

DTrace: beyond Instruments

Frank Lefebvre

What you'll learn

- What is DTrace
- Use dtrace on the command line
- Write scripts in the D language
- Create custom instruments
- Add your own probes to your code
- Use DTrace for fine-grained logging

About DTrace

- Overview
- History
- Dynamic: providers, consumers
- Systemwide
- Zero disable cost

Security considerations

- Privileges
 - dtrace: root
 - Instruments: requests admin password
- Restrictions
 - Supported platforms
 - System Integrity Protection

System Integrity Protection

- Your applications on a development system
 - It just works
- Your applications in production
 - `com.apple.security.get-task-allow` entitlement
 - or disable System Integrity Protection
- Other executables
 - Disable System Integrity Protection

Disabling SIP

- Check current status

```
csrutil status
```

- Reboot in Recovery mode

```
csrutil enable --without dtrace
```

```
csrutil disable
```

```
csrutil clear
```

Hands on

- Disable System Integrity Protection
- How many probes on your system?
 - `dtrace -l`
- Compare your results

Probe specifiers

- probe ID
- `provider:module:function:name`
 - wildcards
 - escaping

Providers

- General providers
 - `syscall`, `fbt`, `io`, `pid`, `profile`...
- Language providers
 - `objc_runtime`, `python`, `sh`...
- Special probes
 - `dtrace:::BEGIN`, `dtrace:::END`

DTrace one-liners

- probes

```
'syscall::open*:entry, syscall::open*:return'
```

- probes {actions}

```
'syscall::open*:entry {printf("%s\n", copyinstr(arg0));}'
```

- probes /predicate/ {actions}

```
'syscall::open*:entry /execname == "Safari"/  
{printf("%s\n", copyinstr(arg0));}'
```

Using one-liners

- Run globally

```
dtrace -n (program)
```

- Attach to existing process

```
dtrace -n (program) -p (pid)
```

- Launch executable (restrictions apply)

```
dtrace -n (program) -c (command)
```

System-provided scripts

- `man -k dtrace`
- `/usr/share/examples/DTTk`
- Try scripts
 - `execsnoop -av`
 - `filebyproc.d`

Structure of a D program

```
#!/usr/sbin/dtrace -s
```

Header

```
provider:module:function:name
```

```
/predicate/
```

```
{
```

```
    action statement;
```

```
    action statement;
```

```
}
```

Clause

```
...
```

Clause

Comments & options

- Command-line option equivalents
 - `#pragma D option quiet`
 - `#pragma D option destructive`
- C-style comments
 - `/* This is a comment. */`
 - `// This does not compile.`

Hello, World!

Hello, World!

```
#!/usr/sbin/dtrace -s
```

```
#pragma D option quiet
```

```
dtrace:::BEGIN
```

```
{
```

```
    printf("Hello World!\n");
```

```
    exit(0);
```

```
}
```


Action statements

- C-like syntax
- No control flow
- No function definitions
- Variables
- Built-in variables

Built-in variables

- `arg0...arg9, args[]`
- `cpu`
- `curpsinfo`
- `curthread`
- `errno`
- `execname`
- `pid, ppid`
- `stackdepth`
- `timestamp`
- `uid`
- `vtimestamp`
- `walltimestamp`

Variable scopes

- Global

`variable`

- Thread local

`self->variable`

- Clause local

`this->variable`

Aggregations

- Simple aggregation

```
@variable = function(...);
```

- Key-based aggregation

```
@variable[key] = function(...);
```

- Aggregating functions

```
count, sum, avg, min, max, quantize...
```

Top 5 processes

```
#!/usr/sbin/dtrace -s
```

```
#pragma D option quiet
```

```
profile:::tick-1001
```

```
{  
    @counters[execname] = count();  
}
```

```
profile:::tick-1sec
```

```
{  
    trunc(@counters, 5);  
    printf("\n");  
    printa("%s --> %@d\n", @counters);  
    trunc(@counters);  
}
```

Instruments & DTrace

- printf statements not supported
- explicit list of returned fields

The screenshot shows the configuration window for a DTrace instrument. At the top, the 'Name' field is 'DTrace Instrument' and the 'Category' is 'Custom Instruments'. The 'Description' field contains 'DTrace Instrument (Created 14/09/16 23:07)'. Below these are three large text areas labeled 'DATA', 'BEGIN', and 'END'. The 'BEGIN' section contains a configuration for 'Probe 1 - syscall : : o...'. It specifies conditions: 'Probe 1' of type 'System Call' hits 'open' at 'entry'. Below this is a script area and a data recording section. The data recording section has two rows: 'Record in Instruments' with fields 'arg0' and 'Path' of type 'String', and 'Record No Data'. At the bottom, there is a 'User Stack Trace' dropdown and a checkbox for 'Preserve previous instrument and recorded data'. 'Cancel' and 'Save' buttons are at the bottom right.

Name: DTrace Instrument Category: Custom Instruments

Description: DTrace Instrument (Created 14/09/16 23:07)

DATA

BEGIN

Probe 1 - syscall : : o...

If the following conditions are met:

Probe Probe 1 of type System Call hits open entry

Perform the following script:

Record the following data:

Record in Instruments arg0 Path String

Record No Data

END

+ - User Stack Trace

☐ Preserve previous instrument and recorded data

Cancel Save

Instruments & DTrace

Probe 1 - syscall : : o...

If the following conditions are met:

Probe	Probe 1	of type	System Call	hits	open	entry	+
-------	---------	---------	-------------	------	------	-------	---

Perform the following script:

Record the following data:

Record in Instruments	arg0	Path	String	-	+	
Record No Data					-	+

Creating your own probes

- Create provider definition file
- Generate C header file
- Import in Objective-C code
- Add probe macros to source code
- Profit!

Create provider definition file

```
/* provider.d */
```

```
provider frenchkit {  
    probe some_action(int);  
    probe another_action(char*);  
};
```

Generate header file

```
dtrace -h -s provider.d
```

```
/* provider.h */
```

```
FRENCHKIT_SOME_ACTION( )
```

```
FRENCHKIT_SOME_ACTION_ENABLED( )
```

```
FRENCHKIT_ANOTHER_ACTION( )
```

```
FRENCHKIT_ANOTHER_ACTION_ENABLED( )
```

Add probe macros

```
#import "provider.h"
```

```
FRENCHKIT_SOME_ACTION(value);
```

```
if (FRENCHKIT_ANOTHER_ACTION_ENABLED()) {  
    NSString* s = some_expensive_call();  
    FRENCHKIT_ANOTHER_ACTION([s UTF8String]);  
}
```

Using your probes in Swift code

```
// File: provider_functions.h
```

```
#import "provider.h"
```

```
static inline void FrenchKitSomeAction(int arg) {  
    FRENCHKIT_SOME_ACTION(arg);  
}
```

Your custom probe in DTrace

- Provider: frenchkit<pid>
- Module: <your executable name>
- Function: FrenchKitSomeAction
- Name: some_action
- Arg0: <value>

Recap

Q&A