembly/C code.
2. Typical infrastructure code, e.g. targeting the driver framework, UEFI/ROM, direct I/O, virtualization checks, etc.
3. JavaScript/PDF vulnerability and corresponding exploit analysis

**Your skills:**

1. Vulnerability and malware analysis skills for x86/JavaScript.
2. 0day exploits.
3. Static and dynamic code analysis and debugging tools like Metasploit, Burp Suite, angr, afl, BinaryAnalysisPlatform bap, KLEE, bindiff, BitBlaze, Ghidra-sre, x64dbg, IDA Pro, OllyDbg, AutoSploit, MetaSploit, Mimikatz, Kali Linux, snort, tcpdump, Wireshark, ClamAV, Yara, Malware analysis sandboxes like Cuckoo
4. Unpackers, PDF parsing, decompression & JS extraction.
5. Disassemblers, decompilers.
6. Analyzing sample exploits or their descriptions and extending them to provide all relevant details.
7. Analyzing binary patches regarding vulnerabilities and fixes.
8. Coding sample programs with vulnerable code and coding exploits, e.g. based on CERT info.
9. Application security, digital forensics, penetration testing, vulnerability assessment.
10. Ideally headless automation of tools.

The core task is nevertheless initially simple: Just taking existing vulnerability descriptions and exploits, gathering missing information by e.g. decompiling and reformatting the info with possibly additional comments to be optimally suited for machine learning. Later less experienced people or initially professionals could get higher payments.

We would be willing to employ you on a monthly basis with a salary as remote position. Thus please give your monthly salary as budget proposal. Please state your relevant experience regarding projects and tools.

**Skills:**

Security Software & Tools

Kali Linux Metasploit Wireshark

Information Security Skills

Application Security Digital Forensics

Other

Penetration Testing Vulnerability Assessment

**Freelancer.com auto-generated text:**

<https://www.freelancer.com/projects/c-programming/Vulnerability-Exploit-Analyst-helping/proposals>

I am looking for a skilled X86 Vulnerability and Exploit Analyst to assist in the development of our SOC Detection AIs. The ideal candidate will have experience in analyzing existing exploits and enhancing existing detection capabilities.

Specific tasks for this project include:

- Analyzing existing exploits and identifying vulnerabilities

- Post-analysis, the AI should have a basic understanding

- The desired outcome of the SOC Detection AI is to enhance existing detection capabilities

Skills and experience required:

- Proficiency in X86 vulnerability and exploit analysis

- Strong understanding of existing exploits

- Familiarity with SOC detection AI development

If you have the necessary skills and experience, please submit your proposal for consideration.

We propose to initially pay $40 per such tuple generated by you for any still relevant vulnerabilities since Spectre in 2017. Later, we will try to automate efforts and re-assess payment.