Exploração da ferramenta DTrace - TPC3 -

Carlos Sá - A59905
carlos.sa01@gmail.com

April 10, 2016

Abstract

Este relatório é o resultado de um estudo feito sobre a ferramenta DTrace. Esta ferramenta é uma ferramenta de análise disponível ao nível do sistema operativo que permite fazer o traçado dinâmico de programas e permite obter informações sobre o traçado de chamadas ao sistema que ocorrem ao nível do kernel tais como: informação sobre os processos e de hierarquia de chamadas ao sistema realizadas, valores de retorno de funções etc.

Para este estudo foram construídas scripts em linguagem d com objectivo de fazer o traçado das chamadas a sistema e retirar informações como: PID, valores de retorno, informações sobre os processos a correr no sistema, UID de utilizador, GID do grupo, entre outras. Todo o trabalho será realizado no sistema operativo Solaris 11 e o tratamento estatistico será feito com base nos resultados obtidos nesta plataforma.

O teste das scripts baseia-se na execução de 4 comandos cat diferentes e análise da captura feita com o comando dtrace.

1. Introdução

O objectivo deste trabalho consiste em exercitar o comando **dtrace** e resolver uma série de exercícios propostos. Um primeiro exercício que consiste em criar um programa em dtrace e imprimir informação variada sobre o traçado realizado da chamada ao sistema *openat()* que em Solaris 11 é o *openat()* (mais genérico).

Da variada informação relevante a retirar pretende-se retirar informação sobre a identificação do processo (PID), utlizador (UID) e GID (grupo) valor de retorno, caminho para o ficheiro, entre outros. Depois de retirada essa informação pretende-se testar o programa com um conjunto de diferentes hipóteses com recurso ao comando **cat**. Num segundo exercicio pretende-se gerar um conjunto de estatisticas sobre a chamada ao sistema openat: número de tentativas de abrir ficheiros existentes, tentativas de criação de ficheiros, número de tentativas bem sucedidas e repetir a experiência com um dado período para imprimir hora e dia atual, e estatisticas recolhidas por PID com o seu respectivo nome.

2. O comando dtrace

O comando dtrace é um front-end genérico que implementa uma interface simples de chamada da linguagem **D**, capaz de fornecer informação sobre o traçado de aplicações utilizando o kernel e um conjunto de rotinas que permite realizar a formatação e impressão dos dados traçados nessas mesmas aplicações. Como iremos perceber ao longo deste trabalho, este comando fornece um conjunto de serviços que permite ao programador obter informação variada sobre o programa: valores de retorno, caminhos para ficheiros abertos, tentativas de criação de ficheiros, PID, UID de utilizador etc. A linguagem **D** é a linguagem fundamental da utilização desta ferramenta e as informações obtidas sobre o programa "a traçar" são conseguidas com a utilização e interpretação de scripts em linguagem **D**.

3. Exercício 1 - Traçado da *syscall openat()*

Para este exercício pretendia-se fazer o traçado da *syscall* e imprimir informação variada por linha tal como:

- Nome do executável (execname);
- PID do processo;
- UID do utilizador;
- GID do grupo;
- A path para o ficheiro que for aberto;
- As flags da chamada da syscall openat();
- O respectivo valor de retorno;

Para a resolução deteste primeiro exercício foi necessário criar uma script em linguagem D. A script abaixo já contém o código completo do exercicio 1

(à excepção da alteração necessária para o exercicio opcional que requere a adição de um predicado como irei abordar mais a frente em 5). Abaixo será feita a análise da script por forma a tornar explicita a forma como a informação relativa aos 4 pontos é imprimida.

```
/** Flags for exercise 1 point 3 for Solaris 11
inline int O_WRONLY = 1;
inline int O_RDWR = 2;
inline int O_APPEND = 8;
inline int O_CREAT = 256;
**/

this string s_flags;

syscall::openat*:entry {
    self->path = copyinstr(arg1);
    self->flags = arg2;
}

syscall::openat*:return {

 this->s_flags = strjoin (
       self->flags & O_WRONLY ? "O_WRONLY"
       : self->flags & O_RDWR ? "O_RWR" : "O_RDONLY",
       strjoin ( self->flags & O_APPEND ? "|O_APPEND" : "",
                 self->flags & O_CREAT ? "|O_CREAT" : ""));
    printf("EXECNAME,PID,UID,GID,PATH,FLAGS,RETURN_VALUE\n");
    printf("%s,%d,%d,%d,%s,%s,%d\n",
        execname, pid, uid, gid, self->path, this->s_flags, arg1);
}
```

Para a resolução do 1º ponto do exercicio um, apenas utilizamos precisámos de nos focar no último **printf** dentro do **return**. Como podemos verificar as diferentes variáveis:

- execname (nome do executável);
- pid (correspondente process id do executável);
- uid (identificador de utilizador);
- gid (identificação do grupo);

Imprimem a informação pretendida para este 1º ponto deste exercicio. O caminho do absoluto para o ficheiro que for aberto que está em arg1 e é guardada no campo **path** da estrutura **self** e que também é imprimido. A impressão das flags do openat é feita capturando o valor de *arg*2 no probe entry e tratada no **return** para posterior impressão consoante o ficheiro tenha sido aberto para leitura, escrita etc. Para o tratamento das flags foi necessário utilizar uma variável **s_flags** e a função **strjoin** para fazer join das strings. Para imprimir o valor de retorno basta imprimir o valor da variável *arg1*.

4. Teste do programa com cat

Após a criação do programa, foi necessário fazer um conjunto de testes utilizando o comando cat. Notar que nestes foi utilizada a script acima (aínda sem a adição do predicado que restringiria o traçado à directoria /etc/ - como pedido no exercicio opcional). Todos os testes foram realizados na máquina com Solaris 11. Cada comando **cat** foi executado à vez, depois de correr o comando dtrace com o programa que criei: *syscall_open_ex_um.d*.

```
$ dtrace -qs syscall_open_ex_um.d >> cat_all.csv
```

Nas subsecções seguintes podemos visualizar 4 tabelas. Cada uma delas diz respeito ao resultado imprimido pelo comando dtrace na execução do programa após executar um comando cat. Para realizar estes testes, executo o comando dtrace num terminal e no outro terminal executo um comando cat. O resultado da execução desse comando dtrace é guardado em ficheiro **csv**. O mesmo é feito para os comandos cat restantes e os resultados

são acrescentados ao ficheiro **csv**. No final esses dados são divididos e tratados dando origem aos resultados apresentados nas tabelas das subsecções seguintes. Algum output desnecessário foi também retirado para melhor visualização dos resultados para cada um dos testes.

4.1. Teste com **cat /etc/inittab > /tmp/test_a59905**

O primeiro teste foi realizado utilizando o comando:

```
$ cat /etc/inittab > /tmp/test_a59905
```

cat /etc/inittab > /tmp/test_a59905						
EXECNAME	**PID**	**UID**	**GID**	**PATH**	**FLAGS**	**RETURN_VALUE**
nfsmapid	1351	1	12	/system/volatile/rpc_door/rpc_100172.1	O_RDONLY	-1
nfsmapid	1351	1	12	/system/volatile/rpc_door/rpc_100172.1	O_RDONLY	-1
bash	21856	29214	5000	/tmp/test_a59905	O_WRONLY|O_CREAT	4
cat	21856	29214	5000	/var/ld/ld.config	O_RDONLY	-1
cat	21856	29214	5000	/lib/libc.so.1	O_RDONLY	3
cat	21856	29214	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY	3
cat	21856	29214	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY	3
cat	21856	29214	5000	/etc/inittab	O_RDONLY	3

Figure 1: *Resultados do dtrace com o programa* ***syscall_open_ex_um.d***

Note-se que para todos os testes foi verificado especificamente o **UID**, para verificar que os resultados dizem respeito a registos de acções realizadas na máquina provocadas pela minha conta de utilizador (29214). Uma forma de perceber qual era o meu **UID** é executar o comando **id** na bash.

4.2. Teste com **cat /etc/inittab » /tmp/test_a59905**

O segundo teste foi realizado utilizando o comando:

```
$ cat /etc/inittab >> /tmp/test_a59905
```

cat /etc/inittab >> /tmp/test_a59905						
EXECNAME	**PID**	**UID**	**GID**	**PATH**	**FLAGS**	**RETURN_VALUE**
bash	21891	29214	5000	/tmp/test_a59905	O_WRONLY|O_APPEND|O_CREAT	4
cat	21891	29214	5000	/var/ld/ld.config	O_RDONLY	-1
cat	21891	29214	5000	/lib/libc.so.1	O_RDONLY	3
cat	21891	29214	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY	3
cat	21891	29214	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY	3
cat	21891	29214	5000	/etc/inittab	O_RDONLY	3

Figure 2: *Resultados do dtrace com o programa* ***syscall_open_ex_um.d***

4.3. Teste com **cat /etc/inittab | tee /tmp/test_a59905**

O terceiro teste foi realizado utilizando o comando:

```
$ cat /etc/inittab | tee /tmp/test_a59905
```

cat /etc/inittab | tee /tmp/test_a59905						
EXECNAME	PID	UID	GID	PATH	FLAGS	RETURN_VALUE
tee	21896	29214	5000	/var/ld/ld.config	O_RDONLY	-1
tee	21896	29214	5000	/lib/libc.so.1	O_RDONLY	3
tee	21896	29214	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY	3
tee	21896	29214	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY	3
tee	21896	29214	5000	/tmp/test_a59905	O_WRONLY|O_CREAT	3
cat	21895	29214	5000	/var/ld/ld.config	O_RDONLY	-1
cat	21895	29214	5000	/lib/libc.so.1	O_RDONLY	3
cat	21895	29214	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY	3
cat	21895	29214	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY	3
cat	21895	29214	5000	/etc/inittab	O_RDONLY	3

Figure 3: *Resultados do dtrace com o programa* ***syscall_open_ex_um.d***

4.4. Teste com **cat /etc/inittab | tee -a /tmp/test_a59905**

O quarto teste foi realizado utilizando o comando:

```
$ cat /etc/inittab | tee -a /tmp/test_a59905
```

cat /etc/inittab | tee -a /tmp/test_a59905						
EXECNAME	PID	UID	GID	PATH	FLAGS	RETURN_VALUE
tee	21955	29214	5000	/var/ld/ld.config	O_RDONLY	-1
tee	21955	29214	5000	/lib/libc.so.1	O_RDONLY	3
tee	21955	29214	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY	3
tee	21955	29214	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY	3
tee	21955	29214	5000	/tmp/test_a59905	O_WRONLY|O_APPEND|O_CREAT	3
cat	21954	29214	5000	/var/ld/ld.config	O_RDONLY	-1
cat	21954	29214	5000	/lib/libc.so.1	O_RDONLY	3
cat	21954	29214	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY	3
cat	21954	29214	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY	3
cat	21954	29214	5000	/etc/inittab	O_RDONLY	3

Figure 4: *Resultados do dtrace com o programa* ***syscall_open_ex_um.d***

5. Exercicio 1 (opcional) - Detetar apenas os ficheiros em /etc

Para a resolução do exercicio 1 opcional, criei uma cópia do programa em **d** utilizado anteriormente porque praticamente todo o código pode ser aproveitado. A ideia neste exercício opcional é de restringir o domínio de deteção por forma a que apenas os ficheiros na diretoria */etc* sejam detetados. Do resultado da pesquisa realizada percebi que poderia utilizar a função **strstr** para capturar todas as acções cujos **paths** que possuam "/etc". Então, para que seja possível fazer essa restrição basta adicionar um predicado:

```
/strstr(self->path,"/etc") != NULL/
```

No programa mostrado em 3 a seguir à parte correspondente ao **return** do *openat()*. Assim, para obter o resultado pretendido para este exercicio basta substituír o corpo de **syscall::openat*:return** pelo apresentado abaixo com predicado necessário à restrição do domínio de detecção:

```
syscall::openat*:return
/strstr(self->path,"/etc") != NULL/
{
    this->s_flags = strjoin (
        self->flags & O_WRONLY ? "O_WRONLY"
        : self->flags & O_RDWR ? "O_RWR" : "O_RDONLY",
        strjoin ( self->flags & O_APPEND ? "|O_APPEND" : "",
                  self->flags & O_CREAT ? "|O_CREAT" : ""));
    printf("EXECNAME,PID,UID,GID,PATH,FLAGS,RETURN_VALUE\n");
    printf("%s,%d,%d,%d,%s,%s,%d\n",
        execname, pid, uid, gid, self->path, this->s_flags, arg1);
}
```

Por forma a verificar o correto funcionamento do programa, igualmente corri o comando dtrace com o programa em **d** já com a adição do predicado. A execução do comando dtrace foi feita de forma análoga às anteriores.

Do lado direito encontram-se os resultados da execução do comando dtrace com os resultados obtidos e guardados no ficheiro **ex_opcional.csv**:

EXECNAME	PID	UID	GID	PATH	FLAGS	RETURN_VALUE
ls	22908	29214	5000	/etc/	O_RDONLY	3
nfsmapid	1351	1	12	/etc/resolv.conf	O_RDONLY	10
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
ls	22909	29214	5000	/etc/log	O_RDONLY	3
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
bash	22895	29214	5000	/etc/	O_RDONLY	4
nfsmapid	1351	1	12	/etc/resolv.conf	O_RDONLY	10
ls	22910	29214	5000	/etc/log/	O_RDONLY	3
bash	22895	29214	5000	/etc/	O_RDONLY	4
nfsmapid	1351	1	12	/etc/resolv.conf	O_RDONLY	10

Figure 5: *Resultados do dtrace com o programa **syscall_openat_ex_um_opcional.d***

Podemos verificar que aparentemente o "filtro" está a ser feito corretamente uma vez que só são capturados os ficheiros que pertencem à *path* "/etc". Claro que a tabela ao lado não exprime um resultado muito exaustivo. Apenas foi aqui representado na tabela output relevante à análise do problema. Por forma a capturar algum output relevante, realizei alguns comandos ls sobre a directoria /etc, /etc/log etc, e verifiquei que todos eles eram corretamente capturados pelo programa.

6. Exercicio 2

Nesta secção poderemos encontrar a resolução do exercício 2 completo. Tal significa que apenas um programa em **d** foi criado por forma a responder às duas alineas: **a**) e **b**).

O programa em d que a cada iteração imprime número de tentativas:

```
/* Counting attemps for open create and succeed openat */

syscall::openat*:entry
/(arg2 & O_CREAT) == O_CREAT/
{
    @create[execname, pid] = count();
}

syscall::openat*:entry
/(arg2 & O_CREAT) == 0/
{
    @open[execname, pid] = count();
}

syscall::openat*:return
/arg1 > 0/
{
    @success[execname, pid] = count();
}

tick-$1s {
    printf("%Y\n",walltimestamp);
    printf("%12s,%6s,%6s,%6s,%6s\n", "EXECNAME","PID","CREATE","OPEN","SUCCESS");
    printa("%12s,%6d,%@6d,%@6d,%@d\n", @create, @open, @success);
    trunc(@create);
    trunc(@open);
    trunc(@success);
}
```

Através da análise da script, podemos verificar que uma acção semelhante é realizada sempre que o predicado entre "/" é verificado. Os diferentes probes **entry** cobrem os casos em que se verifica uma tentativa de criação de um ficheiro, abertura de ficheiros existentes e o mesmo para as tentativas bem sucedidas.

A contagem é feita com o recurso à função **count()**, e o resultado é imprimido no final dentro do corpo tick-$1s, como pretendido para a alinea **a)**.

Para a resolução da alinea **b)**, como o objectivo passa por imprimir iterações dado um determinado periodo, foi necessário incluir as impressões dos valores dentro do corpo tick-$1s.

Desta forma consigo fazer com que os resultados de cada iteração sejam imprimidos de acordo com um determinado período. Para tal basta que se execute esta script utilizando o comando dtrace da seguinte forma:

```
$ dtrace -qs syscall_openat_ex_dois.d ↩
    4
```

O último parâmetro passado, corresponde ao número de segundos que queremos entre cada registo. De acordo com o comando acima, significa que queremos imprimir os resultados de cada iteração a cada 4 segundos. Um dos objectivos para a alínea **b)** do exercício 2 passava por, em cada linha, imprimir a data e a hora num formato legível. Assim, depois de pesquisar percebi que existem diferentes formas de o fazer. Existem as funções **gettimeofday()**, **ctime()** e a variável **walltimestamp**. Dos três, escolhi o **walltimestamp** como se verifica no primeiro printf realizado. Por último são também recolhidos o **execname** e o **PID** de cada registo e igualmente imprimido no final. Este trabalho é feito sempre que uma cada um dos predicados é verificado e execu-

- abrir ficheiros existentes;
- criar ficheiros;
- bem-sucedidas.

Encontra-se abaixo e para o qual passarei a analisar os resultados obtidos.

tada a respectiva a acção dentro do corpo de **entry** e **return**. A acção a realizar tem subjacente a utilização de **agregação** em linguagem **d** com uso de arrays.

A execução do programa faz com que seja imprimido o seguinte output com o resultado pretendido:

2016 Apr 10 14:40:01				
EXECNAME	PID	CREATE	OPEN	SUCCESS
nfsmapid	1351	0	2	0
dtrace	23189	0	2	2
utmpd	259	1	5	6
2016 Apr 10 14:40:05				
EXECNAME	PID	CREATE	OPEN	SUCCESS
nfsmapid	1351	0	2	0
2016 Apr 10 14:40:09				
EXECNAME	PID	CREATE	OPEN	SUCCESS
nfsmapid	1351	0	2	0
2016 Apr 10 14:40:13				
EXECNAME	PID	CREATE	OPEN	SUCCESS
2016 Apr 10 14:40:17				
EXECNAME	PID	CREATE	OPEN	SUCCESS
nfsmapid	1351	0	2	0
2016 Apr 10 14:40:21				
EXECNAME	PID	CREATE	OPEN	SUCCESS
nfsmapid	1351	0	1	1

Figure 6: *Resultados do dtrace com o programa* ***syscall_openat_ex_dois.d***

7. Conclusão

Graças à realização do presente trabalho fui capaz de desenvolver algumas competências em linguagem **d** na criação de scripts utilizáveis para realizar traçados utilizando a ferramenta dtrace. Todos os exercícios do TPC foram realizados e testados. Além do traçado por si, este trabalho teve especial importância para explorar os conceitos de **probes**, e **agregação**.

Neste trabalho fez-se um traçado da chamada ao sistema *openat()* mas o trabalho realizado sobre o traçado desta syscall poderia ser facilmente reprodutível para outras.

A nível de trabalho futuro poderia ser explorado o traçado de outras *syscall* e realizada a mesma análise de resultados.

[1]

References

[1] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.