ABDK CONSULTING

SMART CONTRACT AUDIT

NotionalV2

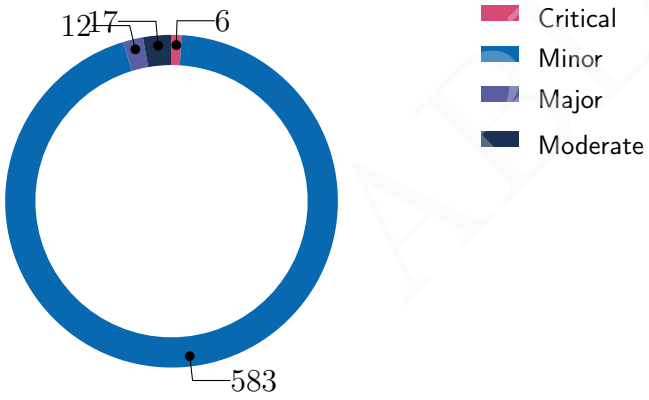
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
6th September 2021

We've been asked to review the 46 files in a Github repo.
We found 6 critical, 12 major, and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Bad datatype	Opened
CVF-3	Minor	Bad naming	Opened
CVF-4	Minor	Bad datatype	Opened
CVF-5	Minor	Suboptimal	Opened
CVF-6	Minor	Bad datatype	Opened
CVF-7	Minor	Suboptimal	Opened
CVF-8	Minor	Procedural	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Minor	Suboptimal	Opened
CVF-11	Minor	Documentation	Opened
CVF-12	Minor	Suboptimal	Opened
CVF-13	Minor	Suboptimal	Opened
CVF-14	Minor	Readability	Opened
CVF-15	Minor	Suboptimal	Opened
CVF-16	Minor	Readability	Opened
CVF-17	Minor	Bad naming	Opened
CVF-18	Minor	Unclear behavior	Opened
CVF-19	Minor	Documentation	Opened
CVF-20	Minor	Readability	Opened
CVF-21	Minor	Unclear behavior	Opened
CVF-22	Minor	Suboptimal	Opened
CVF-23	Minor	Readability	Opened
CVF-24	Minor	Suboptimal	Opened
CVF-25	Minor	Procedural	Opened
CVF-26	Minor	Bad datatype	Opened
CVF-27	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-28	Minor	Unclear behavior	Opened
CVF-29	Minor	Documentation	Opened
CVF-30	Minor	Documentation	Opened
CVF-31	Minor	Bad datatype	Opened
CVF-32	Minor	Documentation	Opened
CVF-33	Minor	Bad datatype	Opened
CVF-34	Minor	Bad datatype	Opened
CVF-35	Minor	Documentation	Opened
CVF-36	Minor	Suboptimal	Opened
CVF-37	Minor	Readability	Opened
CVF-38	Minor	Bad datatype	Opened
CVF-39	Minor	Suboptimal	Opened
CVF-40	Minor	Procedural	Opened
CVF-41	Minor	Suboptimal	Opened
CVF-42	Minor	Readability	Opened
CVF-43	Minor	Readability	Opened
CVF-44	Minor	Procedural	Opened
CVF-45	Minor	Documentation	Opened
CVF-46	Minor	Suboptimal	Opened
CVF-47	Minor	Suboptimal	Opened
CVF-48	Minor	Suboptimal	Opened
CVF-49	Minor	Procedural	Opened
CVF-50	Minor	Bad datatype	Opened
CVF-51	Minor	Procedural	Opened
CVF-52	Minor	Unclear behavior	Opened
CVF-53	Minor	Suboptimal	Opened
CVF-54	Minor	Suboptimal	Opened
CVF-55	Minor	Suboptimal	Opened
CVF-56	Minor	Readability	Opened
CVF-57	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-58	Minor	Suboptimal	Opened
CVF-59	Minor	Suboptimal	Opened
CVF-60	Minor	Suboptimal	Opened
CVF-61	Minor	Suboptimal	Opened
CVF-62	Minor	Bad datatype	Opened
CVF-63	Minor	Bad datatype	Opened
CVF-64	Minor	Procedural	Opened
CVF-65	Moderate	Flaw	Opened
CVF-66	Minor	Procedural	Opened
CVF-67	Minor	Procedural	Opened
CVF-68	Minor	Procedural	Opened
CVF-69	Minor	Procedural	Opened
CVF-70	Minor	Bad datatype	Opened
CVF-71	Minor	Bad naming	Opened
CVF-72	Minor	Suboptimal	Opened
CVF-73	Minor	Suboptimal	Opened
CVF-74	Minor	Bad datatype	Opened
CVF-75	Minor	Suboptimal	Opened
CVF-76	Minor	Overflow/Underflow	Opened
CVF-77	Minor	Suboptimal	Opened
CVF-78	Minor	Documentation	Opened
CVF-79	Minor	Suboptimal	Opened
CVF-80	Minor	Suboptimal	Opened
CVF-81	Moderate	Overflow/Underflow	Opened
CVF-82	Minor	Documentation	Opened
CVF-83	Minor	Documentation	Opened
CVF-84	Minor	Documentation	Opened
CVF-85	Major	Suboptimal	Opened
CVF-86	Minor	Flaw	Opened
CVF-87	Minor	Procedural	Opened

ID	Severity	Category	Status
CVF-88	Minor	Suboptimal	Opened
CVF-89	Minor	Suboptimal	Opened
CVF-90	Minor	Documentation	Opened
CVF-91	Minor	Bad naming	Opened
CVF-92	Minor	Flaw	Opened
CVF-93	Minor	Flaw	Opened
CVF-94	Minor	Bad datatype	Opened
CVF-95	Minor	Suboptimal	Opened
CVF-96	Minor	Suboptimal	Opened
CVF-97	Minor	Readability	Opened
CVF-98	Minor	Suboptimal	Opened
CVF-99	Minor	Suboptimal	Opened
CVF-100	Minor	Readability	Opened
CVF-101	Minor	Suboptimal	Opened
CVF-102	Minor	Suboptimal	Opened
CVF-103	Minor	Suboptimal	Opened
CVF-104	Moderate	Flaw	Opened
CVF-105	Minor	Suboptimal	Opened
CVF-106	Minor	Suboptimal	Opened
CVF-107	Minor	Suboptimal	Opened
CVF-108	Moderate	Flaw	Opened
CVF-109	Moderate	Flaw	Opened
CVF-110	Minor	Suboptimal	Opened
CVF-111	Major	Flaw	Opened
CVF-112	Major	Flaw	Opened
CVF-113	Minor	Flaw	Opened
CVF-114	Minor	Flaw	Opened
CVF-115	Minor	Suboptimal	Opened
CVF-116	Minor	Procedural	Opened
CVF-117	Minor	Flaw	Opened

ID	Severity	Category	Status
CVF-118	Minor	Procedural	Opened
CVF-119	Minor	Flaw	Opened
CVF-120	Minor	Suboptimal	Opened
CVF-121	Minor	Bad datatype	Opened
CVF-122	Minor	Readability	Opened
CVF-123	Minor	Suboptimal	Opened
CVF-124	Minor	Suboptimal	Opened
CVF-125	Minor	Suboptimal	Opened
CVF-126	Moderate	Flaw	Opened
CVF-127	Minor	Flaw	Opened
CVF-128	Moderate	Flaw	Opened
CVF-129	Minor	Procedural	Opened
CVF-130	Minor	Procedural	Opened
CVF-131	Minor	Procedural	Opened
CVF-132	Moderate	Flaw	Opened
CVF-133	Minor	Suboptimal	Opened
CVF-134	Minor	Suboptimal	Opened
CVF-135	Minor	Procedural	Opened
CVF-136	Minor	Readability	Opened
CVF-137	Minor	Procedural	Opened
CVF-138	Minor	Procedural	Opened
CVF-139	Minor	Suboptimal	Opened
CVF-140	Minor	Readability	Opened
CVF-141	Minor	Readability	Opened
CVF-142	Minor	Suboptimal	Opened
CVF-143	Minor	Suboptimal	Opened
CVF-144	Minor	Flaw	Opened
CVF-145	Minor	Readability	Opened
CVF-146	Minor	Flaw	Opened
CVF-147	Minor	Bad naming	Opened

ID	Severity	Category	Status
CVF-148	Minor	Suboptimal	Opened
CVF-149	Minor	Flaw	Opened
CVF-150	Minor	Readability	Opened
CVF-151	Minor	Overflow/Underflow	Opened
CVF-152	Minor	Bad naming	Opened
CVF-153	Minor	Procedural	Opened
CVF-154	Minor	Suboptimal	Opened
CVF-155	Minor	Procedural	Opened
CVF-156	Minor	Readability	Opened
CVF-157	Minor	Suboptimal	Opened
CVF-158	Minor	Readability	Opened
CVF-159	Critical	Flaw	Opened
CVF-160	Minor	Suboptimal	Opened
CVF-161	Minor	Suboptimal	Opened
CVF-162	Moderate	Suboptimal	Opened
CVF-163	Minor	Flaw	Opened
CVF-164	Minor	Readability	Opened
CVF-165	Minor	Readability	Opened
CVF-166	Minor	Suboptimal	Opened
CVF-167	Minor	Readability	Opened
CVF-168	Minor	Readability	Opened
CVF-169	Minor	Unclear behavior	Opened
CVF-170	Minor	Procedural	Opened
CVF-171	Minor	Procedural	Opened
CVF-172	Minor	Procedural	Opened
CVF-173	Minor	Suboptimal	Opened
CVF-174	Minor	Documentation	Opened
CVF-175	Minor	Documentation	Opened
CVF-176	Minor	Suboptimal	Opened
CVF-177	Minor	Readability	Opened

ID	Severity	Category	Status
CVF-178	Minor	Suboptimal	Opened
CVF-179	Minor	Overflow/Underflow	Opened
CVF-180	Minor	Documentation	Opened
CVF-181	Minor	Suboptimal	Opened
CVF-182	Minor	Suboptimal	Opened
CVF-183	Minor	Suboptimal	Opened
CVF-184	Minor	Readability	Opened
CVF-185	Minor	Readability	Opened
CVF-186	Minor	Procedural	Opened
CVF-187	Minor	Bad naming	Opened
CVF-188	Minor	Readability	Opened
CVF-189	Minor	Bad naming	Opened
CVF-190	Minor	Suboptimal	Opened
CVF-191	Minor	Bad naming	Opened
CVF-192	Minor	Documentation	Opened
CVF-193	Minor	Readability	Opened
CVF-194	Minor	Documentation	Opened
CVF-195	Minor	Readability	Opened
CVF-196	Minor	Procedural	Opened
CVF-197	Minor	Documentation	Opened
CVF-198	Minor	Suboptimal	Opened
CVF-199	Minor	Suboptimal	Opened
CVF-200	Minor	Suboptimal	Opened
CVF-201	Minor	Suboptimal	Opened
CVF-202	Minor	Documentation	Opened
CVF-203	Moderate	Overflow/Underflow	Opened
CVF-204	Minor	Suboptimal	Opened
CVF-205	Minor	Suboptimal	Opened
CVF-206	Minor	Suboptimal	Opened
CVF-207	Minor	Procedural	Opened

ID	Severity	Category	Status
CVF-208	Minor	Procedural	Opened
CVF-209	Minor	Procedural	Opened
CVF-210	Minor	Suboptimal	Opened
CVF-211	Minor	Suboptimal	Opened
CVF-212	Minor	Flaw	Opened
CVF-213	Critical	Flaw	Opened
CVF-214	Minor	Overflow/Underflow	Opened
CVF-215	Minor	Suboptimal	Opened
CVF-216	Minor	Procedural	Opened
CVF-217	Minor	Suboptimal	Opened
CVF-218	Minor	Flaw	Opened
CVF-219	Minor	Procedural	Opened
CVF-220	Minor	Bad datatype	Opened
CVF-221	Minor	Flaw	Opened
CVF-222	Minor	Suboptimal	Opened
CVF-223	Minor	Readability	Opened
CVF-224	Major	Flaw	Opened
CVF-225	Minor	Unclear behavior	Opened
CVF-226	Minor	Flaw	Opened
CVF-227	Minor	Procedural	Opened
CVF-228	Minor	Procedural	Opened
CVF-229	Minor	Unclear behavior	Opened
CVF-230	Minor	Suboptimal	Opened
CVF-231	Minor	Suboptimal	Opened
CVF-232	Minor	Procedural	Opened
CVF-233	Minor	Suboptimal	Opened
CVF-234	Minor	Suboptimal	Opened
CVF-235	Minor	Bad datatype	Opened
CVF-236	Minor	Procedural	Opened
CVF-237	Minor	Documentation	Opened

ID	Severity	Category	Status
CVF-238	Minor	Procedural	Opened
CVF-239	Minor	Suboptimal	Opened
CVF-240	Minor	Procedural	Opened
CVF-241	Minor	Overflow/Underflow	Opened
CVF-242	Minor	Suboptimal	Opened
CVF-243	Minor	Readability	Opened
CVF-244	Minor	Readability	Opened
CVF-245	Minor	Readability	Opened
CVF-246	Minor	Procedural	Opened
CVF-247	Minor	Suboptimal	Opened
CVF-248	Minor	Suboptimal	Opened
CVF-249	Minor	Suboptimal	Opened
CVF-250	Minor	Overflow/Underflow	Opened
CVF-251	Minor	Suboptimal	Opened
CVF-252	Minor	Suboptimal	Opened
CVF-253	Moderate	Flaw	Opened
CVF-254	Minor	Unclear behavior	Opened
CVF-255	Minor	Bad naming	Opened
CVF-256	Minor	Readability	Opened
CVF-257	Minor	Bad naming	Opened
CVF-258	Minor	Bad naming	Opened
CVF-259	Minor	Suboptimal	Opened
CVF-260	Minor	Documentation	Opened
CVF-261	Minor	Readability	Opened
CVF-262	Major	Flaw	Opened
CVF-263	Minor	Documentation	Opened
CVF-264	Minor	Procedural	Opened
CVF-265	Minor	Suboptimal	Opened
CVF-266	Minor	Documentation	Opened
CVF-267	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-268	Minor	Readability	Opened
CVF-269	Minor	Readability	Opened
CVF-270	Minor	Suboptimal	Opened
CVF-271	Minor	Documentation	Opened
CVF-272	Minor	Flaw	Opened
CVF-273	Minor	Procedural	Opened
CVF-274	Minor	Readability	Opened
CVF-275	Minor	Suboptimal	Opened
CVF-276	Minor	Suboptimal	Opened
CVF-277	Minor	Suboptimal	Opened
CVF-278	Minor	Suboptimal	Opened
CVF-279	Minor	Suboptimal	Opened
CVF-280	Minor	Bad naming	Opened
CVF-281	Minor	Procedural	Opened
CVF-282	Minor	Documentation	Opened
CVF-283	Minor	Unclear behavior	Opened
CVF-284	Minor	Documentation	Opened
CVF-285	Minor	Suboptimal	Opened
CVF-286	Minor	Suboptimal	Opened
CVF-287	Minor	Suboptimal	Opened
CVF-288	Minor	Documentation	Opened
CVF-289	Minor	Suboptimal	Opened
CVF-290	Critical	Flaw	Opened
CVF-291	Minor	Suboptimal	Opened
CVF-292	Minor	Suboptimal	Opened
CVF-293	Minor	Suboptimal	Opened
CVF-294	Minor	Flaw	Opened
CVF-295	Minor	Suboptimal	Opened
CVF-296	Minor	Suboptimal	Opened
CVF-297	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-298	Minor	Suboptimal	Opened
CVF-299	Minor	Suboptimal	Opened
CVF-300	Minor	Suboptimal	Opened
CVF-301	Minor	Suboptimal	Opened
CVF-302	Minor	Suboptimal	Opened
CVF-303	Minor	Suboptimal	Opened
CVF-304	Minor	Suboptimal	Opened
CVF-305	Minor	Readability	Opened
CVF-306	Minor	Procedural	Opened
CVF-307	Minor	Documentation	Opened
CVF-308	Minor	Bad naming	Opened
CVF-309	Minor	Overflow/Underflow	Opened
CVF-310	Minor	Bad naming	Opened
CVF-311	Minor	Readability	Opened
CVF-312	Minor	Suboptimal	Opened
CVF-313	Minor	Bad naming	Opened
CVF-314	Minor	Suboptimal	Opened
CVF-315	Minor	Suboptimal	Opened
CVF-316	Minor	Suboptimal	Opened
CVF-317	Minor	Suboptimal	Opened
CVF-318	Minor	Suboptimal	Opened
CVF-319	Minor	Suboptimal	Opened
CVF-320	Minor	Bad datatype	Opened
CVF-321	Minor	Procedural	Opened
CVF-322	Minor	Suboptimal	Opened
CVF-323	Minor	Readability	Opened
CVF-324	Minor	Suboptimal	Opened
CVF-325	Minor	Suboptimal	Opened
CVF-326	Minor	Documentation	Opened
CVF-327	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-328	Minor	Bad naming	Opened
CVF-329	Minor	Readability	Opened
CVF-330	Minor	Readability	Opened
CVF-331	Minor	Readability	Opened
CVF-332	Minor	Suboptimal	Opened
CVF-333	Minor	Bad datatype	Opened
CVF-334	Minor	Suboptimal	Opened
CVF-335	Minor	Overflow/Underflow	Opened
CVF-336	Minor	Bad naming	Opened
CVF-337	Minor	Procedural	Opened
CVF-338	Minor	Flaw	Opened
CVF-339	Minor	Readability	Opened
CVF-340	Minor	Bad naming	Opened
CVF-341	Minor	Documentation	Opened
CVF-342	Minor	Suboptimal	Opened
CVF-343	Minor	Bad naming	Opened
CVF-344	Minor	Suboptimal	Opened
CVF-345	Minor	Suboptimal	Opened
CVF-346	Minor	Suboptimal	Opened
CVF-347	Minor	Suboptimal	Opened
CVF-348	Moderate	Flaw	Opened
CVF-349	Minor	Suboptimal	Opened
CVF-350	Minor	Suboptimal	Opened
CVF-351	Minor	Suboptimal	Opened
CVF-352	Minor	Procedural	Opened
CVF-353	Minor	Suboptimal	Opened
CVF-354	Minor	Suboptimal	Opened
CVF-355	Minor	Bad datatype	Opened
CVF-356	Minor	Suboptimal	Opened
CVF-357	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-358	Minor	Suboptimal	Opened
CVF-359	Minor	Suboptimal	Opened
CVF-360	Minor	Suboptimal	Opened
CVF-361	Minor	Suboptimal	Opened
CVF-362	Minor	Bad datatype	Opened
CVF-363	Minor	Suboptimal	Opened
CVF-364	Minor	Suboptimal	Opened
CVF-365	Major	Flaw	Opened
CVF-366	Minor	Suboptimal	Opened
CVF-367	Minor	Readability	Opened
CVF-368	Moderate	Overflow/Underflow	Opened
CVF-369	Minor	Bad naming	Opened
CVF-370	Minor	Flaw	Opened
CVF-371	Minor	Bad naming	Opened
CVF-372	Minor	Flaw	Opened
CVF-373	Minor	Suboptimal	Opened
CVF-374	Minor	Suboptimal	Opened
CVF-375	Major	Flaw	Opened
CVF-376	Minor	Suboptimal	Opened
CVF-377	Minor	Bad naming	Opened
CVF-378	Minor	Suboptimal	Opened
CVF-379	Minor	Bad datatype	Opened
CVF-380	Minor	Readability	Opened
CVF-381	Minor	Procedural	Opened
CVF-382	Minor	Unclear behavior	Opened
CVF-383	Minor	Unclear behavior	Opened
CVF-384	Minor	Bad datatype	Opened
CVF-385	Minor	Unclear behavior	Opened
CVF-386	Minor	Overflow/Underflow	Opened
CVF-387	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-388	Minor	Documentation	Opened
CVF-389	Minor	Suboptimal	Opened
CVF-390	Minor	Documentation	Opened
CVF-391	Minor	Documentation	Opened
CVF-392	Minor	Overflow/Underflow	Opened
CVF-393	Minor	Readability	Opened
CVF-394	Minor	Suboptimal	Opened
CVF-395	Minor	Suboptimal	Opened
CVF-396	Minor	Suboptimal	Opened
CVF-397	Minor	Bad datatype	Opened
CVF-398	Minor	Suboptimal	Opened
CVF-399	Minor	Suboptimal	Opened
CVF-400	Minor	Suboptimal	Opened
CVF-401	Minor	Readability	Opened
CVF-402	Minor	Suboptimal	Opened
CVF-403	Minor	Bad datatype	Opened
CVF-404	Minor	Suboptimal	Opened
CVF-405	Minor	Procedural	Opened
CVF-406	Minor	Flaw	Opened
CVF-407	Minor	Suboptimal	Opened
CVF-408	Minor	Documentation	Opened
CVF-409	Minor	Documentation	Opened
CVF-410	Minor	Suboptimal	Opened
CVF-411	Minor	Procedural	Opened
CVF-412	Moderate	Flaw	Opened
CVF-413	Minor	Flaw	Opened
CVF-414	Minor	Procedural	Opened
CVF-415	Minor	Overflow/Underflow	Opened
CVF-416	Minor	Suboptimal	Opened
CVF-417	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-418	Minor	Suboptimal	Opened
CVF-419	Minor	Documentation	Opened
CVF-420	Minor	Bad naming	Opened
CVF-421	Minor	Bad naming	Opened
CVF-422	Minor	Suboptimal	Opened
CVF-423	Minor	Unclear behavior	Opened
CVF-424	Minor	Readability	Opened
CVF-425	Minor	Suboptimal	Opened
CVF-426	Critical	Flaw	Opened
CVF-427	Minor	Suboptimal	Opened
CVF-428	Minor	Suboptimal	Opened
CVF-429	Minor	Bad naming	Opened
CVF-430	Minor	Suboptimal	Opened
CVF-431	Minor	Bad naming	Opened
CVF-432	Minor	Suboptimal	Opened
CVF-433	Minor	Bad naming	Opened
CVF-434	Minor	Bad naming	Opened
CVF-435	Critical	Flaw	Opened
CVF-436	Minor	Readability	Opened
CVF-437	Minor	Procedural	Opened
CVF-438	Minor	Bad naming	Opened
CVF-439	Moderate	Procedural	Opened
CVF-440	Minor	Overflow/Underflow	Opened
CVF-441	Minor	Suboptimal	Opened
CVF-442	Minor	Bad naming	Opened
CVF-443	Minor	Bad naming	Opened
CVF-444	Minor	Documentation	Opened
CVF-445	Minor	Bad naming	Opened
CVF-446	Minor	Bad naming	Opened
CVF-447	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-448	Minor	Flaw	Opened
CVF-449	Minor	Suboptimal	Opened
CVF-450	Minor	Suboptimal	Opened
CVF-451	Moderate	Flaw	Opened
CVF-452	Minor	Suboptimal	Opened
CVF-453	Minor	Procedural	Opened
CVF-454	Minor	Documentation	Opened
CVF-455	Minor	Suboptimal	Opened
CVF-456	Minor	Suboptimal	Opened
CVF-457	Minor	Suboptimal	Opened
CVF-458	Minor	Suboptimal	Opened
CVF-459	Minor	Suboptimal	Opened
CVF-460	Minor	Suboptimal	Opened
CVF-461	Minor	Suboptimal	Opened
CVF-462	Minor	Documentation	Opened
CVF-463	Minor	Suboptimal	Opened
CVF-464	Minor	Procedural	Opened
CVF-465	Minor	Suboptimal	Opened
CVF-466	Minor	Procedural	Opened
CVF-467	Minor	Suboptimal	Opened
CVF-468	Minor	Suboptimal	Opened
CVF-469	Minor	Suboptimal	Opened
CVF-470	Minor	Suboptimal	Opened
CVF-471	Minor	Overflow/Underflow	Opened
CVF-472	Minor	Documentation	Opened
CVF-473	Minor	Suboptimal	Opened
CVF-474	Minor	Suboptimal	Opened
CVF-475	Minor	Procedural	Opened
CVF-476	Minor	Suboptimal	Opened
CVF-477	Minor	Readability	Opened

ID	Severity	Category	Status
CVF-478	Minor	Readability	Opened
CVF-479	Minor	Suboptimal	Opened
CVF-480	Minor	Suboptimal	Opened
CVF-481	Minor	Procedural	Opened
CVF-482	Minor	Documentation	Opened
CVF-483	Minor	Suboptimal	Opened
CVF-484	Minor	Suboptimal	Opened
CVF-485	Minor	Suboptimal	Opened
CVF-486	Minor	Suboptimal	Opened
CVF-487	Minor	Suboptimal	Opened
CVF-488	Minor	Suboptimal	Opened
CVF-489	Minor	Suboptimal	Opened
CVF-490	Minor	Suboptimal	Opened
CVF-491	Critical	Procedural	Opened
CVF-492	Minor	Procedural	Opened
CVF-493	Minor	Suboptimal	Opened
CVF-494	Minor	Readability	Opened
CVF-495	Minor	Suboptimal	Opened
CVF-496	Minor	Unclear behavior	Opened
CVF-497	Minor	Suboptimal	Opened
CVF-498	Minor	Suboptimal	Opened
CVF-499	Minor	Suboptimal	Opened
CVF-500	Major	Flaw	Opened
CVF-501	Minor	Documentation	Opened
CVF-502	Minor	Suboptimal	Opened
CVF-503	Minor	Suboptimal	Opened
CVF-504	Minor	Suboptimal	Opened
CVF-505	Minor	Suboptimal	Opened
CVF-506	Minor	Suboptimal	Opened
CVF-507	Major	Overflow/Underflow	Opened

ID	Severity	Category	Status
CVF-508	Minor	Suboptimal	Opened
CVF-509	Minor	Suboptimal	Opened
CVF-510	Minor	Readability	Opened
CVF-511	Minor	Suboptimal	Opened
CVF-512	Minor	Bad datatype	Opened
CVF-513	Minor	Overflow/Underflow	Opened
CVF-514	Minor	Suboptimal	Opened
CVF-515	Minor	Procedural	Opened
CVF-516	Minor	Readability	Opened
CVF-517	Major	Flaw	Opened
CVF-518	Minor	Suboptimal	Opened
CVF-519	Major	Unclear behavior	Opened
CVF-520	Minor	Unclear behavior	Opened
CVF-521	Minor	Suboptimal	Opened
CVF-522	Minor	Flaw	Opened
CVF-523	Minor	Documentation	Opened
CVF-524	Moderate	Suboptimal	Opened
CVF-525	Minor	Suboptimal	Opened
CVF-526	Minor	Suboptimal	Opened
CVF-527	Minor	Procedural	Opened
CVF-528	Minor	Suboptimal	Opened
CVF-529	Minor	Flaw	Opened
CVF-530	Minor	Suboptimal	Opened
CVF-531	Minor	Suboptimal	Opened
CVF-532	Minor	Bad naming	Opened
CVF-533	Minor	Suboptimal	Opened
CVF-534	Minor	Suboptimal	Opened
CVF-535	Minor	Procedural	Opened
CVF-536	Minor	Suboptimal	Opened
CVF-537	Minor	Readability	Opened

ID	Severity	Category	Status
CVF-538	Minor	Suboptimal	Opened
CVF-539	Minor	Unclear behavior	Opened
CVF-540	Minor	Suboptimal	Opened
CVF-541	Minor	Suboptimal	Opened
CVF-542	Minor	Suboptimal	Opened
CVF-543	Minor	Suboptimal	Opened
CVF-544	Minor	Suboptimal	Opened
CVF-545	Minor	Flaw	Opened
CVF-546	Minor	Flaw	Opened
CVF-547	Minor	Overflow/Underflow	Opened
CVF-548	Minor	Bad naming	Opened
CVF-549	Minor	Suboptimal	Opened
CVF-550	Minor	Suboptimal	Opened
CVF-551	Minor	Suboptimal	Opened
CVF-552	Minor	Suboptimal	Opened
CVF-553	Minor	Suboptimal	Opened
CVF-554	Minor	Suboptimal	Opened
CVF-555	Minor	Procedural	Opened
CVF-556	Minor	Suboptimal	Opened
CVF-557	Minor	Suboptimal	Opened
CVF-558	Minor	Procedural	Opened
CVF-559	Minor	Suboptimal	Opened
CVF-560	Minor	Suboptimal	Opened
CVF-561	Minor	Suboptimal	Opened
CVF-562	Minor	Readability	Opened
CVF-563	Minor	Overflow/Underflow	Opened
CVF-564	Minor	Procedural	Opened
CVF-565	Minor	Suboptimal	Opened
CVF-566	Minor	Suboptimal	Opened
CVF-567	Minor	Overflow/Underflow	Opened

ID	Severity	Category	Status
CVF-568	Minor	Suboptimal	Opened
CVF-569	Minor	Documentation	Opened
CVF-570	Minor	Readability	Opened
CVF-571	Minor	Suboptimal	Opened
CVF-572	Minor	Readability	Opened
CVF-573	Minor	Bad naming	Opened
CVF-574	Minor	Overflow/Underflow	Opened
CVF-575	Minor	Documentation	Opened
CVF-576	Minor	Documentation	Opened
CVF-577	Minor	Bad naming	Opened
CVF-578	Minor	Flaw	Opened
CVF-579	Major	Documentation	Opened
CVF-580	Minor	Documentation	Opened
CVF-581	Minor	Suboptimal	Opened
CVF-582	Minor	Overflow/Underflow	Opened
CVF-583	Minor	Suboptimal	Opened
CVF-584	Minor	Procedural	Opened
CVF-585	Minor	Suboptimal	Opened
CVF-586	Minor	Suboptimal	Opened
CVF-587	Minor	Procedural	Opened
CVF-588	Minor	Documentation	Opened
CVF-589	Minor	Documentation	Opened
CVF-590	Minor	Suboptimal	Opened
CVF-591	Minor	Bad datatype	Opened
CVF-592	Minor	Unclear behavior	Opened
CVF-593	Minor	Procedural	Opened
CVF-594	Minor	Readability	Opened
CVF-595	Minor	Suboptimal	Opened
CVF-596	Minor	Suboptimal	Opened
CVF-597	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-598	Minor	Suboptimal	Opened
CVF-599	Minor	Suboptimal	Opened
CVF-600	Minor	Suboptimal	Opened
CVF-601	Minor	Unclear behavior	Opened
CVF-602	Minor	Suboptimal	Opened
CVF-603	Minor	Suboptimal	Opened
CVF-604	Minor	Overflow/Underflow	Opened
CVF-605	Minor	Documentation	Opened
CVF-606	Minor	Suboptimal	Opened
CVF-607	Minor	Overflow/Underflow	Opened
CVF-608	Minor	Flaw	Opened
CVF-609	Minor	Flaw	Opened
CVF-610	Minor	Flaw	Opened
CVF-611	Minor	Suboptimal	Opened
CVF-612	Minor	Suboptimal	Opened
CVF-613	Minor	Suboptimal	Opened
CVF-614	Minor	Readability	Opened
CVF-615	Minor	Suboptimal	Opened
CVF-616	Minor	Suboptimal	Opened
CVF-617	Minor	Suboptimal	Opened
CVF-618	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	September 5, 2021	D. Khovratovich	Initial Draft
0.2	September 5, 2021	D. Khovratovich	Minor revision
1.0	September 6, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations. We have reviewed the next files:

- external/actions/AccountAction.sol
- external/actions/BatchAction.sol
- external/actions/ERC1155Action.sol
- external/actions/GovernanceAction.sol
- external/actions/InitializeMarketsAction.sol
- external/actions/LiquidateCurrencyAction.sol
- external/actions/LiquidatefCashAction.sol
- external/actions/TradingAction.sol
- external/actions/nTokenAction.sol
- external/actions/nTokenMintAction.sol
- external/actions/nTokenRedeemAction.sol
- external/adapters/NotionalV1Migrator.sol
- external/adapters/cTokenAggregator.sol
- external/adapters/nTokenERC20Proxy.sol
- external/governance/GovernorAlpha.sol
- external/governance/NoteERC20.sol
- external/governance/Reservoir.sol
- external/FreeCollateralExternal.sol
- external/Router.sol
- external/SettleAssetsExternal.sol
- external/Views.sol
- global/Constants.sol
- global/StorageLayoutV1.sol
- global/Types.sol

- internal/balances/BalanceHandler.sol
- internal/balances/Incentives.sol
- internal/balances/TokenHandler.sol
- internal/liquidation/LiquidateCurrency.sol
- internal/liquidation/LiquidatefCash.sol
- internal/liquidation/LiquidationHelpers.sol
- internal/markets/AssetRate.sol
- internal/markets/CashGroup.sol
- internal/markets/DateTime.sol
- internal/markets/Market.sol
- internal/portfolio/BitmapAssetsHandler.sol
- internal/portfolio/PortfolioHandler.sol
- internal/portfolio/TransferAssets.sol
- internal/settlement/SettleBitmapAssets.sol
- internal/settlement/SettlePortfolioAssets.sol
- internal/valuation/AssetHandler.sol
- internal/valuation/ExchangeRate.sol
- internal/valuation/FreeCollateral.sol
- internal/AccountContextHandler.sol
- internal/nTokenHandler.sol
- math/Bitmap.sol
- math/SafeInt256.sol

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StorageLayoutV1.sol+ Description

Description Should be "̂0.7.0" rather than ">0.7.0" as Solidity 0.8.x has a number of non backward compatible changes. Also relevant for the next files: Types.sol, Router.sol, SettleAssetsExternal.sol, FreeCollateralExternal.sol, Views.sol,Types.sol, SettleAssetsExternal.sol, Router.sol, NotionalV1Migrator.sol, nTokenERC20Proxy.sol, ERC1155Action.sol, LiquidateCurrencyAction.sol, nTokenAction.sol,TradingAction.sol, AccountAction.sol, BatchAction.sol, GovernanceAction.sol, InitializeMarketsAction.sol, LiquidatefCashAction.sol, nTokenMintAction.sol, nTokenRedeemAction.sol, LiquidatefCashAction.sol, InitializeMarketsAction.sol, nTokenAction.sol, LiquidateCurrencyAction.sol, GovernanceAction.sol, BatchAction.sol, AccountAction.sol, ERC1155Action.sol, Bitmap.sol, SafeInt256.sol, SettlePortfolioAssets.sol, SettleBitmapAssets.sol, BitmapAssetsHandler.sol, TransferAssets.sol, Market.sol, DateTime.sol, CashGroup.sol, AssetRate.sol, LiquidationHelpers.sol, LiquidatefCash.sol, LiquidateCurrency.sol, BalanceHandler.sol, TokenHandler.sol, Incentives.sol, nTokenHandler.sol, AccountContextHandler.sol, ExchangeRate.sol, AssetHandler.sol, FreeCollateral.sol, AssetRate.sol, CashGroup.sol, DateTime.sol, TokenHandler.sol,LiquidateCurrency.sol, Market.sol, LiquidationHelpers.sol, LiquidatefCash.sol, StorageLayoutV1.sol, Incentives.sol, BalanceHandler.sol, FreeCollateral.sol, SettleBitmapAssets.sol, AssetHandler.sol, SafeInt256.sol, Bitmap.sol, ExchangeRate.sol, SettlePortfolioAssets.sol, TransferAssets.sol, BitmapAssetsHandler.sol, nERC1155Interface.sol.

Listing 1:

```
2 solidity >0.7.0;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** StorageLayoutV1.sol + Description

Description The type of this variable should probably be more specific.

Listing 2:

```
25 address internal token;
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** StorageLayoutV1.sol

Description The semantics of the keys in this mapping is unclear. Consider adding a documentation comment.

Listing 3:

```
37 mapping(address => mapping(address => bool)) internal
    ↪ accountAuthorizedTransferOperator;
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** StorageLayoutV1.sol

Description The key type for this mapping should be "Token".

Listing 4:

```
41 mapping(address => uint16) internal tokenAddressToCurrencyId;
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description It is a bad practice to treat token amounts as fraction numbers. All token amounts are integers when measured in base units, and the "decimals" property is basically a hint for UI that helps rendering token amounts in a user-friendly way. Usually, smart contracts don't need to care about decimals at all.

Listing 5:

```
6 // Token precision used for all internal balances, TokenHandler
    ↪ library ensures that we
// limit the dust amount caused by precision mismatches
int256 internal constant INTERNAL_TOKEN_PRECISION = 1e8;
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Constants.sol

Description In most cases currency IDs are represented as uint16 values, but here it is uint256. Consider using the same type everywhere.

Listing 6:

```
11 uint256 internal constant ETH_CURRENCY_ID = 1;
```

3.7 CVF-7

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description The number of decimals in ether amounts is just a UI feature for human-readable rendering of such amounts. Internally, all ether amounts are integer numbers denominated in Wei, and smart contracts usually treat them like integers. Treating them as fixed-point fraction numbers just introduces unnecessary complexity. Consider measuring ether amounts in Wei.

Listing 7:

```
12 int256 internal constant ETH_DECIMAL_PLACES = 18;
   int256 internal constant ETH_DECIMALS = 1e18;
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Constants.sol

Description This constant is not used. Consider removing it.

Listing 8:

```
12 int256 internal constant ETH_DECIMAL_PLACES = 18;
```

3.9 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description If the constant would be initialized using the expression in the comment, the comment would be redundant.

Listing 9:

```
16 uint256 internal constant DELEVERAGE_BUFFER = 30000000; // 300 *
    ↪  Constants.BASIS_POINT
```

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description Hardcoding addresses is a bad practice. It makes development, testing, and deployment harder and more error-prone. Consider passing all necessary addresses as constructor arguments and storing them as immutable variables.

Listing 10:

```
21 address constant NOTE_TOKEN_ADDRESS = 0
    ↪ x46B2efE8BE4a97F05826264E88148fc083D595BD;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Constants.sol

Description The terms "most significant" and "less significant" make sense for binary numbers, whereas this constant is a vector of 32 bytes. Consider using term "leftmost bit" or something like this.

Listing 11:

```
23 // Most significant bit
bytes32 internal constant MSB=0
    ↪ x8000000000000000000000000000000000000000000000000000000000000000
    ↪ ;
```

3.12 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description This constant is a low-level technical thing rather than a business-level thing. Consider moving it to some other place.

Listing 12:

```
24 bytes32 internal constant MSB =
```

3.13 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description Using only two decimals for percentages is very limiting, It allows 2% and 3%,but not 2.5%. Consider using more decimals.

Listing 13:

```
28 int256 internal constant PERCENTAGE_DECIMALS = 100;
```

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Constants.sol

Description This could be written as: DAY = 1 days;

Listing 14:

```
36 uint256 internal constant DAY = 86400;
```

3.15 CVF-15

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Constants.sol

Description In order to emphasize that all the intervals divide evenly, it would be better to say: MONTH = WEEK * 5; QUARTER = MONTH * 3;

Listing 15:

```
39 uint256 internal constant MONTH = DAY * 30;
40 uint256 internal constant QUARTER = DAY * 90;
```

3.16 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Constants.sol

Description This value could be rendered as "360 days".

Listing 16:

```
55 uint256 internal constant IMPLIED_RATE_TIME = 31104000;
```

3.17 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Constants.sol

Description The name is confusing. One could think that this is the basis point denominator (10^4), but actually this is one basis point in RATE_PRECISION terms. Consider adding a documentation comment and/or renaming.

Listing 17:

```
58 uint256 internal constant BASIS_POINT = uint256(RATE_PRECISION /
   ↪  10000);
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Constants.sol

Description What is the reason to use bytes1 instead of bool?

Listing 18:

```
69 bytes1 internal constant BOOL_FALSE = 0x00;
70 bytes1 internal constant BOOL_TRUE = 0x01;
```

3.19 CVF-19

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Constants.sol

Description The units of these numbers are unclear. Consider documenting. Also, consider using the same fractional number format across all the code. Currently there are too many different formats.

Listing 19:

```
92 int256 internal constant DEFAULT_LIQUIDATION_PORTION = 40;

94 int256 internal constant TOKEN_REPO_INCENTIVE_PERCENT = 10;

96 int256 internal constant LIQUIDATION_DUST = 10;

99 uint256 internal constant ANNUAL_INCENTIVE_MULTIPLIER_PERCENT =
   ↪ 50;
```

3.20 CVF-20

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Constants.sol

Description It is a bad practice to deal with storage offsets manually. Leaving this to Solidity compiler would make the code simpler and less error-prone.

Listing 20:

```
104 // Internally used storage slots are set at 1000000 offset from
    ↪ the solidity provisioned storage slots to minimize
    // the possibility of clashing.
    uint256 internal constant ACCOUNT_CONTEXT_STORAGE_OFFSET =
    ↪ 1000001;
    uint256 internal constant NTOKEN_CONTEXT_STORAGE_OFFSET =
    ↪ 1000002;
    uint256 internal constant NTOKEN_ADDRESS_STORAGE_OFFSET =
    ↪ 1000003;
    uint256 internal constant NTOKEN_DEPOSIT_STORAGE_OFFSET =
    ↪ 1000004;
110 uint256 internal constant NTOKEN_INIT_STORAGE_OFFSET = 1000005;
    uint256 internal constant BALANCE_STORAGE_OFFSET = 1000006;
    uint256 internal constant TOKEN_STORAGE_OFFSET = 1000007;
    uint256 internal constant SETTLEMENT_RATE_STORAGE_OFFSET =
    ↪ 1000008;
    uint256 internal constant CASH_GROUP_STORAGE_OFFSET = 1000009;
    uint256 internal constant MARKET_STORAGE_OFFSET = 1000010;
    uint256 internal constant ASSETS_BITMAP_STORAGE_OFFSET =
    ↪ 1000011;
    uint256 internal constant IFCASH_STORAGE_OFFSET = 1000012;
    uint256 internal constant PORTFOLIO_ARRAY_STORAGE_OFFSET =
    ↪ 1000013;
```

3.21 CVF-21

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Types.sol

Description What if a Compound token backed by a non-mintable token will ever be registered in the protocol? Should it be possible to change the type of the token from "NonMintable" to "UnderlyingToken"?

Listing 21:

```
10 – NonMintable: tokens that do not have an underlying (therefore
    ↪ not cTokens)
```

3.22 CVF-22

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Types.sol

Description As most of the trade actions have some unused bytes when backed into 32 bytes. Packing a series of actions into a bytes array without unused bytes would be more efficient.

Listing 22:

```
13 @notice Specifies the different trade action types in the system
   ↪ . Each trade action type is
encoded in a tightly packed bytes32 object. Trade action type is
   ↪ the first big endian byte of the
32 byte trade action object. The schemas for each trade action
   ↪ type are defined below.
```

3.23 CVF-23

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Types.sol

Description This structure could be simplified to: struct BalanceActionWithTrades { BalanceAction balanceAction; byte32 [] trades; }

Listing 23:

```
75 BalanceActionWithTrades {
```

3.24 CVF-24

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Types.sol

Description As currency ID is only 16 bits wide, it would be possible to pack a settle amount into a single 256-bits word leaving 240 bits for the signed net cash change value.

Listing 24:

```
88 SettleAmount {
```

3.25 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Types.sol

Description While in most cases current IDs are represented as uint16 values, here uint256 is used. Consider using the same type across the code.

Listing 25:

```
 89 uint256 currencyId;
165 uint256 currencyId;
196 uint256 currencyId;
205 uint256 currencyId;
340 uint256 currencyId;
```

3.26 CVF-26

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Types.sol

Description This field should have type "ERC20".

Listing 26:

```
95 address tokenAddress;
```

3.27 CVF-27

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Types.sol

Description In ERC-20, the “decimals” property is 8 bits wide, so the whole structure could be packed into a single 256-bit word: 160 bits for the address and 8 bits per other field.

Listing 27:

```
97 int256 decimals;
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Types.sol

Description This looks like a floating point number manually implemented. Is it really necessary? Consider using fixed-point numbers with good enough precision.

Listing 28:

```
151 // The decimals (i.e. 10^rateDecimalPlaces) of the exchange rate
    int256 rateDecimals;
    // The exchange rate from base to ETH (if rate invert is
       ↪ required it is already done)
    int256 rate;

188 // The exchange rate from base to quote (if invert is required
       ↪ it is already done)
    int256 rate;
190 // The decimals of the underlying, the rate converts to the
       ↪ underlying decimals
    int256 underlyingDecimals;
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Types.sol

Description The format of the buffer and haircut values is unclear. Consider explaining in a comment.

Listing 29:

```
155 // Amount of buffer to apply to the exchange rate for negative
       ↪ balances.

157 // Amount of haircut to apply to the exchange rate for positive
       ↪ balances
```

3.30 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Types.sol

Description The format of the discount value is unclear. Consider explaining in a comment.

Listing 30:

```
159 // Liquidation discount for this currency
```

3.31 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Types.sol

Description The type of this field should be more specific.

Listing 31:

```
187 address rateOracle;
```

3.32 CVF-32

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Types.sol

Description The formats of these values is unclear. Consider explaining in the corresponding documentation comments.

Listing 32:

```
228 uint256 lastImpliedRate;

230 uint256 oracleRate;
```

3.33 CVF-33

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Types.sol

Description This field should have type "ERC20".

Listing 33:

```
263 address tokenAddress;
```

3.34 CVF-34

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Types.sol

Description The type of this field should be more specific.

Listing 34:

```
272 address rateOracle;

289 address rateOracle;
```

3.35 CVF-35

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Types.sol

Description This object doesn't actually store asset rate. Consider renaming or explaining in the documentation comment, why the current name is fine.

Listing 35:

```
286 @dev Asset rate object as it is represented in storage, total
    ↪ storage is 21 bytes.
AssetRateStorage {
```

3.36 CVF-36

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Types.sol

Description Dynamic arrays are quite expensive. As the maximum sized of these arrays are known to be 7, consider using the "bytes7" type instead of "uint8[]". This would also make it cheaper to pack/unpack these arrays.

Listing 36:

```
318 uint8[] liquidityTokenHaircuts;

320 uint8[] rateScalars;
```

3.37 CVF-37

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Router.sol

Description These variables should be properly typed for readability and to allow type-safe calls.

Listing 37:

```
30 address public immutable GOVERNANCE;
   address public immutable VIEWS;
   address public immutable INITIALIZE_MARKET;
   address public immutable NTOKEN_ACTIONS;
   address public immutable NTOKEN_REDEEM;
   address public immutable BATCH_ACTION;
   address public immutable ACCOUNT_ACTION;
   address public immutable ERC1155;
   address public immutable LIQUIDATE_CURRENCY;
   address public immutable LIQUIDATE_FCASH;
40 address public immutable cETH;
```

3.38 CVF-38

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Router.sol

Description These arguments should be properly typed for readability.

Listing 38:

```
43 address governance_,
   address views_,
   address initializeMarket_,
   address nTokenActions_,
   address nTokenRedeem_,
   address batchAction_,
   address accountAction_,
50 address erc1155_,
   address liquidateCurrency_,
   address liquidatefCash_,
   address cETH_
```

3.39 CVF-39

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Router.sol

Description This wouldn't be necessary in case GovernanceAction would be a library, as library function doesn't need to check access.

Listing 39:

```
73 // Allow list currency to be called by this contract for the
   ↪ purposes of
// initializing ETH as a currency
owner = msg.sender;
```

3.40 CVF-40

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Router.sol

Description Libraries is a standard way to split large logic among several contracts deployed separately. Consider using them instead of type-unsafe 'abi.encodeWithSelector' plus explicit delegate calls.

Listing 40:

```
78 address(GOVERNANCE).delegatecall(
```

3.41 CVF-41

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Router.sol

Description This function basically replicates what the call dispatcher of a contract usually does. More common and type-safe way would be to write type-safe wrappers for all the functions that ought to be routed like this:
function funcN (... args ...) public returns (... rets ...) { return LibraryM.funcN (... args ...); }

Listing 41:

```
99 function getRouterImplementation(bytes4 sig) public view returns
   ↪ (address) {
```

3.42 CVF-42

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Router.sol

Description Should be "else if" for readability.

Listing 42:

```
107 if (
123 if (
133 if (
140 if (
154 if (
163 if (
172 if (sig == InitializeMarketsAction.initializeMarkets.selector) {
176 if (
```

3.43 CVF-43

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Router.sol

Description Should be "else return" for readability.

Listing 43:

```
194 return VIEWS;
```

3.44 CVF-44

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Router.sol

Description This contract should be moved to a separate file named "nTransparentUpgradeableProxy".

Listing 44:

```
233 nTransparentUpgradeableProxy is TransparentUpgradeableProxy {
```

3.45 CVF-45

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Router.sol

Description It is a good practice to put a comment into an empty block to explain why it is empty.

Listing 45:

```
240 receive() external payable override {}
```

3.46 CVF-46

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleAssetsExternal.sol

Description The expression "settleAmounts[0]" is calculated twice. Consider using structure literal assignment like this: settleAmounts[0] = SettleAmount ({ currencyId, settledCache });

Listing 46:

```
128 settleAmounts[0].currencyId = currencyId;
    settleAmounts[0].netCashChange = settledCash;
```

3.47 CVF-47

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateralExternal.sol

Description All the function inside this library are very simple and probably not worth to be made external. Making them internal would make it cheaper to use them.

Listing 47:

```
9 FreeCollateralExternal {
```

3.48 CVF-48

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateralExternal.sol

Description This variable is used only once and probably is not needed.

Listing 48:

```
34 uint256 blockTime = block.timestamp;
```

3.49 CVF-49

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Views.sol

Description Returning zero on unlisted currency is error-prone. Consider revering in such a case.

Listing 49:

```
31 /// @notice Returns a currency id, a zero means that it is not
   ↪ listed.
function getCurrencyId(address tokenAddress)
```

3.50 CVF-50

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Views.sol

Description The argument type should be "Token".

Listing 50:

```
32 function getCurrencyId(address tokenAddress)
```

3.51 CVF-51

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Views.sol

Description There is no range check for the argument. Consider reverting in case the "currencyId" value is zero or exceeds the maximum listed currency ID.

Listing 51:

```
 43 function getCurrency(uint16 currencyId)

 54 function getRateStorage(uint16 currencyId)

 66 function getCurrencyAndRates(uint16 currencyId)

 84 function getCashGroup(uint16 currencyId)

 94 function getCashGroupAndAssetRate(uint16 currencyId)

105 function getInitializationParameters(uint16 currencyId)

119 function getDepositParameters(uint16 currencyId)

133 function nTokenAddress(uint16 currencyId) external view override
    ↪  returns (address) {

145 function getSettlementRate(uint16 currencyId, uint40 maturity)

155 function getActiveMarkets(uint16 currencyId)

167 function getActiveMarketsAtBlockTime(uint16 currencyId, uint32
    ↪ blockTime)

176 function _getActiveMarketsAtBlockTime(uint256 currencyId,
    ↪ uint256 blockTime)

192 function getReserveBalance(uint16 currencyId)

248         currencyId,

333 function getAccountBalance(uint16 currencyId, address account)

375     uint256 currencyId,

382 function getAssetsBitmap(address account, uint256 currencyId)

405 function calculateNTokensToMint(uint16 currencyId, uint88
    ↪ amountToDepositExternalPrecision)
```

(433, 463)

3.52 CVF-52

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Views.sol

Description This function is just a union of two other functions: "getCurrency" and "getRateStorage". It is really necessary?

Listing 52:

```
66 function getCurrencyAndRates(uint16 currencyId)
```

3.53 CVF-53

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description This function looks redundant. It is not useful on-chain, and the current owner could be efficiently found off-chain from the logged events.

Listing 53:

```
138 function getOwner() external view override returns (address) {
```

3.54 CVF-54

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description Getting all the markets could be suboptimal in case the caller needs only some of them. Consider implementing functions to obtain the number of active markets and getting the markets with indexes in certain range.

Listing 54:

```
155 function getActiveMarkets(uint16 currencyId)

167 function getActiveMarketsAtBlockTime(uint16 currencyId, uint32
    ↪ blockTime)
```

3.55 CVF-55

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description The function always returns an array of 10 elements even if the actual number of the balance is smaller. Consider returned a properly sized array. Note, that there is possible to shrink an already allocated array using assembly.

Listing 55:

```
273 AccountBalance[] memory accountBalances,
```

3.56 CVF-56

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Views.sol

Description This variable is not explicitly initialized. Consider initializing to zero for readability.

Listing 56:

```
280 uint256 i;
```

3.57 CVF-57

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description The expression "accontBalances[i]" is calculated several times. Consider calculating once and reusing, or consider using struct literal assignment.

Listing 57:

```
283 accountBalances[i].cashBalance,
    accountBalances[i].nTokenBalance,
    accountBalances[i].lastClaimTime,
    accountBalances[i].lastClaimSupplyAmount

299 accountBalances[i].cashBalance,
300 accountBalances[i].nTokenBalance,
    accountBalances[i].lastClaimTime,
    accountBalances[i].lastClaimSupplyAmount
```

3.58 CVF-58

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description The explicit conversion to uint256 it redundant.

Listing 58:

```
293 accountBalances[i].currencyId = uint256(
```

3.59 CVF-59

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description The conversion to uint16 performs a shift here. This shift could be avoided by extracting the lowest 16 bits instead of the highest ones: uint16 (uint256 (currencies)) & UNMASK_FLAGS. In such a case, right shift should be used at the end of the loop instead of left shift.

Listing 59:

```
294 uint16(bytes2(currencies) & Constants.UNMASK_FLAGS)
```

3.60 CVF-60

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description The value "accountBalances[i].currencyId" that was just written into the memory is read here. Consider storing the value in a local variable and reusing.

Listing 60:

```
296 if (accountBalances[i].currencyId == 0) break;

303 ) = BalanceHandler.getBalanceStorage(account, accountBalances[i
    ↪ ].currencyId);
```

3.61 CVF-61

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Views.sol

Description This function is redundant. Solidity compiler automatically rejects all unrecognized calls in case there is no fallback function.

Listing 61:

```
482 fallback() external {
        revert("Method not found");
    }
```

3.62 CVF-62

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** cTokenAggregator.sol

Description This variable should have type "CTokenInterface".

Listing 62:

```
13 address public override token;
```

3.63 CVF-63

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** cTokenAggregator.sol

Description The "_cToken" argument should have type "CTokenInterface".

Listing 63:

```
22 constructor(address _cToken) {
```

3.64 CVF-64

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** cTokenAggregator.sol

Description This code, that performs safe conversion from uint256 to int256, is written twice. Consider extracting into a utility function.

Listing 64:

```
34 require(exchangeRate <= uint256(type(int256).max), "
     ↪ cTokenAdapter: overflow");

36 return int256(exchangeRate);

41 require(exchangeRate <= uint256(type(int256).max), "
     ↪ cTokenAdapter: overflow");

43 return int256(exchangeRate);
```

3.65 CVF-65

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** cTokenAggregator.sol

Description The formula used by this function is different from the formula recommended by the Compound documentation: https://compound.finance/docs#protocol-math Consider using the recommended formula.

Listing 65:

```
46 function getAnnualizedSupplyRate() external view override
     ↪ returns (uint256) {
```

3.66 CVF-66

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NotionalV1Migrator.sol

Description This interface should be moved to a separate file "WETH9.sol".

Listing 66:

```
9 WETH9 {
```

3.67 CVF-67

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NotionalV1Migrator.sol

Description This interface should be moved to a separate file "IEscrow.sol".

Listing 67:

```
15 IEscrow {
```

3.68 CVF-68

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NotionalV1Migrator.sol

Description This interface should be moved to a separate file "UniswapPair.sol".

Listing 68:

```
19 UniswapPair {
```

3.69 CVF-69

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NotionalV1Migrator.sol

Description This interface should be moved to a separate file "INotionalV1Erc1155".

Listing 69:

```
28 INotionalV1Erc1155 {
```

3.70 CVF-70

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NotionalV1Migrator.sol

Description In Ethereum, timestamps are usually represented as uint256 values, so consider using uint256 here.

Listing 70:

```
64 uint32 maxTime,
```

3.71 CVF-71

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** NotionalV1Migrator.sol

Description It is uncommon to start variable names with capital letter. Such names look like contract names and make code harder to read. Consider renaming.

Listing 71:

```
72 IEscrow public immutable Escrow;
   NotionalProxy public immutable NotionalV2;
   INotionalV1Erc1155 public immutable NotionalV1Erc1155;

76 WETH9 public immutable WETH;
   IERC20 public immutable WBTC;
```

3.72 CVF-72

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NotionalV1Migrator.sol

Description Is this function really necessary? Consider calling "approve" on WBTC in the constructor.

Listing 72:

```
111 function enableWBTC() external {
```

3.73 CVF-73

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NotionalV1Migrator.sol

Description The returned value is ignored. Probably, not an issue.

Listing 73:

```
112 WBTC.approve(address(NotionalV2), type(uint256).max);
```

3.74 CVF-74

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NotionalV1Migrator.sol

Description Inner conversions to bytes32 are redundant, as bitwise "or" could be applied directly to the uint256 values.

Listing 74:

```
186 (bytes32(uint256(TradeActionType.Borrow)) << 248) |
        (bytes32(uint256(v2MarketIndex)) << 240) |
        (bytes32(uint256(v2fCashAmount)) << 152) |
        (bytes32(uint256(v2MaxBorrowImpliedRate)) << 120)
```

3.75 CVF-75

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NotionalV1Migrator.sol

Description This could be simplified as: int256 collateralBalance = balances[v1CollateralId];

Listing 75:

```
202 int256 collateralBalance =
        (v1CollateralId == V1_ETH ? balances[V1_ETH] : balances[
            ↪ V1_WBTC]);
```

3.76 CVF-76

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** NotionalV1Migrator.sol

Description Overflow is possible here.

Listing 76:

```
218 uint256 swapAmount = (uint256(collateralBalance) * 996) / 1000;
```

3.77 CVF-77

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NotionalV1Migrator.sol

Description In case the value of the "v1CollateralId" variable is neither V1_ETH nor V2_WBTC, the function silently does nothing. Consider reverting in such a case.

Listing 77:

```
223 }
```

```
232 }
```

3.78 CVF-78

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** NotionalV1Migrator.sol

Description This comment is confusing, as there is no transfer directly behind it.

Listing 78:

```
258 // transfer tokens to original caller
```

3.79 CVF-79

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NotionalV1Migrator.sol

Description This variable is redundant, as its value is always zero.

Listing 79:

```
305 uint256 debtIndex = 0;
```

3.80 CVF-80

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NotionalV1Migrator.sol

Description The expression "tradeExecution[debtIndex]" is calculated several times. Consider calculating once and reusing.

Listing 80:

```
306 tradeExecution[debtIndex].actionType = DepositActionType.None;
    tradeExecution[debtIndex].currencyId = v2DebtCurrencyId;
    tradeExecution[debtIndex].withdrawEntireCashBalance = true;
    tradeExecution[debtIndex].redeemToUnderlying = true;
310 tradeExecution[debtIndex].trades = new bytes32[](1);
    tradeExecution[debtIndex].trades[0] = tradeData;
```

3.81 CVF-81

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** NotionalV1Migrator.sol

Description Overflow is possible here.

Listing 81:

```
335 withdraws[0].amount = uint128(collateralAmount);
```

3.82 CVF-82

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** NotionalV1Migrator.sol

Description This can be made a precompile constant or at least a documentation comment should be there.

Listing 82:

```
356 return 0xf23a6e61;
```

3.83 CVF-83

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenERC20Proxy.sol

Description Should be "nToken (...).sumbol ()".

Listing 83:

```
15 /// @notice Will be "n{Underlying Token}.symbol()"
```

3.84 CVF-84

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenERC20Proxy.sol

Description This constant is named "INTERNAL_TOKEN_PRECISION".

Listing 84:

```
18 /// @notice Inherits from Constants.INTERNAL_TOKEN_DECIMALS
```

3.85 CVF-85

- **Severity** Major
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenERC20Proxy.sol

Description These events are logged even when the corresponding operation returned false, i.e. was unsuccessful. Consider logging events only on successful operations, or explicitly requiring the operations to be successful.

Listing 85:

```
69  emit Approval(msg.sender, spender, amount);

80  emit Transfer(msg.sender, to, amount);

99  emit Transfer(from, to, amount);
100 emit Approval(msg.sender, from, newAllowance);
```

3.86 CVF-86

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** nTokenERC20Proxy.sol

Description Emitting Approval on transferFrom call is not fully compliant to the EIP20 standard.

Listing 86:

```
100 emit Approval(msg.sender, from, newAllowance);
```

3.87 CVF-87

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenERC20Proxy.sol

Description These functions are not defined by ERC20, thus they are not required for ERC20 compliance. Consider moving them into some other place.

Listing 87:

```
106 function getPresentValueAssetDenominated() external view returns
    ↪ (int256) {

111 function getPresentValueUnderlyingDenominated() external view
    ↪ returns (int256) {

116 function onERC1155Received(

127 function onERC1155BatchReceived(
```

3.88 CVF-88

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenERC20Proxy.sol

Description These functions are redundant. The common way to forbid incoming ERC1155 transfers is to just not define these functions.

Listing 88:

```
116 function onERC1155Received(

127 function onERC1155BatchReceived(
```

3.89 CVF-89

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description These two variables could be merged into a single dynamic array of Proposal structures.

Listing 89:

```
39 uint256 public proposalCount;

76 mapping(uint256 => Proposal) public proposals;
```

3.90 CVF-90

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** GovernorAlpha.sol

Description The meaning of the keys in this mapping is unclear. Consider explaining in the documentation comment.

Listing 90:

```
79 mapping(uint256 => mapping(address => Receipt)) public receipts;
```

3.91 CVF-91

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** GovernorAlpha.sol

Description Events are usually named via nouns, such as "Proposal", "Vote", "ProposalCancellation" etc.

Listing 91:

```
92  event ProposalCreated(

103 event VoteCast(address voter, uint256 proposalId, bool support,
    ↪ uint256 votes);

106 event ProposalCanceled(uint256 id);

109 event ProposalQueued(uint256 id, uint256 eta);

112 event ProposalExecuted(uint256 id);

115 event UpdateQuorumVotes(uint96 newQuorumVotes);

118 event UpdateProposalThreshold(uint96 newProposalThreshold);

121 event UpdateVotingDelayBlocks(uint32 newVotingDelayBlocks);

124 event UpdateVotingPeriodBlocks(uint32 newVotingPeriodBlocks);
```

3.92 CVF-92

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description The proposal ID parameter should be indexed in all events.

Listing 92:

```
93      uint256 id,

103 event VoteCast(address voter, uint256 proposalId, bool support,
    ↪ uint256 votes);

106 event ProposalCanceled(uint256 id);

109 event ProposalQueued(uint256 id, uint256 eta);

112 event ProposalExecuted(uint256 id);
```

3.93 CVF-93

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description There are no range checks for these parameters. Consider adding necessary checks.

Listing 93:

```
127 uint96 quorumVotes_,
    uint96 proposalThreshold_,
    uint32 votingDelayBlocks_,
130 uint32 votingPeriodBlocks_,

133 uint256 minDelay_
```

3.94 CVF-94

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** GovernorAlpha.sol

Description This argument should have type "NoteInterface".

Listing 94:

```
131 address note_,
```

3.95 CVF-95

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description This variable is redundant, as accessing 'block.number' is cheaper than accessing a local variable.

Listing 95:

```
159 uint256 blockNumber = block.number;
```

3.96 CVF-96

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description It would be more reasonable to enforce blockNumber <= type(uint32).max, in order to make it safe to convert the block number to uint32.

Listing 96:

```
160 require(blockNumber > 0 && blockNumber < type(uint32).max);
```

3.97 CVF-97

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** GovernorAlpha.sol

Description Should be ">=".

Listing 97:

```
163 note.getPriorVotes(msg.sender, blockNumber - 1) >
    ↪ proposalThreshold,
```

3.98 CVF-98

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description The expression "targets.length" is calculated several times. Consider calculating once and reusing.

Listing 98:

```
167     targets.length == values.length && targets.length ==
          ↪ calldatas.length,

170 require(targets.length != 0, "GovernorAlpha::propose: must
    ↪ provide actions");

172     targets.length <= proposalMaxOperations,
```

3.99 CVF-99

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description This check would be simpler and cheaper in case a simple boolean flag would be stored for each proposer telling whether this proposer has an alive proposal. Storing last proposal ID is redundant and suboptimal.

Listing 99:

```
177 uint256 latestProposalId = latestProposalIds[msg.sender];
    if (latestProposalId != 0) {
        ProposalState proposersLatestProposalState = state(
          ↪ latestProposalId);
180     require(
            proposersLatestProposalState != ProposalState.Active,
            "GovernorAlpha::propose: one live proposal per proposer,
              ↪ found an already active proposal"
        );
        require(
            proposersLatestProposalState != ProposalState.Pending,
            "GovernorAlpha::propose: one live proposal per proposer,
              ↪ found an already pending proposal"
        );
    }
```

3.100 CVF-100

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** GovernorAlpha.sol

Description These two lines could be merged into one: uint256 newProposalId = ++proposalCount;

Listing 100:

```
191 uint256 newProposalId = proposalCount + 1;
    proposalCount = newProposalId;
```

3.101 CVF-101

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description Using "newProposalId" instead of "newProposal.id" would be cheaper.

Listing 101:

```
211 proposals[newProposal.id] = newProposal;
    latestProposalIds[newProposal.proposer] = newProposal.id;

215     newProposal.id,

223 return newProposal.id;
```

3.102 CVF-102

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description The hashes calculated here will be calculated again inside "scheduleBatch" and "executeBatch" calls.This is suboptimal. Consider refactoring to avoid this.

Listing 102:

```
251 bytes32 computedOperationHash = _computeHash(targets, values,
      ↪ calldatas, proposalId);

289 bytes32 computedOperationHash = _computeHash(targets, values,
      ↪ calldatas, proposalId);
```

3.103 CVF-103

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description This function is redundant, as it is very simple and is used only once. Consider removing it.

Listing 103:

```
263 function _scheduleBatch(

298 function _executeBatch(
```

3.104 CVF-104

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description This allows a proposer to cancel his proposal even if it already succeeded. This could be used for various attacks. A proposer shouldn't have any special rights on his proposals, as proposers could stay anonymous and in general are cannot be trusted.

Listing 104:

```
319 require(
320     msg.sender == guardian ||
          note.getPriorVotes(proposal.proposer, blockNumber - 1) <
            ↪ proposalThreshold,
      "GovernorAlpha::cancel: proposer above threshold"
);
```

3.105 CVF-105

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description This function is redundant as the "receipts" mapping is already public.

Listing 105:

```
335 function getReceipt(uint256 proposalId, address voter) public
      ↪ view returns (Receipt memory) {
```

3.106 CVF-106

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description These conditions are always true, as if either of them would be false, the function would return at line 357.

Listing 106:

```
361 proposal.forVotes > proposal.againstVotes &&
    proposal.forVotes > quorumVotes &&
```

3.107 CVF-107

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description This condition is always true here, as in case it is false, the function would return at like 355.

Listing 107:

```
363 blockNumber >= proposal.endBlock
```

3.108 CVF-108

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description The "Succeeded" state is returned even for queued proposals.

Listing 108:

```
365 return ProposalState.Succeeded;
```

3.109 CVF-109

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description This line is unreachable.

Listing 109:

```
367 return ProposalState.Queued;
```

3.110 CVF-110

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description Calculating the proposal state is more expensive than checking that a proposal is active. Probably not worth optimizing.

Listing 110:

```
411 state(proposalId) == ProposalState.Active,
```

3.111 CVF-111

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description This doesn't allow changing a casted vote. Could be dangerous in some situations, for example when a bug is found in a proposal that already gained much support, or when a duplicate proposals are discovered, both having much support, etc. Consider implementing an ability to revoke and change a vote.

Listing 111:

```
416 require(receipt.hasVoted == false, "GovernorAlpha::_castVote:
    ↪ voter already voted");
```

3.112 CVF-112

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description So the time window is the same for both, support and non-support votes. This could be abused. A powerful user may cast a number of support votes at a failing vote few seconds before the end of the voting, so opponents will not have enough time to case enough non support votes. Casting much non-support votes on an already failing proposal is basically waste of gas. Common approach is to make time window for support votes shorter than for non-support votes.

Listing 112:

```
419 if (support) {
420     proposal.forVotes = _add96(proposal.forVotes, votes);
} else {
    proposal.againstVotes = _add96(proposal.againstVotes, votes)
       ↪ ;
}
```

3.113 CVF-113

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description There are no range checks for the arguments. Consider adding necessary checks.

Listing 113:

```
438 function updateProposalThreshold(uint96 newProposalThreshold)
      ↪ external {

444 function updateVotingDelayBlocks(uint32 newVotingDelayBlocks)
      ↪ external {

450 function updateVotingPeriodBlocks(uint32 newVotingPeriodBlocks)
      ↪ external {
```

3.114 CVF-114

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernorAlpha.sol

Description These event are emitted even if the corresponding parameters didn't actually change.

Listing 114:

```
441 emit UpdateProposalThreshold(newProposalThreshold);

447 emit UpdateVotingDelayBlocks(newVotingDelayBlocks);

453 emit UpdateVotingPeriodBlocks(newVotingPeriodBlocks);
```

3.115 CVF-115

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernorAlpha.sol

Description This function could be simplified as: function _getChainId() private pure returns (uint256 chainId) { assembly { chainId := chainid() } }

Listing 115:

```
477 function _getChainId() private pure returns (uint256) {
        uint256 chainId;
        assembly {
480         chainId := chainid()
        }
        return chainId;
    }
```

3.116 CVF-116

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** GovernorAlpha.sol

Description This interface should be moved to a separate file named "NoteInterface".

Listing 116:

```
486 NoteInterface {
```

3.117 CVF-117

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Reservoir.sol

Description There is no range check for this argument. Consider adding necessary checks, e.g. require that the value is greater than zero.

Listing 117:

```
37 uint256 dripRate_,
```

3.118 CVF-118

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Reservoir.sol

Description This could be moved to the variable declaration.

Listing 118:

```
45 dripped = 0;
```

3.119 CVF-119

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Reservoir.sol

Description This function should log some event.

Listing 119:

```
51 function drip() public returns (uint256 amountToDrip) {
```

3.120 CVF-120

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Reservoir.sol

Description This check is redundant and could cause problems when calling this function from smart contracts before certain operations. Consider just doing nothing in case of empty reservoir, rather than reverting the transaction.

Listing 120:

```
53 require(reservoirBalance > 0, "Reservoir empty");
```

3.121 CVF-121

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NoteERC20.sol

Description Using uint96 for allowances and balances doesn't save storage space but makes the code less efficient. Consider using uint256 instead.

Listing 121:

```
28 mapping(address => mapping(address => uint96)) internal
     ↪ allowances;

31 mapping(address => uint96) internal balances;
```

3.122 CVF-122

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** NoteERC20.sol

Description These two variables could be merged into one: mapping(address => Checkpoint []) public checkpoints.

Listing 122:

```
43 mapping(address => mapping(uint32 => Checkpoint)) public
     ↪ checkpoints;

46 mapping(address => uint32) public numCheckpoints;
```

3.123 CVF-123

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NoteERC20.sol

Description These previous values are redundant, as they could be derived from the previous events.

Listing 123:

```
62 address indexed fromDelegate,

69 uint256 previousBalance,
```

3.124 CVF-124

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NoteERC20.sol

Description An address and the corresponding grant amount could be packed into a single 32-bytes word saving calldata space and allowing more initial accounts to be passed.

Listing 124:

```
84 address[] calldata initialAccounts,
   uint96[] calldata initialGrantAmount,
```

3.125 CVF-125

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NoteERC20.sol

Description The expression "initialAccounts[i]" is calculated several times. Consider calculating once and reusing.

Listing 125:

```
93 totalGrants = _add96(totalGrants, initialGrantAmount[i], "");
   balances[initialAccounts[i]] = initialGrantAmount[i];

96 emit Transfer(address(0), initialAccounts[i], initialGrantAmount
   ↪ [i]);
```

3.126 CVF-126

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** NoteERC20.sol

Description In case the same address is specified multiple times in the "initialAccounts' array, this will overwrite the already set value, thus, only the last grant amount of this address will be granted, however, all grant amounts will be counted in "totalGrants". This could lead to a situation then the total supply is inconsistent with the actual balances. Consider either requiring that "initialAccount[i]" is zero, before setting it, or using (a safe version of) "+=" instead of "=" here.

Listing 126:

```
94 balances[initialAccounts[i]] = initialGrantAmount[i];
```

3.127 CVF-127

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** NoteERC20.sol

Description This check doesn't guarantee that the sum of all the balances equals to the total supply, as the same address could be specified multiple times in the "initialAccounts" array.

Listing 127:

```
99 require(totalGrants == totalSupply);
```

3.128 CVF-128

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** NoteERC20.sol

Description This could cause problems with contracts that call "approve" with large values different from uint(-1). Consider treating all the values, that don't fit into uint96 as uint96(-1).

Listing 128:

```
122 amount = _safe96(rawAmount, "Note::approve: amount exceeds 96
    ↪ bits");
```

3.129 CVF-129

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NoteERC20.sol

Description So when a user calls "transferFrom" to transfer from himself or uses uint96(-1) allowance, the allowance is not decreased. This behavior makes the code more complicated, less efficient and is not fully compliant with ERC20 standard. Consider implementing more conventional behavior.

Listing 129:

```
164 if (spender != src && spenderAllowance != uint96(-1)) {
```

3.130 CVF-130

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NoteERC20.sol

Description Emitting the "Approval" event on "transferFrom" calls is not defined by the ERC-20 standard.

Listing 130:

```
173 emit Approval(src, spender, newAllowance);
```

3.131 CVF-131

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NoteERC20.sol

Description This check should be done earlier, before the expensive signature check.

Listing 131:

```
212 require(block.timestamp <= expiry, "Note::delegateBySig:
    ↪ signature expired");
```

3.132 CVF-132

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** NoteERC20.sol

Description The number of unclaimed votes is calculated for the current block number, not for the block number passed as an argument. Thus, the function called for the same account and block number, could return different values.

Listing 132:

```
270 getUnclaimedVotes(account),
```

3.133 CVF-133

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NoteERC20.sol

Description This event is emitted even if the delegatee has not been actually changed.

Listing 133:

```
291 emit DelegateChanged(delegator, currentDelegate, delegatee);
```

3.134 CVF-134

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NoteERC20.sol

Description This event is emitted even if the number of votes has not been actually changed.

Listing 134:

```
367 emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
```

3.135 CVF-135

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NoteERC20.sol

Description Consider using "type(uint32).max" syntax as it used in other places in the code.

Listing 135:

```
371 require(n < 2**32, errorMessage);
```

3.136 CVF-136

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** NoteERC20.sol

Description This could be simplified as: function _safe32(uint256 n, string memory errorMessage) private pure returns (uint32 r) { require ((r = uint32 (n)) == n, errorMessage); }

Listing 136:

```
371 require(n < 2**32, errorMessage);
    return uint32(n);

376 require(n < 2**96, errorMessage);
    return uint96(n);
```

3.137 CVF-137

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NoteERC20.sol

Description Consider using "type(uint96).max" syntax as it used in other places in the code.

Listing 137:

```
376 require(n < 2**96, errorMessage);
```

3.138 CVF-138

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NoteERC20.sol

Description This utility function appears in several files. Consider moving it into a low-level library. Also, this function could be simplified: function _getChainId() private pure returns (uint256 chainId) { assembly { chainId := chainid() } }

Listing 138:

```
399 function _getChainId() private pure returns (uint256) {
400     uint256 chainId;
        assembly {
            chainId := chainid()
        }
        return chainId;
    }
```

3.139 CVF-139

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Description The formula in the comment could be used to initialize the constant. No need to hardcode the precomputed value.

Listing 139:

```
18  // bytes4(keccak256("onERC1155Received(address,address,uint256,
        ↪ uint256,bytes)"))
    bytes4 internal constant ERC1155_ACCEPTED = 0xf23a6e61;
20  // bytes4(keccak256("onERC1155BatchReceived(address,address,
        ↪ uint256[],uint256[],bytes)"))
    bytes4 internal constant ERC1155_BATCH_ACCEPTED = 0xbc197c81;
```

3.140 CVF-140

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** ERC1155Action.sol

Description Should be "else if" for readability.

Listing 140:

```
79 if (assetType != Constants.FCASH_ASSET_TYPE) return 0;
```

3.141 CVF-141

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** ERC1155Action.sol

Description Should be "else return" for readability.

Listing 141:

```
81 return BitmapAssetsHandler.getifCashNotional(account, currencyId
   ↪ , maturity);
```

3.142 CVF-142

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Description It would be more efficient to decode the token ID into currency ID, maturity, and asset type once, rather than to encode every asset in a loop.

Listing 142:

```
92 TransferAssets.encodeAssetId(
       portfolio[i].currencyId,
       portfolio[i].maturity,
       portfolio[i].assetType
   ) == id
```

3.143 CVF-143

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Description The value "portfolio[i]" is calculated several times. Consider calculating once and reusing.

Listing 143:

```
93              portfolio[i].currencyId,
                portfolio[i].maturity,
                portfolio[i].assetType

97  ) return portfolio[i].notional;
```

3.144 CVF-144

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** ERC1155Action.sol

Description It is not checked, that the arrays have the same length. Consider adding an explicit check.

Listing 144:

```
208 function decodeToAssets(uint256[] calldata ids, uint256[]
    ↪ calldata amounts)
```

3.145 CVF-145

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** ERC1155Action.sol

Description This could be replaced with a simple assembly block:
assembly { sig := mload (add (data, 0x20)) }

Listing 145:

```
282 if (data.length >= 32) {
        // Method signature is not abi encoded so decode to bytes32
        ↪ first and take the first 4 bytes. This works
        // because all the methods we want to call below require
        ↪ more than 32 bytes in the calldata
        bytes32 tmp = abi.decode(data, (bytes32));
        sig = bytes4(tmp);
    }
```

3.146 CVF-146

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** ERC1155Action.sol

Description The "sig" variable remains uninitialized in case data.length < 32. Consider explicitly initializaing to zero in such a case.

Listing 146:

```
287 }
```

3.147 CVF-147

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidateCurrencyAction.sol

Description Events are usually named via nouns, such as "LocalCurrencyLiquidation" or "CollateralCurrencyLiquidation".

Listing 147:

```
15 event LiquidateLocalCurrency(

22 event LiquidateCollateralCurrency(
```

3.148 CVF-148

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrencyAction.sol

Description In order to be stateful, the function could calculate the settled state, but don't store it.

Listing 148:

```
32 /// @notice Calculates the net local currency required by the
   ↪ liquidator. This is a stateful method
/// because it may settle the liquidated account if required.
   ↪ However, it can be called using staticcall
/// off chain to determine the net local currency required
   ↪ before liquidating.
```

3.149 CVF-149

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** LiquidateCurrencyAction.sol

Description Even when called off-chain, static call would fail in case the called contract would try to modify the blockchain state.

Listing 149:

```
33 /// because it may settle the liquidated account if required.
   ↪ However, it can be called using staticcall
```

3.150 CVF-150

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** LiquidateCurrencyAction.sol

Description In most cases currency IDs are represented as uint16 while here they are represented as uint256. This makes code harder to read and more error-prone. Consider using the same type for currency IDs everywhere.

Listing 150:

```
 41 uint256 localCurrency,

 63 uint256 localCurrency,

117 uint256 localCurrency,
    uint256 collateralCurrency,

165 uint256 localCurrency,
    uint256 collateralCurrency,

236 uint256 localCurrency,

253 uint256 localCurrency,

295 uint256 localCurrency,
    uint256 collateralCurrency,

349 uint256 localCurrency,
350 uint256 collateralCurrency,
```

3.151 CVF-151

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** LiquidateCurrencyAction.sol

Description Overflow is possible here. Consider using safe cast.

Listing 151:

```
 96 uint16(localCurrency),

205 uint16(localCurrency),

243 uint16(localCurrency),
    uint16(collateralBalanceState.currencyId),
```

3.152 CVF-152

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidateCurrencyAction.sol

Description Consider giving descriptive names to the returned values for readability. This would also allow simplifying the code.

Listing 152:

```
124 int256,
    int256,
    int256

174 int256,
    int256,
    int256

258 int256,
    BalanceState memory,
260 PortfolioState memory,
    AccountContext memory,
    MarketParameters[] memory markets

302 int256,
    BalanceState memory,
    PortfolioState memory,
    AccountContext memory,
    MarketParameters[] memory markets
```

3.153 CVF-153

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidateCurrencyAction.sol

Description Delegating event emitting to a function is a bad practice as it makes it harder to find where events are emitted.

Listing 153:

```
203 _emitCollateralEvent(
        liquidateAccount,
        uint16(localCurrency),
        localAssetCashFromLiquidator,
        collateralBalanceState
    );
```

3.154 CVF-154

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrencyAction.sol

Description This function is redundant, as it is very simple and is used only once.

Listing 154:

```
234 function _emitCollateralEvent(
```

3.155 CVF-155

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidateCurrencyAction.sol

Description This like introduces an non-obvious side effect to the function and should probably be moved to the calling code.

Listing 155:

```
376 liquidatorContext.setAccountContext(msg.sender);
```

3.156 CVF-156

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** LiquidateCurrencyAction.sol

Description This could be simplified as: return collateralBalanceState.netAssetTransferInternalPrecision.sub (collateralBalanceState.netCashChange);

Listing 156:

```
385 collateralBalanceState.netCashChange.neg().add(
        collateralBalanceState.netAssetTransferInternalPrecision
```

3.157 CVF-157

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description As this is not a standard function anyway, consider returning a signed integer rather than reverting on negative numbers. A caller may always do the check and revert if necessary.

Listing 157:

```
58 require(nTokenBalance >= 0); // dev: negative nToken balance
   return uint256(nTokenBalance);
```

3.158 CVF-158

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenAction.sol

Description Should be "else return" for readability.

Listing 158:

```
75 return nTokenAllowance[owner][spender][currencyId];
```

3.159 CVF-159

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** nTokenAction.sol

Description This code makes it impossible to revoke or modify an existing allowance.

Listing 159:

```
92  uint256 allowance = nTokenAllowance[owner][spender][currencyId];
    require(allowance == 0, "Allowance not zero");

165 uint256 allowance = nTokenWhitelist[msg.sender][spender];
    require(allowance == 0, "Allowance not zero");
```

3.160 CVF-160

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description The function always return true. Consider not returning any value at all.

Listing 160:

```
96  return true;

171 return true;

353 return true;
```

3.161 CVF-161

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description The total amount of tokens that could be transferred is basically the whitelisted amount plus allowed amount, however, an particular invocation of the "nTokenTransferFrom" function may reduce only one of these two accounts. This is inconvenient. For example, a user has 100 whitelisted tokens and 100 allowed tokens, thus he may transfer 200 tokens in total, but has to call "transferFrom" twice. The first call should transfer exactly the whitelisted amount. Consider implementing an ability to transfer both, whitelisted and allowed tokens in one invocation.

Listing 161:

```
136 if (allowance > 0) {
        // This whitelist allowance supersedes any specific
            ↪ allowances
        require(allowance >= amount, "Insufficient allowance");
        allowance = allowance.sub(amount);
140     nTokenWhitelist[from][spender] = allowance;
    } else {
        // This is the specific allowance for the nToken.
        allowance = nTokenAllowance[from][spender][currencyId];
        require(allowance >= amount, "Insufficient allowance");
        allowance = allowance.sub(amount);
        nTokenAllowance[from][spender][currencyId] = allowance;
    }
```

3.162 CVF-162

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description Now, in case both, whitelisted and approved allowances exists, the whitelisted allowance is being spent first. This is bad for users, as whitelisted allowance is more valuable (it could be used for different tokens). Consider spending approved allowance first.

Listing 162:

```
137 // This whitelist allowance supersedes any specific allowances
    require(allowance >= amount, "Insufficient allowance");
    allowance = allowance.sub(amount);
140 nTokenWhitelist[from][spender] = allowance;
```

3.163 CVF-163

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** nTokenAction.sol

Description The variable is not initialized. Consider explicitly initializing to zero.

Listing 163:

```
180 uint256 totalIncentivesClaimed;
```

3.164 CVF-164

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenAction.sol

Description This could be simplified to: totalIncentivesClaimed = BalanceHandler.claimIncentivesManual(balanceState, account); as the "totalIncentivesClaimed" variable is guaranteed to be zero here.

Listing 164:

```
186 totalIncentivesClaimed = totalIncentivesClaimed.add(
        BalanceHandler.claimIncentivesManual(balanceState, account)
    );
```

3.165 CVF-165

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenAction.sol

Description This could be simplified as: currencies «= 16;

Listing 165:

```
203 currencies = currencies << 16;
```

3.166 CVF-166

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description This variable is redundant. Just us "totalIncentivesClaimable" instead.

Listing 166:

```
229 uint256 incentivesToClaim,
```

3.167 CVF-167

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenAction.sol

Description This could be simplified to: totalIncentivesClaimable = incentivesToClaim; as the "totalIncentivesClaimable" variable is guaranteed to be zero here.

Listing 167:

```
238 totalIncentivesClaimable = totalIncentivesClaimable.add(
    ↪ incentivesToClaim);
```

3.168 CVF-168

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenAction.sol

Description This could be simplified as: currencies «= 16;

Listing 168:

```
262 currencies = currencies << 16;
```

3.169 CVF-169

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** nTokenAction.sol

Description These braces are redundant. Are they here to prevent "stack too deep" error?

Listing 169:

```
321 {
332 }
```

3.170 CVF-170

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TradingAction.sol

Description Both parameters should probably be indexed.

Listing 170:

```
25 event BatchTradeExecution(address account, uint16 currencyId);
```

3.171 CVF-171

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TradingAction.sol

Description The first two parameters should probably be indexed.

Listing 171:

```
26 event SettledCashDebt(address settledAccount, uint16 currencyId,
    ↪ int256 amountToSettleAsset);
```

3.172 CVF-172

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TradingAction.sol

Description The first two parameters should probably be indexed.

Listing 172:

```
27 event nTokenResidualPurchase(uint16 currencyId, uint40 maturity,
    ↪ int256 fCashAmountToPurchase);
```

3.173 CVF-173

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description One two fields are used from this parameter: “bitmapCurrencyId” and “nextSettleTime”, while all other fields just waste space in calldata. Consider passing just these two needed values as separate arguments.

Listing 173:

```
42 AccountContext calldata accountContext,
```

3.174 CVF-174

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** TradingAction.sol

Description As this is an external function, consider describing the encoding used in this array, or giving a reference to where this encoding is described. Currently there are no clues where to find the encoding documentation.

Listing 174:

```
43 bytes32[] calldata trades

89 bytes32[] calldata trades
```

3.175 CVF-175

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** TradingAction.sol

Description The semantics of the returned values is unclear. Consider giving descriptive named to the returned values and/or adding a documentation comment.

Listing 175:

```
44 ) external returns (int256, bool) {

90 ) external returns (PortfolioState memory, int256) {
```

3.176 CVF-176

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description The expression "accountContext.bitmapCurrencyId" is calculated several times. Consider calculating once and reusing.

Listing 176:

```
46      CashGroup.buildCashGroupStateful(accountContext.
            ↪ bitmapCurrencyId);

49      BitmapAssetsHandler.getAssetsBitmap(account, accountContext.
            ↪ bitmapCurrencyId);

65          accountContext.bitmapCurrencyId,

77  BitmapAssetsHandler.setAssetsBitmap(account, accountContext.
        ↪ bitmapCurrencyId, ifCashBitmap);
    BalanceHandler.incrementFeeToReserve(accountContext.
        ↪ bitmapCurrencyId, c.totalFee);

80  emit BatchTradeExecution(account, uint16(accountContext.
        ↪ bitmapCurrencyId));
```

3.177 CVF-177

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TradingAction.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero for readability.

Listing 177:

```
54  for (uint256 i; i < trades.length; i++) {

96  for (uint256 i; i < trades.length; i++) {
```

3.178 CVF-178

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description Conditional statements are expensive. Probably the following variant would be cheaper: didIncurDebt |= c.fCashAmount < 0;

Listing 178:

```
72 if (c.fCashAmount < 0) didIncurDebt = true;
```

3.179 CVF-179

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** TradingAction.sol

Description Overflow is possible when converting the bitmap currency ID to uint16. Consider using safe conversion.

Listing 179:

```
80 emit BatchTradeExecution(account, uint16(accountContext.
   ↪ bitmapCurrencyId));
```

3.180 CVF-180

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** TradingAction.sol

Description This comment says nothing about what this function actually does. Consider describing.

Listing 180:

```
134 /// @dev used to clear the stack
```

3.181 CVF-181

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description Conversion to bytes1 applies mask and conversion to uint8 performs a shift, so there are three operations: shift+mask+shift, while only two operations are actually needed. Consider implementing like this:
uint256 marketIndex = (uint256 (trade) » 240) & 0xff;

Listing 181:

```
192 uint256 marketIndex = uint256(uint8(bytes1(trade << 8)));

259 uint256 marketIndex = uint256(uint8(bytes1(trade << 8)));
```

3.182 CVF-182

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description Conversion to bytes11 applies mask and conversion to uint88 performs a shift, so there are three operations: shift+mask+shift, while only two operations are actually needed. Consider implementing like this:
cashAmount = int256 ((uint256 (trade) » 152) & type(uint88).max);

Listing 182:

```
199     cashAmount = int256(uint88(bytes11(trade << 16)));

212     tokens = int256(uint88(bytes11(trade << 16)));

262 int256 fCashAmount = int256(uint88(bytes11(trade << 16)));
```

3.183 CVF-183

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description The same expression is calculated on both, "then" and "else" branches. Consider calculating it once before the conditional statement and storing the result in a local variable.

Listing 183:

```
199 cashAmount = int256(uint88(bytes11(trade << 16)));

212 tokens = int256(uint88(bytes11(trade << 16)));
```

3.184 CVF-184

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TradingAction.sol

Description It would be more logical to reorder these lines: require (netCash > 0); cashAmount = netCash;

Listing 184:

```
205 cashAmount = netCash;
    require(cashAmount > 0, "Invalid cash roll");
```

3.185 CVF-185

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TradingAction.sol

Description These lines could be optimized as: uint256 minImpliedRate = (uint256(trade) » 120) & type(uint32).max); uint256 minImpliedRate = (uint256(trade) » 88) & type(uint32).max);

Listing 185:

```
218     uint256 minImpliedRate = uint256(uint32(bytes4(trade << 104)
          ↪ ));
        uint256 maxImpliedRate = uint256(uint32(bytes4(trade << 136)
          ↪ ));

274 uint256 rateLimit = uint256(uint32(bytes4(trade << 104)));
```

3.186 CVF-186

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TradingAction.sol

Description The brackets are redundant here.

Listing 186:

```
242 return (cashAmount);
```

3.187 CVF-187

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** TradingAction.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or adding a documentation comment.

Listing 187:

```
254 int256 ,
    int256 ,
    int256

296 uint256 ,
    int256 ,
    int256

381 uint256 ,
    int256 ,
    int256
```

3.188 CVF-188

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TradingAction.sol

Description These lines could be optimized as: address counterparty = address (uint20 ((uint256 (trade) » 88) & type(uint20).max)); int256 amountToSettleAsset; int256 amount-ToSettleAsset = int88 (uint88 (uint256 (trade) & type (uint88).max));

Listing 188:

```
301 address counterparty = address(bytes20(trade << 8));
    int256 amountToSettleAsset = int256(int88(bytes11(trade << 168))
    ↪ );
```

3.189 CVF-189

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** TradingAction.sol

Description The variable name is inconsistent with how its value is set. Consider renaming to 'quarterMaturity'.

Listing 189:

```
324 uint256 threeMonthMaturity = DateTime.getReferenceTime(blockTime
    ↪ ) + Constants.QUARTER;
```

3.190 CVF-190

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TradingAction.sol

Description The expression "cashGroup.currencyId" is calculated twice. Consider calculating once and reusing.

Listing 190:

```
333     assets[0].currencyId = cashGroup.currencyId;

345 emit SettledCashDebt(counterparty, uint16(cashGroup.currencyId),
    ↪  amountToSettleAsset);
```

3.191 CVF-191

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** TradingAction.sol

Description Consider renaming the argument to 'quarterMaturity'.

Listing 191:

```
353 uint256 threeMonthMaturity,
```

3.192 CVF-192

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** TradingAction.sol

Description The semantics of the returned value is unclear. Consider giving a descriptive name to it and/or adding a documentation comment.

Listing 192:

```
356 ) private view returns (int256) {
```

3.193 CVF-193

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TradingAction.sol

Description These lines could be optimized as: uint256 maturity = (uint256 (trade) » 216) & type (uint32).max; int256 fCashAmountToPurchase = int88 (uint88 ((uint256 (trade) » 128) & type (uint88).max))

Listing 193:

```
386 uint256 maturity = uint256(uint32(bytes4(trade << 8)));
```

3.194 CVF-194

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** TradingAction.sol

Description Usually, the message in "require" statement describes the failure condition, not the successful one. Here the failure condition is a valid (or, in other words, non idiosyncratic) maturity. So the message should be "Non idiosyncratic maturity'".

Listing 194:

```
393 "Invalid maturity"
```

3.195 CVF-195

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TradingAction.sol

Description The 3600 value could be rendered as "1 hours".

Listing 195:

```
411 uint256(uint8(parameters[Constants.RESIDUAL_PURCHASE_TIME_BUFFER
    ↪ ])) * 3600
```

3.196 CVF-196

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TradingAction.sol

Description There should be a named constant for 10 * BASIS_POINT.

Listing 196:

```
464 10 *
Constants.BASIS_POINT;
```

3.197 CVF-197

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AccountAction.sol

Description There is no explicit check in the code that ensures that the account doesn't have any assets. Consider either adding such check explicitly or clarifying in the comment, why such check is not necessary.

Listing 197:

```
16 /// @notice Enables a bitmap currency for msg.sender, account
    ↪ cannot have any assets when this call
```

3.198 CVF-198

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountAction.sol

Description This variable is redundant, as "msg.sender" is cheaper to access than a local variable.

Listing 198:

```
23 address account = msg.sender;
```

3.199 CVF-199

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountAction.sol

Description This function has common code with the "_settleAccountIfRequiredAndFinalize" function. Consider refactoring to eliminate code duplication.

Listing 199:

```
34 function settleAccount(address account) external {
```

3.200 CVF-200

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountAction.sol

Description In case the account doesn't need to be settled, the function silently does nothing. Consider reverting in such case, or returning a flag telling whether the account as actually settled.

Listing 200:

```
36 if (accountContext.mustSettleAssets()) {
```

3.201 CVF-201

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountAction.sol

Description It is a bad practice to rely on checks performed in the called functions, as such checks could be removed of modified. Consider using a safe cast here.

Listing 201:

```
63 // Int conversion overflow check done inside this method call

99 // Int conversion overflow check done inside this method call,
   ↪ useCashBalance is set to false. msg.sender
```

3.202 CVF-202

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AccountAction.sol

Description The comment is not accurate, as zero value is also forbidden.

Listing 202:

```
72 require(assetTokensReceivedInternal > 0); // dev: asset tokens
   ↪ negative

113 require(assetTokensReceivedInternal > 0); // dev: asset tokens
   ↪ negative
```

3.203 CVF-203

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** AccountAction.sol

Description Overflow is possible when converting to int256.

Listing 203:

```
143 balanceState.netAssetTransferInternalPrecision = int256(
   ↪ amountInternalPrecision).neg();
```

3.204 CVF-204

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountAction.sol

Description The code flow looks like it is always executed, while it is executed only when "accountContext.mustSettleAssets()" is false. Consider putting the rest of the function into explicit "else" branch.

Listing 204:

```
163 }
```

3.205 CVF-205

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description This function looks redundant, as it is equivalent to the "batchBalanceAndTradeAction" function call with no trades. Consider removing this function.

Listing 205:

```
26 function batchBalanceAction(address account, BalanceAction[]
   ↪ calldata actions)
```

3.206 CVF-206

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description It is unclear where the 'msg.value' is actually consumed, and where it is guaranteed that the value will be consumed once and only once. Consider refactoring the code to make this more clear, or add a comment explaining this.

Listing 206:

```
28 payable

76 payable
```

3.207 CVF-207

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BatchAction.sol

Description The variable is used without being explicitly initialized. Consider explicitly initializing to zero.

Listing 207:

```
36 uint256 settleAmountIndex;

45         settleAmountIndex,

86 uint256 settleAmountIndex;

94         settleAmountIndex,
```

3.208 CVF-208

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BatchAction.sol

Description The variable "i" is uninitialized. Explicit initialization to zero would make code easier to read.

Listing 208:

```
38 for (uint256 i; i < actions.length; i++) {

88 for (uint256 i; i < actions.length; i++) {
```

3.209 CVF-209

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BatchAction.sol

Description More convectional way to ensure monotonicity is to store the previous currencyId in a local variable. The initial value for this variable could be -1, thus the conditional operation wouldn't be necessary.

Listing 209:

```
39 if (i > 0) {
40     require(actions[i].currencyId > actions[i - 1].currencyId, "
       ↪ Unsorted actions");
}

89 if (i > 0) {
90     require(actions[i].currencyId > actions[i - 1].currencyId, "
       ↪ Unsorted actions");
}
```

3.210 CVF-210

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description The offset of "actions[i]" is calculated multiple times. Consider calculating once and reusing: BalanceAction calldata action = actions [i];

Listing 210:

```
40  require(actions[i].currencyId > actions[i - 1].currencyId, "
      ↪ Unsorted actions");

46  actions[i].currencyId,

50  actions[i].actionType,
    actions[i].depositActionAmount

58  actions[i].withdrawAmountInternalPrecision,
    actions[i].withdrawEntireCashBalance,
60  actions[i].redeemToUnderlying

90  require(actions[i].currencyId > actions[i - 1].currencyId, "
      ↪ Unsorted actions");

95  actions[i].currencyId,

99  actions[i].actionType,
100 actions[i].depositActionAmount

114         actions[i].trades

124         actions[i].currencyId,

126         actions[i].trades

139 actions[i].withdrawAmountInternalPrecision,
140 actions[i].withdrawEntireCashBalance,
    actions[i].redeemToUnderlying
```

3.211 CVF-211

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description The constant should go first to make evaluation faster.

Listing 211:

```
117 accountContext.hasDebt = accountContext.hasDebt | Constants.
    ↪ HAS_ASSET_DEBT;
```

3.212 CVF-212

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** BatchAction.sol

Description The state is written here even if no trades were executed. Consider setting a flag when a trade is executed, and after the loop write the state only in case the flag is true.

Listing 212:

```
146 accountContext.storeAssetsAndUpdateContext(account,
    ↪ portfolioState, false);
```

3.213 CVF-213

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** BatchAction.sol

Description In is not guaranteed here that settleAmounts[settleAmountIndex].currencyId == currencyId, so netCashChange value for a wrong currency could be used and updated here.

Listing 213:

```
177 balanceState.netCashChange = settleAmounts[settleAmountIndex].
    ↪ netCashChange;

179 settleAmounts[settleAmountIndex].netCashChange = 0;
```

3.214 CVF-214

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** BatchAction.sol

Description Overflow is possible here. Consider using safe cast.

Listing 214:

```
285 int256 withdrawAmount = int256(withdrawAmountInternalPrecision);
```

3.215 CVF-215

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description The code below this line looks like it is always executed, while it is executed only when accountContext.mustSettleAssets() is false. Consider putting the rest of the function explicitly into and "else" branch.

Listing 215:

```
344 }
```

```
362 }
```

3.216 CVF-216

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** GovernanceAction.sol

Description This contract should probably be turned into a library.

Listing 216:

```
16 GovernanceAction is StorageLayoutV1, NotionalGovernance {
```

3.217 CVF-217

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GgovernanceAction.sol

Description This check is redundant, as it is still possible to transfer ownership to a dead address.

Listing 217:

```
26 require(newOwner != address(0), "Ownable: new owner is the zero
   ↪ address");
```

3.218 CVF-218

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernanceAction.sol

Description The event is logged even if the new owner is the same as the old one.

Listing 218:

```
27 emit OwnershipTransferred(owner, newOwner);
```

3.219 CVF-219

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** GovernanceAction.sol

Description This function should return the ID of just listed currency.

Listing 219:

```
40 function listCurrency(
```

3.220 CVF-220

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** GovernanceAction.sol

Description The type or asset oracle parameters should be "AssetRAteOracle" rather than just "address".

Listing 220:

```
43      address rateOracle,

85      address assetRateOracle,

223 function updateAssetRate(uint16 currencyId, address rateOracle)
    ↪ external override onlyOwner {

238     address rateOracle,

277 function _updateAssetRate(uint256 currencyId, address rateOracle
    ↪ ) internal {

313     address rateOracle,
```

3.221 CVF-221

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernanceAction.sol

Description There are no range checks for these parameters, while the documentation comment above describes valid ranges for them. Consider adding explicit checks.

Listing 221:

```
45 uint8 buffer,
   uint8 haircut,
   uint8 liquidationDiscount
```

3.222 CVF-222

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description Representing buffers, haircuts, discounts etc as integer number of percents is very limiting. It does allow values like 2% and 3%, but not 2.5%. Consider using basic points (1/10000) or even finer units, such as WADs (10^18). Alternatively, consider using 64.64 binary fixed point numbers.

Listing 222:

```
 45 uint8 buffer,
    uint8 haircut,
    uint8 liquidationDiscount

187 uint8 residualPurchaseIncentive10BPS,
    uint8 pvHaircutPercentage,
    uint8 residualPurchaseTimeBufferHours,
190 uint8 cashWithholdingBuffer10BPS,
    uint8 liquidationHaircutPercentage

240 uint8 buffer,
    uint8 haircut,
    uint8 liquidationDiscount

315 uint8 buffer,
    uint8 haircut,
    uint8 liquidationDiscount
```

3.223 CVF-223

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** GovernanceAction.sol

Description This could be simplified as: uint16 currencyId = ++maxCurrencyId;

Listing 223:

```
49 uint16 currencyId = maxCurrencyId + 1;
50 // Set the new max currency id
   maxCurrencyId = currencyId;
```

3.224 CVF-224

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** GovernanceAction.sol

Description This also allows a situation when the token type is "Ether" but the token address is not zero. Consider explicitly forbidding such case.

Listing 224:

```
61 underlyingToken.tokenAddress != address(0) ||
   // Ether has a token address of zero
   underlyingToken.tokenType == TokenType.Ether
```

3.225 CVF-225

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** GovernanceAction.sol

Description The event should probably contain more details, such as asset and underlying token addresses. Logging only the currency ID is almost useless.

Listing 225:

```
71 emit ListCurrency(currencyId);
```

3.226 CVF-226

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernanceAction.sol

Description There is no explicit check for validity of this argument. Consider adding such check.

Listing 226:

```
 84     uint16 currencyId,

119     uint16 currencyId,

137     uint16 currencyId,

151 function updateIncentiveEmissionRate(uint16 currencyId, uint32
    ↪ newEmissionRate)

186     uint16 currencyId,

211 function updateCashGroup(uint16 currencyId, CashGroupSettings
    ↪ calldata cashGroup)

223 function updateAssetRate(uint16 currencyId, address rateOracle)
    ↪ external override onlyOwner {

237     uint16 currencyId,
```

3.227 CVF-227

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** GovernanceAction.sol

Description There is a high-level type-safe syntax for deploying smart contracts via CEATE2 opcode: https://docs.soliditylang.org/en/v0.7.0/control-structures.html#salted-contract-creations-create2

Listing 227:

```
94  address nTokenAddress =
        Create2.deploy(
            0,
            bytes32(uint256(currencyId)),
            abi.encodePacked(
                type(nTokenERC20Proxy).creationCode,
100             abi.encode(address(this), currencyId, underlyingName
                    ↪ , underlyingSymbol)
            )
        );
```

3.228 CVF-228

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** GovernanceAction.sol

Description There are no range checks for these arguments. Consider adding relevant checks.

Listing 228:

```
120 uint32[] calldata depositShares,
    uint32[] calldata leverageThresholds

138 uint32[] calldata rateAnchors,
    uint32[] calldata proportions
```

3.229 CVF-229

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** GovernanceAction.sol

Description This event should probably contain more parameters. Currently it is almost useless.

Listing 229:

```
124 emit UpdateDepositParameters(currencyId);
142 emit UpdateInitializationParameters(currencyId);
204 emit UpdateTokenCollateralParameters(currencyId);
274 emit UpdateCashGroup(uint16(currencyId));
347 emit UpdateETHRate(uint16(currencyId));
```

3.230 CVF-230

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description The event is logged even when the new rate is the same as the current one.

Listing 230:

```
162 emit UpdateIncentiveEmissionRate(currencyId, newEmissionRate);
```

3.231 CVF-231

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description This check looks redundant. It doesn't guarantee anything. Setting a single-owner wallet as an operator would not be much different from setting EOA.

Listing 231:

```
257 uint256 codeSize;
    assembly {
        codeSize := extcodesize(operator)
260 }
    // Sanity check to ensure that operator is a contract, not an
       ↪ EOA
    require(codeSize > 0, "Operator must be a contract");
```

3.232 CVF-232

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** GovernanceAction.sol

Description Here ‘currencyId‘ is expanded to u256 before truncating it back in an event, whereas the function does not use the extended representation. Consider using longer types when they are really needed, or use u256 everywhere.

Listing 232:

```
268 function _updateCashGroup(uint256 currencyId, CashGroupSettings
    ↪ calldata cashGroup) internal {

274     emit UpdateCashGroup(uint16(currencyId));

277 function _updateAssetRate(uint256 currencyId, address rateOracle
    ↪ ) internal {

308     emit UpdateAssetRate(uint16(currencyId));

312     uint256 currencyId,

347     emit UpdateETHRate(uint16(currencyId));
```

3.233 CVF-233

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description This check repeats several times. Consider extracting it to a function or a modifier.

Listing 233:

```
269 require(currencyId != 0, "G: invalid currency id");
270 require(currencyId <= maxCurrencyId, "G: invalid currency id");

278 require(currencyId != 0, "G: invalid currency id");
    require(currencyId <= maxCurrencyId, "G: invalid currency id");

319 require(currencyId != 0, "G: invalid currency id");
320 require(currencyId <= maxCurrencyId, "G: invalid currency id");
```

3.234 CVF-234

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description The code behind this line looks like it is always executed, while it is executed only when the rate oracle is not zero. Consider putting hte rest of the function into an explicit "else" branch.

Listing 234:

```
288 }
```

3.235 CVF-235

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** InitializeMarketsAction.sol

Description Events are usually named via nouns, such as "MarketsInitialization" or "InitializedMarkets".

Listing 235:

```
39 event MarketsInitialized(uint16 currencyId);
```

3.236 CVF-236

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** InitializeMarketsAction.sol

Description There is not validity check for the "currencyId" argument. Consider adding such check.

Listing 236:

```
48 function _getGovernanceParameters(uint256 currencyId, uint256
   ↪ maxMarketIndex)

103     uint256 currencyId,

136     uint256 currencyId,

592     uint256 currencyId,
```

3.237 CVF-237

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** InitializeMarketsAction.sol

Description The semantics of the returned value is unclear. Consider giving a descriptive name to the returned value and/or adding a documentation comment.

Listing 237:

```
69 returns (bytes32)
```

3.238 CVF-238

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** InitializeMarketsAction.sol

Description There is no length check for this argument, so it is not guaranteed that all the previous markets will fit into the array or that the array will be fully populated. Also, there is no reliable way to know how many elements of the array were actually populated. Consider returning the actual number of populated elements, also consider stopping the look in case the end of the array reached. Also, consider accepting an additional parameter specifying the offset inside nToken.portfoilioState.storedAssets to start with.

Listing 238:

```
106 MarketParameters[] memory previousMarkets
```

3.239 CVF-239

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description The address of "nToken.portfolioState.storedAssets" is calculated multiple times. Consider calculating once and reusing.

Listing 239:

```
117 for (uint256 i = 1; i < nToken.portfolioState.storedAssets.
    ↪ length; i++) {

121             nToken.portfolioState.storedAssets[i].maturity,
```

3.240 CVF-240

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** InitializeMarketsAction.sol

Description Using different formants for different values makes the code harder to read and is error-prone. Consider using the same format everywhere, such as WAD or 64.64.

Listing 240:

```
181 // This buffer is denominated in 10 basis point increments. It
    ↪ is used to shift the withholding rate to ensure
// that sufficient cash is withheld for negative fCash balances.
```

3.241 CVF-241

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** InitializeMarketsAction.sol

Description Underflow is possible here. Consider making an explicit range check for blockTime.

Listing 241:

```
206 blockTime - Constants.QUARTER
```

3.242 CVF-242

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description This is equivalent to: nToken.cashBalance = assetCashWithholding;

Listing 242:

```
260 nToken.cashBalance = nToken.cashBalance.subNoNeg(
    ↪ netAssetCashAvailable);
```

3.243 CVF-243

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** InitializeMarketsAction.sol

Description This could be simplified as: int128 expValue = ABDKMath64x64.divi (exchangeRate.sub(rateAnchor).mul(rateScalar), 1000000000);

Listing 243:

```
317 int128 expValue = ABDKMath64x64.fromInt(exchangeRate.sub(
    ↪ rateAnchor).mul(rateScalar));
// Scale this back to a decimal in abdk
expValue = ABDKMath64x64.div(expValue, Constants.
    ↪ RATE_PRECISION_64x64);
```

3.244 CVF-244

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** InitializeMarketsAction.sol

Description This could be simplified as: return ABDKMath64x64.muli (proportion, 1000000000);

Listing 244:

```
326 // Scale this back to 1e9 precision
proportion = ABDKMath64x64.mul(proportion, Constants.
    ↪ RATE_PRECISION_64x64);

return ABDKMath64x64.toInt(proportion);
```

3.245 CVF-245

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** InitializeMarketsAction.sol

Description This code could be simplified by calculating the interpolation as a signed number and then returning the maximum of the calculated value and zero.

Listing 245:

```
348 if (longRate >= shortRate) {

355 } else {
```

3.246 CVF-246

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** InitializeMarketsAction.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 246:

```
430 for (uint256 i; i < nToken.cashGroup.maxMarketIndex; i++) {
```

3.247 CVF-247

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description The expression "DateTime.getReferenceTime(blockTime)" is calculated on every loop iteration. Consider calculating once and reusing.

Listing 247:

```
432 newMarket.maturity = DateTime.getReferenceTime(blockTime).add(

499             DateTime.getReferenceTime(blockTime)

510             DateTime.getReferenceTime(blockTime).add(DateTime.
                    ↪ getTradedMarket(i));
```

3.248 CVF-248

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description The same formula is calculated in several different places. Consider extracting it to a utility function.

Listing 248:

```
468 int256 fCashAmount =
        underlyingCashToMarket.mul(parameters.proportions[i]).div(
470         Constants.RATE_PRECISION.sub(parameters.proportions[i])
        );

536     newMarket.totalfCash = underlyingCashToMarket.mul(proportion
        ↪ ).div(
          Constants.RATE_PRECISION.sub(proportion)
        );

550     newMarket.totalfCash = underlyingCashToMarket.mul(proportion
        ↪ ).div(
          Constants.RATE_PRECISION.sub(proportion)
        );
```

3.249 CVF-249

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description The value “parameters.propertions[i]” is calculated twice. Consider calculating once and reusing.

Listing 249:

```
469 underlyingCashToMarket.mul(parameters.proportions[i]).div(
470     Constants.RATE_PRECISION.sub(parameters.proportions[i])
```

3.250 CVF-250

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** InitializeMarketsAction.sol

Description Overflow is possible here. Consider using safe cast. The comment is confusing, as this line will never revert.

Listing 250:

```
478 int256(parameters.rateAnchors[i]), // Will revert on out of
    ↪ bounds error here
```

3.251 CVF-251

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description The value "DateTime.getTradeMarket(i)" was already calculated at the previous loop iteration. Consider reusing it.

Listing 251:

```
510 DateTime.getReferenceTime(blockTime).add(DateTime.
    ↪ getTradedMarket(i));
```

3.252 CVF-252

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description The expression "parameters.leverageThresholds[i]" is calculated twice. Consider calculating once and reusing.

Listing 252:

```
534 if (proportion > parameters.leverageThresholds[i]) {
      proportion = parameters.leverageThresholds[i];
```

3.253 CVF-253

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** InitializeMarketsAction.sol

Description The calculated 'newMarket.totalfCash' value could be zero even if proportion is not zero. Consider changing this line to: if (newMarket.totalfCash < 1) newMarket.totalfCash = 1;

Listing 253:

```
557 if (proportion == 0) newMarket.totalfCash = 1;
```

3.254 CVF-254

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** InitializeMarketsAction.sol

Description This event should probably have more parameters, for example the maturities of the newly initialized markets. With only currency ID the event is almost useless.

Listing 254:

```
587 emit MarketsInitialized(uint16(currencyId));
```

3.255 CVF-255

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCashAction.sol

Description The word "event" is redundant in an event name. Also, events are usually named via nouns, like "CashLiquidation".

Listing 255:

```
14 event LiquidatefCashEvent(
```

3.256 CVF-256

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** LiquidatefCashAction.sol

Description In most cases currency IDs are represented as uint16 values, but here uint256 is used. This makes the code harder to read and more error prone. Consider using the same type for currency IDs everywhere.

Listing 256:

```
 36 uint256 localCurrency,

 64 uint256 localCurrency,

112 uint256 localCurrency,
    uint256 fCashCurrency,

143 uint256 localCurrency,
    uint256 fCashCurrency,

184 uint256 localCurrency,

210 uint256 localCurrency,
    uint256 fCashCurrency,
```

3.257 CVF-257

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCashAction.sol

Description Consider giving descriptive names to the returned values for readability.

Listing 257:

```
 39 ) external returns (int256[] memory, int256) {

 67 ) external returns (int256[] memory, int256) {

116 ) external returns (int256[] memory, int256) {

147 ) external returns (int256[] memory, int256) {
```

3.258 CVF-258

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCashAction.sol

Description Probably a more descriptive name would make the code easier to read.

Listing 258:

```
150 LiquidatefCash.fCashContext memory c =
189 LiquidatefCash.fCashContext memory c;
```

3.259 CVF-259

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenMintAction.sol

Description This check should be done earlier to save gas on failure.

Listing 259:

```
52 require(tokensToMint >= 0, "Invalid token amount");
```

3.260 CVF-260

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenMintAction.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to the returned values and/or adding a documentation comment.

Listing 260:

```
65 ) internal view returns (int256, bytes32) {
```

3.261 CVF-261

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenMintAction.sol

Description Should be "else return" for readability.

Listing 261:

```
84 return (amountToDepositInternal.mul(nToken.totalSupply).div(
   ↪ assetCashPV), ifCashBitmap);
```

3.262 CVF-262

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** nTokenMintAction.sol

Description As "i" variable is unsigned, the condition "i >= 0" always holds. The loop is terminated not because of this condition, but because of the "break" statement at the end of the loop body. Consider rewriting the loop statement like this: for (uint256 i = nToken.cashGroup.maxMarketIndex; i –> 0;) Alternatively, as market indexes are 1-based, consider looping like this: for (uint256 i = nToken.cashGroup.maxMarketIndex; i > 0; i–)

Listing 262:

```
110 for (uint256 i = nToken.cashGroup.maxMarketIndex - 1; i >= 0; i
    ↪ --) {
```

3.263 CVF-263

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenMintAction.sol

Description It is unclear what method is referred here. Consider adding an explicit assert statement for this condition.

Listing 263:

```
122 // We know from the call into this method that assetCashDeposit
    ↪ is positive
```

3.264 CVF-264

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenMintAction.sol

Description Market indexes are 1-based in most cases, but not here. Consider passing i + 1 here, instead of incrementing inside the function.

Listing 264:

```
133 i,
```

3.265 CVF-265

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenMintAction.sol

Description This would not be necessary if the loop will be refactored as suggested above.

Listing 265:

```
153 // Reached end of loop
    if (i == 0) break;
```

3.266 CVF-266

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenMintAction.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to the returned values and/or adding a documentation comment.

Listing 266:

```
186 ) private returns (int256, int256) {

270 ) private returns (int256, int256) {
```

3.267 CVF-267

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenMintAction.sol

Description This variable wouldn't be necessary if the function would be restructured like this:
if (_isMarketOverLeveraged (...)) { _deleverageMarket (...); if (_isMarketOverLeveraged (...)) { return (fCash, perMarketDeposit); } }
fCashAmount = fCashAmount.add (...); return (fCashAmount, 0);

Listing 267:

```
188 bool marketOverLeveraged =
```

3.268 CVF-268

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenMintAction.sol

Description This could be simplified as: return (fCashAmount.add (...), 0);

Listing 268:

```
209 fCashAmount = fCashAmount.add(
210     _addLiquidityToMarket(nToken, market, index,
        ↪ perMarketDeposit)
);
// No residual cash if we're adding liquidity
return (fCashAmount, 0);
```

3.269 CVF-269

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenMintAction.sol

Description Should be "else return" for readability.

Listing 269:

```
216 return (fCashAmount, perMarketDeposit);
```

3.270 CVF-270

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenMintAction.sol

Description In order to mitigate possible rounding errors, this condition could be checked as: market.totalfCash.mul(Constans.RATE_PRECISION) > leverageThreshold.mul (market.totalCash.add (totalCashUnderlying))

Listing 270:

```
229 int256 proportion =
230     market.totalfCash.mul(Constants.RATE_PRECISION).div(
            market.totalfCash.add(totalCashUnderlying)
        );

// If proportion is over the threshold, the market is over
   ↪ leveraged
return proportion > leverageThreshold;
```

3.271 CVF-271

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenMintAction.sol

Description This assumes that the current storage state is not "Delete". Consider adding an explicit assert for this.

Listing 271:

```
257 asset.storageState = AssetStorageState.Update;
```

3.272 CVF-272

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** nTokenMintAction.sol

Description The code below this line looks like it is always executed, while it is executed only when 'netAssetCash' is not zero. Consider putting the rest of the function into an explicit "else" branch.

Listing 272:

```
299 if (netAssetCash == 0) return (perMarketDeposit, 0);
```

3.273 CVF-273

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenRedeemAction.sol

Description The "currencyId" parameter should probably be indexed.

Listing 273:

```
24 event nTokenSupplyChange(address indexed account, uint16
   ↪ currencyId, int256 tokenSupplyChange);
```

3.274 CVF-274

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenRedeemAction.sol

Description Current IDs in some cases are represented as uint16, while in other places as uint256. This makes the code harder to read and error prone. Consider using the same type for currency IDs everywhere.

Listing 274:

```
32 function nTokenRedeemViaBatch(uint256 currencyId, int256
   ↪ tokensToRedeem)

60     uint16 currencyId,

96     uint256 currencyId,
```

3.275 CVF-275

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description Making the function internal would make this check unnecessary and would make batch redemptions cheaper.

Listing 275:

```
36 require(msg.sender == address(this), "Unauthorized caller");
```

3.276 CVF-276

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description External calls are more expansive than internal ones. Consider moving most of the logic of this function into an internal function. and calling this internal function when call is made from within the same contract. External function should just check the caller and call the internal function. In such case, the "|| msg.sender == address(this)" check would not be necessary.

Listing 276:

```
64 // ERC1155 can call this method during a post transfer event
   require(msg.sender == redeemer || msg.sender == address(this), "
   ↪ Unauthorized caller");
```

3.277 CVF-277

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description This variable is redundant, as it is always equal to the "tokensToRedeem_" argument. Consider removing this variable and using "tokendToRedeem_" instead. In would be convenient to have "using SafeInt256 for uint96;" statement in the beginning of the smart contract.

Listing 277:

```
68 int256 tokensToRedeem = int256(tokensToRedeem_);
```

3.278 CVF-278

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description The expression "tokensToRedeem.neg()" is calculated twice. Consider calculating once and reusing.

Listing 278:

```
75 balance.netNTokenSupplyChange = tokensToRedeem.neg();

88 emit nTokenSupplyChange(redeemer, currencyId, tokensToRedeem.neg
   ↪ ());
```

3.279 CVF-279

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description This should be done at the very end of the function to save gas on failure.

Listing 279:

```
88 emit nTokenSupplyChange(redeemer, currencyId, tokensToRedeem.neg
   ↪ ());
```

3.280 CVF-280

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** nTokenRedeemAction.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to the returned values and/or adding a documentation comment.

Listing 280:

```
103 int256 ,
    bool ,
    PortfolioAsset [] memory
```

3.281 CVF-281

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenRedeemAction.sol

Description Variable "i" is not initialized. Consider explicitly initializing to zero for readability.

Listing 281:

```
132 for (uint256 i; i < markets.length; i++) {

216 for (uint256 i; i < nToken.portfolioState.storedAssets.length; i
    ↪ ++) {

259 for (uint256 i; i < markets.length; i++) {
```

3.282 CVF-282

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenRedeemAction.sol

Description The function actually returns two objects. Consider documenting them both and/or giving them descriptive names.

Listing 282:

```
139 /// @notice Removes nToken assets and returns the net amount of
    ↪ asset cash owed to the account.

145 ) private returns (PortfolioAsset[] memory, int256) {
```

3.283 CVF-283

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** nTokenRedeemAction.sol

Description This sounds like hack. Is there a more elegant way to handle the situation where some tokens are redeemed to zero while other are not?

Listing 283:

```
191 // This can happen if a liquidity token is redeemed down to zero
    ↪ . It's possible that due to dust amounts
// one token is reduced down to a zero balance while the others
    ↪ still have some amount remaining. In this case
// the mint nToken will fail in `addLiquidityToMarket`, an
    ↪ account must accept redeeming part of their
// nTokens and leaving some dust amount behind.
```

3.284 CVF-284

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenRedeemAction.sol

Description The semantics of the returned value is unclear. Consider giving the returned value a descriptive name and/or explaining in the documentation comment.

Listing 284:

```
213 ) private view returns (int256) {
```

3.285 CVF-285

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description The conversion "int256(totalSupply)" is redundant as the "totalSupply" variable already has type "int256".

Listing 285:

```
218 int256 tokensToRemove = asset.notional.mul(tokensToRedeem).div(
    ↪ int256(totalSupply));
```

3.286 CVF-286

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description Consider adding an assert that the current asset state is not “Delete‘“.

Listing 286:

```
220 asset.storageState = AssetStorageState.Update;
```

3.287 CVF-287

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description Linear search is inefficient. Consider using binary search.

Listing 287:

```
232 uint256 ifCashIndex;
    while (newifCashAssets[ifCashIndex].maturity != asset.maturity)
       ↪ {
        ifCashIndex += 1;
        require(ifCashIndex < newifCashAssets.length, "Error
           ↪ removing tokens");
    }
```

3.288 CVF-288

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** nTokenRedeemAction.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing in the documentation comment.

Listing 288:

```
254 ) private returns (int256, bool) {
```

3.289 CVF-289

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description This dynamic array is used as two independent variables. It would be more gas efficient to just declare two local variables instead, or at least make the array length static.

Listing 289:

```
255 int256[] memory values = new int256[](2);
```

3.290 CVF-290

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** nTokenRedeemAction.sol

Description This will skip the current market in case the current asset has zero notional value, even if there is another asset afterwards with non-zero notional, whose maturity matches the maturity of the current market.

Listing 290:

```
260 if (fCashAssets[fCashIndex].notional == 0) {
        fCashIndex += 1;
        continue;
    }
```

3.291 CVF-291

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description The value "markets[i].maturity" is calculated twice. Consider calculating once and reusing.

Listing 291:

```
265 while (fCashAssets[fCashIndex].maturity < markets[i].maturity) {

272 if (fCashAssets[fCashIndex].maturity > markets[i].maturity)
    ↪ continue;
```

3.292 CVF-292

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description Linear search is suboptimal. Consider using binary search.

Listing 292:

```
265 while (fCashAssets[fCashIndex].maturity < markets[i].maturity) {
        // Skip an idiosyncratic fCash asset, if this happens then
           ↪ we know there is a residual
        // fCash asset
        fCashIndex += 1;
        hasResidual = true;
270 }
```

3.293 CVF-293

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenRedeemAction.sol

Description If 'fCashIndex' will exceed the capacity of the "fCashAssets" array, the transaction will be reverted. Probably not an issue.

Listing 293:

```
265 while (fCashAssets[fCashIndex].maturity < markets[i].maturity) {

268     fCashIndex += 1;
```

3.294 CVF-294

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Bitmap.sol

Description It is not ensured that split bitmap parts don't contain any set bits outside corresponding ranges. Consider adding corresponding checks.

Listing 294:

```
33 (splitBitmap.dayBits |
        (splitBitmap.weekBits >> Constants.WEEK_BIT_OFFSET) |
        (splitBitmap.monthBits >> Constants.MONTH_BIT_OFFSET) |
        (splitBitmap.quarterBits >> Constants.QUARTER_BIT_OFFSET));
```

3.295 CVF-295

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Bitmap.sol

Description Zero-based bit indexes would be more efficient.

Listing 295:

```
41 /// @notice Set a bit on or off in a bitmap, index is 1-indexed
```

3.296 CVF-296

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Bitmap.sol

Description It would be more efficient to have two functions: one to set bit and another to clear bit.

Listing 296:

```
45 bool setOn
```

3.297 CVF-297

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Bitmap.sol

Description This function is very inefficient. Consider using more efficient approaches. For example:
function setBitsCount2 (uint x) public pure returns (uint) { x = x - (x » 1 & 0x5555555555555555555555555555555555555555555555555555555555555555); x = (x & 0x3333333333333333333333333333333333333333333333333333333333333333) + (x » 2 & 0x3333333333333333333333333333333333333333333333333333333333333333); x = (x + (x » 4)) & 0x0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F; return (x * 0x0001010101010101010101010101010101010101010101010101010101010101 » 248) + (x & 0x0F); }

Listing 297:

```
63 function totalBitsSet(bytes32 bitmap) internal pure returns (
   ↪ uint256) {
```

3.298 CVF-298

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This constant is redundant. Just use "type(int256).min" instead.

Listing 298:

```
5 int256 private constant _INT256_MIN = -2**255;
```

3.299 CVF-299

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This function is suboptimal. Here is how I would implement it:
function mul(int256 a, int256 b) internal pure returns (int256 c) { c = a * b; if (a == -1) require (b == 0 || c / b == a); else require (a == 0 || c / a == b); }

Listing 299:

```
16 function mul(int256 a, int256 b) internal pure returns (int256)
   ↪ {
```

3.300 CVF-300

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This variable is redundant. Just give a name to the returned value and use it instead.

Listing 300:

```
26 int256 c = a * b;
```

3.301 CVF-301

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This check is redundant, as Solidity automatically reverts on division by zero.

Listing 301:

```
44 require(b != 0); // dev: int256 div by zero
```

3.302 CVF-302

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This variable is redundant. Just give a name to the returned value and use it instead.

Listing 302:

```
47 int256 c = a / b;
```

3.303 CVF-303

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This function is suboptimal. Consider implementing like this:
function neg (int256 x) internal pure returns (int256 y) { require ((y = -x) x̂ < 0); }

Listing 303:

```
60 function neg(int256 x) internal pure returns (int256) {
```

3.304 CVF-304

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SafeInt256.sol

Description This variable is redundant. Just give a name fo the returned value and use it instead.

Listing 304:

```
70 int256 z = sub(x, y);
```

3.305 CVF-305

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** SettlePortfolioAssets.sol

Description This variable is not initialized. Consider explicitly initializing to zero.

Listing 305:

```
26 uint256 lastCurrencyId;
```

3.306 CVF-306

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** SettlePortfolioAssets.sol

Description The loop condition "i >= 0" is always true because "i" is unsigned. Consider replacing the loop condition with "true" constant to make code easier to read.

Listing 306:

```
30 for (uint256 i = portfolioState.storedAssets.length - 1; i >= 0;
   ↪  i--) {
```

3.307 CVF-307

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** SettlePortfolioAssets.sol

Description It is unclear where is this loop. Consider giving a reference to the function where this happens.

Listing 307:

```
48 // Actual currency ids will be set in the loop
```

3.308 CVF-308

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** SettlePortfolioAssets.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to them and/or describing the returned values in the documentation comment.

Listing 308:

```
59 int256,
60 int256,
   SettlementMarket memory
```

3.309 CVF-309

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** SettlePortfolioAssets.sol

Description Phantom overflows are possible here. Consider using muldiv function or other safe approach: https://2π.com/21/muldiv/index.html.

Listing 309:

```
67 int256 assetCash = market.totalAssetCash.mul(asset.notional).div
   ↪ (market.totalLiquidity);
int256 fCash = market.totalfCash.mul(asset.notional).div(market.
   ↪ totalLiquidity);
```

3.310 CVF-310

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** SettlePortfolioAssets.sol

Description The semantics of the first returned value is unclear. Consider giving a descriptive name to it and/or describing in the documentation comment.

Listing 310:

```
82 ) private view returns (int256, SettlementMarket memory) {

94     returns (int256, SettlementMarket memory)
```

3.311 CVF-311

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** SettlePortfolioAssets.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 311:

```
139 for (uint256 i; i < portfolioState.storedAssets.length; i++) {
```

3.312 CVF-312

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettlePortfolioAssets.sol

Description The expression "settleAmounts[settleAmountIndex]" is calcualted twice. consider calculating once and reusing.

Listing 312:

```
176 settleAmounts[settleAmountIndex].netCashChange = settleAmounts[
    ↪ settleAmountIndex]
      .netCashChange
```

3.313 CVF-313

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** SettleBitmapAssets.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to them and/or describing them in the corresponding documentation comments.

Listing 313:

```
 26 ) private returns (bytes32, int256) {
 60 ) internal returns (bytes32, int256) {
225 ) private pure returns (bytes32) {
```

3.314 CVF-314

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description The current implementation processes one bit per invocation and just does nothing in case the bit is "zero". It would be more efficient to find the most (of the least) significant "one" bit and process it regardless of how many "zero" bits are before it.

Listing 314:

```
29 if ((bits & Constants.MSB) == Constants.MSB) {
```

3.315 CVF-315

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description These assembly blocks could be merged together.

Listing 315:

```
34 assembly {
       ifCash := sload(ifCashSlot)
   }

43 assembly {
       sstore(ifCashSlot, 0)
   }
```

3.316 CVF-316

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description This function is overcomplicated. It could be simplified like like this:
1. Iterate over "one" bits in the bitmap 2. Convert the index of a "one" bit into maturity 3. Settle the maturity if necessary 4. Otherwise, convert the maturity into a new bit index and set the corresponding bit in the new bitmap 5. Set the new bitmap.

Listing 316:

```
55 function settleBitmappedCashGroup(
```

3.317 CVF-317

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description This call should be postponed until all the cheap checks are made.

Listing 317:

```
64 SplitBitmap memory splitBitmap = bitmap.splitAssetBitmap();
```

3.318 CVF-318

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description There are more efficient ways to iterate through all "one" bits, than to check every bit separately. Consider checking several bits at once using logical "and" or using some more advanced tricks. For example the lowest set bit could be found as:
$x \,\&\, (\tilde{x} + 1)$

Listing 318:

```
76  for (uint256 bitNum = 1; bitNum <= lastSettleBit; bitNum++) {

232 for (uint256 i; i < bitsToShift; i++) {
```

3.319 CVF-319

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description Checking these conditions on every loop iteration is suboptimal. It would be cheaper to iterate through all the day bits in one loop, then through week bets in another loop etc.

Listing 319:

```
77  if (bitNum <= Constants.WEEK_BIT_OFFSET) {

97  if (bitNum <= Constants.MONTH_BIT_OFFSET) {

116 if (bitNum <= Constants.QUARTER_BIT_OFFSET) {

136 if (bitNum <= 256) {
```

3.320 CVF-320

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** SettleBitmapAssets.sol

Description If lastSettleBit can exceed 256, this value should be a named constant; otherwise it is redundant.

Listing 320:

```
136 if (bitNum <= 256) {
```

3.321 CVF-321

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** SettleBitmapAssets.sol

Description This comment should be removed from the production code

Listing 321:

```
162 /// @dev Marked as internal rather than private so we can mock
    ↪ and test directly
```

3.322 CVF-322

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SettleBitmapAssets.sol

Description This function is overcomplicated. It could be replaced with a function similar to this:
function remap (uint x, uint fromOffset, uint toOffset, uint fromStep, uint toStep, uint count) internal pure returns (uint result) { while (count -> 0) { result |= (x » fromOffset & 1) « toOffset; fromOffset += fromStep; toOffset += toStep; } }

Listing 322:

```
218 function remapBitSection(
```

3.323 CVF-323

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** SettleBitmapAssets.sol

Description The variable "i" is not initialized. Consider explicitly initializing it to zero.

Listing 323:

```
232 for (uint256 i; i < bitsToShift; i++) {
```

3.324 CVF-324

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BitmapAssetsHandler.sol

Description It would be more efficient to calculate hash once passing all the keys and the offset as the input.

Listing 324:

```
25  return
        keccak256(
            abi.encode(
                account,
                keccak256(abi.encode(currencyId, Constants.
                    ↪ ASSETS_BITMAP_STORAGE_OFFSET))
30          )
        );

61  return
        keccak256(
            abi.encode(
                maturity,
                keccak256(
                    abi.encode(
                        currencyId,
                        keccak256(abi.encode(account, Constants.
                            ↪ IFCASH_STORAGE_OFFSET))
                    )
70              )
            )
        );
```

3.325 CVF-325

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BitmapAssetsHandler.sol

Description This variable is redundant. Just give a name to the returned value and assign it in the assembly block.

Listing 325:

```
36  bytes32 data;
```

3.326 CVF-326

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BitmapAssetsHandler.sol

Description The comment is confusing. Consider making it more clear.

Listing 326:

```
93 require(currencyId != 0); // dev: invalid account in set ifcash
   ↪ assets
```

3.327 CVF-327

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BitmapAssetsHandler.sol

Description The expression "assets[i].notional" is calculated twice. Consider calculating once and reusing.

Listing 327:

```
97  if (assets[i].notional == 0) continue;

107     assets[i].notional,
```

3.328 CVF-328

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** BitmapAssetsHandler.sol

Description The semantics of the returned value is unclear. Consider giving them descriptive names and/or describing the returned values in the documentation comment.

Listing 328:

```
127 ) internal returns (bytes32, int256) {
```

3.329 CVF-329

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BitmapAssetsHandler.sol

Description Should be "else if" for readability.

Listing 329:

```
153 if (notional != 0) {
```

3.330 CVF-330

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BitmapAssetsHandler.sol

Description The code below looks like it is always executed, while it is executed only when the maturity is in the future. Consider putting the rest of the function into explicit "else" branch.

Listing 330:

```
177 if (maturity <= blockTime) return notional;
```

3.331 CVF-331

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BitmapAssetsHandler.sol

Description Should be "else return" for readability.

Listing 331:

```
191 return AssetHandler.getPresentValue(notional, maturity,
    ↪ blockTime, oracleRate);
```

3.332 CVF-332

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BitmapAssetsHandler.sol

Description Iterating through all the bits is suboptimal, taking into account that there couldn't be too many set bits. Consider checking several bits at once using bitwise operations. Also, if bit itself instead of the bit index would be used to identify maturity, then even more efficient ways to iterate through set bits could be used, based on the fact that (x & (x̃ + 1)) is the lowest set bit inside x.

Listing 332:

```
208 while (assetsBitmap != 0) {

244 while (assetsBitmap != 0) {

278 while (assetsBitmap != 0) {
```

3.333 CVF-333

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** TransferAssets.sol

Description The types used for asset properties during encoding and decoding are different. Consider using the same types for consistency.

Listing 333:

```
21      uint16 currencyId,
        uint40 maturity,
        uint8 assetType

33  uint256 currencyId,
    uint256 maturity,
    uint256 assetType
```

3.334 CVF-334

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TransferAssets.sol

Description The conversions to bytes32 and then back to uint256 are redundant.

Listing 334:

```
26 currencyId = uint16(uint256(bytes32(id) >> 48));
   maturity = uint40(uint256(bytes32(id) >> 8));
   assetType = uint8(uint256(bytes32(id)));
```

3.335 CVF-335

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** TransferAssets.sol

Description Overflows are possible during conversions to the "uint16", "uint40", and "uint8" types. Consider using safe conversions.

Listing 335:

```
39 (bytes32(uint256(uint16(currencyId))) << 48) |
40     (bytes32(uint256(uint40(maturity))) << 8) |
       bytes32(uint256(uint8(assetType)))
```

3.336 CVF-336

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** TransferAssets.sol

Description The names of the two functions called here are similar, while their behavior is very different, as the former function just updates structures in-memory, while the latter actually writes the changes into the storage. Consider renaming the second function to emphasize that it actually stores the changes.

Listing 336:

```
83 portfolioState.addMultipleAssets(assets);

98 BitmapAssetsHandler.addMultipleifCashAssets(account,
     ↪ accountContext, assets);
```

3.337 CVF-337

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Market.sol

Description This constant is declared as "internal", while other similar constants are declared as "private". Consider using consistent access levels.

Listing 337:

```
23 bytes1 internal constant STORAGE_STATE_INITIALIZE_MARKET = 0x03;
   ↪ // Both settings are set
```

3.338 CVF-338

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Market.sol

Description This value is rounded down, i.e. toward the user. Consider rounding up, towards the protocol.

Listing 338:

```
42 int256 fCash = market.totalfCash.mul(assetCash).div(market.
   ↪ totalAssetCash);
```

3.339 CVF-339

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Market.sol

Description "market.totalfCash <= fCashToAmount" would be more readable.

Listing 339:

```
89 if (market.totalfCash.sub(fCashToAccount) <= 0) return (0, 0);
```

3.340 CVF-340

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Market.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing in the documentation comment.

Listing 340:

```
152 int256 ,
    int256 ,
    int256
```

3.341 CVF-341

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Market.sol

Description "Would result" sounds less fatal.

Listing 341:

```
160 // This will result in a divide by zero
```

3.342 CVF-342

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The expression “preFeeCashToAccount.sub(fee)” is calculating twice. Consider calculating once and reusing.

Listing 342:

```
244 preFeeCashToAccount.sub(fee),

246 (preFeeCashToAccount.sub(fee).add(cashToReserve)).neg(),
```

3.343 CVF-343

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Market.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or adding a documentation comment.

Listing 343:

```
257 ) private view returns (int256, int256) {
```

3.344 CVF-344

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description These two lines could be simplified as: int128 rateScaled = ABDKMath64x64.divi(exchangeRate, Constants.RATE_PRECISION);

Listing 344:

```
326 int128 rate = ABDKMath64x64.fromInt(exchangeRate);
    int128 rateScaled = ABDKMath64x64.div(rate, Constants.
        ↪ RATE_PRECISION_64x64);
```

3.345 CVF-345

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description This line could be simplified as: uint256 lnRate = ABDKMath.mulu(lnRateScaled, Constants.RATE_PRECISION);

Listing 345:

```
331 uint256 lnRate =
        ABDKMath64x64.toUInt(ABDKMath64x64.mul(lnRateScaled,
            ↪ Constants.RATE_PRECISION_64x64));
```

3.346 CVF-346

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description These lines could be simplified as; int128 expValueScaled = ABDKMath64x64.divi (impliedRate.mul(timeToMaturity).div(Constants.IMPLIED_RATE_TIME), Constants.RATE_PRECISION);

Listing 346:

```
349 int128 expValue =
350     ABDKMath64x64.fromUInt(
            impliedRate.mul(timeToMaturity).div(Constants.
              ↪ IMPLIED_RATE_TIME)
        );
    int128 expValueScaled = ABDKMath64x64.div(expValue, Constants.
      ↪ RATE_PRECISION_64x64);
```

3.347 CVF-347

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description These lines could be simplified as: return ABDKMath64x64.muli(expResult, Constants.RATE_PRECISION);

Listing 347:

```
355 int128 expResultScaled = ABDKMath64x64.mul(expResult, Constants.
      ↪ RATE_PRECISION_64x64);

    return ABDKMath64x64.toInt(expResultScaled);
```

3.348 CVF-348

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** Market.sol

Description As the "proportion" value could be negative, it is also necessary to check the lower bound.

Listing 348:

```
401 if (proportion > MAX64) return (0, false);
```

3.349 CVF-349

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description This line could be simplified as: int256 result = ABDKMath64x64.muli (ABDKMath64x64.ln(abdkProportion), Constants.RATE_PRECISION);

Listing 349:

```
408 int256 result =
        ABDKMath64x64.toUInt(
410         ABDKMath64x64.mul(ABDKMath64x64.ln(abdkProportion),
                ↪ Constants.RATE_PRECISION_64x64)
        );
```

3.350 CVF-350

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description It would be cheaper to calculate hash once, passing all the keys and the offset as inputs.

Listing 350:

```
467 return
        keccak256(
            abi.encode(
470             maturity,
                keccak256(
                    abi.encode(
                        settlementDate,
                        keccak256(abi.encode(currencyId, Constants.
                            ↪ MARKET_STORAGE_OFFSET))
                    )
                )
            )
        );
```

3.351 CVF-351

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description This variable is redundant. Just assign a name to the returned value and use it instead of this variable.

Listing 351:

```
484 int256 totalLiquidity;
```

3.352 CVF-352

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Market.sol

Description This means that a market actually has at least two storage slots: the one referred by the "storageSlot" field and, and the next one. This could be confusing. Consider renaming the "storageSlot" field to "firstStorageSlot" or "storageOffset" to emphasize that this is only one of several consequent storage slots used by the market.

Listing 352:

```
485 bytes32 slot = bytes32(uint256(market.storageSlot) + 1);
```

3.353 CVF-353

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description This code repeats several times. Consider extracting a function.

Listing 353:

```
500 bytes32 slot = getSlot(currencyId, settlementDate, maturity);
    bytes32 data;

503 assembly {
        data := sload(slot)
    }

537 bytes32 slot = getSlot(currencyId, settlementDate, maturity);
    bytes32 data;

540 assembly {
        data := sload(slot)
    }

563 bytes32 slot = market.storageSlot;

567     bytes32 oldData;
        assembly {
            oldData := sload(slot)
570     }

655 uint256 slot = uint256(getSlot(currencyId, settlementDate,
      ↪ maturity));

657 bytes32 data;

659 assembly {
660     data := sload(slot)
    }
```

3.354 CVF-354

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The final conversions to the "uint256" type are redundant.

Listing 354:

```
507 uint256 lastImpliedRate = uint256(uint32(uint256(data >> 160)));
    uint256 oracleRate = uint256(uint32(uint256(data >> 192)));
    uint256 previousTradeTime = uint256(uint32(uint256(data >> 224))
        ↪ );
```

3.355 CVF-355

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Market.sol

Description The hardcoded offsets should be turned into named constants.

Listing 355:

```
507 uint256 lastImpliedRate = uint256(uint32(uint256(data >> 160)));
    uint256 oracleRate = uint256(uint32(uint256(data >> 192)));
    uint256 previousTradeTime = uint256(uint32(uint256(data >> 224))
        ↪ );

547 market.totalAssetCash = int256(uint80(uint256(data >> 80)));
    market.lastImpliedRate = uint256(uint32(uint256(data >> 160)));
    market.oracleRate = uint256(uint32(uint256(data >> 192)));
550 market.previousTradeTime = uint256(uint32(uint256(data >> 224)))
        ↪ ;

582             (bytes32(market.totalAssetCash) << 80) |
                (bytes32(market.lastImpliedRate) << 160) |
                (bytes32(market.oracleRate) << 192) |
                (bytes32(market.previousTradeTime) << 224));
```

3.356 CVF-356

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description This check could be done earlier, right after unpacking the oracle rate.

Listing 356:

```
515 require(oracleRate > 0, "Market not initialized");
```

3.357 CVF-357

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The final conversions are redundant.

Listing 357:

```
546 market.totalfCash = int256(uint80(uint256(data)));
    market.totalAssetCash = int256(uint80(uint256(data >> 80)));
    market.lastImpliedRate = uint256(uint32(uint256(data >> 160)));
    market.oracleRate = uint256(uint32(uint256(data >> 192)));
550 market.previousTradeTime = uint256(uint32(uint256(data >> 224)))
      ↪ ;
```

3.358 CVF-358

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The first part of the check is redundant, as the "lastImpliedRate" field is unsigned.

Listing 358:

```
576 require(market.lastImpliedRate >= 0 && market.lastImpliedRate <=
      ↪ type(uint32).max); // dev: market storage lastImpliedRate
      ↪ overflow
```

3.359 CVF-359

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The first part of the check is redundant, as the "oracleRate" field is unsigned.

Listing 359:

```
577 require(market.oracleRate >= 0 && market.oracleRate <= type(
    ↪ uint32).max); // dev: market storage oracleRate overflow
```

3.360 CVF-360

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The first part of the check is redundant, as the "previousTradeTime" field is unsigned.

Listing 360:

```
578 require(market.previousTradeTime >= 0 && market.
    ↪ previousTradeTime <= type(uint32).max); // dev: market
    ↪ storage previous trade time overflow
```

3.361 CVF-361

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The final conversions are redundant.

Listing 361:

```
663 int256 totalfCash = int256(uint80(uint256(data)));
    int256 totalAssetCash = int256(uint80(uint256(data >> 80)));
```

3.362 CVF-362

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Market.sol

Description This mask should be turned into a named constant.

Listing 362:

```
667 data = data & 0
    ↪ xffffffffffffffffffffffff0000000000000000000000000000000000000000
    ↪ ;
```

3.363 CVF-363

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The conversion to the "uint256" type is redundant.

Listing 363:

```
669 slot = uint256(slot) + 1;
```

3.364 CVF-364

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description Doing increment in assembly would make it unnecessary to add and then subtract 1: totalLiquidity := sload (add (slot, 1))

Listing 364:

```
669 slot = uint256(slot) + 1;

672     totalLiquidity := sload(slot)

677         storageSlot: bytes32(slot - 1),
```

3.365 CVF-365

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** Market.sol

Description This expression produces incorrect result in case "market.data" already has values for total fCash and total asset cash. Consider either requiring these fields to be zero inside "market.data" or marking them out before doing bitwise "or".

Listing 365:

```
692 data = (bytes32(market.totalfCash) |
        (bytes32(market.totalAssetCash) << 80) |
        bytes32(market.data));
```

3.366 CVF-366

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description This variable is redundant as its value is used only once. Just use "market.totalLiquidity" instead.

Listing 366:

```
703 bytes32 totalLiquidity = bytes32(market.totalLiquidity);
```

3.367 CVF-367

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Market.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 367:

```
744 for (uint8 i; i < 250; i++) {
```

3.368 CVF-368

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Market.sol

Description Overflow is possible when converting to "int256".

Listing 368:

```
766 if (delta.abs() <= int256(maxDelta)) return
    ↪ fCashChangeToAccountGuess;
```

3.369 CVF-369

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** DateTime.sol

Description The name is confusing. The function actually returns the beginning of the quarter the given time belongs to. Consider renaming to something like "getQuarter" or "getQuarterStart".

Listing 369:

```
12 function getReferenceTime(uint256 blockTime) internal pure
    ↪ returns (uint256) {
```

3.370 CVF-370

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** DateTime.sol

Description Should be ">=", as the output is the same for blockTime == QUARTER and blockTime == QUARTER + 1.

Listing 370:

```
13 require(blockTime > Constants.QUARTER);
```

3.371 CVF-371

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** DateTime.sol

Description The name is confusing. The function actually returns the beginning of the day the given time belongs to. Consider renaming to something like "getDay" or "getDayStart".

Listing 371:

```
18 function getTimeUTC0(uint256 time) internal pure returns (
   ↪ uint256) {
```

3.372 CVF-372

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** DateTime.sol

Description Should be ">=", as the output is the same for blockTime == DAY and blockTime == DAY + 1.

Listing 372:

```
19 require(time > Constants.DAY);
```

3.373 CVF-373

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** DateTime.sol

Description This function could be optimized like this: require (index >= 1 && index <=7); return uint32 (0x25143000128a180009450c0003b5380001da9c0000ed4e000076a70000000000 » (index « 5));

Listing 373:

```
25 function getTradedMarket(uint256 index) internal pure returns (
     ↪ uint256) {
     require(index != 0); // dev: get traded market index is zero

     if (index == 1) return Constants.QUARTER;
     if (index == 2) return 2 * Constants.QUARTER;
30   if (index == 3) return Constants.YEAR;
     if (index == 4) return 2 * Constants.YEAR;
     if (index == 5) return 5 * Constants.YEAR;
     if (index == 6) return 10 * Constants.YEAR;
     if (index == 7) return 20 * Constants.YEAR;

     revert("CG: invalid index");
```

3.374 CVF-374

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** DateTime.sol

Description This function could be optimized as: return maturity >= blockTime && maturity % Constants.QUARTER == 0 && ((0x700000000000000000000000000006000000000000005000000004003088 » (maturity - getReferenceTime (blockTime)) / Constants.QUARTER * 3) & 0x07) - 1 <= maxMarketIndex;

Listing 374:

```
40 function isValidMarketMaturity(
```

3.375 CVF-375

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** DateTime.sol

Description The maximum market index allowed by the "getTradedMarket" function is 7, but here the maximum allowed value is 9. Consider changing the condition to "maxMarketIndex <= Constants.MAX_TRADED_MARKET_INDEX".

Listing 375:

```
46 require(maxMarketIndex < 10, "CG: market index bound");

80 require(maxMarketIndex < 10, "CG: market index bound");
```

3.376 CVF-376

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** DateTime.sol

Description This function could be optimized in a way similar to how the "isValidMarketMaturity" function could be optimized.

Listing 376:

```
74 function getMarketIndex(
```

3.377 CVF-377

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** DateTime.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing the returned value in the documentation comment.

Listing 377:

```
78 ) internal pure returns (uint256, bool) {
```

3.378 CVF-378

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** DateTime.sol

Description The "blockTimeUTC0" value is always divided by Constants.DAY. Consider dividing once and using the divided value.

Listing 378:

```
100 uint256 blockTimeUTC0 = getTimeUTC0(blockTime);

106 uint256 daysOffset = (maturity - blockTimeUTC0) / Constants.DAY;

117             (blockTimeUTC0 % Constants.WEEK) /
                Constants.DAY;

128             (blockTimeUTC0 % Constants.MONTH) /
                Constants.DAY;

138             (blockTimeUTC0 % Constants.QUARTER) /
                Constants.DAY;
```

3.379 CVF-379

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** DateTime.sol

Description These "6", "30", and "90" values should be named constants.

Listing 379:

```
121 return (Constants.WEEK_BIT_OFFSET + offset / 6, (offset % 6) ==
    ↪ 0);

131 return (Constants.MONTH_BIT_OFFSET + offset / 30, (offset % 30)
    ↪ == 0);

141 return (Constants.QUARTER_BIT_OFFSET + offset / 90, (offset %
    ↪ 90) == 0);
```

3.380 CVF-380

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** DateTime.sol

Description For readability these ‘if‘ clauses should be made ‘else if‘

Listing 380:

```
164 if (bitNum <= Constants.MONTH_BIT_OFFSET) {

173 if (bitNum <= Constants.QUARTER_BIT_OFFSET) {

182 firstBit =
```

3.381 CVF-381

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** CashGroup.sol

Description This constants should be moved to “Constants.sol”.

Listing 381:

```
20  uint256 private constant RATE_ORACLE_TIME_WINDOW = 8;
    uint256 private constant TOTAL_FEE = 16;
    uint256 private constant RESERVE_FEE_SHARE = 24;
    uint256 private constant DEBT_BUFFER = 32;
    uint256 private constant FCASH_HAIRCUT = 40;
    uint256 private constant SETTLEMENT_PENALTY = 48;
    uint256 private constant LIQUIDATION_FCASH_HAIRCUT = 56;

28  uint256 private constant LIQUIDITY_TOKEN_HAIRCUT = 64;

30  uint256 private constant RATE_SCALAR = 136;
```

3.382 CVF-382

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** CashGroup.sol

Description The relationship between the comments and the constants is unclear.

Listing 382:

```
27 // 9 bytes allocated per market on the liquidity token haircut
   uint256 private constant LIQUIDITY_TOKEN_HAIRCUT = 64;
   // 9 bytes allocated per market on the rate scalar
30 uint256 private constant RATE_SCALAR = 136;
```

3.383 CVF-383

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** CashGroup.sol

Description There is no upper bound check for the "marketIndex" value, while too big market index would result in reading bits outside those allocated for rate scalars, or even bits outside the "cashData.data" value. Consider adding an upper bound check.

Listing 383:

```
40 require(marketIndex >= 1); // dev: invalid market index
```

3.384 CVF-384

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CashGroup.sol

Description The value "10" here should be turned into a named constant.

Listing 384:

```
42 int256 scalar = int256(uint8(uint256(cashGroup.data >> offset)))
   ↪  * 10;
```

3.385 CVF-385

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** CashGroup.sol

Description Probably, returning zero here, and handling it in the calling code would be safer. With the current implementation the caller cannot predict nor prevent transaction revert which could potentially be used for denial of service attacks. A common good practice is that getter functions should revert only on incorrect arguments or in situations that meant to be impossible.

Listing 385:

```
46 // At large time to maturities it's possible for the rate scalar
   ↪ to round down to zero
require(rateScalar > 0, "CG: rate scalar underflow");
```

3.386 CVF-386

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** CashGroup.sol

Description Should be "assetType >= MIN_LIQUIDITY_TOKEN_INDEX" to explain why underflow is not possible below.

Listing 386:

```
58 require(assetType > 1); // dev: liquidity haircut invalid asset
   ↪ type
```

3.387 CVF-387

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description The final conversion to the "uint256" type is redundant.

Listing 387:

```
61 uint256 liquidityTokenHaircut = uint256(uint8(uint256(cashGroup.
   ↪ data >> offset)));
```

3.388 CVF-388

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** CashGroup.sol

Description The result is actually denominated in rate precision terms, not in basis points, however the result is guaranteed to be a factor of one basis point.

Listing 388:

```
65 /// @notice Total trading fee denominated in basis points
```

3.389 CVF-389

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description The final conversion to "int256" is redundant.

Listing 389:

```
76 return int256(uint8(uint256(cashGroup.data >> RESERVE_FEE_SHARE)
   ↪ ));
```

3.390 CVF-390

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** CashGroup.sol

Description The result is actually denominated in rate precision terms, not in basis points, however the result is guaranteed to be a factor of 5 basis point.

Listing 390:

```
79 /// @notice fCash haircut for valuation denominated in basis
   ↪ points

85 /// @notice fCash debt buffer for valuation denominated in basis
   ↪  points

100 /// @notice Penalty rate for settling cash debts denominated in
   ↪ basis points

111 /// @notice Haircut for fCash during liquidation denominated in
   ↪ basis points
```

3.391 CVF-391

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** CashGroup.sol

Description The value is denominated in seconds but is a factor of one minute. Consider explaining this in the documentation comment.

Listing 391:

```
90 /// @notice Time window factor for the rate oracle denominated
   ↪ in seconds
```

3.392 CVF-392

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** CashGroup.sol

Description The underflow would not be a problem if calculations would be performed in signed numbers.

Listing 392:

```
155 // It's possible that the rates are inverted where the short
    ↪ market rate > long market rate and
// we will get an underflow here so we check for that
```

3.393 CVF-393

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** CashGroup.sol

Description The code below looks like it is always executed, while it is executed only in case idiosyncratic is true. Consider putting the rest of the function into explicit "else" branch.

Listing 393:

```
192 }
```

3.394 CVF-394

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description The expression "DateTime.getReferenceTime(blockTime)" is calculated twice. Consider calculating once and reusing.

Listing 394:

```
195 DateTime.getReferenceTime(blockTime).add(DateTime.
    ↪ getTradedMarket(marketIndex));

206 shortMaturity = DateTime.getReferenceTime(blockTime).add(
```

3.395 CVF-395

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description This calculates storage address of a mapping element in a way similar to how Solidity calculates it. Consider using plain Solidity maps to make code easier to read, less error-prone, and more efficient.

Listing 395:

```
222 bytes32 slot = keccak256(abi.encode(currencyId, Constants.
    ↪ CASH_GROUP_STORAGE_OFFSET));

242 bytes32 slot = keccak256(abi.encode(currencyId, Constants.
    ↪ CASH_GROUP_STORAGE_OFFSET));
```

3.396 CVF-396

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description This variable is redundant. Just assign a name to the returned value and use in the assembly block below.

Listing 396:

```
223 bytes32 data;
```

3.397 CVF-397

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CashGroup.sol

Description The value "31" should be turned to a named constant.

Listing 397:

```
235 return uint8(data[31]);

311 uint8 maxMarketIndex = uint8(data[31]);
```

3.398 CVF-398

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description The slot address calculation should be performed just before writing data to this slot. This will save gas on unsuccessful invocation.

Listing 398:

```
242 bytes32 slot = keccak256(abi.encode(currencyId, Constants.
    ↪ CASH_GROUP_STORAGE_OFFSET));
```

3.399 CVF-399

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description This conditions is always true, as the "maxMarketIndex" field is unsigned. Consider removing this condition.

Listing 399:

```
244 cashGroup.maxMarketIndex >= 0 &&
```

3.400 CVF-400

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description This check could be merged into the previous one, as they both check validity of the same field.

Listing 400:

```
251 require(cashGroup.maxMarketIndex != 1, "CG: invalid market index
    ↪ ");
```

3.401 CVF-401

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** CashGroup.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 401:

```
281 for (uint256 i; i < cashGroup.liquidityTokenHaircuts.length; i
    ↪ ++) {

293 for (uint256 i; i < cashGroup.rateScalars.length; i++) {

315 for (uint8 i; i < maxMarketIndex; i++) {
```

3.402 CVF-402

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description These looks repack byte arrays into works. As byte arrays are stored in the memory in a compact form, it is possible to read a whole array at once, no need for a loop.

Listing 402:

```
281 for (uint256 i; i < cashGroup.liquidityTokenHaircuts.length; i
    ↪ ++) {

293 for (uint256 i; i < cashGroup.rateScalars.length; i++) {
```

3.403 CVF-403

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CashGroup.sol

Description Named constants should be used instead of plain numbers here

Listing 403:

```
316 tokenHaircuts[i] = uint8(data[23 - i]);
    rateScalars[i] = uint8(data[14 - i]);

323         rateOracleTimeWindowMin: uint8(data[30]),
            totalFeeBPS: uint8(data[29]),
            reserveFeeShare: uint8(data[28]),
            debtBuffer5BPS: uint8(data[27]),
            fCashHaircut5BPS: uint8(data[26]),
            settlementPenaltyRate5BPS: uint8(data[25]),
            liquidationfCashHaircut5BPS: uint8(data[24]),
```

3.404 CVF-404

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CashGroup.sol

Description The final conversion to the "uint256" is redundant.

Listing 404:

```
341 uint256 maxMarketIndex = uint256(uint8(uint256(data)));
```

3.405 CVF-405

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** AssetRate.sol

Description These constants should be moved to the "Constants.sol" file.

Listing 405:

```
 14 uint256 private constant ASSET_RATE_STORAGE_SLOT = 2;

 17 int256 private constant ASSET_RATE_DECIMAL_DIFFERENCE = 1e10;
```

3.406 CVF-406

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** AssetRate.sol

Description These functions always round down. A good practice is to round towards protocol, i.e. against user.

Listing 406:

```
22 function convertToUnderlying(AssetRateParameters memory ar,
   ↪ int256 assetBalance)

40 function convertFromUnderlying(AssetRateParameters memory ar,
   ↪ int256 underlyingBalance)
```

3.407 CVF-407

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AssetRate.sol

Description This check is redundant. It slightly optimizes a very rare case of zero balance, but makes all other cases more expensive. Consider removing it.

Listing 407:

```
27 if (assetBalance == 0) return 0;

45 if (underlyingBalance == 0) return 0;
```

3.408 CVF-408

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AssetRate.sol

Description Should be "rate * balance * ...".

Listing 408:

```
30 // rateDecimals * balance * internalPrecision / rateDecimals *
   ↪ underlyingPrecision
```

3.409 CVF-409

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AssetRate.sol

Description Should be "... / rate * ...".

Listing 409:

```
48 // rateDecimals * balance * underlyingPrecision / rateDecimals *
   ↪ internalPrecision
```

3.410 CVF-410

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AssetRate.sol

Description The conversion to "address" performs a shift, so there are two shifts in total, while zero shifts are actually needed. Consider rewriting the expression like this: rateOracle = address(uint256(data));

Listing 410:

```
81 rateOracle = address(bytes20(data << 96));
```

3.411 CVF-411

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** AssetRate.sol

Description These two function have much in common. Consider merging them together or extracting common parts into utility function to reduce code duplication.

Listing 411:

```
87 function _getAssetRateView(uint256 currencyId)

113 function _getAssetRateStateful(uint256 currencyId)
```

3.412 CVF-412

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** AssetRate.sol

Description Rates have 18 decimals, so the identity rate should be 10Î18, not 10Î10.

Listing 412:

```
100 // If no rate oracle is set, then set this to the identity
    rate = ASSET_RATE_DECIMAL_DIFFERENCE;

125 // If no rate oracle is set, then set this to the identity
    rate = ASSET_RATE_DECIMAL_DIFFERENCE;
```

3.413 CVF-413

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** AssetRate.sol

Description For identity rate, the underlying precision should be set to internal precision.

Listing 413:

```
102 underlyingDecimalPlaces = 0;

127 underlyingDecimalPlaces = 0;
```

3.414 CVF-414

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** AssetRate.sol

Description These function have much in common. Consider merging them together or extracting common parts into utility functions to reduce code duplication.

Listing 414:

```
137 function buildAssetRateView(uint256 currencyId)

154 function buildAssetRateStateful(uint256 currencyId)
```

3.415 CVF-415

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** AssetRate.sol

Description Overflow is possible here. Consider using safe power or range checking the underlying decimals.

Listing 415:

```
149             underlyingDecimals: int256(10**underlyingDecimalPlaces)

165             underlyingDecimals: int256(10**underlyingDecimalPlaces)

218 return AssetRateParameters(address(0), settlementRate, int256
    ↪ (10**underlyingDecimalPlaces));

261 return AssetRateParameters(address(0), settlementRate, int256
    ↪ (10**underlyingDecimalPlaces));
```

3.416 CVF-416

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AssetRate.sol

Description This calculates a storage address for a mapping value in a way similar to how Solidty does this. Consider using plain Solidity mappings and this would make the code simpler, easier to read, and probably more efficient.

Listing 416:

```
179 bytes32 data;
180 slot = keccak256(
        abi.encode(
            currencyId,
            keccak256(abi.encode(maturity, Constants.
                ↪ SETTLEMENT_RATE_STORAGE_OFFSET))
        )
    );
```

3.417 CVF-417

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AssetRate.sol

Description The final conversion to the "int256" type is redundant.

Listing 417:

```
191 settlementRate = int256(uint128(uint256(data >> 40)));
```

3.418 CVF-418

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AssetRate.sol

Description The condition "blockTime != 0" looks redundant. Consider removing it.

Listing 418:

```
244 require(blockTime != 0 && blockTime <= type(uint40).max); // dev
    ↪ : settlement rate timestamp overflow
```

3.419 CVF-419

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidationHelpers.sol

Description The function returns not only liquidation factors, but also account context and portfolio state. Consider describing this in the documentation comment.

Listing 419:

```
25 /// @notice Settles accounts and returns liquidation factors for
    ↪  all of the liquidation actions.
```

3.420 CVF-420

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidationHelpers.sol

Description The name is too generic and doesn't reflect what this function actually does. Consider renaming.

Listing 420:

```
26 function preLiquidationActions(
```

3.421 CVF-421

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidationHelpers.sol

Description The names of these arguments are confusing, as it is unclear whether they are currency amounts or currency IDs. Consider renaming to "localCurrencyId" and "collateralCurrencyId". Also, currency IDs are usually represented by the "uint16" type, but here "uint256" is used. Consider using the same type for currency IDs across the code.

Listing 421:

```
28 uint256 localCurrency,
   uint256 collateralCurrency
```

3.422 CVF-422

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidationHelpers.sol

Description The value of "localCurrency" cannot be zero here, so "collateralCurrency == 0' implies "collateralCurrency != localCurrency". Thus, the first part of the expression is redundant.

Listing 422:

```
40 require(collateralCurrency == 0 || collateralCurrency !=
   ↪ localCurrency);
```

3.423 CVF-423

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** LiquidationHelpers.sol

Description Why this limit is needed? What prevents a liquidator from purchasing more collateral by performing several transactions?

Listing 423:

```
63 /// @notice We allow liquidators to purchase up to Constants.
   ↪ DEFAULT_LIQUIDATION_PORTION percentage of collateral
```

3.424 CVF-424

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** LiquidationHelpers.sol

Description This assignment could be overwritten in the conditional statement below. Consider putting it into an explicit else branch for efficiency and readability.

Listing 424:

```
78 int256 result = initialAmountToLiquidate;
```

3.425 CVF-425

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidationHelpers.sol

Description This check is redundant, as the liquidation amount is anyway capped by the maxAllowedAmount which is less than the total balance.

Listing 425:

```
80 if (initialAmountToLiquidate > maxTotalBalance) {
       result = maxTotalBalance;
   }
```

3.426 CVF-426

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** LiquidationHelpers.sol

Description Here, the maximum allowed amount is used as a lower, rather than upper cap.

Listing 426:

```
84 if (initialAmountToLiquidate < maxAllowedAmount) {
       // Allow the liquidator to go up to the max allowed amount
       result = maxAllowedAmount;
   }
```

3.427 CVF-427

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidationHelpers.sol

Description The expressions "factors.collateralETHRate.liquidationDiscount" and "factors.localETHRate.liquidationDiscount" are potentially calculated twice. Consider using a "max" function instead.

Listing 427:

```
115 if (
        factors.collateralETHRate.liquidationDiscount > factors.
            ↪ localETHRate.liquidationDiscount
    ) {
        liquidationDiscount = factors.collateralETHRate.
            ↪ liquidationDiscount;
    } else {
120     liquidationDiscount = factors.localETHRate.
            ↪ liquidationDiscount;
    }
```

3.428 CVF-428

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidationHelpers.sol

Description This line does nothing. Consider removing it.

Listing 428:

```
123 return (assetCashBenefitRequired, liquidationDiscount);
```

3.429 CVF-429

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidationHelpers.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing the returned values in the documentation comment.

Listing 429:

```
133 ) internal pure returns (int256, int256) {
```

3.430 CVF-430

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidationHelpers.sol

Description The expression "factors.localAssetAvailable.neg()" is calculated twice. Consider calculating once and reusing.

Listing 430:

```
148 if (localAssetFromLiquidator > factors.localAssetAvailable.neg()
    ↪ ) {

153             .mul(factors.localAssetAvailable.neg())
```

3.431 CVF-431

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidationHelpers.sol

Description There is a named constant for the FALSE value. Consider using it here.

Listing 431:

```
182 liquidatorContext.hasDebt == 0x00,
```

3.432 CVF-432

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidationHelpers.sol

Description This function always returns the liquidator context passed as an argument. This liquidator context is anyway available to the caller, so returning it seems redundant. Consider not returning any value.

Listing 432:

```
197     AccountContext memory liquidatorContext,

203 ) internal returns (AccountContext memory) {
```

3.433 CVF-433

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description It is unclear what these two factors are. Consider giving descriptive names to the returned values and/or describing them in the documentation comment.

Listing 433:

```
25 /// @notice Calculates the two discount factors relevant when
   ↪ liquidating fCash.

30 ) private view returns (int256, int256) {
```

3.434 CVF-434

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description The semantics of the returned value is unclear. Consider giving a descriptive name to the returned value and/or describing it in the documentation comment.

Listing 434:

```
57 ) private view returns (int256) {
```

3.435 CVF-435

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** LiquidatefCash.sol

Description The notional value is calculated but not returned.

Listing 435:

```
59 int256 notional =
```

3.436 CVF-436

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** LiquidatefCash.sol

Description The variable “i” is not initialized. Consider explicitly initializing to zero.

Listing 436:

```
65  for (uint256 i; i < portfolio.length; i++) {
119 for (uint256 i; i < fCashMaturities.length; i++) {
191 for (uint256 i; i < fCashMaturities.length; i++) {
446 for (uint256 i; i < assets.length; i++) {
```

3.437 CVF-437

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidatefCash.sol

Description This structure should probably be moved to "Types.sol".

Listing 437:

```
82 struct fCashContext {
```

3.438 CVF-438

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description Most of the function arguments in this library have long descriptive names, however these arguments have single-letter names. Consider giving them descriptive names for consistency and readability.

Listing 438:

```
100 fCashContext memory c,
174 fCashContext memory c,
212 fCashContext memory c,
285 fCashContext memory c,
377 fCashContext memory c
415 fCashContext memory c
```

3.439 CVF-439

- **Severity** Moderate
- **Status** Opened
- **Category** Procedural
- **Source** LiquidatefCash.sol

Description It is a bad practice to use compiler-enforced checks to enforce business-level constrains. Also, division by zero check behaves like assert rather than like revert, i.e. it consumes all of the available gas. Consider adding an explicit check and revert in case haircut is zero.

Listing 439:

```
109 // If the haircut is zero then this will revert which is the
    ↪ correct result. A currency with
110 // a haircut to zero does not affect free collateral.
.div(c.factors.localETHRate.haircut);
```

3.440 CVF-440

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** LiquidatefCash.sol

Description Overflow is possible here. Consider using safe conversion.

Listing 440:

```
141 int256(maxfCashLiquidateAmounts[i])

200 int256(maxfCashLiquidateAmounts[i]),
```

3.441 CVF-441

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Description Not all slots in this array are actually filled, and unfilled elements may be left in the middle of the array. Consider populating the array without gaps and sizing it according to the actual number of populated slots. Note, that it is possible to truncate in-memory array via assembly.

Listing 441:

```
180 c.fCashNotionalTransfers = new int256[](fCashMaturities.length);
```

3.442 CVF-442

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description The total benefit is divided by this value, not multiplied. Consider renaming.

Listing 442:

```
235 int256 benefitMultiplier;
```

3.443 CVF-443

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing them in the documentation comment.

Listing 443:

```
289 ) private pure returns (int256, int256) {
```

3.444 CVF-444

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidatefCash.sol

Description The semantics of the function is unclear. Consider adding a documentation comment.

Listing 444:

```
338 function _calculateLocalToPurchaseUnderlying(
```

3.445 CVF-445

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to the returned values and/or adding a documentation comment.

Listing 445:

```
343 ) internal pure returns (int256, int256) {
```

3.446 CVF-446

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidatefCash.sol

Description The semantics of the returned values is unclear. Consider giving descriptive names to the returned values and/or describing them in the documentation comment.

Listing 446:

```
378 ) internal returns (int256[] memory, int256) {
```

3.447 CVF-447

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Description Returning these value seems redundant, as the fCash context (known here as "c") is anyway passed by reference and is available to the caller.

Listing 447:

```
406 return (c.fCashNotionalTransfers, c.localAssetCashFromLiquidator
    ↪ );
```

3.448 CVF-448

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** LiquidatefCash.sol

Description It is not checked that these arrays have the same length. Consider adding such check.

Listing 448:

```
442 uint256[] calldata fCashMaturities,
    int256[] memory fCashNotionalTransfers
```

3.449 CVF-449

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Description Here, "fCashMaturities.length" and "assets.length" are the same, so using them both is confusing. Consider either using "fCashMaturities.length" everywhere, or storing this value in a local variable and using the variable.

Listing 449:

```
445 PortfolioAsset[] memory assets = new PortfolioAsset[](
    ↪ fCashMaturities.length);
    for (uint256 i; i < assets.length; i++) {
```

3.450 CVF-450

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Description The expression "assets[i]" is calculated several times. Consider calculating once and reusing or using a structure literal assignment.

Listing 450:

```
447 assets[i].currencyId = fCashCurrency;
    assets[i].assetType = Constants.FCASH_ASSET_TYPE;
    assets[i].notional = fCashNotionalTransfers[i];
450 assets[i].maturity = fCashMaturities[i];
```

3.451 CVF-451

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** LiquidateCurrency.sol

Description This will return true even if all liquidity tokens for the given currency are in Delete state. Consider ignoring such liquidity tokens.

Listing 451:

```
32 if (
       portfolio[i].currencyId == currencyId &&
       AssetHandler.isLiquidityToken(portfolio[i].assetType)
   ) {
       return true;
   }
```

3.452 CVF-452

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description The call to '_hasLiquidityTokens' is redundant, as the subsequent call to '_withdrawLocalLiquidityTokens' will basically do the same job again. Consider removing the "_hasLiquidityTokens" call.

Listing 452:

```
68 if (_hasLiquidityTokens(portfolio.storedAssets, localCurrency))
   ↪ {

70     (w, assetBenefitRequired) = _withdrawLocalLiquidityTokens(
```

3.453 CVF-453

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidateCurrency.sol

Description Relying on checks made in other parts of the code makes the code more fragile, as the importance of that checks is not obvious for those who look at them. Considir using safe subtraction.

Listing 453:

```
83 // This will not underflow, checked when saving parameters

86         uint8(factors.nTokenParameters[Constants.
               ↪ LIQUIDATION_HAIRCUT_PERCENTAGE]) -
             uint8(factors.nTokenParameters[Constants.
                 ↪ PV_HAIRCUT_PERCENTAGE])
```

3.454 CVF-454

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidateCurrency.sol

Description These formulas are confusing. They are not derived from each other. Consider adding come plain text explanation what all these formulas are about.

Listing 454:

```
90 // fullNTokenPV = haircutTokenPV / haircutPercentage
   // benefitGained = nTokensToLiquidate * (liquidatedPV -
      ↪ freeCollateralPV)
   // benefitGained = nTokensToLiquidate * (fullNTokenPV *
      ↪ liquidatedPV - fullNTokenPV * pvHaircut)
   // benefitGained = nTokensToLiquidate * fullNTokenPV * (
      ↪ liquidatedPV - pvHaircut) / totalBalance
   // benefitGained = nTokensToLiquidate * (haircutTokenPV /
      ↪ haircutPercentage) * (liquidationHaircut - pvHaircut) /
      ↪ totalBalance
   // benefitGained = nTokensToLiquidate * haircutTokenPV * (
      ↪ liquidationHaircut - pvHaircut) / (totalBalance *
      ↪ haircutPercentage)
```

3.455 CVF-455

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description This formula doesn't exactly match the code after it. Consider using the same terms in teh formula and in the code. Also consider adding a plain-text explanation.

Listing 455:

```
96 // nTokensToLiquidate = (benefitGained * totalBalance *
   ↪ haircutPercentage) / (haircutTokenPV * (liquidationHaircut
   ↪  - pvHaircut))
```

3.456 CVF-456

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description This conversion is redundant.

Listing 456:

```
106 int256(maxNTokenLiquidation)
```

3.457 CVF-457

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description The conversion to the "int256" type is redundant.

Listing 457:

```
146 ) = _calculateCollateralToRaise(factors, int256(
   ↪ maxCollateralLiquidation));
```

3.458 CVF-458

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description Should be ">=0" and the "else" branch is more expensive.

Listing 458:

```
150 if (balanceState.storedCashBalance > collateralAssetRemaining) {
```

3.459 CVF-459

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description It is usually possible to address by splitting large functions into smaller ones. Also, enclosing parts of a large functions into blocks could help.

Listing 459:

```
174 // This is a hack and ugly but there are stack issues in `
    ↪ LiquidateCurrencyAction.liquidateCollateralCurrency`
```

3.460 CVF-460

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description This conversion is redundant.

Listing 460:

```
191 int256(maxNTokenLiquidation)
```

3.461 CVF-461

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description This expression is calculated twice. Consider calculating once and reusing.

Listing 461:

```
204 requiredCollateralAssetCash.sub(collateralAssetRemaining),
    requiredCollateralAssetCash.sub(collateralAssetRemaining)
```

3.462 CVF-462

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidateCurrency.sol

Description The format of this value is unclear. Consider explaining in the documentation comment.

Listing 462:

```
223 int256 liquidationDiscount
```

3.463 CVF-463

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description The conversion to the "int256" type is redundant.

Listing 463:

```
289 int256(uint8(factors.nTokenParameters[Constants.
    ↪ LIQUIDATION_HAIRCUT_PERCENTAGE]));

291 int256(uint8(factors.nTokenParameters[Constants.
    ↪ PV_HAIRCUT_PERCENTAGE]));
```

3.464 CVF-464

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidateCurrency.sol

Description This structure should probably be moved to the "Types.sol" file.

Listing 464:

```
321 struct WithdrawFactors {
```

3.465 CVF-465

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description These two functions have very much in common. Consider extracting common parts into utility functions to reduce code duplication.

Listing 465:

```
331 function _withdrawLocalLiquidityTokens(

438 function _withdrawCollateralLiquidityTokens(
```

3.466 CVF-466

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidateCurrency.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 466:

```
342 for (uint256 i; i < portfolioState.storedAssets.length; i++) {

447 for (uint256 i; i < portfolioState.storedAssets.length; i++) {

527     for (uint256 i; i < markets.length; i++) {
```

3.467 CVF-467

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description These two conditional statements could be merged into one.

Listing 467:

```
344 if (asset.storageState == AssetStorageState.Delete) continue;
    if (
        !AssetHandler.isLiquidityToken(asset.assetType) ||
        asset.currencyId != factors.cashGroup.currencyId
    ) continue;
```

3.468 CVF-468

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description The expression "marketIndex - 1" is calculated several times. Consider calculating once and reusing.

Listing 468:

```
354         factors.markets[marketIndex - 1],

363     (w.assetCash, w.fCash) = asset.getCashClaims(factors.markets[
            ↪ marketIndex - 1]);

370         factors.markets[marketIndex - 1].removeLiquidity(asset.
                ↪ notional);

385         (w.assetCash, w.fCash) = factors.markets[marketIndex - 1].
                ↪ removeLiquidity(
```

3.469 CVF-469

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description The expression "w.netCashIncrease.sub(w.incentivePaid)" is calculated twice. Consider calculating once and reusing.

Listing 469:

```
367     if (w.netCashIncrease.subNoNeg(w.incentivePaid) <=
            ↪ assetAmountRemaining) {

377             w.netCashIncrease.sub(w.incentivePaid);

382                 w.netCashIncrease.subNoNeg(w.incentivePaid)
```

3.470 CVF-470

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description This function updates the passed in-memory structure with the calculated values instead of returning them. Such behavior makes the code harder to read. Consider returning the calculated values and storing them into an in-memory structure on the caller's side.

Listing 470:

```
416 function _calculateNetCashIncreaseAndIncentivePaid(
```

3.471 CVF-471

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** LiquidateCurrency.sol

Description Overflow is possible here. Consider using safe conversion.

Listing 471:

```
427 int256 haircut = int256(factors.cashGroup.getLiquidityHaircut(
    ↪ assetType));
```

3.472 CVF-472

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidateCurrency.sol

Description The semantics of the returned value is unclear. Consider giving a descriptive name to the returned value and/or describing it in the documentation comment.

Listing 472:

```
443 ) internal view returns (int256) {
```

3.473 CVF-473

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description These two conditional operators could be merged into one.

Listing 473:

```
449 if (asset.storageState == AssetStorageState.Delete) continue;
450 if (
        !AssetHandler.isLiquidityToken(asset.assetType) ||
        asset.currencyId != factors.cashGroup.currencyId
    ) continue;
```

3.474 CVF-474

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description The expression "marketIndex - 1" is calculated several times. Consider calculating once and reusing.

Listing 474:

```
459 factors.markets[marketIndex - 1],

465 asset.getCashClaims(factors.markets[marketIndex - 1]);

470 factors.markets[marketIndex - 1].removeLiquidity(asset.notional)
      ↪ ;

478 (cashClaim, fCashClaim) = factors.markets[marketIndex - 1].
      ↪ removeLiquidity(
```

3.475 CVF-475

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BalanceHandler.sol

Description The "currencyId" parameter should be indexed.

Listing 475:

```
19 event CashBalanceChange(address indexed account, uint16
     ↪ currencyId, int256 netCashChange);

21 event nTokenSupplyChange(address indexed account, uint16
     ↪ currencyId, int256 tokenSupplyChange);
```

3.476 CVF-476

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description Two values returned by this function are always equal or equal up to round-off errors. Consider returning one value instead of two.

Listing 476:

```
30 ///  - assetAmountInternal which is the converted asset amount
     ↪ accounting for transfer fees
   ///  - assetAmountTransferred which is the internal precision
     ↪ amount transferred into the account
```

3.477 CVF-477

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BalanceHandler.sol

Description This is basically equivalent to: assetAmountInternal = token.convertToInternal (assetAmountExternalPrecisionFinal); thus, equals to assetAmountTransferred up to round-off errors.

Listing 477:

```
56 assetAmountInternal = assetAmountInternal.sub(
       token.convertToInternal(assetAmountExternal.sub(
         ↪ assetAmountExternalPrecisionFinal))
```

3.478 CVF-478

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BalanceHandler.sol

Description The rest of the function looks like it is always executed, while it is actually executed only when the token doesn't have transfer fee and farce transfer flag is false. Consider putting the code below into an explicit "else" branch for readability.

Listing 478:

```
61 }
```

3.479 CVF-479

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description The assigned value is exactly the same as "assetAmountInternal".

Listing 479:

```
66 assetAmountTransferred = token.convertToInternal(
    ↪ assetAmountExternal);
```

3.480 CVF-480

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description As "underlyingAmountExternal" is guaranteed to be positive here, it would be safer to convert it to uint256, rather than to convert "msg.value" to int256.

Listing 480:

```
89 require(underlyingAmountExternal == int256(msg.value), "Invalid
    ↪ ETH balance");
```

3.481 CVF-481

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BalanceHandler.sol

Description This relies on the assumption, that a value returned by the "underlyingToken.transfer" call above may never be negative. Probably. Relying on such assumption make the code more fragile. Consider adding an explicit assert to check this assumption.

Listing 481:

```
98 assetToken.mint(uint256(underlyingAmountExternal));
```

3.482 CVF-482

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BalanceHandler.sol

Description The messages are misleading. One would think that it was an attempt to withdraw a negative amount, while the problem is different. Consider fixing the messages.

Listing 482:

```
124 "BH: cannot withdraw negative"

133 "BH: cannot withdraw negative"
```

3.483 CVF-483

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description This check should be done inside the "_finalizeTransfers" function after converting the amount to external precision. Otherwise it is possible that non-zero internal amount will be converted to zero external amount.

Listing 483:

```
137 if (balanceState.netAssetTransferInternalPrecision != 0) {
```

3.484 CVF-484

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description Should be balanceState.netCashChange.add(balanceState.netAssetTransferInternalPrecision) != 0 The current conditions evaluates to true when both values are non-zero and their sum is zero, however in such a case there is no need to update the stored balance.

Listing 484:

```
142 balanceState.netCashChange != 0 || balanceState.
    ↪ netAssetTransferInternalPrecision != 0
```

3.485 CVF-485

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description This sum is calculated twice. Consider calculating once and reusing.

Listing 485:

```
154 balanceState.netCashChange.add(balanceState.
    ↪ netAssetTransferInternalPrecision)
```

3.486 CVF-486

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description Should be balanceState.netNTokenTransfer.add(balanceState.netNTokenSupplyChange) != 0 The current conditions evaluates to true when both values are non-zero, but their sum is zero, however in such a case there is no need to update the stored nToken balance.

Listing 486:

```
158 if (balanceState.netNTokenTransfer != 0 || balanceState.
    ↪ netNTokenSupplyChange != 0) {
```

3.487 CVF-487

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description This line does nothing. Consider removing it.

Listing 487:

```
204 return transferAmountExternal;
```

3.488 CVF-488

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description This check is too restrictive. In case assetTransferAmountExternal >= 0, the "redeemToUnderlying" flag should just be ignored.

Listing 488:

```
223 require(assetTransferAmountExternal < 0); // dev: invalid redeem
    ↪ balance
```

3.489 CVF-489

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description These two calls could be merged into one placed after the conditional statement.

Listing 489:

```
233 actualTransferAmountExternal = underlyingToken.transfer(
        account,
        underlyingAmountExternal.neg()
    );

238 assetTransferAmountExternal = assetToken.transfer(account,
    ↪ assetTransferAmountExternal);
```

3.490 CVF-490

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description The value "cashBalance.neg()" is used in both branches of the conditional statement. Consider calculating in one place, before the conditional statement.

Listing 490:

```
267 amountToSettleAsset = cashBalance.neg();

271 require(amountToSettleAsset <= cashBalance.neg(), "Invalid
    ↪ amount to settle");
```

3.491 CVF-491

- **Severity** Critical
- **Status** Opened
- **Category** Procedural
- **Source** BalanceHandler.sol

Description Unlike other similar places, the value of "nTokenBalance" is ignore here. Should be "cashBalance == 0 && nTokenBalance == 0".

Listing 491:

```
277 if (cashBalance == 0) {
```

3.492 CVF-492

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BalanceHandler.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 492:

```
306 for (uint256 i; i < settleAmounts.length; i++) {
```

3.493 CVF-493

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description The expression "settleAmounts[i]" is calculated several times. Consider calculating once and reusing.

Listing 493:

```
307 if (settleAmounts[i].netCashChange == 0) continue;

314 ) = getBalanceStorage(account, settleAmounts[i].currencyId);

316 cashBalance = cashBalance.add(settleAmounts[i].netCashChange);

318     settleAmounts[i].currencyId,

329     uint16(settleAmounts[i].currencyId),
330     settleAmounts[i].netCashChange

335     settleAmounts[i].currencyId,
```

3.494 CVF-494

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BalanceHandler.sol

Description This could be simplified as: accountContext.hasDebt |= Constants.HAS_CASH_DEBT;

Listing 494:

```
324 accountContext.hasDebt = accountContext.hasDebt | Constants.
    ↪ HAS_CASH_DEBT;
```

3.495 CVF-495

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description This function is probably too simple to be extracted.

Listing 495:

```
345 function setBalanceStorageForNToken(
```

3.496 CVF-496

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** BalanceHandler.sol

Description This function emulates storage address calculation logic usually performed by Solidity compiler. Why to do this manually?

Listing 496:

```
363 function _getSlot(address account, uint256 currencyId) private
    ↪ pure returns (bytes32) {
```

3.497 CVF-497

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description The first check is redundant as lastClaimTime is unsigned.

Listing 497:

```
386 require(lastClaimTime >= 0 && lastClaimTime <= type(uint32).max)
    ↪ ; // dev: last claim time overflow
```

3.498 CVF-498

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description The first check is redundant as lastClaimSupply is unsigned.

Listing 498:

```
390 require(lastClaimSupply >= 0 && lastClaimSupply <= type(uint56).
    ↪ max); // dev: last claim supply overflow
```

3.499 CVF-499

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description The final conversions to the "int256" and "uint256" types are redundant.

Listing 499:

```
421 nTokenBalance = int256(uint80(uint256(data)));
    lastClaimTime = uint256(uint32(uint256(data >> 80)));
    lastClaimSupply = uint256(uint56(uint256(data >> 112)));
    cashBalance = int256(int88(int256(data >> 168)));
```

3.500 CVF-500

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** BalanceHandler.sol

Description It is not checked that the currency ID fits into 16 bits or doesn't exceed the maximum registered currency ID. Consider adding such checks.

Listing 500:

```
436 require(currencyId != 0, "BH: invalid currency id");
```

3.501 CVF-501

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BalanceHandler.sol

Description This function not only claims incentives, but also updates the stored balance state. Consider reflecting this fact in the function name and in the documentation comment.

Listing 501:

```
460 function claimIncentivesManual(BalanceState memory balanceState,
    ↪  address account)
```

3.502 CVF-502

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description This function implements storage address calculation in a similar way to how Solidity does it. Using normal mapping would make the code simple and less error-prone.

Listing 502:

```
18 function _getSlot(uint256 currencyId, bool underlying) private
   ↪ pure returns (bytes32) {
```

3.503 CVF-503

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description This expression have only two possible values: one for underlying=false and another for underlying=true. These two values could be precomputed and made compile-time constants.

Listing 503:

```
23 keccak256(abi.encode(underlying, Constants.TOKEN_STORAGE_OFFSET)
   ↪ )
```

3.504 CVF-504

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The conversion to the "address" type does a shift, so there are two shifts in total, while zero shifts are actually required. Consider rewriting the code like this: address tokenAddress = address(uint256(data));

Listing 504:

```
37 address tokenAddress = address(bytes20(data << 96));
```

3.505 CVF-505

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The conversion to the "uint8" type performs a shift, so there are two shifts in total, while only one shift is actually needed. Consider rewriting the like like this: uint8 tokenDecimalPlaces = uint8(unit256(data » 168));

Listing 505:

```
39 uint8 tokenDecimalPlaces = uint8(bytes1(data << 80));
```

3.506 CVF-506

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The conversion to the ""uint8"" type performs a shift, so there are two shifts in total, while only one shift is actually needed. Consider rewriting the like like this: TokenType tokenType = TokenType(unit256(data » 176));

Listing 506:

```
40 TokenType tokenType = TokenType(uint8(bytes1(data << 72)));
```

3.507 CVF-507

- **Severity** Major
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** TokenHandler.sol

Description Overflow is possible here. Consider using safe power and conversion, and/or asserting that "tokenDecimalPlaces" is not too big.

Listing 507:

```
46 decimals: int256(10**tokenDecimalPlaces),
```

3.508 CVF-508

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The conversion to the "bytes20" type performs a shift, so two shifts are performed, while no shifts are actually needed. Also, performing a bitwise "or" operation whose one argument is zero doesn't make much sense, so consider jusr removing this part of the expression.

Listing 508:

```
62 ((bytes32(bytes20(address(0))) >> 96) |
```

3.509 CVF-509

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The conversion to the "bytes1" type is redundant, because the type of the "BOOL_FALSE" constant is already "bytes1".

Listing 509:

```
63 (bytes32(bytes1(Constants.BOOL_FALSE)) >> 88) |
```

3.510 CVF-510

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TokenHandler.sol

Description The code below this line looks like it is always executed, while it is executed only when the token type is not ether or the currency ID is not ETH_CURRENCY_ID. Consider putting the rest of the function into explicit "else" branch.

Listing 510:

```
72 }
```

3.511 CVF-511

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The conversion to "bytes20" performs a shift, so there are two shifts in total, while no shifts are actually needed. Consider rewriting this part of the expression like this: bytes32 (uint256 (tokenStorage.tokenAddress))

Listing 511:

```
100 ((bytes32(bytes20(tokenStorage.tokenAddress)) >> 96) |
```

3.512 CVF-512

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** TokenHandler.sol

Description The zero value here is actually "NO_ERROR" constant from Compound protocol. Consider turning it into a named constant to make code easier to read.

Listing 512:

```
124 require(success == 0, "TH: mint failure");
146 require(success == 0, "Redeem failure");
```

3.513 CVF-513

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** TokenHandler.sol

Description Overflow is possible when converting to int256. Consider using safe conversion.

Listing 513:

```
128 return int256(endingBalance.sub(startingBalance));
156 return int256(endingBalance.sub(startingBalance));
200     return int256(endingBalance.sub(startingBalance));
204 return int256(amount);
```

3.514 CVF-514

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description This reverts in case "netTransferExternal" is zero. Just doing nothing in such as case would make the function more convenient to use.

Listing 514:

```
170 require(netTransferExternal < 0); // dev: cannot transfer ether
```

3.515 CVF-515

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TokenHandler.sol

Description Usage of "transfer" is discouraged. There are better ways to prevent reentrancy.

Listing 515:

```
174 accountPayable.transfer(uint256(netTransferExternal.neg()));
```

3.516 CVF-516

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** TokenHandler.sol

Description The code below this line looks like it is always executed, while it is executed only when the token doesn't have transfer fee. Consider putting the rest of the function into explicit "else" branch.

Listing 516:

```
201 }
```

3.517 CVF-517

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** TokenHandler.sol

Description These functions always round down. A good practice is to always round toward the protocol, i.e. against the user. Consider implementing two versions of each function, one that round up and another that round down, and choose the appropriate function for each particular case.

Listing 517:

```
207 function convertToInternal(Token memory token, int256 amount)
      ↪ internal pure returns (int256) {

212 function convertToExternal(Token memory token, int256 amount)
      ↪ internal pure returns (int256) {
```

3.518 CVF-518

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description The mul+div approach is expensive and is prone to phantom overflows when the final result would fit into the destination type, while some intermediary calculations overflow. In case token.decimals > INTERNAL_TOKEN_PRECISION, a single division would be enough, and in case token.decimals < INTERNAL_TOKEN_PRECISION, a single multiplication is enough, but we never need both.

Listing 518:

```
209 return amount.mul(Constants.INTERNAL_TOKEN_PRECISION).div(token.
      ↪ decimals);

214 return amount.mul(token.decimals).div(Constants.
      ↪ INTERNAL_TOKEN_PRECISION);
```

3.519 CVF-519

- **Severity** Major
- **Status** Opened
- **Category** Unclear behavior
- **Source** TokenHandler.sol

Description In assembly, "not" is a bitwise inversion, so "not(0)" is basically 2^256 - 1. In Solidity, boolean "true" is internally represented as 1, not 2^256 -1. Consider changing "not(0)" to "1" here.

Listing 519:

```
245 success := not(0) // set success to true
```

3.520 CVF-520

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** TokenHandler.sol

Description This overwrites the first 32 bytes of the memory. Consider allocating a static in-memory array:
uint [1] memory result; assembly { returndatacopy (result, 0, 32) }

Listing 520:

```
249 returndatacopy(0, 0, 32)
```

3.521 CVF-521

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Incentives.sol

Description This basically calculates (timeSinceLastClaim * 1e8 / 31104000 * 50 / 100 + 1e8) * (timeSinceLastClaim * 1e8 / 31104000) * emissionRatePerYear. The constant expressions "1e8 / 31104000 * 50 / 100" and "1e8 / 31104000" could be precomputed with reasonable precision. No need to calculate the every time.

Listing 521:

```
25 uint256 proRataYears =
       timeSinceLastClaim.mul(uint256(Constants.
          ↪ INTERNAL_TOKEN_PRECISION)).div(Constants.YEAR);

30     proRataYears
           .mul(Constants.ANNUAL_INCENTIVE_MULTIPLIER_PERCENT)
           .div(uint256(Constants.PERCENTAGE_DECIMALS))
           .add(uint256(Constants.INTERNAL_TOKEN_PRECISION));
```

3.522 CVF-522

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Incentives.sol

Description Here multiplication is used after division, thus rounding errors produced by the division are multiplied as well. Consider doing division only once at the end of calculation to enhance precision.

Listing 522:

```
25 uint256 proRataYears =
       timeSinceLastClaim.mul(uint256(Constants.
          ↪ INTERNAL_TOKEN_PRECISION)).div(Constants.YEAR);

30     proRataYears
           .mul(Constants.ANNUAL_INCENTIVE_MULTIPLIER_PERCENT)
```

3.523 CVF-523

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Incentives.sol

Description The number format for annual emission rate is unclear. Consider documenting it.

Listing 523:

```
67 // Convert this to the appropriate denomination
   emissionRatePerYear.mul(uint256(Constants.
      ↪ INTERNAL_TOKEN_PRECISION))
```

3.524 CVF-524

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** Incentives.sol

Description Using the average total supply is only slightly better then the original or current total supply, and could be manipulated. People are still intensified to claim when the total supply is elevated. The common approach is to store the integral total supply, i.e. the number that increases every second by the current total supply. If S0 is the integral total supply at the begin of an interval, S1 is the integral total supply at the end of the interval, and T it the interval length in seconds, then the time average total supply for this period is (S1 - S0) / T.

Listing 524:

```
71 // Returns the average supply between now and the previous mint
      ↪ time. This is done to dampen the effect of
   // total supply fluctuations when claiming tokens. For example,
      ↪ if someone minted nTokens when the supply was
   // at 100e8 and then claimed incentives when the supply was at
      ↪ 100_000e8, they would be diluted out of part of
   // their token incentives. This will ensure that they claim with
      ↪  an average supply of 50050e8, which is better
   // than not doing the average
   uint256 avgTotalSupply =
       totalSupply.add(lastClaimSupply.mul(uint256(Constants.
          ↪ INTERNAL_TOKEN_PRECISION))).div(
           2
       );
```

3.525 CVF-525

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Incentives.sol

Description As Constants.INTEGRAL_TOKEN_PRECISION is even, it would be better to just multiply by half of Constants.INTEGRAL_TOKEN_PRECISION, thus no division would be needed.

Listing 525:

```
77 totalSupply.add(lastClaimSupply.mul(uint256(Constants.
      ↪ INTERNAL_TOKEN_PRECISION))).div(
       2
   );
```

3.526 CVF-526

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description This function implements a mapping of structs in a way similar to how Solidity implements mappings. Consider using Solidity mappings instead.

Listing 526:

```
31  function getNTokenContext(address tokenAddress)

56  function nTokenAddress(uint256 currencyId) internal view returns
      ↪  (address tokenAddress) {

65  function setNTokenAddress(uint16 currencyId, address
      ↪ tokenAddress) internal {

92  function setNTokenCollateralParameters(

129 function changeNTokenSupply(address tokenAddress, int256
      ↪ netChange) internal returns (uint256) {

154 function setIncentiveEmissionRate(address tokenAddress, uint32
      ↪ newEmissionsRate) internal {

169 function setArrayLengthAndInitializedTime(

204 function setDepositParameters(

235 function setInitializationParameters(

261 function getInitializationParameters(uint256 currencyId, uint256
      ↪  maxMarketIndex)
```

3.527 CVF-527

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenHandler.sol

Description This code repeats several times. Consider extracting a utility function.

Listing 527:

```
42  bytes32 slot = keccak256(abi.encode(tokenAddress, Constants.
      ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));

68  bytes32 currencySlot =
        keccak256(abi.encode(tokenAddress, Constants.
          ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));

100 bytes32 slot = keccak256(abi.encode(tokenAddress, Constants.
      ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));

130 bytes32 slot = keccak256(abi.encode(tokenAddress, Constants.
      ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));

155 bytes32 slot = keccak256(abi.encode(tokenAddress, Constants.
      ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));

174 bytes32 slot = keccak256(abi.encode(tokenAddress, Constants.
      ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));
```

3.528 CVF-528

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The final conversions to the “uint256” type are redundant.

Listing 528:

```
48  currencyId = uint256(uint16(uint256(data)));
    totalSupply = uint256(uint96(uint256(data >> 16)));
50  incentiveAnnualEmissionRate = uint256(uint32(uint256(data >>
      ↪ 112)));
    lastInitializedTime = uint256(uint32(uint256(data >> 144)));
```

3.529 CVF-529

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** nTokenHandler.sol

Description There are no range checks for the arguments. Consider explicitly checking that both arguments are non-zero, as zero values have special meaning.

Listing 529:

```
65 function setNTokenAddress(uint16 currencyId, address
     ↪ tokenAddress) internal {
```

3.530 CVF-530

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The currency slot address should be calculated only when the address slot is empty.

Listing 530:

```
68 bytes32 currencySlot =
       keccak256(abi.encode(tokenAddress, Constants.
         ↪ NTOKEN_CONTEXT_STORAGE_OFFSET));
```

3.531 CVF-531

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description "This variable is redundant, just do: data |= bytes32 (uint256 (residualPurchaseIncentive10BPS)) « 184 | bytes32 (uint256 (pvHaircutPercentage)) « 192 | ...;"

Listing 531:

```
116 bytes32 parameters =
```

3.532 CVF-532

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** nTokenHandler.sol

Description The semantics of the returned value is unclear. Consider giving it a descriptive name and/or describing in the documentation comment.

Listing 532:

```
129 function changeNTokenSupply(address tokenAddress, int256
    ↪ netChange) internal returns (uint256) {
```

3.533 CVF-533

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The conversion to the "int256" type is redundant.

Listing 533:

```
135 int256 totalSupply = int256(uint96(uint256(data >> 16)));
```

3.534 CVF-534

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The conversion to the "uint256" type is redundant.

Listing 534:

```
139 newSupply >= 0 && uint256(newSupply) < type(uint96).max,
```

3.535 CVF-535

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenHandler.sol

Description This code repeats twice. Consider extracting a utility function.

Listing 535:

```
196 uint256 slot =
        uint256(keccak256(abi.encode(currencyId, Constants.
            ↪ NTOKEN_DEPOSIT_STORAGE_OFFSET)));

209 uint256 slot =
210     uint256(keccak256(abi.encode(currencyId, Constants.
            ↪ NTOKEN_DEPOSIT_STORAGE_OFFSET)));
```

3.536 CVF-536

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The slot address calculations should be done later, just before passing the slot address to the "_setParameters" call.

Listing 536:

```
209 uint256 slot =
210     uint256(keccak256(abi.encode(currencyId, Constants.
            ↪ NTOKEN_DEPOSIT_STORAGE_OFFSET)));

240 uint256 slot =
        uint256(keccak256(abi.encode(currencyId, Constants.
            ↪ NTOKEN_INIT_STORAGE_OFFSET)));
```

3.537 CVF-537

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** nTokenHandler.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 537:

```
219 for (uint256 i; i < depositShares.length; i++) {

246 for (uint256 i; i < rateAnchors.length; i++) {

284 for (uint256 i; i < maxMarketIndex; i++) {

313 uint256 i;

432     for (uint256 i; i < nToken.portfolioState.storedAssets.
        ↪ length; i++) {
```

3.538 CVF-538

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The expression "leverageThresholds[i]" is calculated twice. Consider calculating once and reusing.

Listing 538:

```
223 leverageThresholds[i] > 0 && leverageThresholds[i] < Constants.
    ↪ RATE_PRECISION,
```

3.539 CVF-539

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** nTokenHandler.sol

Description While negative interest rates are weird, zero interest rate is actually possible. Should be ">=" here.

Listing 539:

```
247 // Rate anchors are exchange rates and therefore must be greater
    ↪  than RATE_PRECISION
// or we will end up with negative interest rates
require(rateAnchors[i] > Constants.RATE_PRECISION, "PT: invalid
    ↪ rate anchor");
```

3.540 CVF-540

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The expression "proportions[i]" is calculated twice. Consider calculating once and reusing.

Listing 540:

```
252 proportions[i] > 0 && proportions[i] < Constants.RATE_PRECISION,
```

3.541 CVF-541

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description These variables are redundant. Just give names to the returned values and use them instead.

Listing 541:

```
282 int256[] memory array1 = new int256[](maxMarketIndex);
    int256[] memory array2 = new int256[](maxMarketIndex);
```

3.542 CVF-542

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description Consider unrolling the loop for efficiency.

Listing 542:

```
284 for (uint256 i; i < maxMarketIndex; i++) {
```

3.543 CVF-543

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description The conversions to the "int256" type are redundant.

Listing 543:

```
285 array1[i] = int256(uint32(uint256(data)));

287 array2[i] = int256(uint32(uint256(data)));
```

3.544 CVF-544

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description Performing this check on every loop iteration is suboptimal. Consider splitting the loop into two loops: one for i = 0..3 and another for i = 4..maxMarketIndex - 1.

Listing 544:

```
294 if (i == 3) {
```

3.545 CVF-545

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** nTokenHandler.sol

Description There is no check to ensure that these values have the same lengths.

Listing 545:

```
308 uint32[] calldata array1,
    uint32[] calldata array2
```

3.546 CVF-546

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** nTokenHandler.sol

Description It is not checked that the array length doesn't exceed the maximum number of markets.

Listing 546:

```
314 for (; i < array1.length; i++) {
```

3.547 CVF-547

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** nTokenHandler.sol

Description Overflow is possible here. Consider using safe conversion.

Listing 547:

```
358 nToken.totalSupply = int256(totalSupply);
```

3.548 CVF-548

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** nTokenHandler.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing in the documentation comment.

Listing 548:

```
403 returns (int256, bytes32)
```

3.549 CVF-549

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description These two functions implement a mapping of struct is a way similar to how Solidity compiler does this. Consider using normal Solidity mapping.

Listing 549:

```
16 function getAccountContext(address account) internal view
     ↪ returns (AccountContext memory) {

35 function setAccountContext(AccountContext memory accountContext,
     ↪  address account) internal {
```

3.550 CVF-550

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description This check should be done in the very beginning of the function.

Listing 550:

```
97 require(
       currencyId != 0 && currencyId <= Constants.MAX_CURRENCIES,
       "AC: invalid currency id"
100 );
```

3.551 CVF-551

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description The conversion to the "uint256" type is redundant.

Listing 551:

```
105 uint256 cid = uint256(uint16(bytes2(currencies) & Constants.
    ↪ UNMASK_FLAGS));
```

3.552 CVF-552

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description This flag should be calculated only when cid == currencyId.

Listing 552:

```
106 bool isActive =
        bytes2(currencies) & Constants.ACTIVE_IN_BALANCES ==
        ↪ Constants.ACTIVE_IN_BALANCES;
```

3.553 CVF-553

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description This function is overcomplicated and suboptimal. It could be optimized using binary search instead of linear search. Also, it could be split into two functions: one to add an active currency, and another to remove one. Here is code sample that illustrates the idea: https://gist.github.com/3sGgpQ8H/0f52aa464ee58caed64edb78614b135a

Listing 553:

```
122 function setActiveCurrency(
```

3.554 CVF-554

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description Splitting this function into two functions: one that adds a currency and another that removes a currency, would make code simpler and more efficient.

Listing 554:

```
125 bool isActive,
```

3.555 CVF-555

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** AccountContextHandler.sol

Description These variables are not initialized. Consider explicitly initializing to zero.

Listing 555:

```
137 bytes18 prefix;

139 uint256 shifts;
```

3.556 CVF-556

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description The conversion to the "uint256" type is redundant.

Listing 556:

```
155 uint256 cid = uint256(uint16(bytes2(suffix) & Constants.
    ↪ UNMASK_FLAGS));
```

3.557 CVF-557

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description Increasing the "shifts" variable by 16 instead of 1 would make the multiplications unnecessary.

Listing 557:

```
161         (bytes18(flags) >> (shifts * 16));

170     accountContext.activeCurrencies = prefix | (suffix >> (
          ↪ shifts * 16));

176     prefix = prefix | (bytes18(bytes2(uint16(currencyId)) |
          ↪ flags) >> (shifts * 16));

185     accountContext.activeCurrencies = prefix | (suffix >> ((
          ↪ shifts + 1) * 16));

192 prefix = prefix | (bytes18(bytes2(suffix)) >> (shifts * 16));

194 shifts += 1;

219     result = result | (bytes18(bytes2(suffix)) >> (shifts * 16))
          ↪ ;
220     shifts += 1;
```

3.558 CVF-558

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** AccountContextHandler.sol

Description This check should be done before the previous line.

Listing 558:

```
177 // check that the total length is not greater than 9
    require(
        accountContext.activeCurrencies[16] == 0x00 &&
180         accountContext.activeCurrencies[17] == 0x00,
        "AC: too many currencies"
    );
```

3.559 CVF-559

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description This check could be optimized by using bitwise operations.

Listing 559:

```
179             accountContext.activeCurrencies[16] == 0x00 &&
180                 accountContext.activeCurrencies[17] == 0x00,

202 accountContext.activeCurrencies[16] == 0x00 &&
        accountContext.activeCurrencies[17] == 0x00,
```

3.560 CVF-560

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description The function could be optimized by removing unused currencies in-place. Here the a code sample that illustrates the idea: https://gist.github.com/3sGgpQ8H/dc68293adc8d9f39b50e3d731506cb61

Listing 560:

```
211 function _clearPortfolioActiveFlags(bytes18 activeCurrencies)
      ↪ internal pure returns (bytes18) {
```

3.561 CVF-561

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Description Consider unrolling this loop for efficiency.

Listing 561:

```
216 while (suffix != 0x00) {
```

3.562 CVF-562

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** AccountContextHandler.sol

Description The code is very confusing, as the "ASSET_DEBT" bit is not mentioned in the code. Consider rewriting as "... & Constants.HAS_ASSET_DEBT".

Listing 562:

```
246 // Turns off the ASSET_DEBT flag
accountContext.hasDebt = accountContext.hasDebt & Constants.
    ↪ HAS_CASH_DEBT;
```

3.563 CVF-563

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** ExchangeRate.sol

Description Phantom overflows are possible here. Consider using muldiv function or other safe approach: https://2π.com/21/muldiv/index.html

Listing 563:

```
26      balance.mul(er.rate).mul(multiplier).div(Constants.
            ↪ PERCENTAGE_DECIMALS).div(
              er.rateDecimals
        );

42  int256 result = balance.mul(er.rateDecimals).div(er.rate);

56  return baseER.rate.mul(quoteER.rateDecimals).div(quoteER.rate);
```

3.564 CVF-564

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ExchangeRate.sol

Description This code reimplements Solidity mapping. Consider just passing a storage reference to the ETH rate mapping and using plain Solidity code.

Listing 564:

```
61 bytes32 slot = keccak256(abi.encode(currencyId,
     ↪ ETH_RATE_STORAGE_SLOT));
   bytes32 data;

64 assembly {
       data := sload(slot)
   }
```

3.565 CVF-565

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ExchangeRate.sol

Description The conversion to the "address" type performs a shift, so two shifts are preformed, while no shifts are actually needed. Consider simplifying like this: address rateOracle = address (uint160 (uint256 (data)));

Listing 565:

```
76 address rateOracle = address(bytes20(data << 96));
```

3.566 CVF-566

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ExchangeRate.sol

Description The conversion to the "uint8" type performs a shift, so two shifts are performed in total, while one shift would be enough. Consider simplifying like this: uint8 rateDecimalPlaces = uint8 (uint256 (data » 160))

Listing 566:

```
87 uint8 rateDecimalPlaces = uint8(bytes1(data << 88));
```

3.567 CVF-567

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** ExchangeRate.sol

Description Overflow is possible when calculating power and then when converting to the "int256" type. Consider range checking the "rateDecimalPlaces" before the calculation.

Listing 567:

```
88 rateDecimals = int256(10**rateDecimalPlaces);
```

3.568 CVF-568

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ExchangeRate.sol

Description The final conversions to the "int256" type are redundant.

Listing 568:

```
96 int256 buffer = int256(uint8(bytes1(data << 72)));
   int256 haircut = int256(uint8(bytes1(data << 64)));
   int256 liquidationDiscount = int256(uint8(bytes1(data << 56)));
```

3.569 CVF-569

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AssetHandler.sol

Description The comment is confusing as the function doesn't actually multiply the exponent output by the notional value. Consider removing "notional *" from the comment.

Listing 569:

```
42 /// The formula is: notional * e^(-rate * timeToMaturity).
   function getDiscountFactor(uint256 timeToMaturity, uint256
      ↪ oracleRate)
```

3.570 CVF-570

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** AssetHandler.sol

Description This line could be simplified as: int128 expValue = ABDKMath64x64.divu (oracleRate.mul (timeToMaturity), Constants.IMPLIED_RATE_TIME.mul (Constants.RATE_PRECISION));

Listing 570:

```
48 int128 expValue =
       ABDKMath64x64.fromUInt(oracleRate.mul(timeToMaturity).div(
           ↪ Constants.IMPLIED_RATE_TIME));
50 expValue = ABDKMath64x64.div(expValue, Constants.
       ↪ RATE_PRECISION_64x64);
```

3.571 CVF-571

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AssetHandler.sol

Description It would be safer to use the "ABDKMath64x64.neg" function instead of multiplication by -1.

Listing 571:

```
51 expValue = ABDKMath64x64.exp(expValue * -1);
```

3.572 CVF-572

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** AssetHandler.sol

Description There lines could be simplified as: return ABDKMath64x64.muli (expValue, Constants.RATE_PRECISION);

Listing 572:

```
52 expValue = ABDKMath64x64.mul(expValue, Constants.
       ↪ RATE_PRECISION_64x64);
   int256 discountFactor = ABDKMath64x64.toInt(expValue);

   return discountFactor;
```

3.573 CVF-573

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** AssetHandler.sol

Description The function names are confusing as it is not clear whether the functions estimates fCash or liquidity token value. Consider renaming the function to "getPresentfCashValue" and "getRiskAdustedPresentfCashValue".

Listing 573:

```
58 /// @notice Present value of an fCash asset without any risk
   ↪ adjustments.
   function getPresentValue(

74 /// @notice Present value of an fCash asset with risk
   ↪ adjustments. Positive fCash value will be discounted more
   /// heavily than the oracle rate given and vice versa for
   ↪ negative fCash.
   function getRiskAdjustedPresentValue(
```

3.574 CVF-574

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** AssetHandler.sol

Description Phantom overflows are possible here. Consider using muldiv function or other safe approach: https://2π.com/21/muldiv/index.html

Listing 574:

```
71  return notional.mul(discountFactor).div(Constants.RATE_PRECISION
    ↪ );

103 return notional.mul(discountFactor).div(Constants.RATE_PRECISION
    ↪ );

114 assetCash = market.totalAssetCash.mul(token.notional).div(market
    ↪ .totalLiquidity);
    fCash = market.totalfCash.mul(token.notional).div(market.
    ↪ totalLiquidity);

147 return numerator.mul(tokens).mul(haircut).div(Constants.
    ↪ PERCENTAGE_DECIMALS).div(liquidity);
```

3.575 CVF-575

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AssetHandler.sol

Description Then, the "getLiquidityHaircut" function should return uint8.

Listing 575:

```
128 // This won't overflow, the liquidity token haircut is stored as
    ↪ an uint8
int256 haircut = int256(cashGroup.getLiquidityHaircut(token.
    ↪ assetType));
```

3.576 CVF-576

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** AssetHandler.sol

Description In some cases, the function adds the present fCash value to the portfolio instead of returning it. Consider explaining this in the documentation comment. Alternatively consider always returning the fCash value without converting it to cash, so the caller could add it to the fCash notional amount and only then estimate the cash value.

Listing 576:

```
150 /// @notice Returns the asset cash claim and the present value
    ↪ of the fCash asset (if it exists)
```

3.577 CVF-577

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** AssetHandler.sol

Description Consider giving descriptive names to the returned values to make the code more readable.

Listing 577:

```
158 ) internal view returns (int256, int256) {
```

3.578 CVF-578

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** AssetHandler.sol

Description Such tricks are very error-prone. Consider avoiding them.

Listing 578:

```
193 // WARNING: this modifies the portfolio in memory and therefore
    ↪ we cannot store this portfolio!
```

3.579 CVF-579

- **Severity** Major
- **Status** Opened
- **Category** Documentation
- **Source** AssetHandler.sol

Description The comment is misleading, as the second returned value is not the present value of the fCash asset, but rather the updated portfolio index. consider fixing the comment and/or giving descriptive names to the returned values.

Listing 579:

```
218 /// @notice Returns the asset cash claim and the present value
    ↪ of the fCash asset (if it exists)

225 ) internal view returns (int256, uint256) {
```

3.580 CVF-580

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** FreeCollateral.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing the returned values in the documentation comment.

Listing 580:

```
42 returns (int256, int256)
```

3.581 CVF-581

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateral.sol

Description Conversion to the "uint16" type performs a shift. Consider using uint16 for currency bytes to avoid this.

Listing 581:

```
45 uint256 currencyId = uint256(uint16(currencyBytes & Constants.
    ↪ UNMASK_FLAGS));
```

3.582 CVF-582

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** FreeCollateral.sol

Description Phantom overflows are possible here. Consider using muldiv function or other safe approach: https://2π.com/21/muldiv/index.html

Listing 582:

```
78 .mul(nTokenAssetPV)
.mul(int256(uint8(nToken.parameters[Constants.
    ↪ PV_HAIRCUT_PERCENTAGE])))
80 .div(Constants.PERCENTAGE_DECIMALS)
.div(nToken.totalSupply);
```

3.583 CVF-583

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateral.sol

Description The conversion to the "int256" type is redundant.

Listing 583:

```
79 .mul(int256(uint8(nToken.parameters[Constants.
    ↪ PV_HAIRCUT_PERCENTAGE])))
```

3.584 CVF-584

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** FreeCollateral.sol

Description The returned values remain uninitialized in some circumstances. Consider explicitly returning zero in these cases.

Listing 584:

```
97  int256 netPortfolioValue,
    int256 nTokenHaircutAssetValue,
    bytes6 nTokenParameters

137 int256 nTokenHaircutAssetValue,
    bytes6 nTokenParameters
```

3.585 CVF-585

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateral.sol

Description This code could be simplified as:
bytes1 accountContextHasDebt = accountContext.hasDebt; if (bitmapHasDebt != (accountContextHasDebt & Constants.HAS_ASSET_DEBT != 0) [accountContext.hasDebt ≙ Constants.HAS_ASSET_DEBT; factors.updateContext = true; }

Listing 585:

```
180 // Turns off has debt flag if it has changed
    bool contextHasAssetDebt =
        accountContext.hasDebt & Constants.HAS_ASSET_DEBT ==
            ↪ Constants.HAS_ASSET_DEBT;
    if (bitmapHasDebt && !contextHasAssetDebt) {
        accountContext.hasDebt = accountContext.hasDebt | Constants.
            ↪ HAS_ASSET_DEBT;
        factors.updateContext = true;
    } else if (!bitmapHasDebt && contextHasAssetDebt) {
        accountContext.hasDebt = accountContext.hasDebt & ~Constants
            ↪ .HAS_ASSET_DEBT;
        factors.updateContext = true;
    }
```

3.586 CVF-586

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateral.sol

Description The conversion to the "uint256" type is redundant.

Listing 586:

```
245 uint256 currencyId = uint256(uint16(currencyBytes & Constants.
    ↪ UNMASK_FLAGS));
```

3.587 CVF-587

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** FreeCollateral.sol

Description This variable is not initialized. Consider explicitly initializing to zero.

Listing 587:

```
295 uint256 netLocalIndex;
```

3.588 CVF-588

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** FreeCollateral.sol

Description The comment is confusing. It gives no clue regarding what the function actually does.

Listing 588:

```
362 /// @dev this is used to clear the stack frame
```

3.589 CVF-589

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** FreeCollateral.sol

Description The semantics of the returns value is unclear. Consider giving a descriptive name to it and/or describing the returned value in the documentation comment.

Listing 589:

```
369 ) private returns (int256) {
```

3.590 CVF-590

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FreeCollateral.sol

Description The conversion to the "uint256" type is redundant.

Listing 590:

```
370 uint256 currencyId = uint256(uint16(currencyBytes & Constants.
      ↪ UNMASK_FLAGS));

450         uint256 tempId = uint256(uint16(currencyBytes &
              ↪ Constants.UNMASK_FLAGS));

464     uint256 currencyId = uint256(uint16(currencyBytes &
          ↪ Constants.UNMASK_FLAGS));
```

3.591 CVF-591

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** StorageLayoutV1.sol

Description The type of this variable should probably be more specific.

Listing 591:

```
25 address internal token;
```

3.592 CVF-592

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** nERC1155Interface.sol

Description This makes the interface incompatible with the ERC-1155 specification and with existing wallets and token trackers, as they will show negative balances as very large positive ones. Consider returning uint256 balances from the functions defined by ERC-1155 (return zero for negative balances), and add extra non-standard for obtaining negative balances.

Listing 592:

```
26 /// @dev Return value is overridden to be int256 here

29 /// @dev Return value is overridden to be int256 here
```

3.593 CVF-593

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** PortfolioHandler.sol

Description Should be "Û.7.0" rather than ">0.7.0" as Solidity 0.8.x has a number of non backward compatible changes.

Listing 593:

```
2 solidity >0.7.0;
```

3.594 CVF-594

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** PortfolioHandler.sol

Description The variable "i" is not initialized. Consider explicitly initializing to zero.

Listing 594:

```
20 for (uint256 i; i < assets.length; i++) {
41 for (uint256 i; i < assetArray.length; i++) {
133 for (uint256 i; i < newAssets.length; i++) {
164 for (uint256 i; i < portfolioState.storedAssets.length; i++) {
174 for (uint256 i; i < portfolioState.storedAssets.length; i++) {
205 for (uint256 i; i < portfolioState.newAssets.length; i++) {
285 for (uint256 i; i < portfolioState.storedAssets.length; i++) {
384 for (uint256 i; i < length; i++) {
```

3.595 CVF-595

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "assets[i]" is calculated multiple times. Consider calculating once and reusing.

Listing 595:

```
21 if (assets[i].notional == 0) continue;

25     assets[i].currencyId,
       assets[i].maturity,
       assets[i].assetType,
       assets[i].notional,
```

3.596 CVF-596

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description These three conditional statements could be merged into one: if (assetArray[i].assetType != assetType || assetArray[i].currencyId != currencyId || assetArray[i].maturity != maturity) continue;

Listing 596:

```
42 if (assetArray[i].assetType != assetType) continue;
   if (assetArray[i].currencyId != currencyId) continue;
   if (assetArray[i].maturity != maturity) continue;
```

3.597 CVF-597

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression 'assetArray[i]" is calculated several times. Consider calculating once and reusing.

Listing 597:

```
42 if (assetArray[i].assetType != assetType) continue;
   if (assetArray[i].currencyId != currencyId) continue;
   if (assetArray[i].maturity != maturity) continue;

48 require(assetArray[i].storageState != AssetStorageState.Delete);
     ↪ // dev: portfolio handler deleted storage

50 int256 newNotional = assetArray[i].notional.add(notional);

58 assetArray[i].notional = newNotional;
   assetArray[i].storageState = AssetStorageState.Update;
```

3.598 CVF-598

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "portfolioState.newAssets[portfolioState.lastNewAssetIndex]" is calculated several times. Consider using struct literal assignment.

Listing 598:

```
116 portfolioState.newAssets[portfolioState.lastNewAssetIndex].
      ↪ currencyId = currencyId;
    portfolioState.newAssets[portfolioState.lastNewAssetIndex].
      ↪ maturity = maturity;
    portfolioState.newAssets[portfolioState.lastNewAssetIndex].
      ↪ assetType = assetType;
    portfolioState.newAssets[portfolioState.lastNewAssetIndex].
      ↪ notional = notional;
120 portfolioState.newAssets[portfolioState.lastNewAssetIndex].
      ↪ storageState = AssetStorageState
        .NoChange;
```

3.599 CVF-599

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description Extending by one could be suboptimal in case the array is large. Consider extending by several elements at once. Probably the number of new elements should depend on the current size of the array.

Listing 599:

```
132 PortfolioAsset[] memory extendedArray = new PortfolioAsset[](
    ↪ newAssets.length + 1);
```

3.600 CVF-600

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description In case there is no allocated objects in the memory after the end of the array, it is possible to extend an array without allocating a new array.

Listing 600:

```
132 PortfolioAsset[] memory extendedArray = new PortfolioAsset[](
    ↪ newAssets.length + 1);
```

3.601 CVF-601

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** PortfolioHandler.sol

Description Consider adding an explicit assert that the number of assets dont exceed 16.

Listing 601:

```
155 // NOTE: cannot have more than 16 assets or this byte object
    ↪ will overflow. Max assets is
```

3.602 CVF-602

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "portfolioState.storedAssets.length" is calculated twice. Consider calculating once and reusing.

Listing 602:

```
164 for (uint256 i; i < portfolioState.storedAssets.length; i++) {

174 for (uint256 i; i < portfolioState.storedAssets.length; i++) {
```

3.603 CVF-603

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "portfolioState.storedAssets[i]" is already calculated and stored in the "asset" variable. Consider using the value from there instead of recalculating.

Listing 603:

```
186 if (portfolioState.storedAssets[i].storageState ==
    ↪ AssetStorageState.Update) {

189     bytes32 encodedAsset = _encodeAssetToBytes(portfolioState.
        ↪ storedAssets[i]);
```

3.604 CVF-604

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** PortfolioHandler.sol

Description Overflow is possible here. Consider using safe conversion.

Listing 604:

```
228 uint8(assetStorageLength),
    uint40(nextSettleTime)
```

3.605 CVF-605

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** PortfolioHandler.sol

Description The semantics of the returned values is unclear. Consider giving them descriptive names and/or describing the returned value in the documentation comment.

Listing 605:

```
243 bool,
    bytes32,
    uint256
```

3.606 CVF-606

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "asset.getSettlementDate()" is calcualted twice. Consider calculating once and reusing.

Listing 606:

```
248 if (nextSettleTime == 0 || nextSettleTime > asset.
    ↪ getSettlementDate()) {
      nextSettleTime = asset.getSettlementDate();
```

3.607 CVF-607

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** PortfolioHandler.sol

Description Information could be lost here. Consider explicitly checking that the lowest 16 bits are zero before shifting.

Listing 607:

```
253 (portfolioActiveCurrencies >> 16) |
```

3.608 CVF-608

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** PortfolioHandler.sol

Description It is not guaranteed that MAX_LIQUIDITY_TOKEN_INDEX fits into 8 bits as implied by the code below. Consider making its length hardcoded constant.

Listing 608:

```
268 (bytes32(asset.assetType) << 56) |
(bytes32(asset.notional) << 64));
```

3.609 CVF-609

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** PortfolioHandler.sol

Description This logic relies of the fact that the highest 64 bits in the notional value are insignificant. Consider adding an explicit assert for this or at least describing in the comment.

Listing 609:

```
269 (bytes32(asset.notional) << 64));
```

3.610 CVF-610

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** PortfolioHandler.sol

Description It seems that the maximum active slot is guaranteed to be the same as the portfolioState.storedAssetLength here, so it would be simpler to just find the index of the asset whose storageSlot equals to the "portfolioState.storedAssetLength" value.

Listing 610:

```
282 uint256 maxActiveSlot;
```

3.611 CVF-611

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "portfolioState.storedAssets[i]" is calculated twice.

Listing 611:

```
287 portfolioState.storedAssets[i].storageSlot > maxActiveSlot &&
    portfolioState.storedAssets[i].storageState != AssetStorageState
    ↪ .Delete

290 maxActiveSlot = portfolioState.storedAssets[i].storageSlot;
```

3.612 CVF-612

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "portfolioState.storedAssets[maxActiveSlotIndex]" is calculated several times. Consider calculating once and reusing.

Listing 612:

```
304     portfolioState.storedAssets[maxActiveSlotIndex].storageSlot,

308     portfolioState.storedAssets[maxActiveSlotIndex].storageSlot

310 portfolioState.storedAssets[maxActiveSlotIndex].storageState =
    ↪ AssetStorageState.Update;
```

3.613 CVF-613

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "portfolioState.storedAssets[index]" is calculated several times. Consider calculating once and reusing.

Listing 613:

```
305     portfolioState.storedAssets[index].storageSlot

307     portfolioState.storedAssets[index].storageSlot,

311 portfolioState.storedAssets[index].storageState =
    ↪ AssetStorageState.Delete;
```

3.614 CVF-614

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** PortfolioHandler.sol

Description Consider using a struct literal to simplify the code.

Listing 614:

```
339 state.storedAssets = getSortedPortfolio(account,
      ↪ assetArrayLength);
340 state.storedAssetLength = assetArrayLength;
    state.newAssets = new PortfolioAsset[](newAssetsHint);

343 return state;
```

3.615 CVF-615

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description Quick sort could be suboptimal for short arrays. Consider using some non-recursive algorithm.

Listing 615:

```
351 function _quickSortInPlace(
```

3.616 CVF-616

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The encoded ID is calculated for the same asset several times due to recursion. Consider calculating encoded IDs for all the assets once, storing in an array and then sorting this array in parallel with the assets.

Listing 616:

```
360 uint256 pivot = _getEncodedId(assets[uint256(left + (right -
      ↪ left) / 2)]);

362     while (_getEncodedId(assets[uint256(i)]) < pivot) i++;
        while (pivot < _getEncodedId(assets[uint256(j)])) j--;
```

3.617 CVF-617

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The final conversions to uint256 and int256 are redundant.

Listing 617:

```
390 assets[i].currencyId = uint256(uint16(uint256(data)));
    assets[i].maturity = uint256(uint40(uint256(data >> 16)));
    assets[i].assetType = uint256(uint8(uint256(data >> 56)));
    assets[i].notional = int256(int88(uint256(data >> 64)));
```

3.618 CVF-618

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PortfolioHandler.sol

Description The expression "assets[i]" is calculated several times. Consider calculating once and reusing.

Listing 618:

```
390 assets[i].currencyId = uint256(uint16(uint256(data)));
    assets[i].maturity = uint256(uint40(uint256(data >> 16)));
    assets[i].assetType = uint256(uint8(uint256(data >> 56)));
    assets[i].notional = int256(int88(uint256(data >> 64)));
    assets[i].storageSlot = slot;
```