Unleashing the Power of Static Analysis

Manuvir Das

Principal Researcher
Center for Software Excellence
Microsoft Corporation

Talking the talk …

- Static analysis tools can make a huge impact on how software is engineered
- The trick is to properly balance research with a focus on deployment
- The Center for Software Excellence (CSE) at Microsoft is doing this (well?) today

… walking the walk

- CSE impact on Windows Vista
  - Found 100,000+ *fixed* bugs
  - Added 500,000+ specifications
  - Answered thousands of emails
- We are program analysis researchers
  - But we measure our success in *adoption*
  - And we feel the pain of the customer

Context

- The Nail (Windows)
  - Manual processes do not scale to real software
- The Hammer (Static Analysis)
  - Automated methods for searching programs
- The Carpenter (CSE)
  - A systematic, heavily automated, approach to improving the quality of software

What is static analysis?

- grep == static analysis
- static analysis == grep

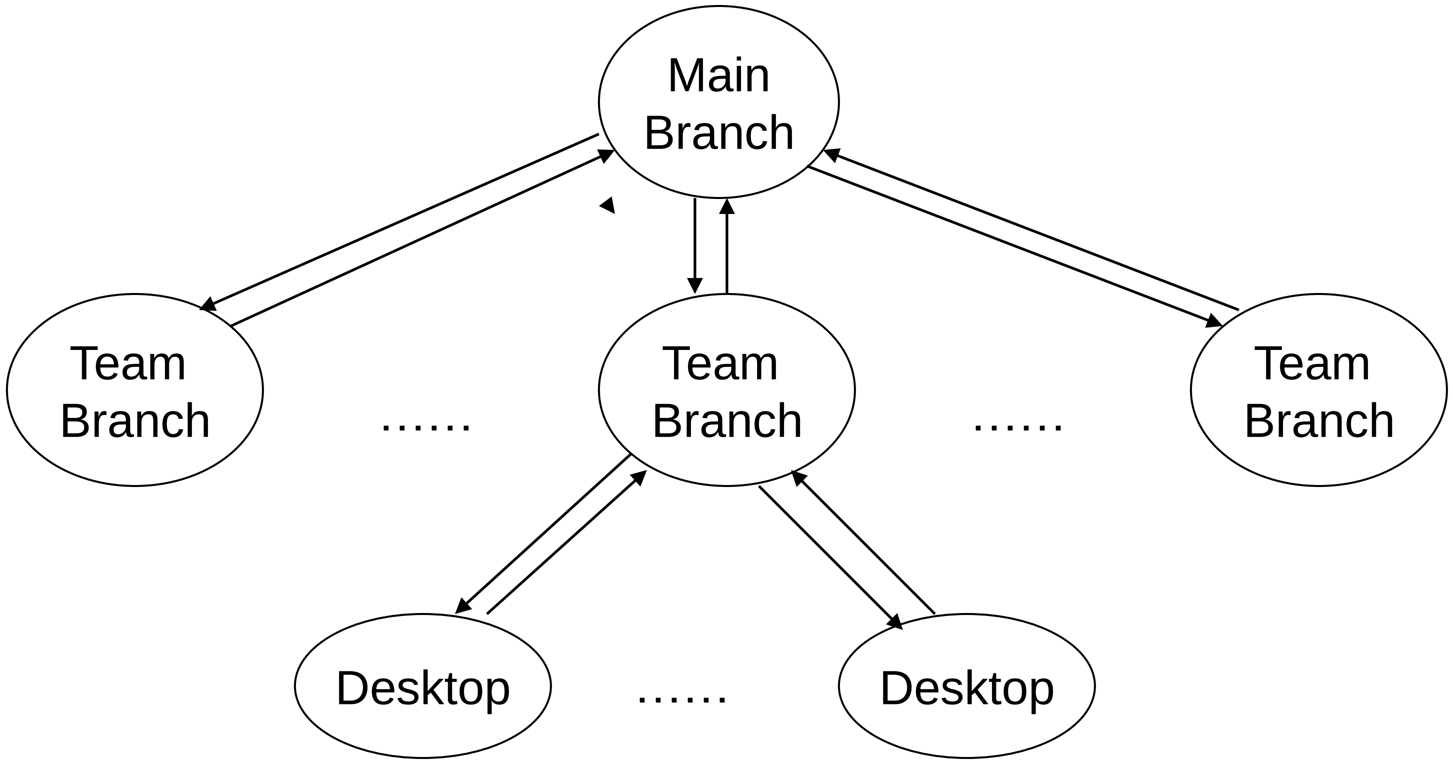
- syntax trees, CFGs, alias analysis, dataflow analysis, dependency analysis, binary analysis, symbolic evaluation, model checking, specifications, …

Roadmap

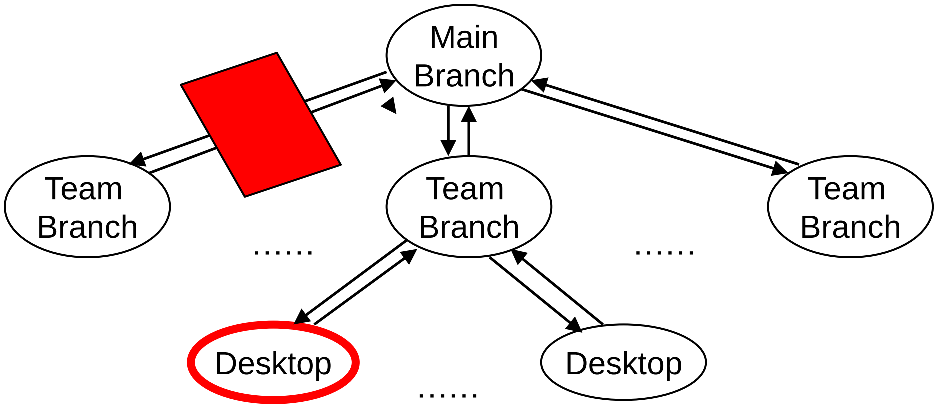
- Engineering process
- Static analysis tools
- Lessons

Engineering process

Build Architecture

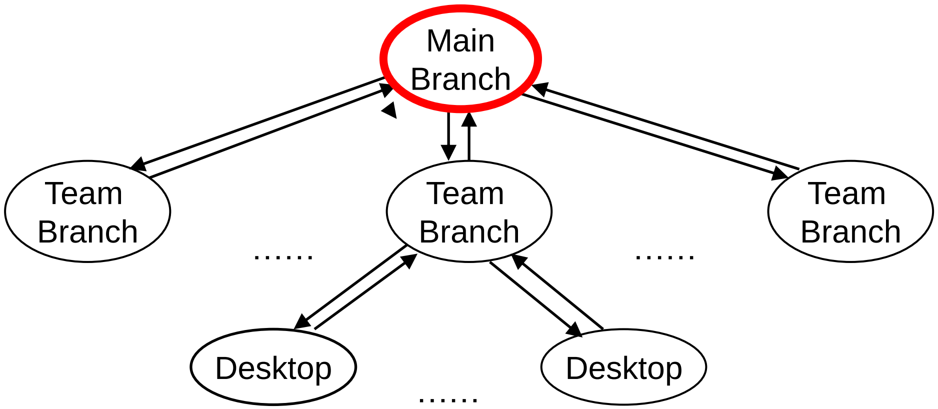


Quality Gates



- Lightweight tools
  - run on developer desktop & feature branches
  - issues tracked within the program artifacts
- Enforced by rejection at gate

Central Bug Filing



- Heavyweight tools
  - run on main branch
  - issues tracked through a central bug database
- Enforced by bug cap

Static analysis tools

1. Code correctness

- Reject code with null pointer dereferences, uninitialized memory, resource leaks, …
- Inter-procedural simulation – PREfix
  - Process the call graph bottom-up
  - Perform symbolic evaluation on a fixed number of paths through every function
  - Build incomplete symbolic function models
  - Use symbolic state to avoid infeasible paths
  - Report defects when bad states arise

2. Integer overflow

- Reject code with potential security holes due to unchecked integer arithmetic

```
size1 = …
size2 = …
data = MyAlloc(size1+size2);
for (i = 0; i < size1; i++)
  data[i] = …
```

- Construct an expression tree for every *interesting* expression in the code
- Ensure that every operation is checked

3. Architecture layering

- Reject code that breaks the component architecture of the product
  - No dependencies from lower layers of the system to higher layers of the system
- Dependency analysis tool – MaX
  - Construct a graph of dependencies between binaries (DLLs) in the system
    - Obvious : call graph
    - Subtle : registry, RPC, …

4. Security

- Problem
  - A security issue is discovered through internal testing, or in the field (MSRC, Watson)
- Diagnosis
  - Identify the code pattern that caused the bug
- Detection (defect by example)
  - Specify the code pattern formally in OPAL
  - Use checkers to find instances of the pattern

RegKey leak defect

```
status = RegOpenKeyExW( HKEY_LOCAL_MACHINE,
   L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Perflib",
   0L, KEY_READ, & hLocalKey);

if (status == ERROR_SUCCESS) bLocalKey = TRUE;

… block of code that uses hLocalKey …

if (bLocalKey)
  CloseHandle(hLocalKey);
```

- Bug: registry key is closed by calling the generic CloseHandle API
  - May fail to clean up some data that is specific to registry key data structures

RegKey leak code pattern

- Search for code paths along which a registry key is opened, and then closed using the generic CloseHandle API
- Specification:
  - define a sequence of relevant actions
  - e.g. A(k)…B(h)
  - define the actions (e.g. A, B, k and h)

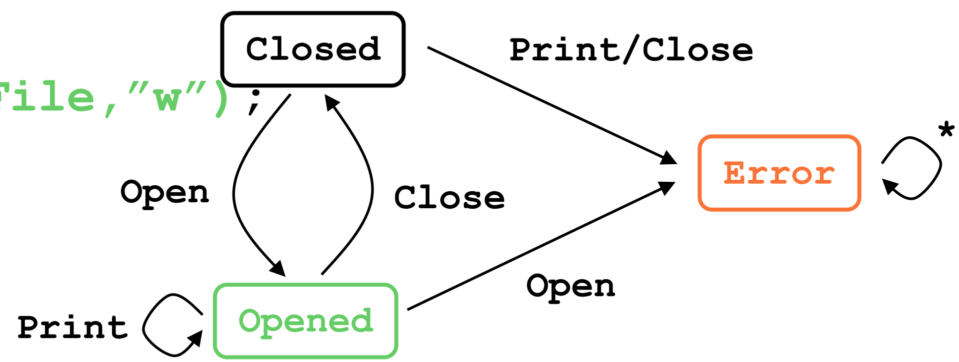
RegKey leak specification

```
defect RegKeyCloseHandle
{
  // A(x)…B(x)
  sequence OpenKey(key);CloseHandle(handle)
  message “Registry key closed using generic CloseHandle API!”

  // A(x)
  pattern OpenKey(key)
     /RegOpenKeyEx[AW](@\d+)?$/ (_,_,_,_,&key)
         where (return == 0)

  // B(x)
  pattern CloseHandle(handle)
     /CloseHandle(@\d+)?$/ (handle)
}
```

This is the entire specification effort for the codebase

Safety properties

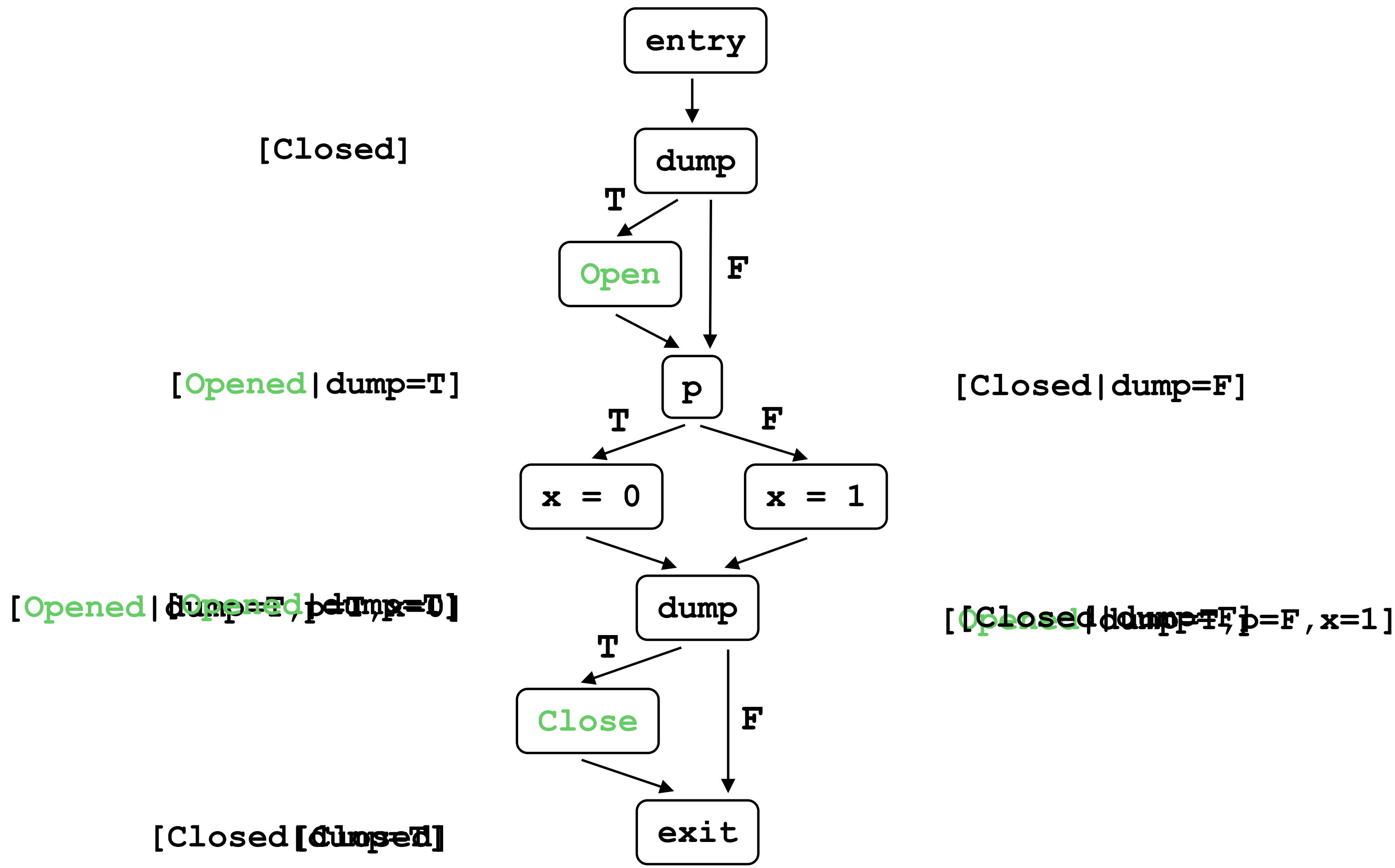
```
void main ()
{
  if (dump)
    fil = fopen(dumpFile,"w");

  if (p)
    x = 0;
  else
    x = 1;

  if (dump)
    fclose(fil);
}
```



ESP

- Symbolic state: FSA + execution state
- Branch points: Does execution state uniquely determine branch direction?
  - Yes: process appropriate branch
  - No: split & update state, and process both branches
- Merge points: Do states agree on FSA?
  - **Yes: merge states**
  - **No: process states separately**

ESP example



entry
dump
T
Open
F
p
T
F
x = 0
x = 1
dump
T
Close
F
exit
[Closed]
[Opened|dump=T]
[Closed|dump=F]

5. Concurrency

- Deadlocks, data races, orphan locks, …
- Sequential analysis of lock sequences at every program point of every thread
  - Cycle in lock ordering: deadlock
  - Access without consistent locking: data race
  - Exit while holding critical section: orphan lock!
- Inter-procedural dataflow analysis – ESPC
  - Instance of ESP – lock sequences control merge
  - Understands Win32 locking semantics

6. Buffer overruns

- Defect: a buffer access index is out of bounds
- Detection: check that index is within bounds

- Problem: where are the buffer bounds stored?
  - Tools must track buffer size from allocation to access
  - Exhaustive global analysis is infeasible

- Solution: turn global analysis into local analysis
  - Standard Annotation Language (SAL)
  - Specify buffer sizes at function interfaces
  - Perform modular (one function at a time) analysis

SAL example 1

- **`wcsncpy`** [precondition] destination buffer must have enough allocated space

```
wchar_t wcsncpy (
 wchar_t *dest, wchar_t *src, size_t num );

wchar_t wcsncpy (
 __pre __notnull __pre __writableTo(elementCount(num))
 wchar_t *dest,
 wchar_t *src, size_t num );

wchar_t wcsncpy (
 __out_ecount(num) wchar_t *dest,
 wchar_t *src, size_t num);
```

SAL example 2

- **`memcpy`**

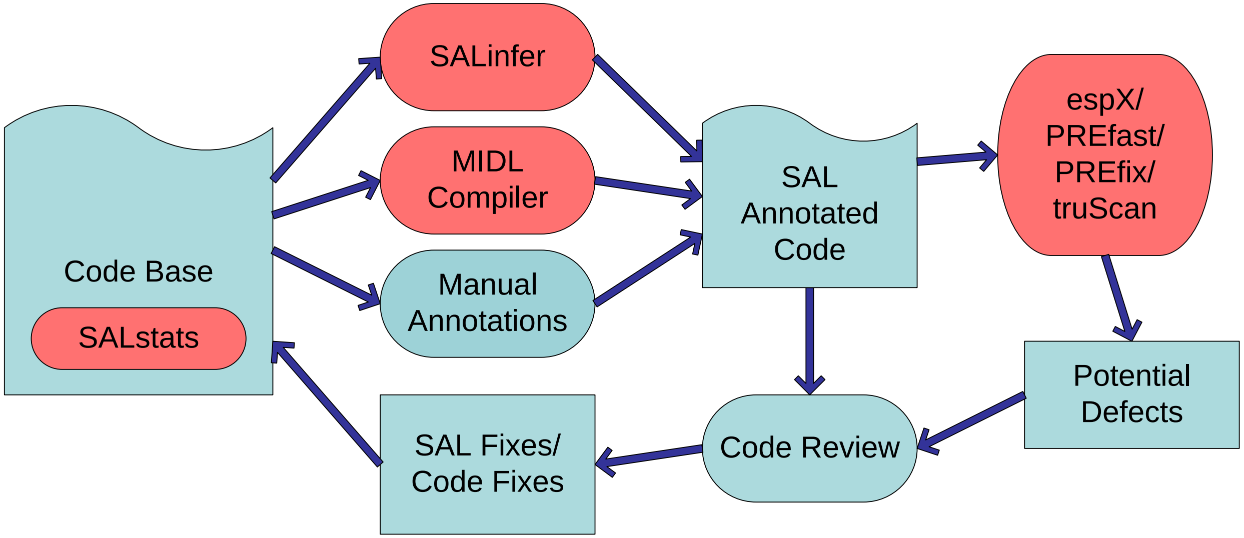
```
void * memcpy ( void * dest, void * src, size_t num );

void * memcpy (
 __pre __notnull __pre __writableTo(byteCount(num))
 __post __readableTo(byteCount(num)) void * dest,
 __pre __notnull __pre __deref __readonly
 __pre __readableTo(byteCount(num)) void * src,
 size_t num );

void * memcpy (
 __out_bcount_full(num) void * dest,
 __in_bcount(num) void * src, size_t num );
```

SAL primer

- Usage example:

  a_0 *RT func(*a_1 *…* a_n *T par)* a_i : SAL annotation
- Interface contracts
  - pre, post, object invariants
- Basic properties
  - null, readonly, valid, range, …
- Buffer extents
  - writableTo(size), readableTo(size)
- Buffer size formats
  - (byte|element)Count, endPointer, sentinel, …

SAL ecosystem



- espX/PREfast/… : Use annotations to find defects
- SALstats : Identify parameters that should be annotated
- MIDL Compiler : Translate MIDL directives to annotations
- SALinfer : Infer annotations using global static analysis

SALinfer example

```
void work() {
    int tmp[200];                          size(tmp,200)
    wrap(tmp, 200);
}

void wrap(int *buf, int len) {             size(buf,len)      write(buf)
    int *buf2 = buf;                       size(buf2,len)
    int len2 = len;                        size(buf2,len2)    write(buf2)
    zero(buf2, len2);
}

void zero(int *buf, int len) {             size(buf,len)      write(buf)
    int i;
    for(i = 0; i <= len; i++)
        buf[i] = 0;                                           write(buf)
}
```

SALinfer example

```
void work() {
    int tmp[200];
    wrap(tmp, 200);
}

void wrap(__out_ecount(len) int *buf, int len) {
    int *buf2 = buf;
    int len2 = len;
    zero(buf2, len2);
}

void zero(__out_ecount(len) int *buf, int len) {
    int i;
    for(i = 0; i <= len; i++)
        buf[i] = 0;
}
```

espX example

```
void zero(__out_ecount(len) int *buf, int len) {
    int i;
    for(i = 0; i <= len; i++)
        buf[i] = 0;
}
```

assume(sizeOf(buf) == len)

```
for(i = 0; i <= len; i++)
```

inv (i >= 0 && i <= len)

assert(i >= 0 && i < sizeOf(buf))

```
buf[i] = 0;
```

Warning: Cannot validate buffer access.
Overflow occurs when i == len

Constraints:

(C1) i >= 0

(C2) i <= len

(C3) sizeOf(buf) == len

Goal: i >= 0 && i < sizeOf(buf)

Subgoal 1: i >=0   by (C1)

Subgoal 2: i < len   FAIL

SAL impact

- Windows Vista
  - Mandate: Annotate 100,000 mutable buffers
  - Developers annotated 500,000+ parameters
  - Developers fixed 20,000+ bugs
- Office 12
  - Developers fixed 6,500+ bugs
- Visual Studio, SQL, Exchange, …
- External customers
  - CRT + Windows headers SAL annotated
  - SAL aware compiler shipped with VS 2005

SAL evaluation

Vista – mutable string buffer parameters

- Annotation cost:
  - [–] 100,000 parameters required annotations
  - [+] 4 out of 10 automatic
- Defect detection value:
  - [+] 1 buffer overrun exposed per 20 annotations
- Locked in progress:
  - [+] 9.4 out of 10 buffer accesses validated

Lessons

Forcing functions for change

- Gen 1: Manual Review
  - Too many code paths to think about
- Gen 2: Massive Testing
  - Inefficient detection of simple errors
- Gen 3: Global Program Analysis
  - Delayed results
- Gen 4: Local Program Analysis
  - Lack of calling context limits accuracy
- Gen 5: Specifications

Acceptance of specifications

- Developers like incremental specs
  - No specifications, no bugs
- Developers like useful specs
  - More specifications, more real bugs
- Developers like informative specs
  - Make implicit information explicit
  - Avoid repeating what the code says

Defect detection myths

- Soundness matters
  - sound == find only real bugs
  - The real measure is Fix Rate
- Completeness matters
  - complete == find all the bugs
  - There will never be a complete analysis
- Developers only fix real bugs
  - Developers fix bugs that are easy to fix, and
  - Unlikely to introduce a regression

Theory is important

- Fundamental ideas have been crucial
  - Hoare logic
  - Dataflow analysis
  - Abstract interpretation
  - Graph algorithms
  - Context-sensitive analysis
  - Alias analysis

Summary

- Static analysis tools can make a huge impact on how software is engineered
- The trick is to properly balance research with a focus on deployment
- The Center for Software Excellence (CSE) at Microsoft is doing this (well?) today

http://www.microsoft.com/cse
http://research.microsoft.com/manuvir