Результаты профайлинга скрипта тестового задания.

В качестве основного инструмента было выбран профайлер XHProf (https://www.php.net/manual/en/book.xhprof.php)

Данный инструмента запускается в виде расширения php, а также имеет базовый web интерфейс для просмотра результатов профайлинга.

Инструкции для запуска необходимого окружения для профилирования находятся в файле README.txt подпапки тестового задания asap-lab-test-profiling.

Для запуска профайлера в требуемом окружении с docker-compose, в docker файл контейнера с php-fpm (asap-lab-test-profiling/php-fpm/Dockerfile) были добавлены инструкции для сборки и установки расширения xhprof. Так же добавлены инструкции в в docker файл контейнера с nginx (asap-lab-test-profiling/nginx/Dockerfile) для копирования и включения web интерфейса профайлера.

Так же в тестовый скрипт добавлены инструкции для старта процесса профайлинга: (xhprof_enable(XHPROF_FLAGS_CPU + XHPROF_FLAGS_MEMORY);

) и остановки с сохранением результата работы профайлера: file_put_contents('/profiles/'.time().'.application.xhprof', serialize(xhprof_disable()));

Для начала работы необходимо запустить выполнение тестового скрипта по адресу http://localhost:8080/test.php

После выполнения скрипта заходим на страницу по адресу http://localhost:8081/index.php для просмотра результатов работы профайлера.

По результатам работы мы видим что большую часть времени (9.4 секунды)(61.2% от общего времени) заняло выполнение запроса curl :

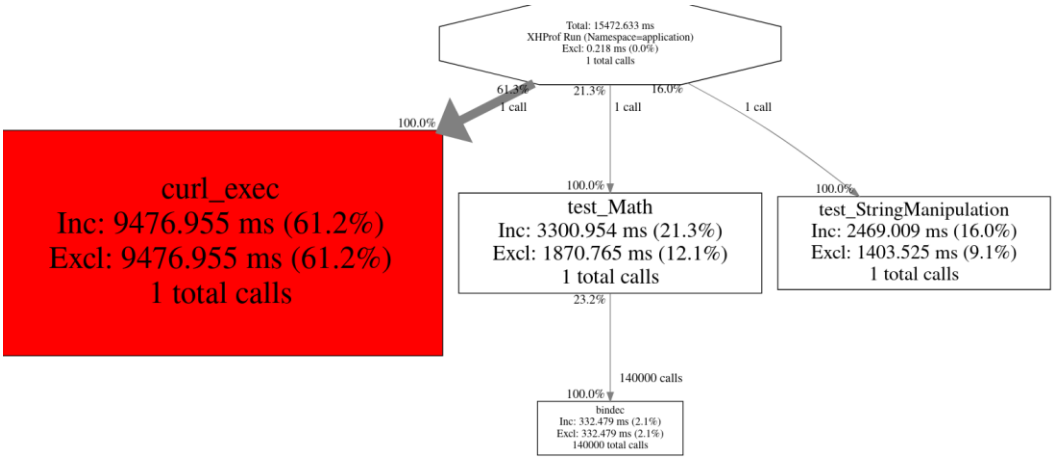
Overall Summary

Total Incl. Wall Time (microsec): 15,472,633 microsecs
Total Incl. CPU (microsecs): 5,917,121 microsecs
Total Incl. MemUse (bytes): 193,112 bytes
Total Incl. PeakMemUse (bytes): 171,840 bytes
Number of Function Calls: 3,240,129

[View Full Callgraph]

Displaying top 100 functions: Sorted by Incl. Wall Time (microsec) [display all]

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Excl. Wall Time (microsec)	EWall%	Incl. CPU (microsecs)	ICpu%	Excl. CPU (microsec)	ECPU%	Incl. MemUse (bytes)	IMemUse%	Excl. MemUse (bytes)	EMemUse%	Incl. PeakMemUse (bytes)	IPeakMemUse%	Excl. PeakMemUse (bytes)	EPeakMemUse%
main()	1	0.0%	15,472,633	100.0%	218	0.0%	5,917,121	100.0%	0	0.0%	193,112	100.0%	-172,336	-89.2%	171,840	100.0%	520	0.3%
curl_exec	1	0.0%	9,476,955	61.2%	9,476,955	61.2%	10,109	0.2%	10,109	0.2%	776	0.4%	776	0.4%	0	0.0%	0	0.0%
test_Math	1	0.0%	3,300,954	21.3%	1,870,765	12.1%	3,216,202	54.4%	1,515,205	25.6%	10,832	5.6%	-4,478,200	-2319.0%	13,816	8.0%	5,104	3.0%
test_StringManipulation	1	0.0%	2,469,009	16.0%	1,403,525	9.1%	2,464,384	41.6%	1,183,343	20.0%	9,560	5.0%	-74,878,504	-38774.7%	6,640	3.9%	4,504	2.6%
bindec	140,000	4.3%	332,479	2.1%	332,479	2.1%	337,859	5.7%	337,859	5.7%	4,480,456	2320.1%	4,480,456	2320.1%	4,152	2.4%	4,152	2.4%
test_Loops	1	0.0%	123,514	0.8%	123,502	0.8%	126,525	2.1%	126,525	2.1%	7,360	3.8%	6,280	3.3%	9,392	5.5%	8,376	4.9%
tan	140,000	4.3%	107,854	0.7%	107,854	0.7%	152,608	2.6%	152,608	2.6%	512	0.3%	512	0.3%	136	0.1%	136	0.1%
sha1	130,000	4.0%	104,996	0.7%	104,996	0.7%	83,262	1.4%	83,262	1.4%	10,400,528	5385.7%	10,400,528	5385.7%	232	0.1%	232	0.1%



Остальное время выполнения было занято математическими расчетами – 3.3 секунды (21.3% общего времени выполнения) :

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Incl. CPU (microsecs)	ICpu%	Incl. MemUse (bytes)	IMemUse%	Incl. PeakMemUse (bytes)	IPeakMemUse%
Current Function										
test_Math	1	0.0%	3,300,954	21.3%	3,216,202	54.4%	10,832	5.6%	13,816	8.0%
Exclusive Metrics for Current Function			1,870,765	56.7%	1,515,205	47.1%	-4,478,200	-41342.3%	5,104	36.9%
Parent function										
main()	1	100.0%	3,300,954	100.0%	3,216,202	100.0%	10,832	100.0%	13,816	100.0%
Child functions										
bindec	140,000	8.3%	332,479	10.1%	337,859	10.5%	4,480,456	41363.1%	4,152	30.1%
tan	140,000	8.3%	107,854	3.3%	152,608	4.7%	512	4.7%	136	1.0%
sin	140,000	8.3%	102,744	3.1%	114,939	3.6%	512	4.7%	136	1.0%
exp	140,000	8.3%	101,015	3.1%	152,517	4.7%	512	4.7%	136	1.0%
acos	140,000	8.3%	100,889	3.1%	113,645	3.5%	520	4.8%	0	0.0%
asin	140,000	8.3%	100,036	3.0%	107,188	3.3%	520	4.8%	144	1.0%
atan	140,000	8.3%	100,028	3.0%	100,612	3.1%	520	4.8%	144	1.0%
floor	140,000	8.3%	98,019	3.0%	97,435	3.0%	520	4.8%	144	1.0%
abs	140,000	8.3%	97,094	2.9%	123,542	3.8%	1,792	16.5%	2,240	16.2%
is_finite	140,000	8.3%	97,075	2.9%	137,422	4.3%	520	4.8%	144	1.0%
is_nan	140,000	8.3%	96,498	2.9%	126,729	3.9%	520	4.8%	144	1.0%
sqrt	140,000	8.3%	96,431	2.9%	136,501	4.2%	520	4.8%	144	1.0%
number_format	1	0.0%	14	0.0%	0	0.0%	560	5.2%	0	0.0%
function_exists	12	0.0%	11	0.0%	0	0.0%	528	4.9%	528	3.8%
microtime	2	0.0%	2	0.0%	0	0.0%	520	4.8%	520	3.8%

И манипуляции с текстовыми данными – 2.4 секунды (16% общего времени выполнения) :

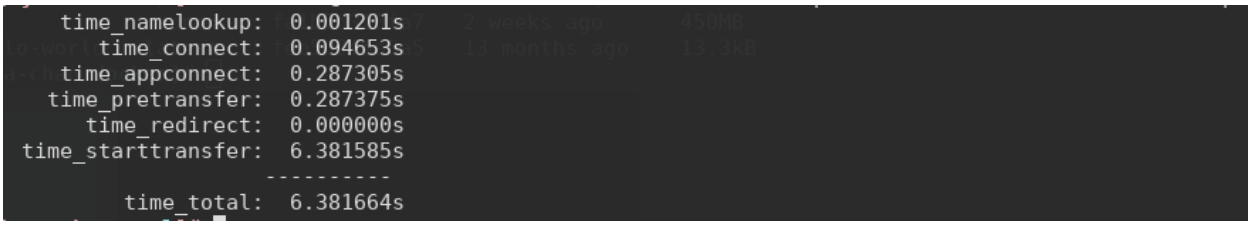
Для более подробного рассмотрения обращения curl тестовый скрипт был декодирован (в онлайн сервисе для декодироввания php скриптов https://www.mobilefish.com/services/eval_gzinflate_base64/eval_gzinflate_base64.php) и получен url обращения https://teststore.host321.net/cron.php

Далее используя тот же инструмент curl, имитируем обращение на тот же url с дополнительной статистикой.

Для этого заходим в директорию curl полученного репозитория и выполняем комманду:

curl -w "@curl-format.txt" -o /dev/null -s "https://teststore.host321.net/cron.php"

В результате выполнения видим что основная задержка была в ожидании начала передачи данных (предполагаю что в скрипте на который происходит обращение внесена намеренная задержка ответа):



Несколько тестовых запусков curl показывают что время до начала передачи варьируется 6-10 секунд.

В случае если бы декодировать скрипт не удалось, то можно добавить использование mitm прокси сервера для перехвата обращения с дальнейшим декодированим трафика, для этого надо развернуть сервис с прокси (например https://mitmproxy.org/), перенаправить трафик докер контейнера на него (встроенные средства докера позволяют определить прокси через переменную окружения) и добавить корневой сертификат сгенерированный прокси сервером в исходные контейнеры.