IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

~ Aloha! ~

Dynamic Tracing for Exploitation and Fuzzing

SHAKACON 2009

Tiller Beauchamp
IOActive

[David Weston]

DTrace Background

- Kernel-based dynamic tracing framework
- Created by Sun Microsystems
- First released with Solaris™ 10 operating System
- Now included with Apple OS X Leopard, QNX
- Re-implemented in FreeBSD 8 (CURRENT) back ported to FreeBSD 7.1 and 7.2 (John Birrell)
- OpenBSD, NetBSD, Linux?

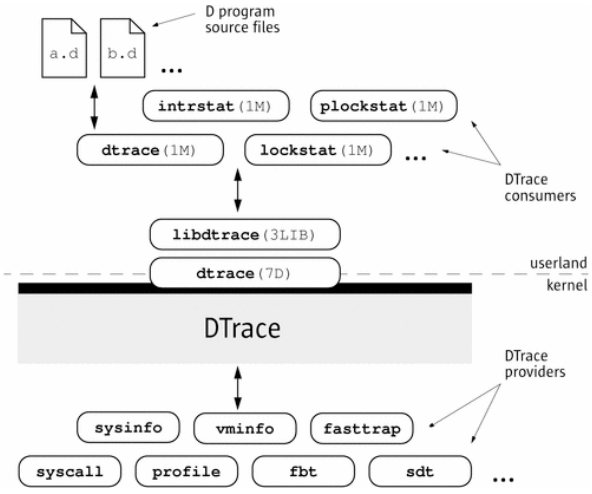
IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Overview

- DTrace is a framework for performance observability and debugging in real time
- Tracing is made possible by “probes” placed “on the fly” throughout the system
- Probes are points of instrumentation in the kernel
- When a program execution passes one of these points, the probe that enabled it is said to have fired
- DTrace can bind a set of actions to each probe

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Architecture



IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Source: Solaris Dynamic Tracing Guide

The D Language

- D is an interpreted, block-structured language
- D syntax is a subset of C
- D programs are compiled into intermediate form
- Intermediate form is validated for safety when your program is first examined by the DTrace kernel software
- The DTrace execution environment handles any runtime errors

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

The D Language

- D does not use control-flow constructs such as if statements and loops
- D program clauses are written as single, straight-line statement lists that trace an optional, fixed amount of data
- D can conditionally trace data and modify control flow using logical expressions called *predicates*
- *A predicate is tested at probe firing before executing any statements*

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Performance

- DTrace is dynamic: probes are enabled only when you need them
- No code is present for inactive probes
- There is no performance degradation when you are not using DTrace
- When the `dtrace` command exits, all probes are disabled and instrumentation removed
- The system is returned to its original state

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Uses

- DTrace takes the power of multiple tools and unifies them with one programmatically accessible interface
- DTrace has features similar to the following:
  - `truss`: tracing system calls, user functions
  - `ptrace`: tracing library calls
  - `prex/tnf`*: tracing kernel functions
  - `lockstat`: profiling the kernel
  - `gdb`: access to kernel/user memory

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Uses

- DTrace combines system performance statistics, debugging information, and execution analysis into one tight package
- A real “Swiss army knife” for reverse engineers
- DTrace probes can monitor every part of the system, giving “the big picture” or zooming in for a closer look
- Can debug “transient” processes that other debuggers cannot

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Creating DTrace Scripts

- Dozens of ready-to-use scripts are included with Sun’s DTraceToolkit; they can be used as templates
  - syscalls by process
  - reads and writes by process
  - file access
  - CPU time,
  - memory r/w statistics
- A lot of information can be gathered with simple DTrace one-liners
- Effective use of DTrace often involves continually revising scripts

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Example: Syscall Count

```
dtrace -n 'syscall:::entry{@[execname] = count();}'.
```

```
Matched 427 probes
Syslogd                   1
DirectoryService          2
Finder                    3
TextMate                  3
Cupsd                     4
Ruby                   4309
vmware-vmx             6899
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Example: File Open Snoop

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

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

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Example: File Snoop Output

vmware-vmx	/dev/urandom
Finder	/Library/Preferences/SystemConfiguration/com.apple.smb.server.plist
iChat	/Library/Preferences/SystemConfiguration/com.apple.smb.server.plist
Microsoft Power	/Library/Preferences/SystemConfiguration/com.apple.smb.server.plist
nmblookup	/System/Library/PrivateFrameworks/ByteRange ... ByteRangeLocking
nmblookup	/dev/dtracehelper
nmblookup	/dev/urandom
nmblookup	/dev/autofs_nowait
Nmblookup	/System/Library/PrivateFrameworks/ByteRange... ByteRangeLocking

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Lingo

- **Probes** - points of instrumentation
- **Providers** - logically grouped sets of probes
  - Pid – Userland process probes
  - Syscall – Syscall probes
  - Fbt – Kernel probes
  - Io – Input/output probes
- **Predicates –** condition that determines if probe fires or not
- **Actions** – statements performed when a probe fires

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Syntax

Generic D Script

Probe: provider:module:function:name
Predicate: /some condition that needs to happen/
{
Action: action1;
action2; (ex: printf();)
}

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace and Reverse Engineering

How Can We Use DTrace?

DTrace for RE

- It is very useful for understanding the way a process behaves and interacts with the rest of the system
- DTrace probes work in a manner very similar to debugger “hooks”
- DTrace probes are useful because they can be described generically and focused later

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace for RE

- Think of DTrace as a rapid development framework for RE tasks and tools
- Dtrace is very fast, minimal impact on application performance
- DTrace can instrument any process on the system without starting or stopping it
- Complex operations can be understood with a succinct one-line script
- You can refine your script as the process continues to run

Ex:

- Trace until a function has a certain user-controlled argument
- Analyze heap allocation patterns at runtime

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Helpful Features

DTrace gives us some valuable functionality:

- Control flow indicators
- Symbol resolution
- Call stack trace
- Function parameter values
- CPU register values
- Both in kernel space and user space!

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Control Flow

```
1        -> -[AIContentController finishSendContentObject:]
1          -> -[AIAdium notificationCenter]
1          <- -[AIAdium notificationCenter]
1          -> -[AIContentController processAndSendContentObject:]
1            -> -[AIContentController
handleFileSendsForContentMessage:]
1            <- -[AIContentController
handleFileSendsForContentMessage:]
1            -> -[AdiumOTREncryption willSendContentMessage:]
1              -> policy_cb
1                -> contactFromInfo
1                  -> -[AIAdium contactController]
1                  <- -[AIAdium contactController]
1                  -> accountFromAccountID
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Symbol and Stack Trace

dyld`strcmp
dyld`ImageLoaderMachO::findExportedSymbol(char
dyld`ImageLoaderMachO::resolveUndefined(...
dyld`ImageLoaderMachO::doBindLazySymbol(unsigned
dyld`dyld::bindLazySymbol(mach_header const*, ...
dyld`stub_binding_helper_interface2+0x15
Ftpd`yylex+0x48
Ftpd`yyparse+0x1d5
ftpd`ftp_loop+0x7c
ftpd`main+0xe46

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Function Parameters

DTrace's copyin* functions allow you to copy data from the process space:

```
printf("arg0=%s", copyinstr( arg0 ))
```

Output:

```
1   -> strcmp     arg0=_isspecial_l
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

CPU Register Values

Uregs array allows access to reading CPU registers

```
printf("EIP:%x", uregs[R_EIP]);
```

Example:

```
EIP: 0xdeadbeef
EAX: 0xffffeae6
EBP: 0xdefacedd
ESP: 0x183f6000
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Destructive Examples

```
#!/usr/sbin/dtrace -w -s
syscall::uname:entry { self->a = arg0; }

syscall::uname:return{
        copyoutstr("Windows", self->a, 257);
        copyoutstr("PowerPC", self->a+257, 257);
        copyoutstr("2010.b17", self->a+(257*2), 257);
        copyoutstr("fud:2010-10-31", self->a+(257*3), 257);
        copyoutstr("PPC", self->addr+(257*4), 257);
}
```

Adapted from: Jon Haslam, http://blogs.sun.com/jonh/date/20050321

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Snooping

```
syscall::write:entry {
   printf("write: %s",
   copyinstr(arg1));
}
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Got Ideas?

Using DTrace:

- Monitor stack overflows
- Code coverage
- Function argument Fuzzing
- Monitor heap overflows
- Record ioctl messages for device fuzzing
- Finding ways to reach vulnerable functions

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace vs. Debuggers

- DTrace is not a debugger
- DTrace does not affect process memory like a debugger
- DTrace does not directly perform exception handling
- Debuggers allow you to control execution, modify memory/registers
- DTrace can instrument both the kernel and user land applications at the same time
- To trace execution, debuggers use instructions to pause and resume execution
- DTrace carries out parallel actions in the kernel when a probe is hit

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace vs. Debuggers

- DTrace can halt process and transfer control to external debugger
- Apple has implemented anti-debugging with use of the PT_DENY_ATTACH ptrace flag
- Landon Fuller released a kernel module to prevent PT_DENY_ATTACH
- Also, the SInAR rootkit hides from dtrace on Solaris (2004)

http://landonf.bikemonkey.org/code/macosx/Leopard_PT_DENY_ATTACH.20080122.html

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace vs. Tracers

- Truss, ltrace, and strace operate one process at a time, with no system-wide capability
- Truss reduces application performance
- In some cases the difference is 65% slower vs. 0.3% slower
- Truss stops threads through procfs, records the arguments for the system call, and then restarts the thread
- Dtrace doesn't stop the application, information is collected into a kernel buffer and read out asynchronously to the process
- Valgrind™ is limited to a single process and only runs on Linux
- But Valgrind™ has promising taint tracing abilities
- Ptrace is much more efficient at instruction level tracing but it is crippled on OS X

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

*Valgrind is Open Source/Free Software and is freely available under the GNU General Public License.

DTrace Limitations

- The D language does not have conditionals or loops
- The output of many functions is to stdout (i.e., stack(), ustack())
- Lack of loops and use of stdout means DTrace is not ideal for processing data
- Cannot modify registers :'(epic sad time
- We can fix this
  - Dtrace for tracing to a specific point
  - Other tools for access debugging API

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace Cautionaries

A few issues to be aware of:

- DTrace drops probes by design
- Tune buffer/frequency options, narrow trace scope to improve performance
- Some libraries and functions behave badly
- DTrace requires symbols

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

MONITORING THE STACK

Writing a Stack Overflow Monitor with RE:Trace

Stack Overflow Monitoring

Programmatic control at the time of overflow allows you to:

- Pinpoint the vulnerable function
- Reconstruct the function call trace
- Halt the process before damage occurs
- Dump and search process memory
- Collect function argument values
- Send feedback to fuzzer
- Attach debugger

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Overflow Detection in One Probe

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

pid$target:::return
   / uregs[R_EIP] == 0x41414141 / {
   printf("Don't tase me bro!!!");
          stop()
          ...
}
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

First Approach

- Store RETURN value at function entry
- uregs[R_SP], NOT uregs[R_ESP]
- Compare EIP to saved RETURN value at function return
- If different, there was an overflow

Simple enough, but false positives from:

- Tail call optimizations
- Functions without return probes

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

DTrace and Tail Calls

- Certain compiler optimizations mess with the standard call/return control flow
- Tail calls are an example of such an optimization
- Two functions use the same stack frame, saves resources, less instruction
- DTrace reports tail calls as a return then a call, even though the return never happens
- EIP on return is not in the original calling function, it is the entry to second
- Screws up simple stack monitor if not aware of it

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

New Approach

- Store RETURN value at function entry
- At function return, compare saved RETURN value with CURRENT value
- Requires saving both the original return value and its address in memory
- Fires when saved RETURN ! = current RETURN and EIP = current RETURN

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

But Missing Return Probes???

Still trouble with functions that “never return”

- Some functions misbehave
- DTrace does not like function jump tables (dyld_stub_*)
- Entry probe but no exit probe

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Determining Missing Returns

Using DTrace – l flag

- List entry/exit probes for all functions
- Find functions with entry but no exit probe

Using DTrace aggregates

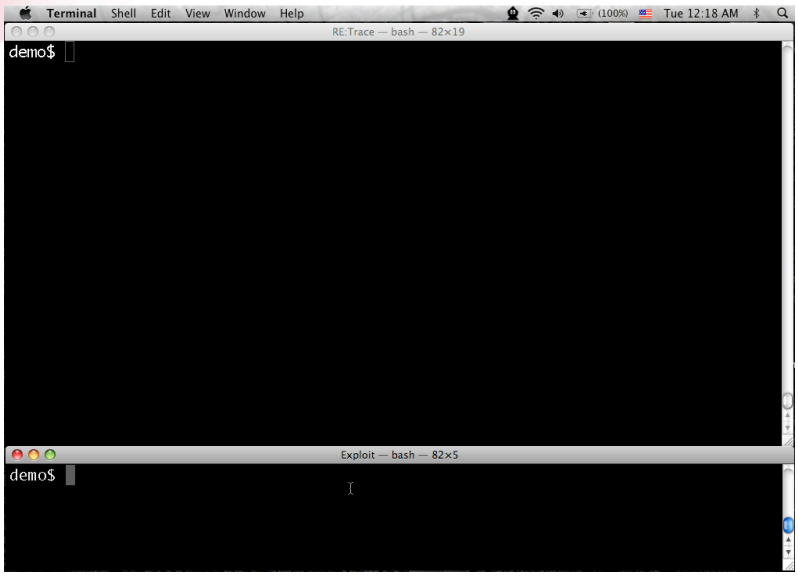
- Run application
- Aggregate on function entries and exits
- Look for mismatches

Exclude these functions with predicates

- / probefunc ! = “everybodyJump” /

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Stack Overflow Video



IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Advanced Tracing

Diving in deeper:

- Instruction-level tracing
- Code coverage with IDA Pro and IdaRub
- Profiling idle and GUI code
- Feedback to the fuzzer, smart/evolutionary fuzzing
- Conditional tracing based on function parameters (reaching vulnerable code paths)
- IOCTL tracing

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Code Coverage

Tracing Function and Instructions

Code Coverage Approach

Approach

- Instruction-level tracing using DTrace
- Must properly scope tracing
- Use IdaRub to send commands to IDA
- IDA colors instructions and code blocks
- Can be done in real time, if you can keep up

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Tracing Instructions

- The last field of a probe is usually `entry` or `return`
- But it can also be a particular instruction within the function
- Leave blank to trace every instruction
- Must map static global addresses to function offset addresses

Print address of every instruction:

pid$target:a.out:: { print(“%d”, uregs[R_EIP]); }

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Tracing Instructions (cont.)

- DTrace to print instructions
- Ruby-Dtrace to combined DTrace with Ruby
- Idarub and rublib to combined Ruby with IDA

Tracing libraries

- When tracing libraries, must know memory layout of program
- `vmmap` on OS X will tell you
- Use offset to map runtime library EIPs to decompiled libraries

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Code Coverage with DTrace

Capabilities:

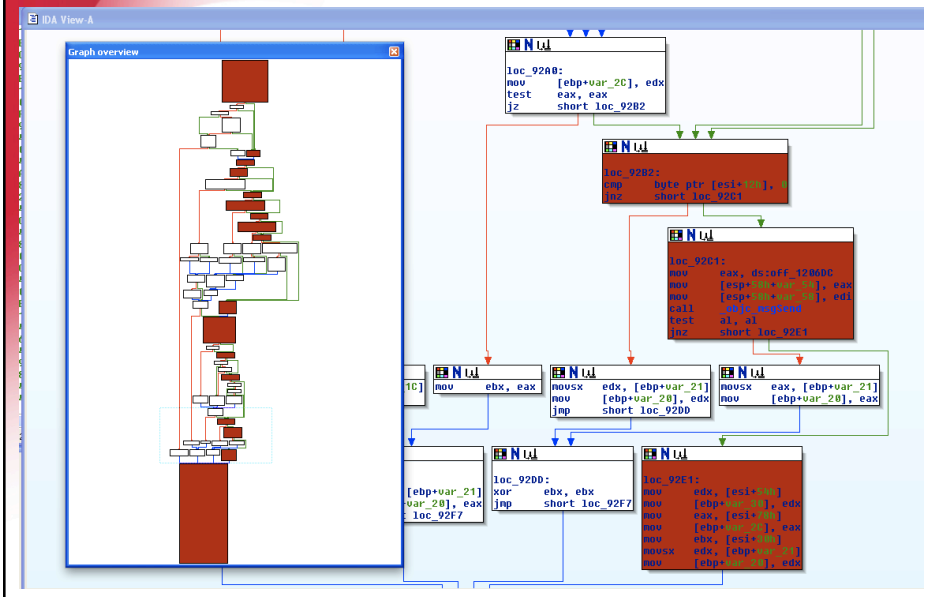
- Associate fuzz runs with code hit
- Visualize code paths
- Record number of times blocks were hit
- Compare idle traces to other traces

Limitations:

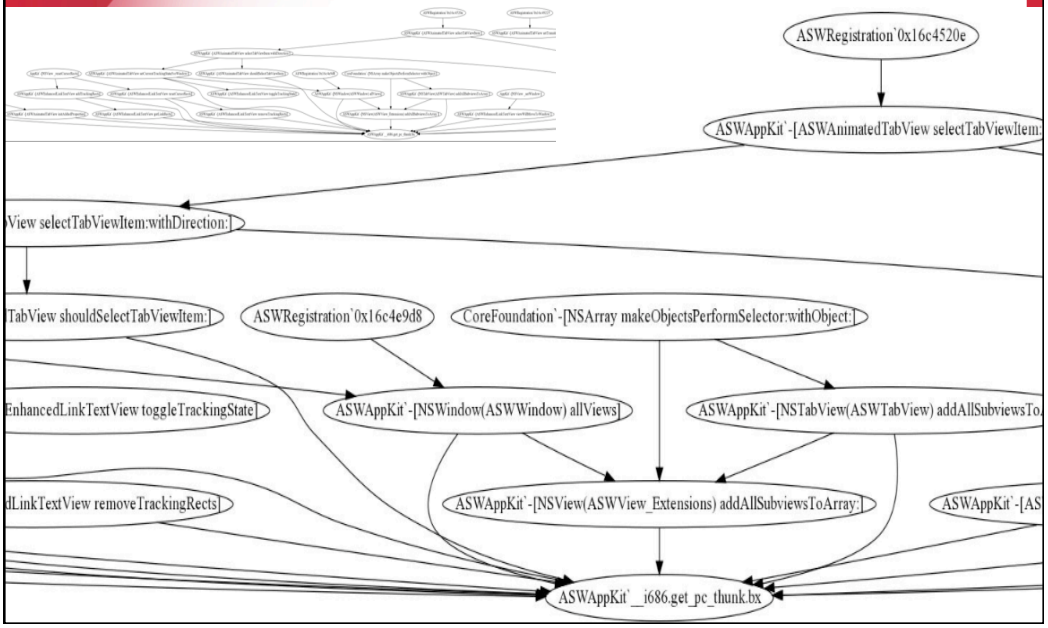
- Instruction tracing can be slow for some applications
- Again, tuning and limiting scope

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Coverage Visualization



Runtime Call Graphs



IOCTL Background

- IOCTL is a function that allows userland applications to communicate with devices or kernel modules

```
int ioctl(int fildes, unsigned long request, ...);
```

Ex:

```
int error = ioctl(fd, IOSNDRQ, data);
```

- Common for privileged code to be placed in kernel module
- Userland application then makes requests to modules with ioctl
- Three parts to request
  - Open File Descriptor
  - Unique Request Code
  - Buffer, can be and 'in' or 'out' buffer

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

IOCTL Fuzzing

- Traditionally the challenge is that the Request Code is not known
- You can brute force it, but makes it time consuming
- Brute forcing the message format can be even less fruitful
- You will not get good code coverage

- With DTrace we can quickly learn the request ID and corresponding message formats

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

First look at ioctl calls - Aggregation

```
syscall::ioctl:entry /execname != "dtrace"/ {

 @ioctlcalls[
    execname,
    fds[self->fd].fi_pathname,
    self->request
  ] = count();

}
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Ioctl Call Aggregation Output

Execname	FD	Pathname	Request
Bash	4	??/bin/cups-config	1074030202
airportd	3	??/<(NULL v_name)>/ dtracehelper	2148034564
SystemUIServer	10	<(not a vnode)>	3223349705
Vmware-vmx	32	??/<(NULL v_name>/ vmmon	3221509658

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Record ioctl calls and message formats

```
syscall::ioctl:entry /execname != "dtrace"/ {

 printf("fd=%d request=%d\n", arg0, arg1);
 printf("filename=%s\n", fds[arg0].fi_pathname);

 this->buf = (uintptr_t *) copyin(arg2, 16);

 printf("\t0x%08x 0x%08x 0x%08x 0x%08x\n",
    this->buf[0], this->buf[1], this->buf[2],
    this->buf[3]);

}
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Tracing ioctl Message Example

```
PID=622 SystemUIServer FD=14 (0x0000000e)
   request=3223349705 (0xc02069c9)
file=<unknown (not a vnode)>
buffer: 0xbfffe8dc
   0x00326e65 0x00000000 0x00000000 0x00000000
   0x0000000c 0x00000000 0x0000005400xbfffe860
   0x00000000 0x00000000 0x00000000 0x00000000
   0x00821200 0x00000002 0x00000002 0x00000000
PID=622 SystemUIServer retval=0
Return buffer: 0xbfffe8dc
   0x00326e65 0x00000000 0x00000000 0x00000000
   0x0000000c 0x00000000 0x0000000300xbfffe860
   0x00000000 0x00000000 0x00000000 0x00000000
   0x00821200 0x00000002 0x00000002 0x00000000
```

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Inline ioctl Fuzzing

- Use dtrace destructive actions
- Modify the message as it is passed from userland to kernel
- Remember this may be the in message or the out message
- Likely to cause the userland app to become unstable
- May crash the kernel module/device

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Inline ioctl Fuzzing – Flipping Bits

```
this uintptr_t* buf;
int flip[1];

syscall::ioctl:entry /execname != "dtrace" / {

    this->buf = (uintptr_t *) copyin(arg2, 4);

    flip[0] = (~ *(int *)this->buf);

    copyout(flip, arg2, 4);
}
```

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Problems with Inline Ioctl Fuzzing

- Many applications rely on ioctl, if the ioctl calls fail, the applications often do not function properly
- The kernel devices can easily corrupt the process' stack

- Ioctl inline fuzzing is useful for specialized situations
- You can easily simulate a badly behaving device
- Good for testing the applications that uses the device

- But if you take the data gathered from dtrace and use a traditional fuzzer approach… gold mine

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Traditional Ioctl Fuzzing with DTrace data

- Create your own application that calls ioctl
  - See Ilja van Sprundel's ioctlfuzz fuzzer
  - Augment it with request codes and buffer data collected with dtrace
  - Win

```
Unsigned char buf[4096]; // <- make sure this is big!
int fd = open("/dev/vmmon", O_RDWR);
unsigned int requests[] =
    {1073829395,1073829451,1074025984,1074025985,107402598
    7,1074026006, … etc

While(1){
  for(int i=0;i<sizeof(requests)/sizeof(*requests);i++){
    ioctl(fd,requests[i],rand_fill(buf));
}}
```

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Dynamic Tracing for Exploitation and Fuzzing

We saw:
- General process inspection
- Stack overflow monitoring
- Code coverage
- IOCTL Device Fuzzing

I didn't show you:
- Heap overflow monitor
- RE:Trace
- RE:dbg
- Malware analysis
- Defensive DTrace
- Higher Level Application tracing USDT

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

~ Mahalo! ~

See the RE:Trace/REDBG framework for implementation:

http://www.poppopret.org/

Questions?

Tiller Beauchamp
IOActive
Tiller.Beauchamp (@) ioactive.com

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES

Kernel Panic Demo!

IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES