Introduction to DTrace

Frank Lefebvre

CocoaHeads Paris
November 2016

@franklefebvre

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
```

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)
```

Hello, World!

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


#pragma D option quiet


dtrace:::BEGIN
{
  printf("Hello World!\n");
  exit(0);
}
```

Structure of a D program

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

Header

```
provider:module:function:name
/predicate/
{
 action statement;
 action statement;
}
```

Clause

```
…
```

Clause

Action statements

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

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);
}
```

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

Generate header file

```
dtrace -h -s provider.d


/* provider.h */


COCOAHEADS_SOME_ACTION()
COCOAHEADS_SOME_ACTION_ENABLED()
COCOAHEADS_ANOTHER_ACTION()
COCOAHEADS_ANOTHER_ACTION_ENABLED()
```

Add probe macros

```
#import "provider.h"

COCOAHEADS_SOME_ACTION(value);

if (COCOAHEADS_ANOTHER_ACTION_ENABLED()) {
  NSString* s = some_expensive_call();
  COCOAHEADS_ANOTHER_ACTION([s UTF8String]);
}
```

Using your probes in Swift code

```
// File: provider_functions.h


#import "provider.h"


static inline void CocoaHeadsSomeAction(int arg) {
  COCOAHEADS_SOME_ACTION(arg);
}
```

Your custom probe in DTrace

- Provider: cocoaheads<pid>
- Module: <your executable name>
- Function: CocoaHeadsSomeAction
- Name: some_action
- Arg0: <value>

Recap

CocoaHeads Paris
November 2016

@franklefebvre

Q&A

CocoaHeads Paris
November 2016

@franklefebvre