Shade Analyzers

application expects (and has reserved enough space) to save all 64 bits of each register. The default is −**win32**.

The *analyzer-options* supported by the analyzers in the standard Shade distribution are:

−**U**

Print a usage message and immediately exit.

−**V**

Print a version message and immediately exit.

−**exec**

If the traced application exec's a new program image, the analyzer will not normally continue tracing the new image but will execute it natively instead. Thus, tracing the shell process, for example, will trace just the shell and not any of the commands it spawns. The −**exec** switch causes the analyzer to trace into the new image. For example, st(analyng]TJ 02.3973-1.11

Any Shade analyzer can trace a variety of different types of applications. If an application uses shared libraries, the analyzer traces the application itself, the shared libraries, and the dynamic loader. If an application is a shell script, the analyzer traces the shell as it interprets the script. Shade analyzers cannot, however, trace setuid or setgid programs unless the owner is the same as the user running Shade.

NAME

   icount − count executed instructions

SYNOPSIS

   icount [−**annul**] [−**perthread**]

DESCRIPTION

   The **icount** analyzer counts and prints the number of instructions executed by each of the specified application programs. In addition to the standard Shade analyzer switches, the **icount** analyzer accepts the following options:

   −**annul**

      Causes **icount** to]

                  −**perthread**

                           Causes

NAME

rcount − count executed instructions per region

SYNOPSIS

rcount [-o outfile] [-sample sample_info] [-skip skip_count] -r regionfile -- command

DESCRIPTION

The **rcount** Shade analyzer counts the number of instructions executed in a set of user defined regions. A region is defined by a starting PC and zero or more ending PC's. When the application executes the instruction at a region's starting PC, that region becomes active. The region remains active until the application executes one of the region's ending PC's. Note, the instructions encompassed by a region

NAME

      spixcounts − generate spix counts file

SYNOPSIS

      spixcounts [−**b** *fmt*] [−**data** *fmt*] [−**merge**] [−**s**

Both the −**b** and −**data** switches require a file name template to be specified. The file name template may contain format specifies which are replaced as follows:

%p   Replaced with the basename of the application program or the basename of the shared library. When used with the −**data** switch, this is always replaced with the basename of the application program.

%n   Replaced with a per-command sequence number. The sequence number starts out at one and is incremented for each application that is traced. This specifier is not allowed in the −**b** file name template when the −**merge** switch is specified.

%i   Replaced with the process ID of the analyzer. This specifier is not allowed in the −**b** file name template when the −**merge** switch is specified.

%%   Replaced with '%'.

If no −**b** switch is specified, **spixcounts** uses the template name "%p.%n.bb" (if the −**merge** switch is not specified), or "%p.bb" (if the −**merge** switch is specified).

THREADS

The **spixcounts** analyzer combines the execution counts for all application LWPs (threads) together.

FORK AND EXEC

If the traced application forks, the **spixcounts** analyzer forks too, and each analyzer then writes its own set of output files. The execution counts reported for the child are exclusive of the counts reported for the parent. The "%i" file name template format can be used to distinguish output files generated by the parent and child.

If the application exec's a new image and the −**exec** switch is specified (see *shade_intro*(1sh)), instruc-

NAME

pairs − instruction pairs analyzer

SYNOPSIS

pairs

addpairs

postpairs [

NAME

trips − instruction triplets analyzer

SYNOPSIS

trips [−**a**]

DESCRIPTION

The **trips** analyzer is like **pairs**(1sh) except it looks at three instructions at a time instead of two.

Normally **trips** truncates its output after printing information for the top 90% of instruction triplets. The −**a** option causes information for all executed instruction triplets to be printed.

Like **pairs**(1sh), **trips** displays statistics by opcode group rather than by opcode.

THREADS

The **trips** analyzer tracks the execution of each LWP (thread) independently, but merges the statistics for all threads together in its output.

FORK AND EXEC

If the traced application forks, the **trips** analyzer forks too, and each analyzer then reports its own set ofs trip4452.0tati1(t

NAME

window − register window analyzer

SYNOPSIS

window

DESCRIPTION

The **window** Shade analyzer tracks the register window usage for one or more applications. The output

Shade Analyzers

Caches are virtually addressed. Annulled instructions cause an instruction (or unified) cache reference,

NAME

      brpred − branch predictor performance analyzer

SYNPOSIS

      brpred [**-u**] [**-single-cpu**] [**-pcs**] *<brpredspec>+*

DESCRIPTION

      The **brpred**

NAME

hist − shade tool to print an application's most recent instructions

SYNOPSIS

hist [−**exit**] [−**pc** address[:count]] [−**ea[rw]** {B│H│W}address[:count]] [−**signal** signal] [−**o** filename] [−**num** number] [−**log** filename] [−**stdenv**] [−**notrace**] [−**traceafter** number] [−**tracepc** address[:count]] [-N] -- application

DESCRIPTION

The **hist** Shade analyzer maintains a trace history of an applications most recently executed instructions. The trace history is printed when the application causes any of several user-definable events to occur. This produces a history of instructions leading up to that event.

The following options allow you to choose when the trace history is printed. More than one of these options may be specified, causing the trace history to be dumped when any of the chosen events occurs.

−**exit**

Causes the trace history to be dumped when the application exits. In this case the trace is a history of the application's final instructions.

−**pc** <address>[:<count>]

Causes the trace history to be dumped immediately after the application executes the instruction at the given address. If 'count' is not specified, the history is dumped each time the application reaches this address. If 'count' is specified, the history is dumped only when the application reaches the address 'count' times.

−**ea[rw]** {B│H│W}<address>[:<count>]

Causes the trace history to be dumped immediately after the application read or writes the memory at the given address(r after -ea means read only, w - write only). Executing an instruction at this address does not count as a read. Use **-pc** for that. One of the characters B, H, or W must precede the address indicating either a byte, half-word (2 byte), or word (4 byte) range of addresses. If 'count' is not specified, the history is dumped each time the

instructions in the tool's trace output. The exact number of instructions in the trace depends both on the size of the buffer and on the mix of instructions the application executes. (Some instructions require more buffer space than others.) You may not use the −**log** option if you specify −**num**.

−**log** <filename>

If the instruction modified a register, the line contains a record of the form $rn=value, indicating that the given value was written to the given register. If the instruction modifies memory, the line contains a record of the form {B|H|W}address=value, indicating that the given value was written to the given memory location (byte, halfword, or word). If the instruction reads memory, the line contains a record of the form ({B|H|W}

NAME

shade_anal, shade_fp, shade_ego, shade_usage, shade_error, shadeuser_initialize, shadeuser_analyze, shadeuser_report, shadeuser_terminate, shadeuser_analusage, shadeuser_analversion − Common Shade analyzer interface and functions that must be defined by user to use the interface.

SYNOPSIS

cc [*flag . . .*] *file . . .* **libshade.a** [

The interface calls **shadeuser_analyze( )** once for each application. If the analyzer detects an error, it should issue a message and return a non-zero value. This causes the interface to ignore any remaining applications and immediately call the analyzer's **shadeuser_terminate( )** function.   If **shadeuser_analyze( )** does not detect an error, it should return zero.

When **shadeuser_analyze( )** exits, the interface calls **shadeuser_report( )** to report any results. Please

035 -w (shen)the interface calls

NAME

       shade_appname, shade_interpname, shade_appbase, shade_interpbase − Retrieve information about Shade application

SYNOPSIS

NAME

shade_appstatus − Return status of application running under Shade

SYNOPSIS

#include <shade.h>

shade

NAME

shade_bench_memory − Return Shade application's base memory address

SYNOPSIS

#include <shade.h>

char ∗shade_bench_memory(void);

DESCRIPTION

The **shade_bench_memory( )** function returns the base memory address of the application loaded under Shade. The analyzer can add this value to any address in the application to yield the corresponding address on the host. The following example shows how this function can be us _

shade us(ва)}Tj /F2 1 Tf 3.73 0 TD 09882 Tw)ue tobtas in thcontentess oe memorpriorue t)16(executowin)16(ca)16((ap

NAME

shade_getopt, shade_

NAME

shade_iset,   shade_iset_newclass,   shade_iset_newtype,   shade_iset_newop,   shade_iset_newcopy,
shade_iset_free, shade_iset_addclass, shade_iset_addtype, shade_iset_addop − Manage sets of instructions for Shade

SYNOPSIS

#include <shade.h>

shade_iset_t ∗**shade_iset_newclass(shade_iclass_t** *iclass*, …**);**

shade_iset_t ∗**shade_iset_newcopy(shade_iset_t** ∗*piset***);**

void shade_iset_free(shade_iset_t ∗*piset***);**

shade_iset_t ∗**shade_iset_addclass(shade_iset_t** ∗ **piset , shade_iclass_t** *iclass*, …**);**

.

#include <shade_ARCH.h>

shade_iset_t ∗**shade_iset_newop(spix_ARCH_iop_t** *iop*, …**);**

shade_iset_t ∗**shade_iset_addop(shade_iset_t** ∗*piset***, spix_ARCH_iop_t** *iop*, …**);**

shade_iset_t ∗**shade_iset_newtype(spix_ARCH_itype_t** *itype*, …**);**

shade_iset_t ∗**shade_iset_addtype(shade_iset_t** ∗*piset***, spix_ARCH_itype_t** *itype*, …**);**

DESCRIPTION

independeclassesets
sh~~exdepd6~~ε

SHADE_ICLASS_UBRANCH
> Selects all unconditional branch instructions.

SHADE_ICLASS_CBRANCH
> Selects all conditional branch instructions.

SHADE_ICLASS_TRAP
> Selects all instructions that explicitly trap to privileged code. Typically, this includes instructions an application uses to request system services, but does not include instructions that trap due to, say, a page fault.

NAME

　　　shade_load, shade_loadp, shade_unload − Load an application under Shade

SYNOPSIS

　　　#include <shade.h>

　　　int shade_load(const char *∗path***, char** ∗ **const** *∗argv***, char** ∗ **const** *∗envp***);**

　　　int shade_loadp(const char *∗file***, char** ∗ **const** *∗argv***, char** ∗ **const** *∗envp***);**

　　　void shade_unload(void);

DESCRIPTION

　　　The **shade_load( )** and **shade_loadp( )** functions load a new application under Shade. Both provide the

NAME

shade_lock, shade_lock_new, shade_lock_delete, shade_lock_set, shade_lock_clr − Lock critical regions of Shade code

SYNOPSIS

#include <shade.h>

shade_lock_t ∗**shade_lock_new(void);**

void shade_lock_delete(shade_lock_t ∗*plock***);**

void shade_lock_set(shade_lock_t ∗*plock***);**

void shade_lock_clr(shade_lock_t ∗*plock***);**

DESCRIPTION

These functions provide a way for Shade analyzers to lock critical regions of code in user-defined trace functions (enabled via **shade_trfun(3sh)**

NAME
    shade_

NAME

      shade_print_opt_info − Function for printing help information.

SYNOPSIS

      cc [*flag . . .*] *file . . .* **libshgetopt.a** [*library . . .*]

      #include <shgetopt.h>

      void shade_print_opt_info(char∗ *anal_info*, **const shade_options_t**

NAME

shade_run − Run and trace application under Shade

SYNOPSIS

#include <shade.h>

unsigned shade_run(shade_trace_t *∗ptrace***, unsigned** *ntrace***);**

DESCRIPTION

The **shade_run( )** function executes the application loaded under Shade and collects any requested trace information. The *ptrace* parameter specifies the start of a buffer of *ntrace* trace records into which Shade stores trace information from the application. The **shade_run( )** call returns when the trace buffer

Shade Library

NAME

      shade_setopt − Enable Shade options

SYNOPSIS

      #include <shade.h>

      int shade_setopt(shade_opt_t *opt***);**

DESCRIPTION

      The **shade_setopt( )** function allows an analyzer to set options that affect the way Shade operates. The following options are supported.

      SHADE_OPT_EXECTRACE

            This option allows an analyzer to trace an application after it execs a new executable image. When this option is in effect, **shade_run(3sh)** returns zero after the application execs a new image. The next cb 0 eooro4ecE 0 TD sc19f 13.11 083.177 wietuade_3sh84 Tw [([(image.)--3.6hade operates

following options are supported.

SHADE_setopt( )

NAME

shade_shell, shade_fshell, shade_sshell − Run application scripts in Shade

SYNOPSIS

#include <shade.h>

int shade_shell(int (∗*anal*)**(int, char** ∗∗**, char** ∗∗**, char** ∗∗**));**

NAME

shade_signal, shade_analsig, shade_sendsig − Manipulate signals in Shade

SYNOPSIS

#include <shade.h>

int shade_analsig(int *sig* **, void** (∗*handle*)**(int, siginfo_t**

NAME

  shade_splitargs − Separate Shade analyzer and application arguments

SYNOPSIS

  #include <shade.h>

  int shade_splitargs(char $**argvin$, **char** $***pargvapp$, **int** $*pargcapp$**);**

DESCRIPTION

  ppvgThe **shade_splitargs( )** function provides a mechanism for separating analyzer and application argument
  lists. It relies on a convention followed by many Shade analyzers of marking the application arguments
  with the string "--". This function search*ments(13666 Tm (DE.2 s 422-1.)-326(<0 TD 0 32.94 (**)Tj /F4 1 Tf 1 2 TD

NAME

      shade_trange, shade_addtrange, shade_subtrange, shade_intrange, shade_argtrange, − Restrict Shade trac-
ing by address range

SYNOPSIS

      #include <shade.h>

      void shade_subtrange(spix_addr_t *addrlo*, **spix_addr_t** *addrhi***);**

      void shade_addtrange(spix_addr_t *addrlo*, **spix_addr_t** *addrhi***);**

      spix_bool_t shade_intrange(spix_addr_t *addr***);**

      int shade_argtrange(const char *∗pstr***);**

DESCRIPTION

      These functions work in concert with **shade_trctl(3sh)** and **shade_trfun(3sh)** to determine which
instructions Shade traces. Shade traces an instruction only if it is selected by **shade_trctl(3sh)** or
shade_trfun(3sh) and if that instruction resides in an address range selected by the **shade_trange( )**
functions. By default, all addresses in the application are selected, so an analyzer need not call the
shade_trange( ) functions unless it wants to restrict the range of traced instructions.

      The **shade_subtrange( )** function disables tracing for instructions residing in the given address range.
The **shade_addtrange( )**

unused.

The **shade_trfun_at( )** function is like **shade_trfun( )** except it only applies to the instruction starting at the given target address. If the instruction at that address is specified by *piset*, the user-defined trace functions are called for that instruction as defined above. If **shade_trfun( )** and **shade_trfun_at( )** specify different functions, the functions in the **shade_trfun_at( )** call prevail. Moreover, the instruction at address *addr* is evaluated dynamically, so the address need not be mapped when the analyzer calls **shade_trfun_at( )**

NAME

shade_trsize − Specify size of Shade trace record

SYNOPSIS

#include <shade.h>

int shade_trsize(size_t *size***);**

DESCRIPTION

This function specifies the size (in bytes) of the Shade analyzer's trace record. Analyzers should call
shade_trsize( ) before calling the **shade_trctl(3sh)** functions in order to tell Shade the size of a tracefecord.

SHADE_TRCTL_TID
Record the ID of the thread executing the traced instruction in the **tr_tid field of the trace**

If an invalid value is passed to **shade_tset_new( )** or **shade_tset_add( )**, the function issues a diagnostic message and returns NULL.

SEE ALSO

shade_trctl(3sh), shade_iset(3sh), shade_ARCH_trctl(5sh).

NAME

shade_version − Shade library version string

SYNOPSIS

#include <**shade.h**>

const char   **shade_version[];**

DESCRIPTION

The **shade_version** character array contains a read-only string representation of the Shade library's version level.

NAME

shade_sparcv9_trctl − SPARC V9 trace parameter codes

SYNOPSIS

#include <