

Informe de Testing

INFORMACIÓN DE GRUPO

Group: C2.043

Repository: <https://github.com/DaniFdezCab/DP2-2324-C1-043.git>

Student #2

UVUS: alvmarmun1

Contact: alvmarmun1@alum.us.es

Student #4

UVUS: alepingar

Contact: alepingar@alum.us.es

Date: Sevilla Junio 27, 2024

TABLA DE CONTENIDOS

Informe de Testing	1
INFORMACIÓN DE GRUPO	1
TABLA DE CONTENIDOS	2
RESUMEN EJECUTIVO	3
TABLA DE REVISIONES.....	3
INTRODUCCIÓN.....	3
CONTENIDOS.....	3
CONCLUSION.....	12
BIBLIOGRAFIA.....	12

RESUMEN EJECUTIVO

El presente informe documenta el proceso de pruebas de testeo del proyecto Acme-SF-D04. Se han llevado a cabo pruebas exhaustivas de los servicios para comprobar la cobertura del código. Adicionalmente, se realizaron intentos de hackeos para evaluar la seguridad del código de las entidades Contract y ProgressLog. Estos esfuerzos garantizan que el sistema no solo funcione correctamente, sino que también sea seguro contra posibles vulnerabilidades.

TABLA DE REVISIONES

Número de Revisión	Fecha	Descripción
1	23/05/2024	Realización de los .hack y .safe.
2	24/05/2024	Realización del Coverage.
3	25/05/2024	Pruebas de rendimiento.
4	27/05/2024	Realización del documento.

INTRODUCCIÓN

Este informe documenta las pruebas exhaustivas realizadas en el proyecto Acme-SF-D04 para asegurar la calidad y seguridad del sistema. Se han implementado y verificado varios requisitos críticos, centrados en la cobertura del código y la seguridad de las entidades Contract y ProgressLog.

Las pruebas de cobertura del código se llevaron a cabo mediante un riguroso análisis de los servicios, asegurando que todas las rutas y funciones del código estén adecuadamente cubiertas y probadas. Esto permite garantizar que el sistema funcione correctamente bajo diversas condiciones y que cualquier posible error sea identificado y corregido de manera oportuna.

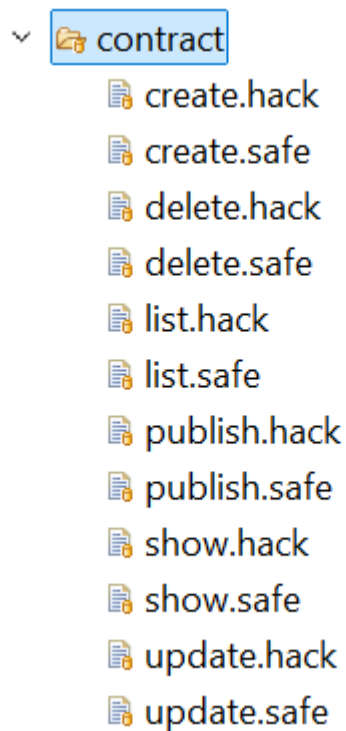
Además, se realizaron intentos de hackeo específicos para evaluar la seguridad de las entidades Contract y ProgressLog. Estas pruebas incluyeron técnicas de penetración para descubrir vulnerabilidades, como inyecciones SQL y problemas de autenticación y autorización.

CONTENIDOS

PRUEBAS FUNCIONALES

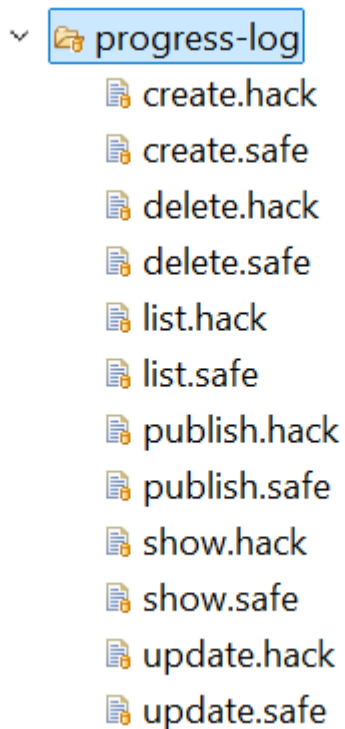
Se realizaron pruebas funcionales .safe y .hack para las dos entidades mencionadas anteriormente (Contract y ProgressLog). Solo se van a explicar detalladamente los tester.trace de Contract ya que el procedimiento es el mismo que para ProgressLog.

Estos son todos los tester.trace que se realizaron para Contract:



Un hack y un safe para cada servicio.

Estos son los de ProgressLog:



Primeramente se van a explicar los .safe:

ClientCreateContract.safe: Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos crear un contrato con todos los campos vacíos. Se rellenan todos los campos correctamente y se empiezan a probar restricciones en todos los campos. Finalmente se ponen datos válidos, se crea el contrato y se para la traza.

ClientDeleteContract.safe: Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos eliminar un contrato que no esté publicado. Una vez se elimina el contrato, nos deslogueamos y paramos la traza.

ClientListContract.safe: Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos listar todos los contratos de un cliente. Una vez se listan los contratos, nos deslogueamos y paramos la traza.

ClientPublishContract.safe: Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos publicar un contrato con todos los campos vacíos. Se rellenan todos los campos correctamente y se empiezan a probar restricciones en todos los campos. Finalmente se ponen datos válidos, se publica el contrato y se para la traza.

ClientShowContract.safe: Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos ver un contrato de un cliente. Una vez se muestra, nos deslogueamos y paramos la traza.

ClientUpdateContract.safe: Se inicia el tester#recorder, nos logueamos como cliente1 y intentamos actualizar un contrato con todos los campos vacíos. Se rellenan todos los campos correctamente y se empiezan a probar restricciones en todos los campos. Finalmente se ponen datos válidos, se actualiza el contrato y se para la traza.

A continuación se explican los .hack:

ClientCreateContract.hack: Se inicia el tester#recorder, nos logueamos como consumer1, sponsor1 y manager1 e intentamos crear un contrato accediendo a esta ruta:
<http://localhost:8082/acme-sf-24.5.0/client/contract/create>

Cuando nos da error de “access not authorised” paramos la trama.

ClientDeleteContract.hack: Se inicia el tester#recorder, nos logueamos como sponsor1, client1 y client2 e intentamos eliminar un contrato publicado, otro editable y uno que no existe a partir de las siguientes rutas:

`http://localhost:8082/acme-sf-24.5.0/client/contract/delete?id=76 -- PUBLICADO`

`http://localhost:8082/acme-sf-24.5.0/client/contract/delete?id=86 -- EDITABLE`

`http://localhost:8082/acme-sf-24.5.0/client/contract/delete?id=1 -- NO EXISTE`

Cuando nos da error de “access not authorised” paramos la trama.

ClientListContract.hack: Se inicia el tester#recorder, nos logueamos como sponsor1, manager1 y consumer1. Intentamos listar contratos a partir de esta ruta:

<http://localhost:8082/acme-sf-24.5.0/client/contract/list>

Cuando nos da error de “access not authorised” paramos la trama.

ClientPublishContract.hack: Se inicia el tester#recorder, nos logueamos como sponsor1, manager1 y consumer1. Intentamos publicar contratos a partir de esta ruta:

<http://localhost:8082/acme-sf-24.5.0/client/contract/publish>

Cuando nos da error de “access not authorised” paramos la trama.

ClientShowContract.hack Se inicia el tester#recorder, nos logueamos como sponsor1, manager1 y client2. Intentamos ver contratos (publicados, editables y que no existen) a partir de esta ruta:

<http://localhost:8082/acme-sf-24.5.0/client/contract/show?id=76> - PUBLICADO

<http://localhost:8082/acme-sf-24.5.0/client/contract/show?id=78> - NO PUBLICADO

<http://localhost:8082/acme-sf-24.5.0/client/contract/show?id=1> - NO EXISTE

Cuando nos da error de “access not authorised” paramos la trama.

ClientUpdateContract.hack: Se inicia el tester#recorder, nos logueamos como sponsor1, manager1 y consumer1. Intentamos listar contratos a partir de esta ruta:

<http://localhost:8082/acme-sf-24.5.0/client/contract/update>

Cuando nos da error de “access not authorised” paramos la trama.

SE REALIZA MISMO PROCEDIMIENTO DE LOS .SAFE Y .HACK CON PROGRESSLOG.

Una vez se han realizado todos los .safe y .hack runeamos tester#replayer y vemos el coverage de cada servicio:

acme.features.client.contract	86,9 %	1.158	174	1.332
> ClientContractController.java	100,0 %	35	0	35
> ClientContractListService.java	94,4 %	67	4	71
> ClientContractShowService.java	96,6 %	115	4	119
> ClientContractDeleteService.java	61,9 %	130	80	210
> ClientContractCreateService.java	93,3 %	249	18	267
> ClientContractUpdateService.java	91,4 %	255	24	279
> ClientContractPublishService.java	87,5 %	307	44	351

Como se puede comprobar, los tester.trace realizados anteriormente cubren el 90% de todos los servicios excepto en Delete por que no se utilizan el unbind de su código en las trazas que se han grabado.

```

@Override
public void unbind(final Contract object) {

    assert object != null;

    Collection<Project> projects;
    SelectChoices choices;

    projects = this.clientContractRepository.findAllProjects();
    choices = SelectChoices.from(projects, "code", object.getProject());

    Dataset dataset;

    dataset = super.unbind(object, "code", "instantiationMoment", "providerName", "customerName", "goals", "budget");
    dataset.put("project", choices.getSelected().getKey());
    dataset.put("projects", choices);

    super.getResponse().addData(dataset);
}

```

En ProgressLog sucede lo mismo:

acme.features.client.progressLog	86,9 %	972	147	1.119
ClientProgressLogController.java	100,0 %	35	0	35
ClientProgressLogListService.java	94,7 %	71	4	75
ClientProgressLogShowService.java	96,7 %	119	4	123
ClientProgressLogDeleteService.java	55,0 %	104	85	189
ClientProgressLogCreateService.java	92,6 %	199	16	215
ClientProgressLogUpdateService.java	92,1 %	220	19	239
ClientProgressLogPublishService.java	92,2 %	224	19	243

```

@Override
public void unbind(final ProgressLog object) {

    assert object != null;

    Dataset dataset;

    dataset = super.unbind(object, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson");

    dataset.put("masterId", object.getContract().getId());
    dataset.put("draftMode", object.isDraftMode());

    super.getResponse().addData(dataset);
}

```

PRUEBAS DE RENDIMIENTO

Con el seguimiento de las diapositivas obtenemos el promedio que tarda en obtener los resultados de cada ruta.

Promedio /	5.77151524
Promedio /anonymous/system/sign-in	5.97288182
Promedio /any/system/welcome	4.06913018
Promedio /authenticated/system/sign-out	4.77814361
Promedio /client/contract/create	72.35112
Promedio /client/contract/delete	47.3927337
Promedio /client/contract/list-mine	20.7346811
Promedio /client/contract/publish	34.9131
Promedio /client/contract/show	15.5730733
Promedio /client/contract/update	58.9678
Promedio /client/progress-log/create	31.8595336
Promedio /client/progress-log/delete	32.4599495
Promedio /client/progress-log/list	14.1954356
Promedio /client/progress-log/publish	31.8557495
Promedio /client/progress-log/show	11.6590993
Promedio /client/progress-log/update	39.8556165
Promedio General	27.0255977



Tras instalar herramientas de para análisis y usar la estadística descriptiva con un nivel de confianza del 95%, obtenemos:

Columna1					
			Interval(ms)	10.9096689	15.9029039
Media	13.4062864		Interval(s)	0.01090967	0.0159029
Error típico	1.26854622				
Mediana	5.5706				
Moda	#N/D				
Desviación estándar	21.7510366				
Varianza de la muestra	473.107595				
Curtosis	53.8852125				
Coeficiente de asimetría	5.78251919				
Rango	254.380699				
Mínimo	1.4672				
Máximo	255.847899				
Suma	3941.4482				
Cuenta	294				
Nivel de confianza(95.0%)	2.49661751				

Añadimos índices a las entidades que estamos testeando:

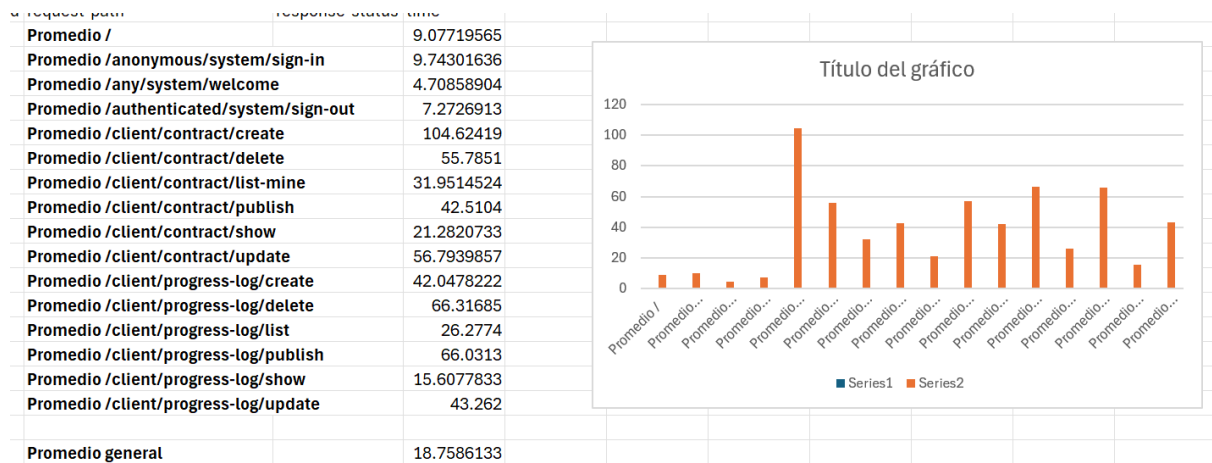
(Contract)

```
@Setter
@Table(indexes = {
    @Index(columnList = "client_id"), @Index(columnList = "project_id"), @Index(columnList = "code")
})
```

(ProgressLog)

```
9 @Setter
10 @Table(indexes = {
11     @Index(columnList = "contract_id")
12 })
```

Obtenemos todos las graficas anteriores tras la implementación de los índices:



Columna1					
			Interval(ms)	14.7808602	22.7363663
Media	18.7586133		Interval(s)	0.01478086	0.02273637
Error típico	2.02112				
Mediana	7.8262				
Moda	5.5596				
Desviación es	34.6549889				
Varianza de la	1200.96826				
Curtosis	105.081105				
Coeficiente d	8.53964064				
Rango	474.6586				
Mínimo	2.3385				
Máximo	476.9971				
Suma	5515.0323				
Cuenta	294				
Nivel de confi	3.97775303				

Vamos a compararlas:

Before	After									
87.794499	141.5444									
33.290599	21.7135									
12.035299	26.0613									
22.240199	47.9436									
5.5638	6.3737									
67.1194	134.0963									
36.565001	77.2704									
255.847899	476.9971									
43.4829	56.8068									
80.9867	91.5674									
62.2918	60.8523									
37.9151	66.0507									
84.590301	79.1886									
49.0589	45.6554									
7.2016	7.3024									
16.547799	8.2948									
4.6622	6.0302									
7.389301	8.8									
4.3361	8.0881									
6.0852	5.3995									
45.1959	41.3652	Interval(ms)		10.9096689	15.9029039	Interval(ms)		14.78086023	22.7363663	
4.5433	7.2923	Interval(s)		0.010909669	0.0159029	Interval(s)		0.01478086	0.02273637	

La mejor forma para compararlo es realizando un Z-Test:

Prueba z para medias de dos muestras		
	Before	After
Media	13.1524017	18.3395491
Varianza (conocida)	473	1200
Observaciones	293	293
Diferencia hipotética de las medias	0	
z	-2.1707731	
P(Z<=z) una cola	0.01497416	
Valor crítico de z (una cola)	1.64485363	
Valor crítico de z (dos colas)	0.02994833	
Valor crítico de z (dos colas)	1.95996398	

Nos debemos de fijar en la fila que esta marcada.

Cuanto mas cerca a cero tenga este valor, mayor habrá sido la mejora de rendimiento. Sin embargo, cuanto mas cerca de 1 sea, menos significantes serán los cambios realizados en cuanto a rendimiento.

En nuestro caso como este valor es 0,0299. Por lo tanto, ha habido una notoria mejora de rendimiento tras añadir los índices.

Utilización de VisualVm

Con el uso de VisualVm podemos ver el tiempo dedicado a cada método mientras se ha realizado el tester#replayer. De esta forma podemos ver los métodos que mas tiempo han tardado en ejecutarse. Esto nos proporciona una idea de los métodos que limitan a otros en cuanto a tiempos de uso.

Results: View: Collected data: Snapshot Thread Dump

Name	Total Time	Total Time (CPU)
http-nio-8082-exec-1	2.355 ms (100 %)	2.113 ms (100 %)
main	172 ms (100 %)	172 ms (100 %)

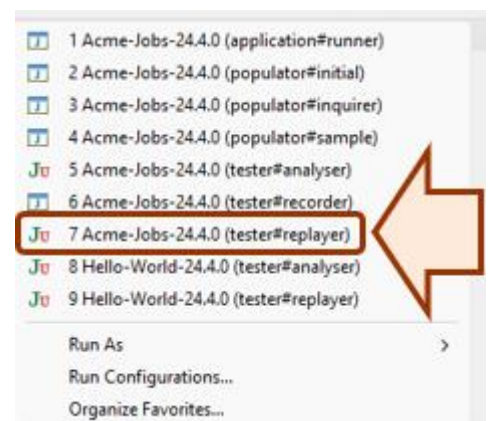
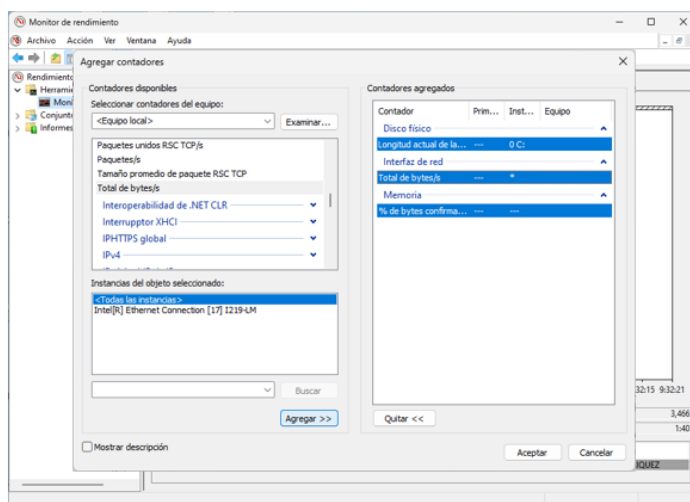
Name	Self Time (CPU)	Total Time (CPU)
acme.features.client.contract.ClientContractCreateService. bind ()	0,0 ms (0 %)	512 ms (22,1 %)
acme.features.client.contract.ClientContractUpdateService. bind ()	0,0 ms (0 %)	299 ms (12,9 %)
acme.features.client.progressLog.ClientProgressLogCreateService. bind ()	0,0 ms (0 %)	274 ms (11,9 %)
acme.features.client.progressLog.ClientProgressLogUpdateService. bind ()	0,0 ms (0 %)	197 ms (8,5 %)
acme.features.client.contract.ClientContractListMineService. load ()	0,0 ms (0 %)	148 ms (6,4 %)
acme.features.client.contract.ClientContractCreateService. unbind ()	0,0 ms (0 %)	103 ms (4,4 %)
acme.features.client.contract.ClientContractUpdateService. unbind ()	0,0 ms (0 %)	51,4 ms (2,2 %)
acme.features.client.contract.ClientContractShowService. authorise ()	0,0 ms (0 %)	49,2 ms (2,1 %)
acme.features.client.contract.ClientContractCreateService. validate ()	0,0 ms (0 %)	48,8 ms (2,1 %)
acme.features.client.progressLog.ClientProgressLogUpdateService. authorise ()	0,0 ms (0 %)	43,8 ms (1,9 %)
acme.features.administrator.banner.AdministratorBannerController. <init> ()	0,0 ms (0 %)	35,6 ms (1,5 %)
acme.features.client.progressLog.ClientProgressLogShowService. authorise ()	0,0 ms (0 %)	35,0 ms (1,5 %)
acme.features.client.progressLog.ClientProgressLogDeleteService. bind ()	0,0 ms (0 %)	30,2 ms (1,3 %)
acme.features.client.contract.ClientContractDeleteService. bind ()	0,0 ms (0 %)	28,7 ms (1,2 %)
acme.features.client.contract.ClientContractDeleteService. perform ()	0,0 ms (0 %)	22,8 ms (1 %)

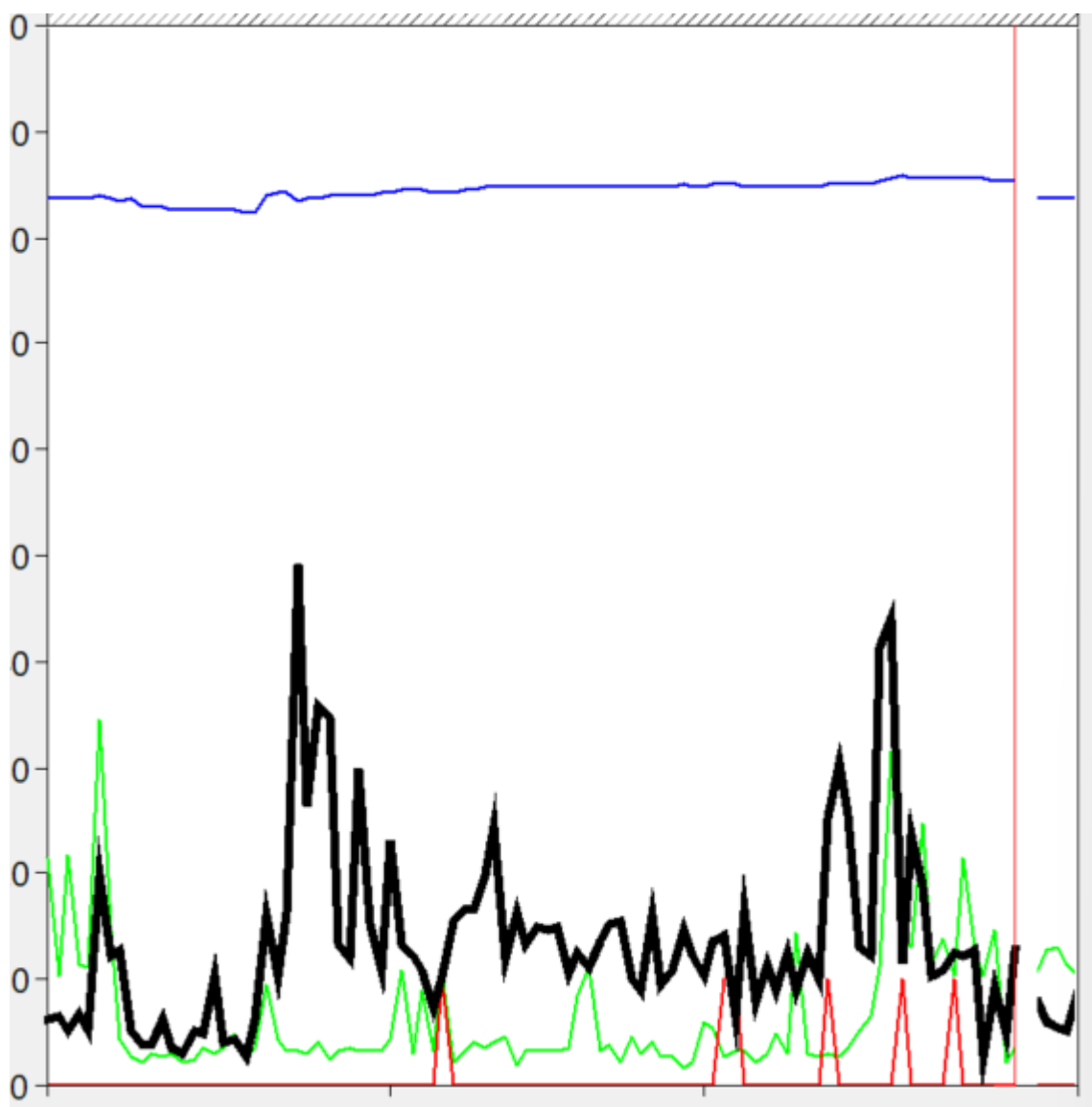
Como se puede ver, el método bind() es el que mas tarda en realizarse con diferencia. Sobre todo el del create con 512ms. El resto de métodos (quitando los bind()) no superan los 50ms.

Hay que puntualizar que el Self Time es de cero milisegundos. Eso significa que no es el método el que está consumiendo demasiado tiempo, sino los métodos que invoca.

MONITOR DE RENDIMIENTO

Iniciamos el monitor de rendimiento y lo configuramos:





Se puede apreciar el rendimiento de la CPU mientras se ejecuta el recorder#replayer. La red de este ordenador está siendo utilizada moderadamente, pero no es un claro cuello de botella ya que no se está usando al 100% del tiempo. La CPU y la memoria están poco utilizadas, por lo que están lejos de ser un cuello de botella.

CONCLUSION

En conclusión, la implementación y el proceso de pruebas del proyecto Acme-SF-D04 han sido exitosos, cumpliendo con los criterios de calidad y seguridad establecidos. Las pruebas exhaustivas de cobertura del código y los intentos de hackeo realizados aseguran que el sistema funcione

correctamente y esté protegido contra posibles vulnerabilidades. Las entidades clave, como Contract y ProgressLog, fueron sometidas a rigurosas evaluaciones tanto funcionales como de seguridad, garantizando su robustez y fiabilidad.

Los resultados de las pruebas funcionales, tanto .safe como .hack, demostraron una cobertura del 90% de los servicios, con algunas excepciones en el Delete debido a la falta de uso del unbind en las trazas grabadas. Las pruebas de rendimiento mostraron mejoras significativas tras la implementación de índices, evidenciadas por un valor de 0,0299 en el Z-Test, lo cual indica una notable mejora en el rendimiento del sistema.

El análisis con VisualVM reveló que los métodos bind() son los que más tiempo de ejecución requieren, mientras que otros métodos se ejecutan en menos de 50ms. Es importante destacar que el Self Time es de cero milisegundos, indicando que el tiempo de ejecución prolongado se debe a los métodos invocados, no al propio método en sí.

El monitoreo de rendimiento indicó que la red está siendo utilizada moderadamente y no representa un cuello de botella, mientras que la CPU y la memoria están subutilizadas. Esto sugiere que el sistema tiene capacidad adicional para manejar cargas de trabajo más intensas sin comprometer el rendimiento.

BIBLIOGRAFIA

Intencionalmente en blanco.