Dealing with Application Crashes

ACCU -- March 2018

About Me

MySQL book not required

- Co-Founder of Backtrace, a software error monitoring and analysis product.
- C & C++ Programmer for 10+ years
- Even split between OS, Driver, and Embedded code and Userspace (web servers, data processing, etc)

@nullisnt0
amathew@backtrace.io

Crash?

Let's just trust Wikipedia

*In computing, a **crash** (or **system crash**) occurs when a computer program, such as a software application or an operating system, stops functioning properly and exits. The program responsible may appear to hang until a crash reporting service reports the crash and any details relating to it. If the program is a critical part of the operating system, the entire system may crash or hang, often resulting in a kernel panic or fatal system error.*

- *"https://en.wikipedia.org/wiki/Crash_(computing)"*

A crash is a condition in which a computer program stops performing as expected and also stops responding to other parts of the system. Often the crashed program will appear to freeze. Other terms for to crash are to hang, to lock up and to bomb.

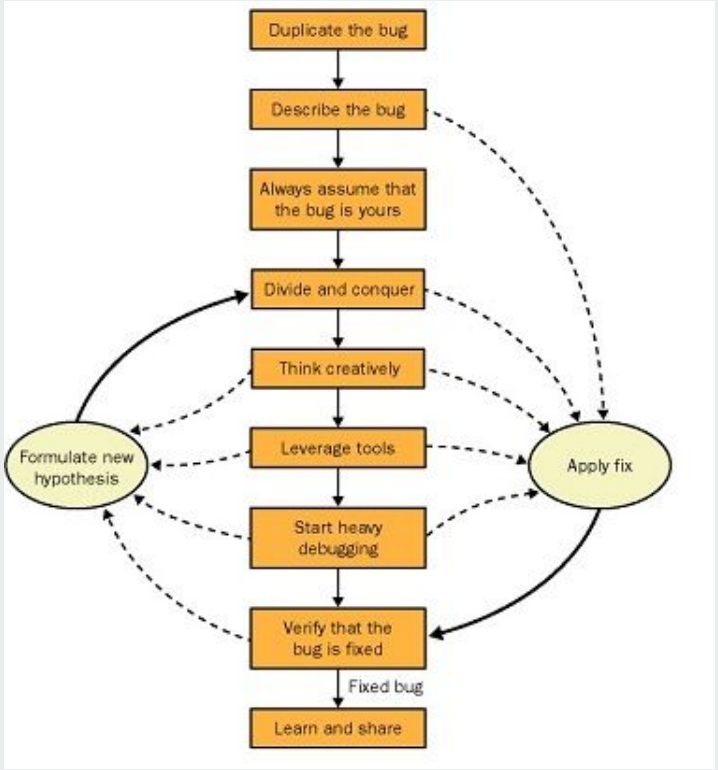
- *"http://www.linfo.org/crash.html'*

What if a hung program doesn't exit?

Crash

Methods + Techniques are not a replacement for static-analysis, model-checking during the development process

The best way to resolve a crash… is to not have a crash at all



aichengxu.com

“Dealing” with Application Crashes?

How do some popular applications:

- **Detect** a crash has happened
- **Capture** relevant data to investigate
- **Diagnose/Understand** the root cause

Applications

- Mozilla Firefox
- Apache Traffic Server
- ~~Applications on Embedded RTOS~~

Disclaimer: I don't work on any of these applications full-time and review maybe based on stale data.

Applications

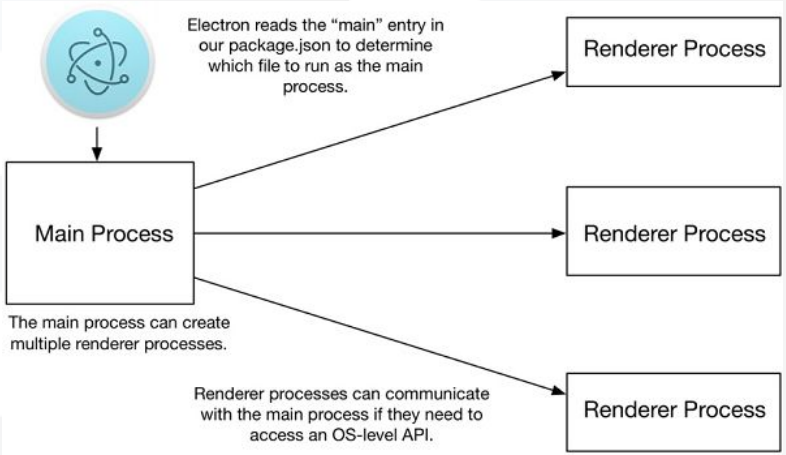
- **Mozilla Firefox**
  - Multi-threaded, Multi-process
    - In some processes, evented
  - Cross-platform (Mac, Windows, Linux, etc)
  - C++, Javascript, Rust, and many more languages
  - Web Browser: Millions of deployments around the world
- **Apache Traffic Server**

Applications

- **Mozilla Firefox**
- **Apache Traffic Server (trafficserver.apache.org)**
  - Multi-process, Multi-threaded
  - Forward & Reverse HTTPS proxy
  - Linux, FreeBSD, OmniOS, Mac OS
  - C++11, PERL, Lua
  - Tens of thousands deployments

Considerations

- **Enough storage space for the necessary data?**
  - What about capturing data **across** processes?
- **Data Privacy/Security**
  - Likelihood and impact of your dump being captured by unwanted eyes
- **Events to capture**
  - Hang? SIGSEGV? Asserts? Exceptions?
- **Available methods for notification**



http://freecontent.manning.com/electron-in-action-article-1/

Detection

Mozilla Firefox

Detect

What: Hangs (plugin, renderer), fatal instructions, invalid app state (see below)

How: Hang monitor, timers, signals, API for explicit kills

```
504  MessageLoop* messageLoop = MessageLoop::current();
505  if (!messageLoop) {
506    sMessageLoop = new MessageLoopForUI(MessageLoop::TYPE_MOZILLA_PARENT);
507    sMessageLoop->set_thread_name("Gecko");
508    // Set experimental values for main thread hangs:
509    // 128ms for transient hangs and 8192ms for permanent hangs
510    sMessageLoop->set_hang_timeouts(128, 8192);
511  } else if (messageLoop->type() == MessageLoop::TYPE_MOZILLA_CHILD) {
512    messageLoop->set_thread_name("Gecko_Child");
513    messageLoop->set_hang_timeouts(128, 8192);
514  }
```

+callers:"mozilla::dom::ContentParent::KillHard(const char *)"

16 results from the mozilla-central tree:

dom / clients / manager / ClientSourceParent.cpp
49 mContentParent->KillHard("invalid ClientSourceParent actor");
dom / filesystem / FileSystemRequestParent.cpp
113 mContentParent->KillHard("This path is not allowed.");
dom / indexedDB / ActorsParent.cpp
964 aContentParent->KillHard("IndexedDB CheckPermission 0");
974 aContentParent->KillHard("IndexedDB CheckPermission 1");
027 aContentParent->KillHard("IndexedDB CheckPermission 2");
034 aContentParent->KillHard("IndexedDB CheckPermission 3");
dom / ipc / ContentParent.cpp
999 KillHard("BridgeToChildProcess");
559 KillHard(aReason);
628 KillHard("DeallocateLayerTreeId");
424 KillHard("SandboxBroker::Create failed");
432 KillHard("SandboxInitFailed");
2692 KillHard("PlaySound only accepts a valid chrome URI.");
3106 self->KillHard("ShutDownKill");
5546 KillHard("FileCreationRequest is not supported.");
ipc / glue / BackgroundParentImpl.cpp
602 mContentParent->KillHard("BroadcastChannel killed: principal::GetOrigin failed.");
607 mContentParent->KillHard("BroadcastChannel killed: origins do not match.");

```
6
7   #ifndef mozilla_HangMonitor_h
8   #define mozilla_HangMonitor_h
9
10  namespace mozilla {
11  namespace HangMonitor {
12
13  /**
14   * Signifies the type of activity in question
15  */
16  enum ActivityType
17  {
18    /* There is activity and it is known to be UI related activity. */
19    kUIActivity,
20
21    /* There is non UI activity and no UI activity is pending */
22    kActivityNoUIAVail,
23
24    /* There is non UI activity and UI activity is known to be pending */
25    kActivityUIAVail,
26
27    /* There is non UI activity and UI activity pending is unknown */
28    kGeneralActivity
29  };
30
31  /**
32   * Start monitoring hangs. Should be called by the XPCOM startup process only.
33   */
34  void Startup();
35
36  /**
   * Stop monitoring hangs and join the thread.
   */
  void Shutdown();

  /**
   * Notify the hang monitor of activity which will reset its internal timer.
   *
   * @param activityType The type of activity being reported.
   * @see ActivityType
   */
  void NotifyActivity(ActivityType activityType = kGeneralActivity);

  /*
   * Notify the hang monitor that the browser is now idle and no detection should
   * be done.
   */
  void Suspend();

  } // namespace HangMonitor
  } // namespace mozilla

  #endif // mozilla_HangMonitor_h
```

https://dxr.mozilla.org/mozilla-central/source/xpcom/threads/HangMonitor.h

Backtrace

Apache Traffic Server

Detect

What: Plugin, invalid instructions

How: Signals & child processes

```
int
main(int /* argc ATS_UNUSED */, const char **argv)
{
  FILE *fp;
  char *logname;
  TSMgmtError mgmterr;
  crashlog_target target;
  pid_t parent = getppid();

  diags = new Diags("traffic_crashlog", "" /* tags */, "" /* actions */, new BaseLogFile("stderr"));

  appVersionInfo.setup(PACKAGE_NAME, "traffic_crashlog", PACKAGE_VERSION, __DATE__, __TIME__, BUILD_MACHINE, BUILD_PERSON, "");

  // Process command line arguments and dump into variables
  process_args(&appVersionInfo, argument_descriptions, countof(argument_descriptions), argv);

  if (wait_mode) {
    EnableDeathSignal(SIGKILL);
    kill(getpid(), SIGSTOP);
  }
```

traffic_crashlog *is a helper process that catches Traffic Server crashes and writes a crash report log to the logging directory. Other than for debugging or development purposes,* ***traffic_crashlog*** *is not intended for users to run directly.*
When ***traffic_server*** *starts, it will launch a* ***traffic_crashlog*** *process and keep it stopped, activating it only if a crash occurs.*

- "https://docs.trafficserver.apache.org/en/5.3.x/reference/commands/traffic_crashlog.en.html?highlight=crash"

```
PR_SET_PDEATHSIG (since Linux 2.1.57)
       Set the parent process death signal of the calling process to arg2 (either a signal value in the range 1..maxsig, or 0 to  clear).
       This  is  the signal that the calling process will get when its parent dies.  This value is cleared for the child of a fork(2) and
       (since Linux 2.4.36 / 2.6.23) when executing a set-user-ID or set-group-ID binary.

PR_GET_PDEATHSIG (since Linux 2.3.15)
       Return the current value of the parent process death signal, in the location pointed to by (int *) arg2.
```

Embedded App (RTOS)

Detect

What: Hardware Interrupts, Software Exceptions
How: OS Signals, explicit interrupt service routines

If a hardware interrupt or a software exception occurs on the embedded machine, the RTOS handles the interrupt and/or exception in an interrupt service routine. If the given hardware interrupt or software exception is critical, or one we wish to capture, a core dump can be initiated.
-- "http://s3.mentor.com/public_documents/whitepaper/resources/mentorpaper_32245.pdf"

Capture

Data to Debug

A complete talk on its own

- **Registers (GPR, status, external)**
- **Instruction Memory**
  - Callstack
- **Data Memory**
- **Symbolic Information**
  - in-memory process -> symbols / code
  - Consider the case stripped vs unstripped binaries
- **Application State**
  - Global, Local variables
- **Characterization Information**
  - Data to describe
  - Software version, hardware revision, request type, etc

Mozilla Firefox

Capture

Minidump through the Google::Breakpad/Crashpad library

- Callstack across threads
- Stack space
- Attributes of the crash environment
- Requires symbolication

Generated and transmitted to collection server

Uptime	30,964 seconds (8 hours, 36 minutes and 4 seconds)
Install Age	200,343 seconds since version was first installed (2 days, 7 hours and 39 minutes)
Install Time	2018-03-19 06:22:32
Product	Firefox
Release Channel	beta
Version	60.0b4
Build ID	20180315232954
OS	Windows 7

```
3804 static void
3805 CreatePairedChildMinidumpAsync(ProcessHandle aTargetPid,
3806                                ThreadId aTargetBlamedThread,
3807                                nsCString aIncomingPairName,
3808                                nsCOMPtr<nsIFile> aIncomingDumpToPair,
3809                                nsIFile** aMainDumpOut,
3810                                xpstring aDumpPath,
3811                                std::function<void(bool)>&& aCallback,
3812                                RefPtr<nsIThread>&& aCallbackThread,
3813                                bool aAsync)
3814 {
3815   AutoIOInterposerDisable disableIOInterposition;
3816
3817 #ifdef XP_MACOSX
3818   mach_port_t targetThread = GetChildThread(aTargetPid, aTargetBlamedThread);
3819 #else
3820   ThreadId targetThread = aTargetBlamedThread;
3821 #endif
3822
3823   // dump the target
3824   nsCOMPtr<nsIFile> targetMinidump;
3825   if (!google_breakpad::ExceptionHandler::WriteMinidumpForChild(
3826         aTargetPid,
3827         targetThread,
3828         aDumpPath,
3829         PairedDumpCallbackExtra,
3830         static_cast<void*>(&targetMinidump)
3831 #ifdef XP_WIN32
3832         , GetMinidumpType()
3833 #endif
3834       )) {
3835     NotifyDumpResult(false, aAsync, Move(aCallback), Move(aCallbackThread));
3836     return;
3837   }
```

Backtrace

https://crash-stats.mozilla.com/report/index/70c62f6c-9c33-4880-980b-6f4d70180321

Apache Traffic Server

Capture

```
88  int
89  main(int /* argc ATS_UNUSED */, const char **argv)
90  {
91    FILE *fp;
92    char *logname;
93    TSMgmtError mgmterr;
94    crashlog_target target;
95    pid_t parent = getppid();
96
97    diags = new Diags("traffic_crashlog", "" /* tags */, "" /* actions */, new BaseLogFile("st
98
99    appVersionInfo.setup(PACKAGE_NAME, "traffic_crashlog", PACKAGE_VERSION, __DATE__, __TIME__
100
101   // Process command line arguments and dump into variables
102   process_args(&appVersionInfo, argument_descriptions, countof(argument_descriptions), argv)
103
104   if (wait_mode) {
105     EnableDeathSignal(SIGKILL);
106     kill(getpid(), SIGSTOP);
107   }
108
109   // If our parent changed, then we were woken after traffic_server exited. There's no point
110   // emit a crashlog because traffic_server is gone.
111   if (getppid() != parent) {
112     return 0;
113   }
```

Uses a child process to capture + log data from parent.

By default generates a log of

- **Callstack across threads**
- **Configuration & System wide information**
- **Global data structures**

Up to user to transmit log to collection server

User has the option of plugging in their own crash data collection mechanism (traffic_crash_log)

This let's your capture a coredump: complete capture of the in-memory process image.

Apache Traffic Server

Capture

```
192     crashlog_write_procname(fp, target);
193     crashlog_write_exename(fp, target);
194     fprintf(fp, LABELFMT "Traffic Server %s\n", "Version:", PACKAGE_VERSION);
195     crashlog_write_uname(fp, target);
196     crashlog_write_datime(fp, target);
197
198     fprintf(fp, "\n");
199     crashlog_write_siginfo(fp, target);
200
201     fprintf(fp, "\n");
202     crashlog_write_registers(fp, target);
203
204     fprintf(fp, "\n");
205     crashlog_write_backtrace(fp, target);
206
207     fprintf(fp, "\n");
208     crashlog_write_procstatus(fp, target);
209
210     fprintf(fp, "\n");
211     crashlog_write_proclimits(fp, target);
212
213     fprintf(fp, "\n");
214     crashlog_write_regions(fp, target);
215
216     fprintf(fp, "\n");
217     crashlog_write_records(fp, target);
218
219     Error("wrote crash log to %s", logname);
```

https://github.com/apache/trafficserver/blob/2ee71379a9a57616ae5b501da04341d08f3e8aca/cmd/traffic_crashlog/traffic_crashlog.cc

```
53  // Suck in a file from /proc/$PID and write it out with the given label.
54  static bool
55  write_procfd_file(const char *filename, const char *label, FILE *fp, const crashlog_target &target)
56  {
57    ats_scoped_fd fd;
58    TextBuffer text(0);
59    fd = procfd_open(target.pid, filename);
60    if (fd != -1) {
61      text.slurp(fd);
62      text.chomp();
63      fprintf(fp, "%s:\n%.*s\n", label, (int)text.spaceUsed(), text.bufPtr());
64    }
65
66    return !text.empty();
67  }
68
69  bool
70  crashlog_write_regions(FILE *fp, const crashlog_target &target)
71  {
72    return write_procfd_file("maps", "Memory Regions", fp, target);
73  }
74
75  bool
76  crashlog_write_procstatus(FILE *fp, const crashlog_target &target)
77  {
78    return write_procfd_file("status", "Process Status", fp, target);
79  }
80
81  bool
82  crashlog_write_proclimits(FILE *fp, const crashlog_target &target)
83  {
84    return write_procfd_file("limits", "Process Limits", fp, target);
85  }
```

https://github.com/apache/trafficserver/blob/2ee71379a9a57616ae5b501da04341d08f3e8aca/cmd/traffic_crashlog/procinfo.cc

Apache Traffic Server

Capture

```
342   Debug("backtrace", "tracing %zd threads for traffic_server PID %ld", threads.size(), (long)lmgmt->watched_process_pid);
343
344   for (auto threadid : threads) {
345     Debug("backtrace", "tracing thread %ld", (long)threadid);
346     // Get the thread name using /proc/PID/comm
347     ats_scoped_fd fd;
348     char threadname[128];
349
350     snprintf(threadname, sizeof(threadname), "/proc/%ld/comm", (long)threadid);
351     fd = open(threadname, O_RDONLY);
352     if (fd >= 0) {
353       text.format("Thread %ld, ", (long)threadid);
354       text.readFromFD(fd);
355       text.chomp();
356     } else {
357       text.format("Thread %ld", (long)threadid);
358     }
359
360     text.format(":\n");
361
362     backtrace_for_thread(threadid, text);
363     text.format("\n");
364   }
```

```
255 static void
256 backtrace_for_thread(pid_t threadid, TextBuffer &text)
257 {
258   int status;
259   unw_addr_space_t addr_space = NULL;
260   unw_cursor_t cursor;
261   void *ap       = NULL;
262   pid_t target   = -1;
      unsigned level = 0;

      // First, attach to the child, causing it to stop.
      status = ptrace(PTRACE_ATTACH, threadid, 0, 0);
      if (status < 0) {
        Debug("backtrace", "ptrace(ATTACH, %ld) -> %s (%d)", (long)threadid, strerror(errno), errno);
        return;
      }

      // Wait for it to stop (XXX should be a timed wait ...)
      target = waitpid(threadid, &status, __WALL | WUNTRACED);
      Debug("backtrace", "waited for target %ld, found PID %ld, %s", (long)threadid, (long)target,
            WIFSTOPPED(status) ? "STOPPED" : "???");
      if (target < 0) {
        goto done;
      }

      ap = _UPT_create(threadid);
      Debug("backtrace", "created UPT %p", ap);
      if (ap == NULL) {
        goto done;
      }
285
286   addr_space = unw_create_addr_space(&_UPT_accessors, 0 /* byteorder */);
287   Debug("backtrace", "created address space %p", addr_space);
288   if (addr_space == NULL) {
289     goto done;
290   }
```

...

Apache Traffic Server

Capture: Coredumps

You can also turn off traffic_crashlog and enable coredumps.

Coredump record the complete state of the process image at the time of crash.

- **This means if your process is using 32GB of memory, the coredump will be ~32GB.**

Need a debugger (GDB, LLDB, etc) to inspect.

Further reading: https://backtrace.io/blog/blog/2015/10/03/whats-a-coredump/index.html

Other Capture Mechanisms

Capture

- **Attach GDB (coredump or live process)**
  - Run GDB-python scripts and output info to a log
- **Process Snapshotting**
  - Backtrace-ptrace: out of process tracer that captures callstack, variables, system-wide information and

Diagnose

Managing the onslaught of bugs

Debugging

- Detected a crash
- Captured data relevant to investigation
- Now what?

Post Mortem Analysis

Analyzing crash after the fact

Root Cause Analysis = (intuition) + (educated guessing) + (data analysis)

Consider

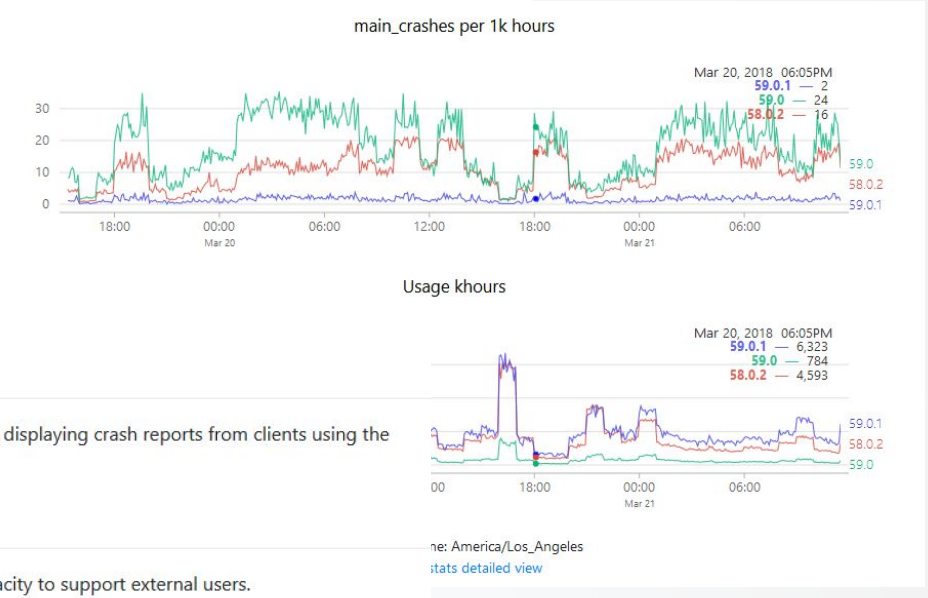
- **Never have a perfect picture of what went wrong**
  - Event recreating bugs can fail to nail down root cause
  - Can't log the state of the world since the start of the world
- **You never have just 1 crash to diagnose**
  - Which crash to fix first? Crash Bucketing can reduce great signal to noise
- **Can you make up for incomplete information by analyzing crashes in aggregate?**
  - Correlating characteristics as a hint of what's going on
  - Data analysis techniques become key in RCA & debugging automation

Mozilla Firefox

Diagnose

https://crash-stats.mozilla.com/home/product/Firefox (Mozilla Socorro)
https://data-missioncontrol.dev.mozaws.net/#/

Used to collect minidumps + attributes
Server-side symbolication
Performs crash bucketing/deduplication
Provides searching across crash set, high level visualization



Apache Traffic Server + Embedded RTOS

The rest of us

Crash collection and analysis are dependent on the developer

Options:

- **Log analysis solutions (Splunk, ELK)**
- **Roll your own**
- **3rd-party crash reporting and analysis solutions**

Log Aggregators

Splunk, ELK

Treat crash dumps like logs:

- **Basic searching**
- **Bucketing based on subset of callstack/frames**
  - Matching crashes by a single frame
- **Requires some work to get metadata analysis**
  - analysis across metadata

MSFT WER / Dr. Watson

Decade of diagnosing Windows, Office, crashes

Crash Bucketing / Impact Analysis

An analysis of the error reports revealed that 96% of the faulting computers were running a specific third-party device driver.

Automated patch distribution (see graph to the right)

Debugging in the (Very) Large:
Ten Years of Implementation and Experience

Kirk Glerum, Kinshuman Kinshumann, Steve Greenberg, Gabriel Aul,
Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

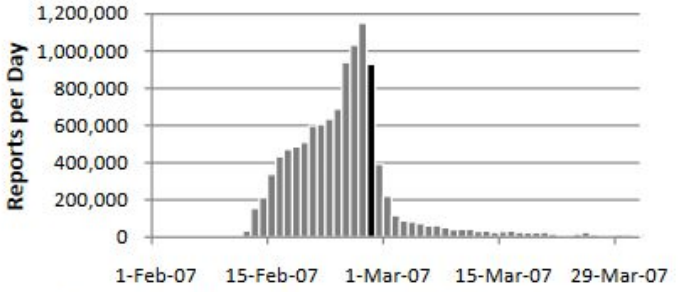


Figure 10. Renos Malware: Number of error reports per day. Black bar shows when the fix was released through WU.

MSFT WER / Dr. Watson

Decade of diagnosing Windows, Office, crashes

Data framework for testing hypothesis against large sets of crashes (!!analyze)

Historic analysis helped resolve “Heisenbugs”

Debugging in the (Very) Large:
Ten Years of Implementation and Experience

Kirk Glerum, Kinshuman Kinshumann, Steve Greenberg, Gabriel Aul,
Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

The WER database can be used to test programmer hypotheses about the root causes of errors. The basic strategy is to construct a debugger test function that can evaluate a hypothesis on a memory dump, and then apply it to thousands of memory dumps to verify that the hypothesis is not violated. For example, one of the Windows programmers was recently debugging an issue related to the plug-and-play lock in the Windows I/O subsystem. We constructed an expression to extract the current holder of the lock from a memory dump and then ran the expression across 10,000 memory dumps to see how many of the reports had the same lock holder. One outcome of the analysis was a bug fix; another was the creation of a new heuristic for `!analyze`.

With WER’s scale, even obscure Heisenbugs [17] can generate enough error reports for isolation. Early in its use WER helped programmers find bugs in Windows NT and Office that had existed for over five years. These failures were hit so infrequently to be impossible to isolate in the lab, but were automatically isolated by WER. A calibrating experiment using a pre-release of MSN Explorer to 3.6 million users found that less than 0.18% of users see two or more failures in a 30 day period.

Crash Reporting and Analysis Solutions

Solutions for the rest of us

Options:

- **Backtrace (that's me)**
- **Bugsplat**
- **Raygun**

Important factors:

- **Attach custom attributes and logs**
- **How easy it to access the data?**
- **Crash data retention periods**
- **Flexibility of data analysis**


Backtrace

Overview

Prepare for the inevitable

Detect & Capture

- **Process Monitor, In-process detection, system functionality**
- **Breakpad/Crashpad, backtrace(2), ptrace(2), coredumps**

Diagnose & Analyze

- **Individual: debuggers, intuition (no silver bullets)**
- **Aggregate: Log aggregators, build your own, 3rd-party solutions**

Follow-up

At least 3 topics that I can go further in-depth in a future talk:

- **Walking the callstack**
- **Symbolication**
  - https://www.slideshare.net/sbahra/symbolic-debugging-with-dwarf
- **Debugging through data analysis**
  - Algorithmic debugging

Let me know if these interest you and I'll take a crack at it!

Dealing with Application Crashes

ACCU -- March 2018
@nullisnt0
amathew@backtrace.io