

How chatGPT4 found an anomaly in a CVSS 4.0 score

[linkedin.com/in/parisel](https://www.linkedin.com/in/parisel)

Needle in a haystack

- The task may sound easy for a human auditor
- It's not for a LLM! There is no way an AI will find anything wrong without preparation and methodology.

No improvisation

- The scope of exploration is huge and chatGPT conversations are limited: we must shrink the scope to where potential problems may lie
- ChatGPT must be trained to reason like a software vulnerability expert

CVSS 4.0 scores partitioning

- There are more than 10 million vectors
- Vectors are grouped into a small number of equivalence classes called **macroVectors** to make the scoring tractable
- Only the highest severity vector in each macroVector is hand-fed with a score

Step 1: prove that the 270 macroVectors form a partition of the CVSS 4.0 vectors



- No tool is needed here. Proof is straightforward by reading CVSS 4 specifications.
- 4 equations are independent (EQ1, EQ2, EQ4 and EQ5), so the partitioning can be proved individually
- Proof that inter-related equations EQ3 and EQ6 form a partition is done by making the cross-product of EQ3 and EQ6

CVSS 4.0 macrovectors ordering

- there are 270 macroVectors, so there are 270 manually-fed scores (with potential repeats)
- macroVectors are six dimensional (along EQ1 , ..., EQ6)
- In 6 dimensions, how to make sure that a lower macroVector isn't attributed a higher score than a higher macroVector?

Step 2: prove that macrovectors ordering is complete and consistent



- We use Z3 to encode all macrovector scores
- We ask Z3 to find a contradiction in the ordering
- Z3 couldn't find a contradiction: proof finished!

Sub-scoping

- CVSS-B is the most common (and most useful) subset of all CVSS: it focuses only on **base scores**
- CVSS-B only uses **11 metrics**
- The cardinality of CVSS-B vectors is 104976 => much smaller than the whole CVSS!

Step 3: find all macrovectors of CVSS-B



- CVSS-B is a tiny subset of CVSS, so it must use a subset of the 270 macrovectors
- We use Z3 to find the CVSS-B macrovectors: there are 36

CVSS-B Vectors sampling

- To reason about CVSS-B vectors, chatGPT needs at least one sample from in each of the 36 macrovectors
- Samples must be random to avoid bias
- Highest and lowest severity vectors must be avoided (unless sampled at random)

Step 4: sample vectors from CVSS-B macrovectors



- We use Z3 to find one random vector from each of the 36 macrovectors

Final ChatGPT reasoning scope

- We started off with 10 millions vectors
- We end up feeding chatGPT with the following 36 vectors:

110200:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
012201:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
011100:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:H/SI:H/SA:H
010200:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
112201:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
201200:/AV:A/AC:L/AT:N/PR:L/UI:P/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
202201:/AV:A/AC:L/AT:N/PR:L/UI:P/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
211200:/AV:A/AC:H/AT:N/PR:L/UI:P/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
101200:/AV:A/AC:L/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
010100:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
000200:/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
212201:/AV:A/AC:H/AT:N/PR:L/UI:P/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
110100:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
111200:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
111100:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:H/SI:H/SA:H
200200:/AV:A/AC:L/AT:N/PR:L/UI:P/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
000100:/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

110200:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
012201:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
011100:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:H/SI:H/SA:H
010200:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
112201:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
201200:/AV:A/AC:L/AT:N/PR:L/UI:P/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
202201:/AV:A/AC:L/AT:N/PR:L/UI:P/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
211200:/AV:A/AC:H/AT:N/PR:L/UI:P/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
101200:/AV:A/AC:L/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
010100:/AV:N/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
000200:/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
212201:/AV:A/AC:H/AT:N/PR:L/UI:P/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L
110100:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H
111200:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:L/SI:L/SA:L
111100:/AV:A/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:H/SI:H/SA:H
200200:/AV:A/AC:L/AT:N/PR:L/UI:P/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L
000100:/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

ChatGPT training

- ChatGPT must be explained the 11 metrics and the vector format
- We must perform sanity checks to make sure the format is grasped

Here are the possible values of each metric. For each metric, metric values are ordered from highest severity to lowest severity:

AV: AV:N (Network), AV:A (adjacent), AV:L (local), AV:P (physical)

AC: AC:L (lo), AC:H (hi)

AT: AT:N (none), AT:P (present)

PR: PR:N (none), PR:L (lo), PR:H (hi)

UI: UI:N (none), UI:P (passive), UI:A (active)

For VC, VI, VA we have three values: N (none), L (low), H (high)

For SC, SI, SA we have three values as well: N (none), L (low), H (high)

To-be-scored vectors are provided in the following sample format:

/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H. Now please use this format to show me the lowest severity vector (where every single metric is at minimum).

Based on the provided metrics and their severity values, the vector representing the lowest severity (minimum impact) would be:

/AV:P/AC:H/AT:P/PR:H/UI:A/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

This vector signifies:

- **Attack Vector (AV):** Physical (the hardest access to exploit)
- **Attack Complexity (AC):** High (more complex to exploit)
- **Attack Requirements (AT):** Present (requires some conditions)
- **Privileges Required (PR):** High (requires high-level privileges)
- **User Interaction (UI):** Active (requires the target user to take an action)
- **Confidentiality Impact to the Vulnerable System (VC):** None (no impact)
- **Integrity Impact to the Vulnerable System (VI):** None (no impact)
- **Availability Impact to the Vulnerable System (VA):** None (no impact)
- **Confidentiality Impact to the Subsequent System (SC):** None (no impact)
- **Integrity Impact to the Subsequent System (SI):** None (no impact)
- **Availability Impact to the Subsequent System (SA):** None (no impact)

This vector essentially represents a vulnerability that's extremely difficult to exploit and, even if exploited, would have minimal impact.

Feeding chatGPT with the sampling

- The LLM is now ready to be fed with the 36 sampled vectors (slide 7)
- It must also be fed with the corresponding 36 hand-fed scores, which can be obtained from the online CVSS 4.0 calculator javascript code:

000100:10.0
000200:9.3
001100:9.3
001200:8.8
002101:7.9
002201:6.9
010100:9.5
010200:9.2
011100:9.2
011200:8.4
012101:7.1
012201:6.3
100100:9.4
100200:8.7
101100:8.6
101200:7.2
102101:6.5
102201:5.3

110100:9.0
110200:7.7
111100:7.4
111200:6.1
112101:5.8
112201:2.3
200100:8.6
200200:7.0
201100:7.2
201200:5.3
202101:4.7
202201:2.4
210100:7.3
210200:5.4
211100:6.1
211200:4.6
212101:2.4
212201:1.0

ChatGPT's finding

- From the point, chatGPT found a vector rated 7.9 (HIGH) which should be rated CRITICAL

Given that only scores ≥ 9.0 are critical, can you find vectors which should be critical and which are not?

Certainly, let's evaluate vectors with SI:H and SC:H (as we previously focused on these) and ensure that their impact is commensurate with a score that should classify them as "critical" (≥ 9.0).

1. **001100:**

- **Vector:** AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:H/SI:H/SA:H
- **Score:** 9.3
- **Classification:** Correctly Critical

2. **011100:**

- **Vector:** AV:N/AC:H/AT:N/PR:N/UI:N/VC:L/VI:H/VA:H/SC:H/SI:H/SA:H
- **Score:** 9.2
- **Classification:** Correctly Critical

3. **002101:**

- **Vector:** AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:L/VA:L/SC:H/SI:H/SA:H
- **Score:** 7.9
- **Classification:** Not Critical

Given the SI and SC are both high in these vectors and given their other attributes, it seems **002101** stands out as an anomaly. While its vulnerabilities are low in impact on the compromised system (VC:L, VI:L, VA:L), the deep impact on subsequent systems (SI:H and SC:H) should heighten its overall severity score, especially since the attack vector is Network-based (AV:N) and the Attack Complexity is Low (AC:L).

Thus, the vector **002101** should potentially be considered critical, or at least closer to the critical threshold, given its deep impact on subsequent systems.

Appendix

Proving macroVectors ordering with Z3

```

def proveOrdering():
    global sol
    sol.push()
    macroVector2score=Function('macroVector2score',BitVecSort(4),BitVecSort(4),BitVecSort(4),BitVecSort(4),BitVecSort(4),BitVecSort(4),BitVecSort(4),BitVecSort(8))
    u1=BitVec('u1',4)
    u2=BitVec('u2',4)
    v1=BitVec('v1',4)
    v2=BitVec('v2',4)
    w1=BitVec('w1',4)
    w2=BitVec('w2',4)
    x1=BitVec('x1',4)
    x2=BitVec('x2',4)
    y1=BitVec('y1',4)
    y2=BitVec('y2',4)
    z1=BitVec('z1',4)
    z2=BitVec('z2',4)

    score1=BitVec("score1",8)
    score2=BitVec("score2",8)

    sol.add(u1>=0,u2>=0,v1>=0,v2>=0,w1>=0,w2>=0,x1>=0,x2>=0,y1>=0,y2>=0,z1>=0,z2>=0)
    sol.add(u1<3,u2<3,v1<3,v2<3,w1<3,w2<3,x1<3,x2<3,y1<3,y2<3,z1<3,z2<3)

    coords=[]
    for entry in macroVectorsDictionary:
        coordstr=re.findall('([0-2])',entry)
        scorestr=macroVectorsDictionary[entry]
        score=int(10.0*float(scorestr))
        z3expr='sol.add(macroVector2score('+coordstr[0]+'+',+coordstr[1]+'+',+coordstr[2]+'+',+coordstr[3]+'+',+coordstr[4]+'+',+coordstr[5]+'')=='+'+str(score)+'')'
        coord=eval(coordstr[0]+'+',+coordstr[1]+'+',+coordstr[2]+'+',+coordstr[3]+'+',+coordstr[4]+'+',+coordstr[5])
        coords.append(coord)
        eval(z3expr)

    for u in range(0,3):
        for v in range(0,3):
            for w in range(0,3):
                for x in range(0,3):
                    for y in range(0,3):
                        for z in range(0,3):
                            found=False
                            for cs in coords:
                                if cs[0]==u and cs[1]==v and cs[2]==w and cs[3]==x and cs[4]==y and cs[5]==z:
                                    found=True
                                    break
                            if found==False:
                                z3expr="sol.add(macroVector2score("+str(u)+"", "+str(v)+"", "+str(w)+"", "+str(x)+"", "+str(y)+"", "+str(z)+"") ==0) "
                                eval(z3expr)
    sol.add(UGT(macroVector2score(u2,v2,w2,x2,y2,z2),0))
    sol.add(score1==macroVector2score(u1,v1,w1,x1,y1,z1))
    sol.add(score2==macroVector2score(u2,v2,w2,x2,y2,z2))
    sol.add(And(UGE(u1,u2),UGE(v1,v2),UGE(w1,w2),UGE(x1,x2),UGE(y1,y2),UGE(z1,z2)))
    # find counter example
    sol.add(ULT(macroVector2score(u2,v2,w2,x2,y2,z2),macroVector2score(u1,v1,w1,x1,y1,z1)))
    if sol.check()==sat:
        print("SAT")
        mdl=sol.model()
        print("1:",mdl.evaluate(u1),mdl.evaluate(v1),mdl.evaluate(w1),mdl.evaluate(x1),mdl.evaluate(y1),mdl.evaluate(z1))
        print("score 1:",mdl.evaluate(score1))
        print("2:",mdl.evaluate(u2),mdl.evaluate(v2),mdl.evaluate(w2),mdl.evaluate(x2),mdl.evaluate(y2),mdl.evaluate(z2))
        print("score 2:",mdl.evaluate(score2))
    else:
        print("Could not find a counter-example: this is perfect!")
    sol.pop()

```

Appendix

CVSS-B Vectors sampler in Z3 (snippet only, code too long)

```

def sampler():
    global sol
    EQ1_0=Bool('EQ1_0')
    EQ1_0=And(AV_N,PR_N,UI_N)
    EQ1_1=Bool('EQ1_1')
    EQ1_1=And(Or(AV_N,PR_N,UI_N),Not(And(AV_N,PR_N,UI_N)),Not(AV_P))
    EQ1_2=Bool('EQ1_2')
    EQ1_2=Or(AV_P,Not(Or(AV_N,PR_N,UI_N)))
    [...]
    EQ6_0=Bool('EQ6_0')
    EQ6_0=Or(VC_H,VI_H,VA_H) # default enviro is 'H' for CR,IR,AR
    EQ6_1=Bool('EQ6_1')
    EQ6_1=Not(Or(VC_H,VI_H,VA_H)) # default enviro is 'H' for CR,IR,AR

    for entry in macroVectorsDictionary:
        sol.push()
        EQ1=True
        EQ2=True
        EQ3=True
        EQ4=True
        EQ5=True
        EQ6=True
        coordstr=re.findall('([0-2])',entry)
        if coordstr[0]=='0':
            EQ1=EQ1_0
        elif coordstr[0]=='1':
            EQ1=EQ1_1
        elif coordstr[0]=='2':
            EQ1=EQ1_2
        else:
            EQ1=False
    [...]
    if coordstr[5]=='0':
        EQ6=EQ6_0
    elif coordstr[5]=='1':
        EQ6=EQ6_1
    else:
        EQ6=False
    sol.add(And(EQ1,EQ2,EQ3,EQ4,EQ5,EQ6))
    sol.add(Or(AV_N,AV_A,AV_L,AV_P))
    sol.add(Or(AC_L,AC_H))
    sol.add(Or(AT_N,AT_P))
    sol.add(Or(PR_N,PR_L,PR_H))
    sol.add(Or(UI_N,UI_P,UI_A))
    sol.add(Or(VC_H,VC_L,VC_N))
    sol.add(Or(VI_H,VI_L,VI_N))
    sol.add(Or(VA_H,VA_L,VA_N))
    sol.add(Or(SC_H,SC_L,SC_N))
    sol.add(Or(SI_H,SI_L,SI_N))
    sol.add(Or(SA_H,SA_L,SA_N))
    sol.check()

```