Oracle Linux

DTrace Release Notes

F35833-17
June 2025

Oracle Linux DTrace Release Notes,

F35833-17

Contents

Preface

The Oracle Linux: DTrace Release Notes provides information about DTrace releases for Oracle Linux and Unbreakable Enterprise Kernel.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1 About DTrace

DTrace is a powerful dynamic tracing tool that's available in Oracle Linux for use with the Unbreakable Enterprise Kernel (UEK). It has a rich feature set, supports most of the common probe providers, and is available for x86_64 and aarch64 architectures. DTrace is developed as an open source project available under the Universal Permissive License (UPL), Version 1.0. You can access source code and more information at https://github.com/oracle/dtrace-utils.

This implementation of DTrace uses existing Linux kernel tracing facilities, such as eBPF, which didn't exist when DTrace was first ported to Linux. The new implementation removes DTrace dependencies on specialized kernel patches.

This implementation is a user space application and is available on:

- Unbreakable Enterprise Kernel 8 (UEK 8) and later kernels on Oracle Linux 10.
- Unbreakable Enterprise Kernel Release 7 (UEK R7) and later kernels on Oracle Linux 9.
- Unbreakable Enterprise Kernel Release 6 (UEK R6) and later kernels on Oracle Linux 8.

DTrace is also available on Unbreakable Enterprise Kernel Release 6 (UEK R6) and later kernels on Oracle Linux 7, and requires the `libdtrace-ctf` library to run. The functionality of the `libdtrace-ctf` library is integrated into the Oracle Linux GNU tool chain for later Oracle Linux releases. Oracle Linux 7 is in Extended Support. Migrate applications and data to Oracle Linux 8, Oracle Linux 9, or Oracle Linux 10, as soon as possible.

Previous versions of UEK continue to include the original DTrace implementation.

Functionality is being delivered as it becomes available, starting with a limited set of capabilities (primarily framework functionality that doesn't offer many user visible features) but ultimately reaching, and then exceeding, earlier support.

For more information about DTrace, see Oracle Linux: Using DTrace for System Tracing.

2

Install DTrace

The following instructions provide steps to install DTrace on different Oracle Linux releases and to verify that the installation was successful.

Install DTrace on Oracle Linux 10

1. Enable the yum repository.

   If running on an x86 platform, enable the `ol10_UEKR8` yum repository for the system.

   ```
   sudo dnf config-manager --enable ol10_UEKR8
   ```

   > **Note:**
   >
   > Oracle releases UEK and DTrace packages in the `baseos` repository for aarch64 platforms. You don't need to enable any other repositories to access the DTrace packages for aarch64 platforms.

2. Install DTrace.

   Install the `dtrace` package.

   ```
   sudo dnf install -y dtrace
   ```

Install DTrace on Oracle Linux 9

1. Enable the yum repository.

   If running on an x86 platform, enable the `ol9_UEKR7` yum repository for the system.

   ```
   sudo dnf config-manager --enable ol9_UEKR7
   ```

   > **Note:**
   >
   > Oracle releases UEK and DTrace packages in the `baseos` repository for aarch64 platforms. You don't need to enable any other repositories to access the DTrace packages for aarch64 platforms.

2. Install DTrace.

   Install the `dtrace` package.

   ```
   sudo dnf install -y dtrace
   ```

Install DTrace on Oracle Linux 8

1. Enable the yum repository.

   If running on an x86 platform, enable either the `ol8_UEKR6` or `ol8_UEKR7` yum repository for the system.

   For example, run:

   ```
   sudo dnf config-manager --enable ol8_UEKR7
   ```

   > **Note:**
   >
   > Oracle releases UEK and DTrace packages in the `baseos` repository for aarch64 platforms. You don't need to enable any other repositories to access the DTrace packages for aarch64 platforms.

2. Install DTrace.

   Install the `dtrace` package.

   ```
   sudo dnf install -y dtrace
   ```

Install DTrace on Oracle Linux 7

> **WARNING:**
>
> Oracle Linux 7 is now in Extended Support. See Oracle Linux Extended Support and Oracle Open Source Support Policies for more information.
>
> Migrate applications and data to Oracle Linux 8, Oracle Linux 9, or Oracle Linux 10, as soon as possible.

1. Enable the `ol7_UEKR6` yum repository.

   For example, if you have `yum-utils` installed, run:

   ```
   sudo yum-config-manager --enable ol7_UEKR6
   ```

2. Install DTrace.

   Install the `dtrace` and `libdtrace-ctf` packages:

   ```
   sudo yum install -y dtrace libdtrace-ctf
   ```

Verify the DTrace Installation

Check that DTrace is installed to the correct location and verify the DTrace version.

1. Confirm DTrace is installed into `/usr/sbin/dtrace`.

   ```
   ls -lah /usr/sbin/dtrace
   ```

2. Display the DTrace version number.

   ```
   dtrace -V
   ```

   The output looks similar to:

   ```
   dtrace: Oracle D 2.0
   ```

3

Example DTrace Usage

The following examples illustrate current functionality in DTrace. These examples assume that `/usr/sbin` is in the `$PATH`.

Example 3-1 List probes

```
sudo dtrace -l
```

The output looks similar to:

```
DTrace 2.0.0 [Pre-Release with limited functionality]
ID   PROVIDER    MODULE                     FUNCTION NAME
1     dtrace                                         BEGIN
2     dtrace                                         END
3     dtrace                                         ERROR
4        fbt   vmlinux     trace_initcall_finish_cb entry
5        fbt   vmlinux     trace_initcall_finish_cb return
...
```

On this particular system, there were:

- 3 dtrace probes
- 87890 fbt probes (based on kprobes)
- 1262 sdt probes (based on Linux tracepoints)
- 666 syscall probes

Example 3-2 Using the `-S` option

This script uses the `-S` option to output the compiled D code as an eBPF program. The `-e` option exits after compilation.

```
sudo dtrace -Sen 'write:entry { trace(1) }'
```

The output looks similar to:

```
Disassembly of clause ::write:entry, <dt_clause_0>:
INS  OFF    OPCODE                   INSTRUCTION
0000 00000: 7b a 1 fff8 00000000    stdw [%fp-8], %r1
0001 00008: bf 0 a 0000 00000000    mov  %r0, %fp
0002 00016: 07 0 0 0000 ffffffa0    add  %r0, -96
0003 00024: 7b a 0 fff0 00000000    stdw [%fp-16], %r0
0004 00032: 79 0 a fff8 00000000    lddw %r0, [%fp-8]
0005 00040: 79 9 0 0018 00000000    lddw %r9, [%r0+24]
0006 00048: 79 0 0 0010 00000000    lddw %r0, [%r0+16]
0007 00056: 7a 0 0 0028 00000000    stdw [%r0+40], 0
0008 00064: 7a 0 0 0030 00000000    stdw [%r0+48], 0
0009 00072: 7a 0 0 0018 00000000    stdw [%r0+24], 0
```

```
0010 00080: 62 0 0 0000 ffffffff    stw  [%r0+0], -1              ! = EPID
0011 00088: 62 0 0 0008 ffffffff    stw  [%r0+8], -1              ! = CLID
0012 00096: 62 9 0 0000 ffffffff    stw  [%r9+0], -1              ! = EPID
0013 00104: 62 9 0 0004 00000000    stw  [%r9+4], 0
[...]
```

Example 3-3 DTrace script

```
sudo dtrace -n '
syscall::write:*
{
    this->x = 3;                 /* clause-local variables */
    this->y = 8;
    trace(this->x * this->y);
    trace(&`max_pfn);
}'
```

In the example script:

- Probe all `write()` system call probes simultaneously using a wildcard.
- Probe with recording the address of a kernel identifier (`max_pfn`) and other data items.
- Associate several probes with a single action.
- Clause-local variables are used.
- The `trace()` action is used to report output.

4

DTrace Changelog

The changelog provided here describes the major features and changes in each release and also lists any known issues.

2.0.3-1 (Jun 10th, 2025)

This release is only available on Oracle Linux 10.

New features:

- USDT probes in executables and shared libraries that are compiled with LTO are now supported. USDT probe definitions are now recorded in .note.usdt ELF notes.
- The pr_psargs member of the psinfo_t translator is now implemented. It uses the new execargs builtin variable.
- The pid provider now supports offset-based instruction probes.

Bugfixes:

- Automatic selection of fprobe vs kprobe based FBT probes has been corrected.
- Trampoline use of the BPF stack for scratch space has been corrected.
- The rawfbt provider has been fixed to perform symbol lookups on the actual true symbol name.
- BPF attach type detection has been fixed.
- The probeprov, probemod, probefunc, and probename builtin variables now report the correct values for late USDT processes.
- The value of arg3 for sched:::enqueue and sched:::dequeue has been corrected.
- No longer register fbt and rawfbt twice.
- Discovery of new probes now correctly loops through all registered providers rather than just the initial providers.

User-visible changes:

- The kernel symbol name filter has been made less strict.
- Failures in enabling a probe now report the probe name.

Internal changes:

- Builtin variables are now implemented with individual functions rather than a single large function, reducing compiled BPF program size.
- Relocation types R_BPF_64_ABS64 and R_BPF_64_ABS32 are now supported.
- The proc provider has been implemented using the standard SDT provider implementation.
- The fbt and rawfbt providers have been consolidated, and significant performance improvements have been applied.

- The hashtable implementation has been moved to libcommon and is now used by dtprobed and libdtrace.

Testsuite changes:

- Many testcases have been improved.
- Use of bash and gawk throughout the testsuite has been made more consistent.
- Tests can now use CC, OBJCOPY, OBJDUMP, NM, ... as shell variables in scripts to allow setting values that apply to all tests.

2.0.2-1 (Dec 6th, 2024)

New features:

- Translators for kernel versions 6.10 and beyond have been added to ensure translator support through (at least) 6.12 kernels.
- FBT return probes based on fexit probes now report return values.
- The print() action has been enhanced with type information for its argument.
- USDT probes that are discovered after tracing started (either because of the use of wildcards in the probe specification or with the -Z option) are now supported.
- USDT probe argument type support has been implemented, incl. translated types and argument mapping.
- A manual page for dtprobed has been added.
- The rawfbt provider has been implemented. This is a new provider that you can use to trace all functions that are available through kprobes. This includes <func>.<suffix> function names that are the result of compiler optimizations.

Bugfixes:

- Since BPF assembler source is written in 'normal' assembler syntax, the -masm=normal option is to be passed to BPF gcc invocations.
- Parsing of bpf_helper_defs.h is more tolerant of version changes.
- The BPF-to-CTF conversion works for kernel modules.
- DTrace no longer tries to obtain a return value for void functions.
- The stddev() aggregation function carry-over computation has been corrected.
- The clear() action now only clears the aggregation for which is it called.
- ERROR probe firings were corrupting probe arguments for the probe whose execution triggered the error.
- ERROR probe firings didn't always report the correct probe id.
- Relocations in the ERROR probe program were being done too early, causing some values to be incorrect.
- Buffer size calculations have been corrected.
- Casting no longer renders an lvalue immutable.
- Multiple USDT providers in a single ELF object now work.
- Self-grab support has been improved.
- The umod/usym/uaddr actions are improved.

- Various memory leaks have been resolved.

User-visible changes:

- The header files to support USDT probes (sdt.h, and so on) have been moved to /usr/lib64/dtrace/include/sys. Support for pkg-config has been added to have a convenient way to obtain installation paths for libraries and header files.
- DTrace provides an appropriate error message when it's run in a mode that requires root privileges.
- The manual pages for dtrace and dtprobed are updated to use better markup.

Internal changes:

- Keys for @[mod()], @[sym()], etc are now consolidated where possible.
- Trace data consumption now processes only one set of records per iteration, to avoid possible starvation issues.
- Various pieces of dead code are removed.
- The association between probes and clauses now operates at the statement level. This more accurately reflects the structure of tracing programs, and is needed to support discovery of probes after tracing has started.
- The EPID concept has been deprecated. A value for the 'epid' built-in variable is still generated for backward compatibility. The value has no semantic meaning, and satisfies the requirement that it's unique for every instance of a particular probe.
- BPF tracing programs can now contain clauses that can be used by probes that are discovered after tracing already started.
- The internal implementation of pid probes, USDT probes, and is-enabled USDT probes has been consolidated (and simplified) in a single trampoline for the underlying uprobe.
- DTrace (specifically dtprobed) no longer depends on libsystemd. It can still report its activity status to systemd.

Testsuite changes:

- Many test cases are adjusted to not depend on optional features, and to have more predictable behavior.
- Use of bash and gawk throughout the testsuite is more consistent.

Build-time:

- All installation paths can be configured to use flexible file system layouts for distributions.
- Support for valgrind is now optional.

Known problems:

- Enabling a large amount of probes (in the order of > 50,000) consumes a significant amount of memory to hold the BPF programs, which might trigger the OOM killer in the kernel.

2.0.1-1 (May 6th, 2024)

New features:

- Function Boundary Tracing (FBT) probes can now access function arguments using argv[n] where n is bound by the number of arguments of the function. The data type of each argument is a generic uint64_t for now.

Internal changes:

- Function Boundary Tracing (FBT) probes are now implemented using the more lightweight fentry/fexit kernel tracing facility. Fallback to kprobes is provided for kernels that don't provide fentry/fexit probing using BPF.

Build-time:

- D translators are now included with the DTrace source code, and are installed for all supported kernel versions. This means that building DTrace no longer requires access to kernel development headers.

2.0.0-1.14 (Mar 5th, 2024)

Fourteenth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features

- The io provider has been implemented.
- The print() action has been implemented.
- The link_ntop() subroutine has been implemented.
- The cleanpath() subroutine has been implemented.
- The d_path() subroutine has been implemented to always return "<unknown>". This is needed to ensure that the io and procfs translators compile.
- The -xcpu option has been implemented.
- The -xaggpercpu option has been implemented.
- The -xlockmem option has been improved. The limit is set before retrieving probe info, and the default behavior is now "unlimited" (meaning most users will not have to worry about this option).
- The pid provider now supports offset-based probe names.
- Aggregations of stacks are now supported.
- The retrieval of rawtp argument information has been improved.
- You can now delete an element in an associative array by assigning a literal 0 to it, regardless of the element datatype.
- The lexer has been improved to support module names that start with a numeral so that they can be used. For example, 9p`v9fs_remove.
- A basic configure script has been added to help building and packaging in various distributions.

- USDT probe information maintained by dtprobed is now stored under /run to ensure it can survive daemon restarts.
- DTrace can now be used for tracing with upstream kernels without requiring any additional patches, albeit with some limitations.
- The ip provider has been implemented.
- The trunc() action has been implemented.
- The pcap() action has been implemented.
- The inet_ntoa6() subroutine has been implemented.
- The inet_ntop() subroutine has been implemented.
- Support for modules.builtin.ranges data from the kernel has been added. This is the new way to determine module name association for kernel symbols that are built into the kernel. Support for kallmodsyms is retained for kernel that don't support modules.builtin.ranges yet.
- A BTF-to-CTF convertor has been added to support using DTrace with kernels that don't provide CTF data. Note that BTF is currently more limited than CTF. For example, BTF doesn't provide datatype information for kernel variables.

Bugfixes

- Drop counter handling is fixed for local-only updates.
- Dedicated space has been introduced for call stacks so that stackdepth and temporary strings don't overwrite one another.
- dt_tp_event_info() has been corrected so as not to overrun its buffer.
- Compilation of BPF code that uses BPF helpers now uses the bpf_helpers.h header file from libbpf-dev[] instead of the (deprecated) bpf-helpers.h header file that the gcc BPF cross provided.
- Because of the need to support DTrace on older kernels, BPF source code files are now compiled using -mcpu=v3 to ensure that the object code is acceptable to the BPF verifier in older kernels.
- When a DTrace instance would trigger the END probe to be processed, any and all other dtrace instances on the system would have their END probe fire as well because the dtrace provider trampolines weren't validating the tgid of the task triggering the probe.
- The initialization of the cpuinfo BPF map could cause a buffer overrun on systems with non-sequential online CPU ids.
- On kernels that support preemptive BPF program execution, probe data could get corrupted. As a temporary fix, concurrent BPF program execution for DTrace probes is blocked.
- Struct and union member access in alloca()-allocated memory no longer cause a BPF verifier violation.
- Bitfield offset calculations have been corrected.
- Disassembler output for endianness conversion instructions has been corrected.
- Bounds checking of array datatypes of size 0 or 1 in the kernel is now skipped because they are commonly used in the kernel as anchors for dynamically sized arrays.
- Zero constants are now checked at compile time wherever NULL pointer argument checking is done.

- Uprobes are now created using the offset in the inode rather than based on an absolute address.
- Building in various forms of kernel builds is now more streamlined.

Internal changes

- Code has been restructured to better support SDT-based providers. While such providers (lockstat, io, and so on.) used to be based on static probes in the kernel source, they're now implemented with fbt, rawtp, and even syscall probes. Probe trampolines can become involved. Changes, notably in cg, better support these providers. Also, the underlying probes are using rawtp more rather than relying solely on fbt.
- Support for compilation in older environments (esp. older compilers) has been improved.
- There have been several build improvements, especially for cross compilation and to build with upstream kernels.
- A bunch of code to parse strings has been removed, relying instead on flex for this support.
- The creation and deletion of USDT probes has moved from dtprobed to dtrace.
- The dtprobed now uses presets for daemon restarting.
- The dependency on waitfd() has been replaced with a mechanism that doesn't depend on this system call.

Testsuite changes

- Test dependence on tick-* probes has further been reduced. The tick-* probes can behave poorly on some kernels, depending on how their timers subsystem is configured (CONFIG*_HZ*). Reducing this dependence has gone on over several releases to improve the robustness of these tests.
- Fix err.* tests that force XFAIL to report so correctly.
- Skip lockstat testing before 5.10.
- Fix the use of syscall::execve:entry args[], because there are two levels of dereferencing userspace addresses, requiring two copyin*().

2.0.0-1.13.1 (Jun 7th, 2023)

Thirteenth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

Bugfixes:

- Upgrading DTrace using RPMs now correctly restarts dtprobed.

Testsuite changes:

- Some tests can leave orphaned tracing events registered with the kernel if the tests timeout and the dtrace process is killed. Such probes are now reported and cleaned up after each test is run.

2.0.0-1.13 (May 26th, 2023)

Thirteenth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- Full support for is-enabled USDT probes.
- An error will be reported when a tracing script requires more space to store aggregation data than is available per the aggsize option value.
- An error will be reported when a tracing script requires more space to store dynamic variables than is available per the dynvarsize option value.
- Support for data drop counters for principal buffers, speculation buffers, aggregations, and dynamic variables.
- The proc:::signal-clear probe has been implemented.
- The sched provider has been implemented for a limited set of probes (and with some limitations). Available probes are: dequeue, enqueue, off-cpu, on-cpu (limited trigger locations), surrender, tick, and wakeup. Note that the cpuinfo argument for dequeue and enqueue (arg1) is NULL due to system limitations. Future releases will incrementally expand this provider.
- The lockstat provider has been implemented. All lockstat probes are implemented, but depending on the runtime kernel configuration, some probe may not trigger in all cases (particularly for lock operations that are forced to be inlined). Also, kernels prior to 5.10.0 contain a bug that can cause kernel deadlock when a kretprobe is used on spinlock functions. The lockstat provider is not enabled for such kernels for safety.
- True NULL strings are now supported.
- The uregs built-in variable is now supported on older kernels as well.
- New option 'linknommap' has been added as a workaround for elfutils bugs related to mmap() usage.

Bugfixes:

- The error message issued by dtprobed to report incorrect helper data size was reporting the expected and received values backwards.
- USDT probes in programs that live in different fs namespaces are now fully supported.
- When multiple USDT probes were specified, only the first one would get provided properly.
- Properly recognize all forms of the 'char' datatype as equivalent.
- Do not allow iregs to be increased beyond its default value (the number of BPF registers).
- The uaddr handling has been fixed to not trigger a segmentation fault for pid 0.
- Tracepoint argument datatypes that are expressed by the kernel using symbolic array size specifiers are now handled correctly.
- The established behaviour of DTrace when storing data in a speculation has been to abort clause execution at any statement that would cause a speculation buffer overflow. Code to perform overflow checks when storing data in a speculation are now generated correctly.
- Some faults were not reporting the PC of the fault location.
- FBT probes are no longer provided for compiler-generated internal symbols. Such symbols cannot be probed anyway.
- Multiple memory leaks were resolved.
- Integers loaded from an associative array are now promoted to 64 bits.

- Failure to allocate a dynamic variable now reports a dynamic variable drop warning, and aborts the clause execution.
- DOF parser crash causes were fixed in dtprobed.
- USDT probes in non-PIE executables are now fully supported.
- Multiple programs providing their DOF to dtprobed simultaneously could cause some of their probes to not get created.
- Support for shared libraries and executables with very large numbers of USDT probes (500+) has been improved.

Internal changes:

- DOF_VERSION_3 has been added for the new-style USDT is-enabled probe mechanism that is not compatible with the previous versions.
- Userspace probe scanning was reworked to resolve performance issues.
- The 'cpuinfo' BPF map can now support configurations where CPU ids are not strictly sequential.
- The GCC BPF support in some gcc/binutils releases did not offer a way to express an atomic add operation. As a workaround, the DTrace source code provides its own atomic_add() construct.
- The handling of associative arrays and TLS variables has been consolidated because they are both implemented using dynamic variables.

Testsuite changes:

- The testsuite can now specify kernel modules that are needed for tests.
- A test has been added to verify whether libctf bug #30264 is present on the system. The libctf bug breaks offsetof() for members of unnamed structs and unions at non-zero offsets.
- A test has been added to test multiple simultaneous dtrace instances tracing multiple processes,
- Various tests using tick-* probes without actually requiring them have been reworked using non-timer based probes for efficiency and stability.
- Test have been added for many dtrace options.
- The testsuite can now support interpreter-style executable .d files using #!dtrace (the actual pathname for dtrace will be substituted during test execution).

Known problems:

- Programs and shared libraries that make use of is-enabled USDT probes and were built using a previous version of dtrace will need to be rebuilt for is-enabled probes to work.

2.0.0-1.12 (Feb 27th, 2023)

Twelfth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The `bcopy()` subroutine no longer enforces that its first argument (source address) can't be an `alloca()'d` memory region. While this restriction is documented, it was never enforced in DTrace and the restriction has no practical reason.
- The `clear()` and `tracemem()` actions have been implemented.
- The `switchrate` and `aggrate` options have been implemented.
- The `cpc` and `proc` providers have been implemented.
- The `copyout()` and `copyoutstr()` subroutines have been implemented.
- The `uregs[]` built-in variable has been implemented.

Bugfixes:

- The maximum `strtab` size has been increased to `SSIZE_MAX`.
- Probe argument information is only be retrieved once per probe.
- Handling string values in `alloca()'d` memory has been fixed.
- The `basename()`, `dirname()`, `strchr()`, `strrchr()`, `inet_ntoa()` subroutines have been updated to fully support using arbitrary address pointers.
- The return value of `copyin()` is now a valid offset into `scratchmem` (native representation of a pointer to `alloca()'d` memory).
- The `arg0` and `arg1` probe arguments for `profile-*` and `tick-*` probes have been corrected. (The `arg2` argument is still unimplemented.)
- The evaluation order of arguments to `bcopy()` has been corrected.
- Runtime bounds checking has been implemented for scalar array access.

Internal changes:

- Selection of the correct arch-dependent asm include hierarchy for building the precompiled BPF function library has been corrected.
- A few potentially unsafe calls to printf-style functions have been fixed.
- The manpage for dtrace has been moved to section 8 (System Management Commands).
- The error handling mechanism between libdtrace and consumer front-ends has been amended to allow error reporting for non-probing related issues.
- The `copyinstr()` subroutine has been updated to use the temporary string mechanism.
- The tracking of pointers to `alloca()'d` memory and pointer to DTrace managed memory has been improved, and explicit tests for it have been added to the testsuite.
- The code generator uses indirect load instructions for pointers to `alloca()'d` and DTrace managed memory for efficiency and to enable the BPF verifier to perform access checks.

Testsuite changes:

- The `copyin*()` tests are now more robust with the use of a distinct trigger.
- Various tests have been moved from XFAIL to PASS status to reflect the implementation of new features and because some bug fixes.
- Various tests were improved.

- Various new tests were added.

Known problems:

- The `uregs[]` built-in variable isn't supported on kernels before 5.15.

2.0.0-1.11 (Nov 9th, 2022)

Eleventh errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The args[] built-in variable has been implemented.
- Support loading scalars from kernel space addresses.
- The copyin(), copyinto(), and copyinstr() subroutines have been implemented.
- A -xlockmem option has been added to adjust the kernel locked-memory limit. When loading BPF maps or programs fails in a way that might indicate that the locked-memory limit is too low, an error message is printed to suggests using this new option.
- Support for aggregations indexed by a key (tuple) has been added.
- Disassembler annotations have been added for aggregation variables.
- The setopt() action has been implemented. A limited number of options is currently supported.
- The pid provider has been changed to ignore compiler-generated internal function names.
- The USDT provider has been implemented for basic use cases. Regular, non-is-enabled probes are supported for executables that are referenced explicitly (by pid) in the probe script. Argument mapping and wildcard probe specifications are not supported yet.

New dependencies:

- The USDT provider support depends on the availability of libfuse version 2 or 3. At build time, preference is given to libfuse 3 if available. The build process supports forcing building against libfuse 2 by passing 'libfuse2=yes' to the make command.

Upgrading:

- The USDT implementation depends on an always-running daemon (dtprobed): the corresponding systemd dtprobed.service is automatically started in relatively early boot in non-rescue scenarios, but when DTrace is first installed, or if it is upgraded from a version before the daemon existed (before 2.0.0-1.11), any probes in programs that were already running before that point will not appear in DTrace's list of available probes until such programs are restarted.

Bugfixes:

- Arguments of sdt-provider probes are now correctly populated using the tracepoint data.
- Argument handling for dtrace:::, fbt:::return, pid:::, and syscall:::return probes has been cleaned up.
- The dtrace utility is now able to handle multiple args after --.
- The -xcpp, -xctfpath, and -xverbose options have been fixed.

- Some bugs with typecasting and internal integer storage have been fixed.
- The libproc search of rtld_global has been improved for glibc changes.
- In procfs.d, projid_t was renamed to resolve a conflict with the kernel.
- In the parser, support has been added for slices of typedefs.
- String comparison involving non-DTrace pointers has been fixed.
- The value of the execname built-in variable is now correctly recognized as a non-DTrace pointer.

Internal changes:

- The code generator is able to adapt to BPF-helper-function availability differences between runtime kernels.
- Read-only blocks of zeros for initializing BPF maps have been consolidated.
- Tuples are now constructed with their component values at predictable offsets based on their datatype rather than their value..
- Support for the BPF dt_bpf_map_next_key() helper to iterate over the keys in a BPF map has been added.
- Support for multiple copies of aggregation data (DT_AGG_NUM_COPIES) is no longer needed and has been removed.
- Support for creating a map (array or hash) of maps has been added, including functions to perform lookups and updates of inner maps.
- The storage of aggregation data has been modified to make use of an array of BPF hash maps, indexed by CPU id. As a result, aggregation data for each CPU is stored in its own BPF hash map and can be modified without affecting the data for other CPUs.
- Error reporting for BPF program load, map creation, CTF, and dlib load has been cleaned up.
- Some code has been refactored and some obsolete code removed.

Testsuite changes:

- Add support for '-e' in test options.
- Tests that are expected to fail have improved xfail messages.
- Support has been added for more stringent, @@nosort checking.
- Problems with "unstable" tests are report as XFAIL.
- Tests that fire many times (historically using tick-n) are more robust.
- Various tests have been moved from XFAIL to PASS status to reflect the implementation of new features and in view of some bug fixes.
- Various tests were improved.

Known problems:

- On some aarch64 systems the copyin(), copyinstr(), and copyinto() subroutines may report a fault due to limitations in the BPF implementation at the kernel level. This problem seems to be related to specific CPU features.

2.0.0-1.10 (Apr 26th, 2022)

Tenth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The trace() action supports array, struct, and union values.
- The execname built-in variables is now implemented.
- The inet_ntoa() subroutine has been implemented.
- The progenyof() subroutine has been implemented.
- The getmajor() and getminor() subroutines have been implemented.
- The mutex_owned(), mutex_owner(), mutex_type_adaptive(), mutex_type_spin(), rw_read_held(), rw_write_held(), and rw_iswriter() have been implemented.
- The alloca() and bcopy() subroutines have been implemented.
- Associative arrays have been implemented. They are supported for both global and TLS variables.
- Disassembler annotations have been added for associative arrays, register spills, and string constants.
- The translators have been updated to support up to kernel series 5.16.
- Faults will now report the PC (program counter) where the fault is reported.

Bugfixes:

- Register allocation leaks were fixed.
- NULL pointer verification has been optimized to avoid checking the same pointer more than once.
- NULL pointers handling in ternary conditionals are now supported.
- Casting of pointers to integers has been fixed.
- Negative (immediate) values in signed conditionals are now printed correctly.
- Disassembler annotations for TLS variables have been corrected.
- The DIFO strtab handling has been reworked to fix multiple bugs.

Internal changes:

- The strlen() subroutine is now implemented using the bpf_probe_read_str() BPF helper.
- Strings are no longer stored using a length prefix.
- BPF functions that are implemented in C or assembly code are no longer statically listed in the DTrace source code. Their existence is determined at runtime when the dlibs are loaded.
- All load-time constants are now handled by the relocation mechanism.
- New function dt_dis_insn() can be used by developers to disassemble a single instruction..
- The implementation of pre and post arithmetic has been optimized.
- Relocation support for the 'add immediate' instructions has been added.

- The substr() subroutine has been optimized to reduce register pressure.

Testsuite changes:

- Various tests have been moved from XFAIL to PASS status in response to the implementation of new features and in view of some bug fixes.
- Various tests were improved.

2.0.0-1.9 (Dec 8th, 2021)

Ninth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The rand() subroutine has been implemented.
- The ftruncate() subroutine has been implemented.
- The basename() and dirname() subroutines have been implemented.
- Thread-local storage (TLS) variables have been implemented. For now, only non-indexed variables are supported.
- The strtok() subroutine has been implemented.

Obsolete features:

- The ctf_module_dump tool has been removed. It is no longer needed.

Bugfixes:

- The substr() and strjoin() subroutines now correctly store the result string length in the string length prefix.

Internal changes:

- The temporary string (tstring) support in the code generator has been improved to provide better development level diagnostics.
- The lifecycle handling of temporary strings has been updated to handle assignments and ternary expressions correctly.
- The substr() and strjoin() subroutines have been reworked to provide a much more optimized implementation.
- More efficient code is new generated for storing a string value in the trace output buffer.
- Improvements were made to the generic hashtable (htab) in libdtrace,

Testsuite changes:

- A results post-processor was added to various tests to work around CTF error message differences between libdtrace-ctf and libctf.

Known problems:

- Complex nested expressions may cause the code generator to run our of usable registers. This is a known problem with the register lifecycle tracking.

2.0.0-1.8 (Oct 15th, 2021)

Eighth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The htonl(), htonll(), htons(), ntohl(), ntohll(), and ntohs() subroutines have been implemented.
- String comparison has been implemented.
- The strchr(), strrchr(), index(), rindex(), strstr(), and lltostr() subroutines have been implemented.
- Support has been added to be able to resolve symbols in compressed kernel modules.
- Speculative tracing has been implemented. Full support for the speculate(), commit(), and discard() actions is available, as is support for the speculaton() subroutine.
- It is now possible to run dtrace under valgrind.

Bugfixes:

- Symbol resolution for loadable modules was broken. This has been corrected.

Internal changes:

- Support has been added for the endianness conversion BPF instruction.
- All uses of perf_event_open() now specify the PERF_FLAG_FD_CLOEXEC flag.

Known problems:

- String sizes greater than 128 characters may pose problems with some string operations due to BPF verifier limitations.

2.0.0-1.7 (Sep 9th, 2021)

Seventh errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- Argumsnts passed for SDT probes can now be retrieved using the arg0 through arg9 builtin variables.
- A -xbpflog option has been added to request the BPF verifier log to be generated and displayed regardles sf the outcome of trying to load BPF programs. The option can also be set with a D option pragma.
- The strjoin() subroutine has been implemented.
- The substr() subroutine has been implemented.

Bugfixes:

- Trampoline generation has been corrected to ensure that the correct probe context is set during code generation.

- The type alignment handling code used to determine the alilgnment size for a given datatype was treating enums as integers, which is incorrect. Proper alignment determination is now done, avoiding libctf-related failures.
- The handling of ERROR probe invocations within the BEGIN probe execution has been fixed.
- The size of string data in the trace output buffer has been corrected to acocunt for the 2-byte length prefix and the terminating NUL byte.
- The data size for value copy operations has been corrected. It was determined solely on the data size of the source data, even if the destination was smaller. It now uses the lesser of the two sizes.

Internal changes:

- Provider implemenations now use standard functions to clear oe copy the CPU register state at the time a probe fires.
- New macros set_upper_bound() and set_lower_bound() are available for use in C-to-BPF source code. They are used to provide hints about value and range boundaries for the BPF verifier.
- Precopiled BPF code can now use the STRSZ BPF symbol to represent the maximum string size.
- The precompiled dt_memcpy() function has been replaced with a call to the bpf_probe_read() BPF helper function.
- Support has been added for the compilation of BPF assembler source files (.S) into object files (.o). This feature makes uses of the GCC BPF cross compiler.
- The generic scratch memory area is now accessible through a pointer to its base address. This pointer can be found in dctx->mem. The stack trace implementation has been updated to make use of this area. This scratch memory area is also used to provide temporary string space to be used in string manipulation functions.
- The length prefix for strings has been changed from a variable-length integer to a 2-byte fixed width integer. This was made necessay due to BPF verifier limitations. This is an interim solution while a more permanent reworking of the string handling code is dveloped.

2.0.0-1.6 (Jun 18th, 2021)

Sixth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- Instruction offsets are printed as 4 digit values to accommodate the larger size of BPF programs used to implement probe programs.
- String constants can be used as values in D clauses, and variables can hold string values. Built-in variables that hold string data can be assigned to variables and they can be used as values in expressions and as action arguments.
- The trace() action supports strings.
- The strlen() subroutine has been implemented.
- The following built-in variables are now supported: probeprov, probemod, probefunc, probename, caller, stackdepth, ucaller, ustackdepth, errno, and walltimestamp.

- The following actions have been implemented: stack(), ustack(), umod(), usym(), and uaddr().

Bugfixes:

- The storage size was not always set correctly for global and local variables causing data corruption. When variables are not declared explicitly, their datatype may not be known until their first use. The storage size is now always set based on the explicit or discovered datatype of the variable.
- Built-in variables are implemented as global variables within a specific variable ID range. Their value is not stored in the global variable storage area and they therefore do not have a storage offset. The variable listing in the disassembly output was printing -1 as offset. The offset will no longer be printed for built-in variables.
- A memory leak related to the ERROR probe has been fixed.
- Relocations of 64-bit data items were being truncated to the lower 32 bits. This has been fixed.
- Storing data in an aggregation was considered a data recording action. This resulted in probe firings being reported by the consumer for clauses that do not actually store data in the probe output buffer. This behaviour was not intended. Aggregation data generation is no longer a data recording action.

Internal changes:

- Global and local variables are now stored more efficiently by taking into account their size and alignment requirements.
- Probe descriptions (id, provider name, module name, function name, and probe name) are now stored in the 'probes' BPF map. The values are offsets into the string constant table.
- The string constant table is loaded into the 'strtab' BPF map as the value of the singleton element with key 0.
- String hash value calculations have been unified into a single function that is called from all code that needs it.
- Variable length integer support has been added. It will primarily be used to store the length of strings inline with the character stream.
- The memory copy function (implemented as pre-compiled C code, compiled to BPF) has been optimized and has been made more robust.

Testsuite changes:

- Various tests have been moved from XFAIL to to PASS status in response to the implementation of new features and in view of some bug fixes.
- Various tests were improved.

Known problems:

- The assignment of values of a datatype that is larger than 256 bytes does not currently work due to limitations in the memory copy implementation.
- While the DTrace option to set a specific maximum string size is accepted by the command line tool, settings beyond 256 bytes do not work correctly.
- The -Z option (allowing clauses that do not match any available probe) does not allow for registering a clause to be enabled at a later time when the probe becomes available.

2.0.0-1.5.1 (Apr 12th, 2021)

Fifth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The pid provider has been implemented, enabling function boundary tracing at the userspace level (in shared libraries and executables). Future development will augment the functionality provided here with arbitrary instruction tracing at the userspace level.
- The ERROR probe (dtrace provider) is implemented. Some error conditions such as division by zero or NULL pointer dereferencing are explicitly checked in the BPF program because they constitute fatal failures in BPF program execution.
- Normalization aggregation actions have been implemented: normalize(), and denormalize().
- Support has been added for global and local variables of size greater than 8 bytes and accessing variables by reference, including allowing struct assignment statements, for sizes up to 256 bytes. Future work will allow larger value sizes.
- A -xbpflogsize=N option has been added to specify the maximum size of the BPF verifier output log. This log is generated when a BPF program cannot to be loaded into the kernel. The option can also be set with a D option pragma.
- The -xdisasm=N support for the -S option has been improved. The list of available disassembly listings has been updated. The value of <N> is the sum of any number of the following available listings:
  - 1 = After compilation and assembly of a clause function.
  - 2 = After constructing a probe program.
  - 4 = After linking dependencies into a probe program.
  - 8 = After all processing, prior to loading a probe program.

Packaging changes:

- Sample scripts have been added for building DTrace on Ubuntu.

Bugfixes:

- Various aggregations bug fixes: resetting aggregations, formatted printa(), not printing aggregations with no data (using per-aggregation latches), etc.
- Bit-field operations have been fixed in a manner that preserves legacy behavior (aligning each bit field to the size of the next largest integer type).

Internal changes:

- The implementation of kernel tracepoint based providers has been reworked for greater consistency and to accommodate the needs of the new pid provider implementation. The pid provider also provides a sample for implementing providers that expose probes that do not map one-to-one to kernel probes.
- There is now a mechanism to turn off dual-copy aggregation code. We anticipate using that mechanism when we migrate to newer kernels, but for the time being it is simply using up excessive BPF map space.

- The eventfd mechanism is used as a replacement for the condition variable that used to signal that one or more processes terminated. This means that process termination notifications are processed together with trace buffer data notifications. The dtrace_sleep() function has been deprecated.
- The source code was refactored for greater stylistic consistency, and a style guide (CODING-STYLE) was added.
- A standard implementation for *_add and _del htab functions was introduced.
- Jump-target relocation for generated BPF code was fixed to handle unlabeled BPF_NOP instructions.
- Handle translators with definitions that vary in more than two kernel releases.
- The get_gvar() and set_gvar() pre-compiled BPF functions have been removed.

Testsuite changes:

- New tests have been added or XFAIL annotations revised for new features.
- A probe to test/unittest/pragma/*libdep* tests has been added to eliminate their reliance on undefined behavior with regards to what library dependencies mean in the absence of any probes.
- There are improvements in aggregation tests.
- Some disassembly tests have been added.

Known problems:

- Some architecture (like aarch64) set aside a hardcoded amount of memory for JIT compiled BPF programs. Each program or sub-program takes up a whole number of pages in memory. If the kernel has been configured with a large pagesize (16k or even 64k), the reserved amount of memory may not be sufficient to support a larger amount of probes to be used at the same time.

  Note that the reserved memory is system-wide so concurrent DTrace tracing sessions will consume memory from the same limited pool of pages.

  There is no known workaround for this at the current time.

2.0.0-1.4 (Dec 9th, 2020)

Fourth errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- Aggregations have been implemented. For now, only non-indexed aggregations are supported, e.g. @, @a, @foo, but not @[1], @a["foo"].
- All aggregation functions have been implemented: avg(), count(), llquantize(), lquantize(), max(), min(), quantize(), stddev(), and sum().
- Argument checking for aggregation functions has been improved.
- The printa() action has been implemented for standard aggregations.

Bugfixes:

- Bitwise negation has been corrected.

- Reporting for quantize() has been corrected. No data was being reported when all values mapped to the last bin.
- END clauses are now executed correctly when the consumer triggers tracing to stop.

Internal changes:

- Aggregations now accumulate data in per-CPU kernel buffers (in a BPF map) and the consumer retrieves a snapshot of all CPU buffers as needed. This means that the only aggregation happening at the consumer level constitutes aggregating the values across all CPUs.
- Macros have been added to support manual generation of BPF code. Each BPF instruction used to take two C statements: the instruction was defined and then appended to a list. Use of the new macros eliminates hundreds of such lines and makes the C code look much more like the BPF it is generating.
- Restore the error message "%s %s( ) may not be called from a D expression (D program context required)", which had been disabled during development.
- Remove the obsolete dt_bpf_builtins.h header file.
- Replace dtrace_aggvarid_t by dtrace_aggid_t and DTRACEAGG_* with DT_AGG_*.

Compilation fixes:

- Pre-compiled BPF functions are now correctly loaded even if they have no relocations.

Testsuite changes:

- Comments, typos, and naming have been cleaned up.
- New tests have been added or XFAIL annotations revised for new features.

2.0.0-1.3 (Oct 2nd, 2020)

Third errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The freopen() action has been implemented for numeric values.
- The system() action has been implemented for numeric values.
- Two additional built-in variables are now available: id and ppid.
- Annotations in the DTrace disassembler have been improved for readability.

Bugfixes:

- The BEGIN and END probe semantics have been corrected to match the documented behaviour. BEGIN will always be the first probe executed, and END will always be the last probe executed. Corrections have also been made to the exit() action in terms of how it interacts with the BEGIN and END probe, and the trace data consumer.
- The behaviour of the default action vs non-data producing actions has been corrected.
- The implementation of the signed divide and modulo operations has been corrected in view of BPF not providing instructions for them.
- The code generated for post-decrement expressions has been corrected.

- A bug fix for a theoretical buffer overflow issue was merged from the 1.2 version of DTrace because the same code exists in this version.

Internal changes:

- Type casting has been optimized to only perform shift operations when needed. The implementation has also been improved to not require an extra register.

Compilation fixes:

- The procfs.d D library makes use of datatypes that are defined in the sched.d D library, but it was missing an explicit dependency on sched.d.
- The yylineno variable was declared in two places, causing a conflict with newer compilers. The primary declaration is now in dt_lex.c and dt_cc.c now has an extern declaration for the variable.
- Distributions place architecture specific include files in different locations. The build system will try different known locations, using the first one that seems valid.
- Some newer compilers do not accept 'const' for the r_debug_offsets and link_map_offsets arrays in the source code generated by mkoffsets.sh.
- Various changes were applied to the source code to resolve compiler warnings that were triggered during compilation.

Testsuite changes:

- A bug fix for the bogus-ioctl testsuite trigger executable was merged from the 1.2 version of DTrace because the same improper use of the open() library function occurs in this version.
- Some testsuite scripts were using local variables (this->n) in places where thread-local variables (self->n) were needed.
- Various testsuite cases have been updated to be more robust.

2.0.0-1.2 (Aug 6th, 2020)

Second errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- The profile provider has been implemented. Both profile-n and tick-n probes are supported, for probes with default fire rates and user-specified rates/intervals.
- The trace() action has been updated to provide more consistent output based on the datatype of its argument (signed vs unsigned, and width).
- The printf() action has been implemented for numeric values.
- The raise() action has been implemented.
- Clauses can now be specified with one or more probe specifications, and each probe specification can contain wildcards.
- Listing probes based on wildcard probe specifications has been implemented.
- It is now possible to specify the same probe for multiple clauses.
- Various built-in variables are available: arg0 through arg9 (for probes that provide arguments), curcpu curthread, epid, gid, pid, tid, uid, and timestamp.

- Expected test outcomes are continually being updated to reflect increasing functionality.
- The locked-memory limit is raised automatically if it is too small, since BPF has relied on a higher limit.

Bugfixes:

- Various memory management issues such as memory leak and unsafe memory access operations were fixed.
- A register allocation leak in predicate handling was fixed.
- The 'timestamp' built-in variable should yield the same value every time it is used within a specific clause. The cached value should not leak into the next clause execution.
- Interrupting dtrace using Ctrl-C or sending a signal could leave uprobes and/or kprobes behind. We now ensure that the interrupt handler is set up early enough to be able to provide proper cleanup.
- There were cases where integer values between INT32_MAX and UINT32_MAX were not processed correctly.
- The fallback support for /proc/kallsyms did not handle the lack of symbol sizes correctly, making it impossible to map an address to a kernel symbol.
- int8_t is now always signed, even on platforms where char is unsigned.

Internal changes:

- The optional predicate for a clause is now compiled as an inline conditional at the beginning of the clause execution.
- The machine state used during clause execution was allocated on the stack in previous releases. This worked fine when BPF code was executed using the kernel interpreter, but when the JIT BPF engine was used it would result in stack overruns. Now, the machine state is stores in a BPF map value to free up stack space.
- The creation of uprobes and kprobes as underlying probing mechanism has been deferred until tracing is actually about to start. This means that listing probes (-l) no longer results in uprobes and/or kprobes being created on the system without being used.
- A significant (but largely invisible) change has been implemented in the D compiler and runtime environment. The compilation of a clause will now merely generate a BPF function. When probe execution is being set up, all probes that are to be part of the tracing session are collected and for each probe a list of associated clauses is created. Finally, when tracing is to commence, a trampoline BPF program is assembled for each probe. Each of the trampoline BPF programs will include calls to the clauses associated with the probe. When the final program for each probe is linked, references to compiled clauses and precompiled BPF utility functions are resolved. At this point, the BPF program for each probe is loaded into the kernel and attached to its probe.

2.0.0-1.0 (Apr 24th, 2020)

First errata of the standalone userspace implementation.

This is a pre-release with limited functionality.

New features:

- BEGIN and END probes are functional. They are implemented using uprobes on trigger functions in libdtrace. The current implementation does not yet satisfy all documented semantics for these special probes. E.g. It is possible for a probe to be reported prior to the BEGIN probe.

- The exit(n) action has been implemented. It terminates probing and will result in dtrace reporting the given return value <n> as its return code.
- The flow-indent option -F has been implemented. Some of the heuristics present in DTrace 1.x are not available in this version - further code analysis is ongoing to determine whether they are necessary.

New options:

- -xdisasm=n: Specify which disassembler listings to generate when the -S option is supplied. The value for <n> is the sum of any of the following:
  - 1 = After compilation and assembly of a program.
  - 2 = After linking in precompiled BPF functions (dependencies).
  - 4 = After final relocation processing (final program).

Bugfixes:

- Various memory management issues such as memory leak and unsafe memory access operations were fixed.
- Using local variables in D clauses could cause the compiler to generate instruction sequences where a load instruction for a local variable occurred before a store took place to that variable. The BPF verifier rejects such sequences. We now ensure that we do not load from stack locations that were never initialized.
- The code generated for the post-increment operation resulted in the new value to be used as value of the expression. The value of the expression is now the old value.
- Various issues were resolved concerning register use in the compiler. We are now using proper register spilling techniques to free up a sufficient amount of general purpose registers.
- FBT return probes were not created correctly if an entry probe also existed for the same function. Both were getting attached to the same probe (either entry or return, depending on which was created first).

Internal changes:

- Various helper functions have been implemented in C code that is compiled to BPF code using the GCC BPF cross compiler. These are available as an ELF object for linking with the dynamically generated code that the DTrace compiler produces. When D code has been compiled into BPF code, we resolve any references to precompiled BPF functions against this ELF object and add any functions used (and their dependencies) to the compiled program.
- Various tests in the testsuite make use of the -xerrtags and -xdroptags options in dtrace to include specific error and drop tags in the error output. The testsuite engine did not enforce validation of these tags. Tests that specify a tag in their name err.<tag>.* and drp.<tag>.*) will now trigger validation that the specified tag is mentioned in the error output that dtrace produces.
- Dynamically created system level probes (kprobes and uprobes are now grouped under a tracepoint group named dt_<pid>_<prv>[_<prb>] where <pid> is the PID of the dtrace process, <prv> the name of the probe provider, and <prb> the probe name. The _<prb> optional suffix is used for FBT probes to separate entry probes from return probes.

2.0.0 (Mar 10th, 2020)

First release of the standalone userspace implementation.

This is a pre-release with limited functionality.

Working components:

- The entire D language, with the exception of aggregations. Valid D clauses are compiled into equivalent BPF programs.
- The vast majority of the DTrace core functionality has been implemented, providing a nearly complete compiler implementation (including predicates), provider API, probe management, and Linux tracing integration.
- Support for pre-compiled BPF function libraries has been added. This is used to implement various D language constructs (global and TLS variable access, string manipulation, ...) and D subroutines. This feature makes use of the BPF support in GCC and binutils as cross compilation tools. The BPF functions are compiled at DTrace build time, so there is no runtime dependency on the cross compilation tools.
- Support has been added for reporting BPF verifier output. When compiled D scripts are loaded as BPF program into the kernel, the BPF verifier performs a static code analysis to ensure safety of the program. When this analysis fails, output is generated and DTrace will report this output to the user.
- Function Boundary Tracing (FBT) probes with functions grouped by module (regardless of whether the module is compiled in or loadable) if the kernel provides this information in /proc/kallsyms (or /proc/kallmodsyms).
- Syscall entry and return probes (systrace provider), with support for typed probe arguments.
- Statically Defined Tracing (SDT) probes based on Linux tracepoints, with support for typed probe arguments.
- The trace data buffer management code has been reworked to work with the perf event ring buffers that are used by BPF to record tracing data.
- The DTrace testsuite has been updated to reflect what tests are expected to pass with the current state of development for this implementation of DTrace. Various tests were also improved to be more robust and to be more focused on what they're meant to be testing.

1.2.1 (Feb 12th, 2019)

Bugfixes:

- Fix a bug causing DTrace to fail to terminate if a process was grabbed using -c, then died while traced with ustack(), usym() or umod() or by the pid or usdt providers.

Compilation fixes:

- Compile on glibc 2.28, which moves makedev() out of <sys/types.h>.
- Improvements to the testsuite in the presence of recent versions of GNU Awk.

1.2.0 (Dec 13th, 2018)

New features:

- New action pcap(struct sk_buff *, proto) where proto is one of the PCAP_* constants from /usr/lib64/dtrace/*/pcap.d. If tshark is installed, this produces formatted packet traces; if it is not, raw memory dumps are produced, as with tracemem(). If freopen() is used to redirect DTrace output to a file, the raw packets are written there.
- Translator changes for the 4.19 kernel (also works with 4.20pre up to -6).

Bugfixes:

- Fix a variety of small memory leaks.
- Improvements to the testsuite.

Build-time:

- Improve the generation of signal.d: work better with newer glibc versions.

1.1.1 (Oct 25th, 2018)

Only testsuite changes.

1.1.0 (Aug 10th, 2018)

New features:

- Translator changes for the 4.15 kernel. (Kernels up to 4.18 work too.)

New options:

- -xctfpath: Specify the name of a CTF archive to use with the running kernel, for when it cannot be found in the usual place under $(-xmodpath)/kernel/vmlinux.ctfa.

Bugfixes:

- Fix a variety of small memory leaks and use-of-uninitialized-data bugs.
- Clean up compiler warnings.

Library interface changes:

- Add more DTRACE_PROBE definitions to sdt.h, for SystemTap compatibility.

1.0.4 (Aug 10th, 2018)

Bugfixes:

- No longer crash when attempting to trace ourselves.
- More fixes for crashes of both DTrace and the traced process when DTrace terminates at the wrong instant.

1.0.3 (Jul 24th, 2018)

Bugfixes:

- No longer crash or deadlock when -c/-p processes terminate at the wrong time.
- No longer deadlock when -c/-p processes create new threads.
- Stop the disassembler coredumping.

1.0.2 (May 10th, 2018)

Bugfixes:

- Mark an erroneously-failing test on ARM as passing.

1.0.1 (Apr 28th, 2018)

New features:

- Both USDT and pid providers are supported on ARM64.

Bugfixes:

- The -c option works on ARM64 now, as does the -x evaltime option.
- Improvements to the testsuite and testsuite runner.

1.0.0 (Mar 27th, 2018)

New architectures:

- ARM64 support.

New features:

- Compile-time array bounds checking. Dereferencing arrays beyond their declared bound is now a compile-time error. To dereference an array regardless, use casts, e.g. ((char *)curlwpsinfo->pr_name)[32].
- Translator support for kernels 4.12 -- 4.14.
- Added initial pid provider support for userspace tracing.
- Redesigned build system now allows change in translators in 4.14.y versions of kernels. When porting to the new kernel, it is no longer required to add the new kernel version to the list of define_for_kernel macros (unless a change is truly required).

Bugfixes:

- Addresses are normalized properly by mod(), so use of mod() in aggregates works better now.
- DTrace will no longer consider symbols in built-in modules or the core kernel to be in the wrong module: its idea of symbol addresses, sizes, and their mapping to names is better in general, particularly with respect to symbols that overlap, symbols whose names are duplicates, and weak symbols.

- An interface problem has been fixed that can cause DTrace consumers to dereference freed memory when victim processes grabbed via ustack(), umod(), usym() or dtrace -c or -p exec(). This requires changes to certain users of libdtrace, and relinking: see "Library interface changes" below.
- The ip provider's ipv6_tclass and ipv6_flow fields were wrong on little- endian machines.
- Fix rare assertion failures at exit.
- dtrace -S now disassembles all actions in statements containing more than one, rather than disassembling only the first.
- A new symbol at address zero introduced by the KPTI changes is eliminated from symbol resolution.
- Improvements to the testsuite and testsuite runner.

Library interface changes:

- The dtrace_proc_*() functions have changed the type they take to an opaque handle, struct dtrace_proc. There is a new function dtrace_proc_getpid() to get the PID from this opaque handle. dtrace_proc_grab() has been renamed to dtrace_proc_grab_pid(). See INCOMPATIBILITIES.

  The library soname has been bumped to libdtrace.so.1 correspondingly. All consumers must relink, but consumers not using the dtrace_proc_*() APIs need no code changes. All places where code changes are needed elicit a compile-time error, so it should be easy to see what needs changing.

Testsuite changes:

- Tagging added via a new @@tag in test files: testing with specific tags can be requested via TEST_TAGS='a !b' in the environment and --tag/--no-tag arguments to runtest.sh. The intersection of all tags is run, so in the example above, only tests tagged with 'a' and not tagged with 'b' would be run. You can specify tags that apply by default in the test/ tags.default file, and tags that apply on only one architecture in test/tags.$arch.default.

0.6.2 (Sep 12th, 2017)

Packaging changes:

- The DTrace headers in /usr/include/linux/dtrace were formerly provided by the dtrace-modules-shared-headers package. They are now pulled in from the kernel-uek-devel package in /usr/src/kernels at dtrace-utils package build time and shipped out into the dtrace-utils-devel package.
- libdtrace-ctf 0.7 or above is now required.
- CTF type information can now be provided in an archive located at /lib/modules/$(uname -r)/kernel/vmlinux.ctfa, cutting startup time when all types referenced are found in the kernel tree rather than in out-of-tree modules.

New features:

- A new llquantize() aggregation, providing log/linear results. Syntax: llquantize(expression, log base, lower exponent, upper exponent, step, [increment])
- The tracemem() action has gained a third argument, the number of bytes to trace: unlike the second argument, which must be a constant, the third argument can be an arbitrary D expression, which can be used to limit a larger second argument to a suitable value. This brings it into parity with tracemem() on Solaris.

- The lockstat provider is implemented.

Bugfixes:

- dtrace_sync() is drastically faster: setup and teardown of large numbers of probes without latency problems and watchdog timer firings is now much more practical.
- The error message given when D argument counts were wrong was itself often wrong or confusing.
- Fixed a segfault at shutdown time if grabbed processes die at precisely the wrong time.
- Structure and union members in the kernel with the same name as D keywords can now be referenced: mostly, this means you can get at members named 'self'.
- lquantize() no longer truncates its value to 32 bits.
- dtrace_update() now merges module address ranges better.
- Fix some places where sleeping inside RCU read critical sections or atomic context could happen (module provide, profile/tick providers, and more general probe and state setup/teardown code).
- After one release without it, the walltimestamp variable reports useful values again.
- One place where failure to allocate memory (for ECBs) could crash the kernel has been fixed.

Build-time:

- A new 'make ctf' target in the kernel tree, for generating the vmlinux.ctfa archive mentioned above: it is no longer linked into in-tree modules. The old CONFIG_DT_DISABLE_CTF option is thus removed, as is the ctf.ko module.
- You can no longer build CTF as root.
- Kernel CTF type generation now understands the DWARF generated by GCC 6; one more problem with representation of bitfields is fixed; and one sort of painfully manually-maintained type-related blocklist is now automated away.
- Some unused variable warnings in the io provider are squashed.

0.6.1 (Aug 7th, 2017)

Licensing changes:

- Userspace is now licensed under the Universal Permissive License (UPL) v1.0. The kernel module is now GPLv2, and is shipped in the same package as other in-tree kernel modules.

  /usr/lib64/dtrace/load_dtrace_modules no longer tries to yum install anything (but will still modprobe modules listed in /etc/dtrace-modules).

New features:

- A new link_ntop() subroutine is provided, which is like inet_ntop() except it returns a human-readable string describing the link-layer address. Ethernet and InfiniBand are currently supported.
- A default set of modules is now provided in /etc/dtrace-modules. The file was supported since 0.4.5, but no /etc/dtrace-modules was shipped by default.
- The TCP and UDP providers are implemented, with associated translators.

- The IO provider has been completely rewritten and is dramatically improved, with support for most local filesystems and explicit support for XFS and NFS.

Bugfixes:

- Do not require sdt.ko to be loaded before allowing the use of the 'cpu' variable.
- Passing an object file through dtrace -G no longer corrupts it on SPARC64, echoing a similar bugfix made to x86 long ago.
- Improve tracking of process state on SPARC64 a bit.
- Bitfields in kernel types are now better-supported, though some bitfields still do not work, notably those crossing machine word boundaries.
- dtrace_print_lquantize() no longer normalizes the name of the lowest bucket, only the bucket contents (as intended).
- jstack() was fetching data from the wrong offset, leading to garbage output at the start of the stack dump.
- Reading of unaligned data from high addresses in traced processes was failing. In practice only SPARC has any data at addresses high enough to cause this, and most of the accesses done are aligned: but the machinery that adjusts to changes in glibc's internal data structures was broken, leading to failures to look up symbols after certain glibc upgrades.
- dtrace -C and -G now search for cpp and ld along the PATH rather than defaulting to /usr/bin/cpp and /usr/bin/ld, fixing failures with compilers in non-default locations, like the Software Collections devtoolset packages.
- dtrace-utils-devel now requires elfutils-libelf-devel. (This dependency was always present in practice but was mistakenly omitted until now.)

0.6.0 (Apr 3rd, 2017)

Kernel release:

4.1.12-97.el6uek

New features:

- Function boundary tracing (FBT) is supported for entry probes to most functions in the core kernel. The current implementation does not support retrieval of function arguments or return probes (except for some limited support on x86_64). These features are currently under development.
- The ip provider is implemented, supporting ip:::send, ip:::receive, ip:::drop-in and ip:::drop-out, with parameters compatible with other implementations and appropriate translators. IPv4 and IPv6 are both supported.
- Userspace tracepoints (USDT) now work on SPARC for both 64- and 32-bit processes.
- The types and translators used by SDT probes are now acquired from the DTRACE_PROBE macros in the kernel source. New probe argument types and translations are picked up automatically without needing to change the module at all. perf-event probe argument types are acquired in the same way.
- The DTRACE_PROBEn() macros used for SDT probes have been supplanted by a new DTRACE_PROBE() macro which works exactly the same except that you don't need to count the arguments any more and misuses (args with no types, etc) are diagnosed even when CONFIG_DTRACE is disabled. Much the same has been done for USDT, except

that the old numbered USDT macros remain available for code that must be compiled with compilers that don't support __VA_ARGS__ comma elision (such as GCC when in c89/strict-ANSI mode). This involves a new, installed, internal header, /usr/include/sys/sdt_internal.h.

- is-enabled probes are now supported for SDT: these are expressions which always return false unless the specified probe is enabled, generally used directly in if statements, and can be used to suppress collection of expensive data only needed for probes until the probes that use them are enabled:

  if (DTRACE_PROBE_ENABLED(probename)) /* expensive stuff */

  Per-provider wrappers for DTRACE_PROBE_ENABLED() can be used, as with DTRACE_PROBE() itself.
- dtrace consumers (including dtrace(1)) can now grab themselves via -p, though symbol resolution is degraded when they do so because they cannot stop themselves. (Previously, such grabs were suppressed but the code to do so was buggy and caused dtrace not to terminate if dtrace was asked to do a self-grab in conjunction with a -c of some other process, even once the other process had terminated.)
- D translators for the ip provider are now available.

Bugfixes:

- Due to a logic error in preemption handling, it was possible that code was being executed under the assumption that preemption was disabled when in fact it was not.
- Probe processing (probe context) is not re-entrant, yet probes firing as a result of processing another probe would cause re-entry into the processing core, with often horrible effects. The processing core has been modified to block any re-entry attempt except for ERROR probe processing. That is a deliberate (and acceptable) exception in the DTrace design.
- The fast path implementation for obtaining the value of the D 'caller' variable for sparc64 has been corrected.
- The implementation of the D 'stack' action has been made more robust, making sure that memory access faults are not fatal.
- The implementation of the D 'ustack' action has been reworked completely to improve stability and accuracy.
- The number of stack frames to skip has been adjusted to changes in the implementation of various providers, ensuring that DTrace related frames are skipped as they should. This makes the D 'stack' action and the D 'caller' variable values correct.
- The implementation of the D 'stackdepth' variable could cause memory writes beyond the end of the DTrace probe scratch buffer.
- Numerous dtrace -c/-p and USDT fixes on SPARC systems, with symptoms varying from a hanging dtrace and child process to a dtrace that runs out of file descriptors.
- Fix memory and fd leaks when a process monitored with -c or -p exec()s frequently.

Changes to user-visible internals:

- DTrace now uses /proc/$pid/map_files, where available.
- The implementation of the D 'ustack' action has been moved into the kernel proper. This change was motivated by the need to access page table structures directly using a lock-free mechanism.

- Probe processing will be bypassed when the system is entering panic mode, This ensures that DTrace will not cause panic related output to be disrupted.
- dtrace -S now dumps the offset of DIF as well as the instruction counter, allowing you to more easily match up DIF disassembly with errors from the kernel.
- Predicate DIF is disassembled in dtrace -S output.

0.5.4 (Nov 8th, 2016)

Workarounds:

- Work around a bug in elfutils causing massive corruption of object files when dtrace -G is used.

0.5.3 (May 25th, 2016)

Kernel release:

4.1.12-43.el6uek

New features:

- It is now possible to have perf-events presented as DTrace SDT probes. This feature is turned on by default in the kernel. The probes will appear with the same names as the perf-events and are grouped under the new 'perf' SDT provider.

  In its current implementation, the perf-events DTrace probes do not offer argument type information as is seen with standard DTrace SDT probes.

Bugfixes:

- On sparc64, it was possible to crash the system by unloading and reloading the sdt DTrace multi-provider module due to the handling of memory that is set aside for SDT probe trampolines. This bug has been fixed.

Crash fixes:

- When dtrace(1) exited at the same instant as a process it had grabbed (e.g. for ustack()) terminated, it could deadlock or crash with an assertion failure or a segmentation fault.

0.5.2 (Feb 3rd, 2016)

Kernel release:

4.1.12-33.el6uek

Crash fixes:

- Programs containing USDT probes can crash at startup or dlopen() time if shared libraries are mapped into the top half of the address space. This never happens on x86-64 but is common on SPARC64: dtrace -G should be rerun on programs on such platforms that contain USDT probes, to link in the fixed ELF constructor.

Performance improvements:

- dtrace(1) no longer wastes time in a CPU-heavy busywaiting loop: previously, the sleeping code was mistakenly picking a time in the past to sleep to roughly half the time

Bugfixes:

- dtrace -c and -p now work on SPARC64.
- When both entry and return probes were enabled for a system call, upon disabling the first, the function pointer in the system call table got reset to its default value even though the 2nd probe might still be active. This could cause race conditions in the state of the system call probing.
- Access to the SPARC64 R_L7 register was consistently failing due to an off-by-one bug.
- It was possible to read past the beginning of the stack for a user process. The mechanism for reading stack slots also got updated to increase efficiency, consistency and reliability across architectures.
- While reading the stack of a userspace process, the stack bias was not being applied for architectures that need it, causing an abundance of essentially invalid values to pollute the result.

0.5.1 (Nov 17th, 2015)

Kernel release:

4.1.12-24.el6uek

Bugfixes:

- When copyout() or copyoutstr() is used in a D script, safety checks are now enforced to protect against unprivileged memory accesses.
- The DTrace modules package no longer prevents automated kernel RPM removal when the install limit is reached.
- It is now possible to access the envp and argv arrays in the psinfo for a task using copyin(). This is the convention across DTrace-capable systems.

Performance improvements:

- dtrace(1) and libdtrace(1) startup speed is improved, both by avoiding a filesystem walk by using the modules.order file to locate available kernel modules, and by avoiding loading all kernel modules to resolve possible types when unqualified probe names that cannot possibly be C identifiers are seen (like 'tick-1sec'). When the disk cache is cold these changes speed up startup by on the order of 2x.

Changes to user-visible internals:

- The DTRACE_DEBUG debugging option could intermingle debugging output in limited ways when multiple threads were emitting debugging at once.

0.5.0 (Aug 10th, 2015)

Kernel release:

4.1.4-4.el6uek

New architectures:

- Linux on SPARC64 is supported with the following providers: dtrace, profile, syscall, and SDT. Userspace tracing doesn't work yet.
- The uid / gid handling has been updated to accommodate namespace support at the kernel level (kuid and kgid). All uid / gid values reported by D subroutines (or obtained from structures) are evaluated based on the initial user namespace.

New options:

- -xuseruid: On non-systemd systems (such as OL6), specify the user ID of the first non-system user. (The default will normally be appropriate.) Processes with uids below this, and which appear to truly be daemons, are only ptrace()d if explicitly specified via dtrace -p or -c.
- -xsysslice: On systemd systems (such as OL7), specify the name of the system slice. (The default will almost always be appropriate.) Processes in this slice or the root slice are considered crucial system daemons and only ptrace()d if explicitly specified, as above.

  The systemd/non-systemd determination is made dynamically, so you can switch init systems freely and everything should still work.

Changes to user-visible internals:

- Translator support for the UEK4 4.1 kernel.
- Accessing kernel memory under NOFAULT protection now implies NOPF (no page fault) as well. Previously, NOPF was an option that could be set in addition to NOFAULT.
- Debugging output has been improved (to be enabled at compile time).
- The datatype formerly known as sdt_instr_t has been renamed asm_instr_t. The rationale behind this change is that it will be used in code beyond the SDT provider and therefore a more generic name is appropriate.

Bugfixes:

- Symbol resolution in non-ptraceable processes is improved.
- dtrace -p with an invalid PID now produces a sensible error message.

0.4.6 (Jun 30th, 2015)

New dependencies:

- dtrace-utils-devel now always pulls in the corresponding version of dtrace-utils, rather than being satisfied with whatever version is installed. [Introduced in DTrace 0.3.0.]
- The DTrace kernel header package was renamed dtrace-modules-shared-headers in dtrace-modules 0.4.4; dtrace-utils now follows this renaming.

New options:

- dtrace -vV now reports information on the released version of dtrace, as well as the internal version-control ID of dtrace(1) and libdtrace(1). (The last two should always be the same unless the installation is faulty.)

Bugfixes:

- Processes that receive SIGTRAP in normal operation now work even when being dtraced for a ustack(), etc. Previously, the SIGTRAP would be ignored. [Introduced -- intentionally -- in DTrace 0.4.5, though this case would have misbehaved in other ways since 0.4.0.]
- DTrace no longer loses track of processes that exec() while DTrace is looking at their dynamic linker state.
- DTrace no longer leaves breakpoints lying around in fork()ed processes, but properly detaches from them and removes its breakpoints.
- DTrace no longer considers that it knows the state of the symbol table of processes it has since stopped monitoring.
- DTrace no longer crashes multithreaded processes that do dlopen() / dlclose(). [All introduced in DTrace 0.4.0.]

Library interface changes:

- Including <dtrace.h> used to fail because of the absence of a Solaris-specific header we did not ship. That header is no longer called for.

Changes to user-visible internals:

- DTrace now loads D libraries (with translators, etc) from directories with a name that depends upon the running kernel, so can support multiple kernels with the same userspace package.

Known problems:

- Multithreaded processes under u{stack,sym,addr,mod}() which do dlopen() in threads other than the first may not have accurate symbol resolution for symbols introduced by such dlopen()s.

0.4.5 (Jun 17th, 2015)

Kernel release:

3.8.13-87.el6uek

New features:

- Provider modules are now automatically loaded from /etc/dtrace-modules when DTrace initializes for the first time, at the same time as dtrace.ko. (Providers that do not come from the dtrace-modules package are not automatically 'yum install'ed.)
- It is now possible to use USDT probes in 32-bit applications on 64-bit hosts.

Bugfixes:

- Fixed a (minor) memory leak problem with the help tracing facility in DTrace. Upon loading the dtrace.ko module, a buffer (by default 64K) was being allocated, and it was never released.
- Stack backtraces are more accurate as a result of various fixes to adjust the number of frames to skip for specific probes.
- Datatypes have been adjusted to be more carefully specified after a detailed audit in preparation for supporting architectures other than x86_64.
- The stack depth was being determined by requesting a backtrace to be written into a temporary buffer that was being allocated (vmalloc), which posed significant problems when probes were executing in a context that does not support memory allocations. The buffer is now obtained from the scratch area of memory that DTrace provides for probe processing.
- It was possible to cause a system crash by passing an invalid pointer to d_path(). Due to its implementation, it is not possible to depend on safe memory accesses to avoid this. Instead, the pointer passed as argument must be validated prior to calling d_path() in the kernel.
- Fix intermittent dtrace crash on failure of initial grabs or creations of processes (via dtrace -p or -c, or via ustack(), usym(), uaddr(), or umod()).
- Fix dtrace -S DIF subr names. [Introduced in DTrace 0.4.0.]
- DTrace can now reliably monitor processes that undergo exec() and processes that are hit by stopping signals and later resumed. (Previously, it would sometimes lose track of the victim process, sometimes kill it with a SIGTRAP, and sometimes crash itself.) Numerous other subtle bugs and deadlocks in this area have been fixed as a side-effect.
- Fix a sign-extension bug in breakpoint-instruction poking which could cause the monitored process to crash. [Introduced in DTrace 0.4.0.]
- DTrace is now more resilient against changes to glibc: many places where non-ABI-guaranteed internals of glibc are relied upon now dynamically search for the correct field offsets, so are resilient against new fields appearing in glibc's internal structures, and against fields changing size.

Library interface changes:

- The dtrace_proc_*() functions have changed the type they take (it is now a small structure passed by value). See INCOMPATIBILITIES.

  There are still no binary-compatibility guarantees for libdtrace consumers.

Changes to user-visible internals:

- The code has been restructured to facilitate supporting architectures other than x86_64 in future releases.
- The d_path() D subroutine requires its argument to be a pointer to a path struct that corresponds to a file that is known to the current task (see bugfixes below).

Known problems:

- Processes under u{stack,sym,addr,mod}() cannot receive SIGTRAP.
- Multithreaded processes under u{stack,sym,addr,mod}() which do dlopen() in threads other than the first may crash.

0.4.4 (Mar 12th, 2015)

Kernel release:

3.8.13-69.el6uek

New options:

- -xcppargs: Additional arguments to pass to the preprocessor when run over D scripts by DTrace.

Bugfixes:

- The DOF ELF object generated by dtrace -G no longer requires an executable stack.
- Renamed the dtrace-modules-headers package to dtrace-modules-shared-headers to work around problems in Yum where a symbol has had both versioned and unversioned provides over time.

0.4.3 (May 1st, 2014)

Kernel release:

3.8.13-33.el6uek

New features:

- Timer based profile-* probes (profile provider). These probes use the omni-present cyclic support in the UEK3 kernel (3.8.13-32 and later) to fire probes at a specific frequency/ interval on every active CPU.

Bugfixes:

- Several memory-allocation, underrun and overrun bugs in process handling were fixed. With sufficient ingenuity these may be exploitable by local users who can craft and run unusual ELF executables and arrange for dtrace to attach to them.
- The pid and ppid variables were being reported based on the kernel task PID, which is not the same as the userspace concept of a PID (for threaded applications). We now pass (more correctly) the thread group id (tgid).
- Since userspace doesn't know about thread kernel level) pids, we are now also passing the tgid in the result of ustack, usym, etc... We pass the tgid in the first slot, and the (kernel) pid in the second slot.
- Major reworking of the dtrace_getufpstack() implementation to handle locking, stack detection, and potential page fault while accessing the stack of a task.

Known problems:

- As a result of earlier code changes to ensure that all memory allocation requests are checked for failures, the test for auto-resize behaviour of the principal buffer allocations results in the dtrace utility aborting processing rather than continuing operation with the reduced buffer size. This is overall a non-harmful regression that will be addressed in a future release.

0.4.2 (Dec 20th, 2013)

Kernel release:

3.8.13-22.el6uek

New features:

- SDT probe points in kernel modules are now supported.
- The 'vtimestamp' D variable has been implemented.

Bugfixes:

- Kill -9'ing a running dtrace will no longer leave breakpoints outstanding in processes with no controlling terminal that were grabbed as a side effect of ustack(), usym(), uaddr() or umod); as a side effect, symbol resolution will be less accurate for such processes. Grabbing a process with no controlling terminal via dtrace -p restores full symbol resolution accuracy for these processes, at the cost of dropping breakpoints in them again. Processes with a controlling terminal are still treated as in prior releases.
- ustack(), usym(), uaddr() and umod() of multithreaded processes no longer crashes the system, oopses the kernel, hangs the process being probed, crashes dtrace(1) itself, or runs dtrace or the system as a whole out of filehandles.
- Interrupting dtrace with a SIGINT while monitored processes are dying simultaneously now consistently stops it rather than hanging forever.
- dtrace's symbol-resolution paths are armoured against various problems which could occur when processes died while lookups were underway.
- pid and ppid are now correctly derived for multithreaded processes, pointing to the POSIX pid and parent respectively rather than the thread and thread group leader.
- Resolving kernel symbols located at the start of modules will no longer cause dtrace userspace to dereference uninitialized memory as a pointer.

0.4.1 (Nov 6th, 2013)

Kernel release:

3.8.13-16.2.1.el6uek

New features:

- DTrace now automatically modprobes for dtrace.ko if needed, and yum installs it if it is not found on the system. Provider modules are not automatically modprobed, but running (for example) dtrace -l is now a good way to make sure that the modules are present on the system so you can modprobe them.
- It is no longer permissible to have non-unique provider names within the context of a userspace process. I.e. it is not permissible for the main executable and a loaded shared library, or two loaded shared libraries, to list the same provider name in their DOF sections.
- A new cyclic implementation has been included in the UEK3 kernel, replacing the more error prone former version. The modules code has been updated to use that new implementation.

- New development tools showUSDT (for dumping of DOF sections) and ctf_module_dump (for dumping of CTF in kernel modules). (The former tool is an example only, and is installed in the documentation directory.)

Bugfixes:

- A lexer bug was fixed which caused spurious errors if D scripts contained a pragma or comment at intervals of 8192 characters, and prevented the use of scripts >16KiB entirely. [Introduced in the original Linux port]
- A variety of memory leaks and uninitialized memory reads are fixed.
- A bug whereby breakpoints could be left outstanding in a process if dtrace was interrupted with an ordinary SIGINT at just the wrong instant is fixed. [Introduced in DTrace 0.4.0.]
- The visibility of .SUNW_dof sections was wrong. [Introduced in DTrace 0.4.0.]
- Fix devinfo_t's dev_statname and dev_pathname for cases where the device does not have partitions. [Introduced in DTrace 0.4.0.]
- drti.o, which contributes a constructor to programs and shared libraries that contain DOF, now has lower overhead when DTrace is not running, emits its errors to stdout, not stderr, and opens its files with O_CLOEXEC. [Introduced in DTrace 0.4.0.]
- Lock ordering problems that were inherited from the original code are fixed.
- Userspace stack memory accesses are now performed in a safe manner.
- A race condition between speculative tracing buffer cleaning and destroying consumer state has been resolved.
- A memory leak related to consumer state has been fixed.
- A provider reference counter calculation problem was resolved.
- The 'errno' D variable now holds the correct value during syscall:::return probe action execution, i.e. 0 if the syscall completed without an error, and a valid error code if the syscall failed.

0.4.0 (Sep 20th, 2013)

Kernel release:

3.8.13-16.el6uek

New features:

- Support for meta-providers, such as fasttrap (used for userspace tracing). A meta-provider implements a framework to instantiate providers dynamically (on demand).
- Userspace Statically Defined Tracing (USDT) provides support for SDT-alike probes in userspace executable and libraries. Two types of probes are available: regular SDT-alike probes, and is-enabled probes. A new header file (sys/sdt.h) is installed in support of USDT.
- The fasttrap provider has been implemented, although it is currently only supporting USDT probes.
- Symbol lookup now works: stack() and ustack() now print symbols, as does &. ustack() can look up symbols in libraries loaded with dlopen() and dlmopen() as well as via DT_NEEDED. Symbol lookup of global symbols in userspace processes respects symbol interposition and all other symbol-ordering trickery. Some of the machinery involved in this only works with programs running against specific versions of the GNU C Library. (It will

always work with the version of glibc shipped with OEL, and falls back to a simpler approach which does not support symbol interposition or dlmopen() if it appears an incompatible glibc is in use).

This depends on new machinery in the kernel, notably waitfd()s and PTRACE_GETMAPFD, so will not work with earlier DTrace kernels.

- -xevaltime={preinit, postinit, main} now work, with a few caveats:
  - postinit (the default) is equivalent to main.
  - On statically linked binaries, preinit is equivalent to exec, and may not skip ld.so initialization (which can happen after main() on such binaries).
  - On stripped, statically linked binaries, postinit and main are equivalent to preinit, because we cannot look up the 'main' symbol when there is no symbol table.
- DTrace options can now be set from environment variables named DTRACE_OPT_*. Example:

  export DTRACE_OPT_INCDIR=/usr/lib64/dtrace:/usr/include/sys

Changes to user-visible internals:

- The ELF section in which CTF data is stored has changed from .dtrace_ctf to .ctf.
- The storage representation of internal kernel symbols is improved, saving DTrace memory usage at startup by a megabyte or so.
- The libdtrace public API header now names its arguments. A few other libdtrace functions have changed prototype: see INCOMPATIBILITIES.
- Two undocumented libproc environment variables from Solaris are removed, because the code whose behaviour they adjusted no longer exists: _LIBPROC_INCORE_ELF and _LIBPROC_NO_QSORT.
- New low-overhead debugging machinery. Exporting DTRACE_DEBUG=signal in the environment will emit debugging output only when DTrace is hit by a SIGUSR1, avoiding all printf() locking overhead until then. This uses a ring buffer to stop debugging output, by default 100Mb in size, changeable via the DTRACE_DEBUG_BUF_SIZE variable (which takes a size in megabytes).
- What was previously defined as a meta-provider (see 0.2.0 below) is in fact better defined as a multi-provider, i.e. a provider framework that handles multiple providers that essentially share (the majority of) a single implementation, such as SDT where probes are grouped together into providers even though they are all provided by the same provider (sdt).
- The DTrace header files in the kernel proper, the kernel modules, and the userspace utility have been restructured to avoid duplication and to offer a more consistent and clean design. This also offers better support for custom consumers or other DTrace-related utilities.
- The systrace provider has been updated to account for changes in the Linux kernel (between 2.6.39 and 3.8.13).

Bugfixes:

- It is now possible to get the correct value for the ERR registers.
- The ustack() and jstack() actions were not passing the PID correctly as the first element in the result array.
- The ustack() action implementation has been replaced.

- Several obscure locking problems have been resolved.
- Correct handling of arg5 through arg9.
- The -h and -G command-line options work.
- Negative values passed to DTrace options that take only positive integers are correctly diagnosed as errors again.

Known problems:

- Presently, kill -9'ing a running dtrace can leave breakpoints outstanding in other processes, which may sooner or later kill them. This will be fixed in due course (by avoiding the use of breakpoints in more cases).

0.3.0 (Sep 14th, 2012)

Kernel release:

2.6.39-201.0.1.el6uek

New features:

- CTF support. This exposes all kernel types declared at the global scope to DTrace scripts (even those private to single files). All global kernel variables not declared static are also available to the ` operator as external variables.

  The module for kernel-wide symbols is known as vmlinux, but genunix can still be used as a name for it to aid script portability.

  Kernel modules from a compatible kernel must be visible to DTrace for this feature to work, as must the kernel-provided file /proc/kallmodsyms. DTrace will work with no kernel modules, with no visible /proc, or with a kernel whose modules do not contain type information, but no kernel types or variables will be available. (See -xprocfspath and -xmodpath below.)
- The curcpu builtin variable has been implemented as a DIF builtin variable on Linux, providing a pointer to the CPU info structure for the CPU that is currently active.
- A new DIF subroutine has been implemented: d_path(). This subroutine takes a pointer to a path structure as argument, and returns a string representing the full pathname for that path.
- The raise() action has been implemented. This action allows a D script to raise a signal in the current task.
- The io provider probes has been implemented. It provides the following SDT probes: start, wait-start, wait-done, and done.
- The proc provider has been implemented. It provides the following SDT probes: create, exec, exec-failure, exit, lwp-create, lwp-exit, lwp-start, signal-clear, signal-discard, signal-handle, signal-send, start.
- The sched provider has been implemented. It provides the following SDT probes: change-pri, dequeue, enqueue, off-cpu, on-cpu, preempt, remain-cpu, sleep, surrender, tick, wakeup.
- Argument mappings have been provided for io, proc, and sched provider probes. This information is used by userspace consumers.

New dependencies:

- DTrace now depends on libdtrace-ctf, a modified, GPLed port of the Solaris libctf type-storage library. Despite its name it cannot read Solaris CTF files: the file formats are incompatible.

New options:

- -xprocfspath: if set, specifies the path to /proc. May be useful in chroots, though glibc and other things may break if /proc is moved to another location.
- -xmodpath: if set, specifies the path to kernel modules, rather than looking in /lib/modules/$(uname -r).

Options removed:

- The undocumented -xlinkmode=primary option is removed: it never worked in DTrace for Linux in any case.

Bugfixes:

- The -c and -p command-line options work.
- Lexer bugs causing aggressive and unnecessary reading of modules are fixed. As a result, when used with typo-free scripts, DTrace now starts much faster than ever it did on Solaris (often taking half the time or less). You may find a few error messages have changed error text (though not error tag) as a result of this bugfix and the following one.
- The SDT provider now describes its argument types to DTrace userspace.
- The types of many DTrace actions and variables are fixed to correspond to the Linux reality.
- The set of available error numbers in errno.d is more complete.
- DTrace libraries are installed to /usr/lib64 now, not /usr/lib.
- Users of dtrace -C can now include <sys/dtrace.h> without incident.
- Various DIF builtin variables that were providing a hardcoded value based on the init task whenever a probe was executing in interrupt context are now providing the actual value from the current task. In Linux, there is always a valid task structure available as 'current'.
- The numbering of the registers for the x86_64 architecture has been updated to match the order of registers pushed onto the stack.
- It is now possible to get the correct value for the DS, ES, FS, and GS registers.
- SDT probes are now correctly cleaned up when the SDT meta-provider module is unloaded from the system.
- The rw_read_held() DIF subroutine will now verify whether it can safely access the passed in argument based on the correct argument datatype.

Changes to user-visible internals:

- A new file /proc/kallmodsyms now exists, like /proc/kallsyms but giving object sizes and listing the module each kernel object would be part of were it built as a module, even if it is currently built in.
- A new module dtrace_ctf.ko is pulled in whenever dtrace.ko is loaded. It is a container for type information.
- The undocumented -B buffer-inspection command-line option no longer crashes DTrace.

- The invalid operand trap logic previously provided to support SDT probes has been made more generic to support any probes that wish to utilize this facility.
- The DTrace core module now depends on the core kernel CTF data-module, to ensure that when DTrace modules are loaded on the system, CTF data for the kernel will be available also.

0.2.5 (Mar 19th, 2012)

Userspace release only.

New features:

- libdtrace is now a shared library, just as on OpenSolaris, with a very similar API. No API or ABI compatibility guarantees are made regarding this library, at present.

0.2.4 (Feb 15th, 2012)

Kernel release:

2.6.39-101.0.1.el6uek

Bugfixes:

- Provider modules now use a reference counter to determine whether any of their probes are currently enabled. Whenever the reference counter has a value greater than zero, the provider module is referenced to ensure that it cannot be unloaded. Once the counter drops down to zero, the reference on the module is released. This prevents providers from being unloaded while some of their probes are still in use (which would typically lead to a kernel panic).

0.2.3 (Feb 10th, 2012)

Kernel release:

2.6.39-101.0.1.el6uek

Internal changes:

- The DTrace core has been updated to support 28 DTrace option settings, to account for the 'quietresize' option that was added to the userspace dtrace consumer utility.

Bugfixes:

- Various assertions in the DTrace core implementation incorrectly used mutex_is_locked() where the test was meant to determine whether the current task holds the mutex. This has been corrected.

0.2.0 (Jan 25th, 2012)

Kernel release:

2.6.39-101.0.1.el6uek

This release brings DTrace for Linux to the 2.6.39 kernel, as an upgrade from the previous release based on 2.6.32.

New features:

- The DTrace core and provider API now support meta-providers, a framework that provides multiple providers using a common implementation.
- The Statically Defined Tracing (SDT) provider is implemented, providing in-kernel static probes. Some of the proc provider is implemented using this facility.

Bugfixes:

- Syscall tracing of stub-based syscalls (such as fork, clone, exit, and sigreturn) now works.
- Invalid memory accesses inside D scripts no longer cause oopses or panics.
- Memory exhaustion inside D scripts no longer emits spurious oopses.
- Several crash fixes.
- Fixes to arithmetic inside aggregations, fixing quantize().
- Improvements to the installed headers.

Internal changes:

- The minimal cyclic implementation has been removed from the DTrace modules because it is now provided by an equivalent GPL implementation in the core kernel.
- CPU core information is now maintained at the core kernel level.
- Kernel and module code can now perform safe memory accesses by setting a flag in the CPU core information structure. If a memory access results in a Page Fault or General Protection Fault, the failure will be noted as a CPU fault, and execution will continue rather than causing a kernel panic.
- Functionality that depends on walking the stack (determining stack depth, or collecting a backtrace) is now provided by a GPL implementation in the core kernel.
- In the interest of consistency, a pseudo kernel module structure is created at the core kernel level, representing the main kernel image. This module structure makes it possible to represent all kernel-level objects equally. This structure provides a list of SDT probe locations in the core kernel.

0.1.0 (Oct 20th, 2011)

First release.

Working components:

- the entire D language, with the exception of parts that depend on symbol lookup or CTF
- The vast majority of the DTrace core functionality has been implemented, providing a nearly complete DIF/DOF implementation (including predicates, aggregates, and speculative tracing support), provider API, ioctl interface for userspace consumers, and direct probe invocation.
- BEGIN, END, and ERROR probes (dtrace provider).
- Syscall entry and return probes (systrace provider), with the exception of the clone probe, which is disabled
- the profile provider (timer-based tick-* probes, no arbitrary-precision profile timers).

- kernel stack tracebacks, but ustack() prints addresses only, no symbols

Major components not yet present include all the other providers, including sdt usdt.