Lecture 2: Program “Correctness” Tools

Aditya V. Nori
Microsoft Research, India

[Thanks to Tom Ball & Sriram Rajamani for material from their lectures]
Righting Software

James R. Larus, Microsoft Research
Thomas Ball, Microsoft Research
Manuvir Das, Microsoft Research
Robert DeLine, Microsoft Research
Manuel Fähndrich, Microsoft Research
Jon Pincus, Microsoft Research
Sriram K. Rajamani, Microsoft Research
Ramanathan Venkatapathy, Microsoft Research
Software Tools

• What tools do you use to develop and debug software?

Editors

XEmacs

Vim the editor

Compilers/Debuggers

GCC

Microsoft Visual Studio
Software Tools

• What about correctness tools?
• Help programmer deal with program errors:
  – Errors using a programming language feature (E.g. referencing an uninitialized variable)
  – Errors in API usage
Two approaches

Testing
• Instrument the property as a “monitor” state machine
• Run the program on several inputs to see if the monitor reaches a bad state
• Limited coverage
• No false errors

Static Analysis
• Build CFG of the program and include only the relevant state
• Flow analysis over the CFG to see if the program reaches a bad state
• Full coverage (proof)
• Can get false errors
Soundness vs. Completeness

- Soundness vs. Completeness
- False errors vs. Missed errors

May (Over)

Actual

Must (Under)
Soundness vs. Completeness: Example

```c
FILE* f;
if (complex_calc1())
    f = fopen(...);
...
if (complex_calc2())
    fclose(f);
```
Elusive triangle

We will let go of this one!

Deep properties

Large programs

Automation
Past, Present and Future

• First generation
  – heuristic tools
  – widely deployed and used

• Second generation
  – sound tools
  – now beginning to be widely used

• Third generation
  – combinations
    • verification with testing
    • program analysis with statistics
  – active area of current research
First Generation of Correctness Tools

- **PREfix**
  - detailed, path-by-path interprocedural analysis
  - heuristic (unsound, incomplete)
  - expensive (a few days on Windows)
  - effective at finding bugs

- **PREfast**
  - user-supplied plug-ins find bugs by traversing AST
  - desktop use, easily customized

- **Widely deployed in Microsoft**
  - 1/6 of bugs fixed in Windows Server 2003 found by these tools
PREfix

- Detects errors in C/C++ code
  - null pointer, memory allocation, uninitialized value, resource state, library usage, ...
- Path-sensitive analysis
- Interprocedural
  - bottom up traversal of call graph
  - models routine by examining limited set of paths (100)
  - apply model at call site
  - expensive, batch computation for large systems
- Large effort to minimize effects of false positives
  - filtering and prioritizing error reports
  - heuristics tuned to reduce noise (at cost of precision)
```c
int myfunc(int j)
{
    int k;
    if ( j == 0 )
    {
        k = 1;
        return k;
    }
}
```
PREfix Summary

• Adhoc, but extremely scalable defect detection tool

• Huge impact in Microsoft
  – Complemented testing by exploring “rare” paths
  – Part of Visual Studio
PREfast

• Front-end of C++ compiler with clean AST
• Plug-ins traverse AST looking for specific patterns
  – Structural “grep”
• Enormously popular
  – Desktop checking (fast & private)
  – Easily customizable
• Shipped in Visual Studio
What's wrong with this code?

```
char *p1 = new char[10];
char *p2 = new char[10];
...
delete[] p2;
delete[] p1;
```

Can leak memory `<pointer>` due to an exception from second new operation.

```
char *p1 = new char[10];
char *p2;
try { p2 = new char[10]; } 
catch (std::bad_alloc *e) {
    delete[] p1;
}
...
delete[] p2;
delete[] p1;
```

Yes, its ugly, and necessary in a non-GC language.
Metal

- Gcc extensions developed by Dawson Engler’s group at Stanford
- Heuristic analyses to scan AST and CFG
- Scales to millions of lines of C/C++ code
- Extensively used on open source code bases to find and fix several errors
- They have formed a company Coverity to sell these tools
Second Generation of Correctness Tools

- **SLAM**: Software Model Checking
  - Sound analysis of programs
What rules can SLAM check for?

- Properties are specified using state machines
- Tool checks for all violations of the rule on the property
- Counterexample driven refinement is used to reduce false errors.
Source Code

Development

Static Driver Verifier

Rules

Precise API Usage Rules (SLIC)

Drive testing tools

Software Model Checking

Testing

Read for understanding

New API rules

Defects

100% path coverage

SLAM
SLAM - Software Model Checking

• **SLAM innovations**
  – boolean programs: a new model for software
  – model creation (c2bp)
  – model checking (bebop)
  – model refinement (newton)

• **SLAM toolkit**
  – built on MSR program analysis infrastructure
SLIC

• Finite state language for stating rules
  – monitors behavior of C code
  – temporal safety properties
  – familiar C syntax

• Suitable for expressing control-dominated properties
  – e.g. proper sequence of events
  – can encode data values inside state
State Machine for Locking

Locked

Unlocked

Error

Acq

Rel

Acq

Rel

Locking Rule in SLIC

```c
state {
    enum {Locked, Unlocked}
    s = Unlocked;
}

KeAcquireSpinLock.entry {
    if (s==Locked) abort;
    else s = Locked;
}

KeReleaseSpinLock.entry {
    if (s==Unlocked) abort;
    else s = Unlocked;
}
```
The SLAM Process

prog. $P$ -> slic -> prog. $P'$

SLIC rule

c2bp -> bebop

boolean program

newton

predicates

path

thumbs up
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);
do {
    KeAcquireSpinLock();
    if(*){
        KeReleaseSpinLock();
    } while (*);
    KeReleaseSpinLock();
Example

Is error path feasible in C program? (newton)

do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets;
    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets; b = true;
    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++; b = b ? false : *;
    }
} while (nPackets != nPacketsOld); !b
KeReleaseSpinLock();
Example

Model checking refined boolean program (bebop)

\[ b : (nPacketsOld == nPackets) \]

do {
    KeAcquireSpinLock();
    b = true;
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while ( !b );

KeReleaseSpinLock();
Example

\begin{verbatim}
example do { 
    KeAcquireSpinLock();
    b = true;
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while ( !b );
KeReleaseSpinLock();
\end{verbatim}

Model checking refined boolean program (bebop)
Observations about SLAM

• **Automatic discovery of invariants**
  – driven by property and a finite set of (false) execution paths
  – predicates are *not* invariants, but *observations*
  – abstraction + model checking computes inductive invariants
    (boolean combinations of observations)

• **A hybrid dynamic/static analysis**
  – *newton* executes path through C code symbolically
  – c2bp+bebop explore all paths through abstraction

• **A new form of program slicing**
  – program code and data not relevant to property are dropped
  – non-determinism allows slices to have more behaviors
Precise API Usage Rules (SLIC)

Rules

Static Driver Verifier

Development

Testing

Source Code

Read for understanding

New API rules

Defects

100% path coverage

Drive testing tools

Software Model Checking

Precise API Usage Rules

Static Driver Verifier
Current status of SDV

- Runs on 100s of Windows drivers
- Finds several bugs, proves several properties
- SDV now transferred from MSR to Windows division
- Shipped as a part of the DDK with Windows Vista!
<table>
<thead>
<tr>
<th>Driver Name</th>
<th>Specialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelSpinLock</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>startIoCancel</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>addDevice</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>lowerDriverReturn</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>TargetRelationNeedsRef</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>DoubleCompletion</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>PrematureSkip</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>KeWaitDeadLock</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>WmiComplete</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>WmiForward</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>IrpProcessingComplete</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>MarkIrpPending</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>PendingCompletedRequest</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>
• **Driver:** Parallel port device driver

• **Rule:** Checks that driver dispatch routines do not call `IoCompleteRequest(...)` twice on the I/O request packet passed to it by the OS or another driver
DoubleCompletion.slic | parallel.h | pdopenp.c | datalink.c | debug.c | sdv-harness.c | fdownmi.c
| ieee1284.c | pdopenp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

116:     return PptPdoPower( DevObj, Irp );
117: } else {
118:     return PptPdoPower( DevObj, Irp );
119: }
120: }
121: □
122: NTSTATUS
123: PptDispatchCreateOpen( PDEVICE_OBJECT DevObj, PIRP Irp ) {
124:    PFDO_EXTENSION fdx = DevObj->DeviceExtension;
125:    P5TraceIrpArrival( DevObj, Irp );
126:    if( DevTypeFdo == fdx->DevType ) {
127:        return PptPdoCreateOpen( DevObj, Irp );
128:    } else {
129:        return PptPdoCreateOpen( DevObj, Irp );
130:    }
131: }
132: □
133: NTSTATUS
134: PptDispatchClose( PDEVICE_OBJECT DevObj, PIRP Irp ) {
135:     PFDO_EXTENSION fdx = DevObj->DeviceExtension;
136:     P5TraceIrpArrival( DevObj, Irp );
137:    if( DevTypeFdo == fdx->DevType ) {
138:        return PptPdoClose( DevObj, Irp );
139:    } else {
140:        return PptPdoClose( DevObj, Irp );
141:    }
142: }
143: □
144: NTSTATUS
145: PptDispatchCleanup( PDEVICE_OBJECT DevObj, PIRP Irp ) {
if (DevTypeFdo == fdx->DevType) {
    return PptPdoCreateOpen( DevObj, Irp );
} else {
    return PptPdoCreateOpen( DevObj, Irp );
}

if (DevTypeFdo == fdx->DevType) {
    return PptPdoClose( DevObj, Irp );
} else {
    return PptPdoClose( DevObj, Irp );
}
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

- 116:     return PptPdoPower( DevObj, Irp );
- 117:     } else {
- 118:         return PptPdoPower( DevObj, Irp );
- 119:     }
- 120: }
- 121: }
- 122: NTSTATUS
- 123: PptDispatchCreateOpen( PDEVICE_OBJECT DevObj, PIRP Irp ) {
- 124:     PFDO_EXTENSION fdx = DevObj->DeviceExtension;
- 125:     PSTraceIrpArrival( DevObj, Irp );
- 126:     if( DevTypeFdo == fdx->DevType ) {
- 127:         return PptPdoCreateOpen( DevObj, Irp );
- 128:     } else {
- 129:         return PptPdoCreateOpen( DevObj, Irp );
- 130:     }
- 131: }
- 132: }
- 133: NTSTATUS
- 134: PptDispatchClose( PDEVICE_OBJECT DevObj, PIRP Irp ) {
- 135:     PFDO_EXTENSION fdx = DevObj->DeviceExtension;
- 136:     PSTraceIrpArrival( DevObj, Irp );
- 137:     if( DevTypeFdo == fdx->DevType ) {
- 138:         return PptPdoClose( DevObj, Irp );
- 139:     } else {
- 140:         return PptPdoClose( DevObj, Irp );
- 141:     }
- 142: }
- 143: }
- 144: NTSTATUS
- 145: PptDispatchCleanup( PDEVICE_OBJECT DevObj, PIRP Irp ) {

File: ../../../dispatchredirect.c, Line: 138, Function 'PptDispatchClose'

Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.
```c
#include "pch.h"

NTSTATUS PptFdoClose(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
)
{
    PFDO_EXTENSION fdx = DeviceObject->DeviceExtension;

    NTSTATUS status;

    PAGED_CODE();

    // Verify that our device has not been SURPRISE_REMOVED. Generally
    // only parallel ports on hot-plug busses (e.g., PCMCIA) and
    // parallel ports in docking stations will be surprise removed.
    //
    if (fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED ) {
        //
        // Our device has been SURPRISE removed, but since this is only
        // a CLOSE, SUCCEED anyway.
        //
        status = F4CompleteRequest( Irp, STATUS_SUCCESS, 0 );
    }
    goto target_exit;
}
```

File: ..../../../../../fdoclose.c, Line: 9, Function 'PptFdoClose'

Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.
```c
#include "pch.h"

NTSTATUS PptFdoClose()
{
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
}
{
    PFDO_EXTENSION fdx = DeviceObject->DeviceExtension;
    NTSTATUS status;

    PAGED_CODE();

    // Verify that our device has not been SUPRISE_REMOVED. Generally
    // only parallel ports on hot-plug busses (e.g., PCMCIA) and
    // parallel ports in docking stations will be surprise removed.
    //
    if ( fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED ) {
        //
        // Our device has been SURPRISE removed, but since this is only
        // a CLOSE, SUCCEEDED anyway.
        //
        status = F4CompleteRequest( Irp, STATUS_SUCCESS, 0 );

        goto target_exit;
    }
```
#include "pch.h"

NTSTATUS
PptFdoClose(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
)
{
    FFDO_EXTENSION fdx = DeviceObject->DeviceExtension;
    NTSTATUS status;
    PAGED_CODE();

    // Verify that our device has not been SUPRISE_REMOVED. Generally only parallel ports on hot-plug busses (e.g., PCMCIA) and parallel ports in docking stations will be surprise removed.

    // Our device has been SURPRISE removed, but since this is only a CLOSE, SUCCEED anyway.
    status = F4CompleteRequest( Irp, STATUS_SUCCESS, 0 );
    goto target_exit;
}
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowm.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclse.c | utils.c

2:
3: NTSTATUS
4: PptFdoClose{
5:    IN PDEVICE_OBJECT DeviceObject,
6:    IN IRP Irp
7: }
8: 
9: FFDO_EXTENSION fdx = DeviceObject->DeviceExtension;
10: NTSTATUS status;
11: 
12: PAGED_CODE();
13: 
14: //
15: // Verify that our device has not been SUPRISE_REMOVED. Generally
16: // only parallel ports on hot-plug busses (e.g., PCMCIA) and
17: // parallel ports in docking stations will be surprise removed.
18: //
19: if( fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED ) {
20: //
21: // Our device has been SURPRISE removed, but since this is only
22: // a CLOSE, SUCCEED anyway.
23: //
24: status = P4CompleteRequest( Irp, STATUS_SUCCESS, 0 );
25: 
26: goto target_exit;
27: }
28: 
29: //
30: //

File: ../../../../../../../../fdoclse.c, Line: 24, Function 'PptFdoClose'

Driver: src\kernel\parport  Rule: DoubleCompletion  Defect: The driver is calling IoCompleteRequest twice.
DoubleCompletion.slic parallel.h pdopnp.c datalink.c debug.c sdv-harness.c fdowmi.c
ieee1284.c fdopnp.c wdmguid.h ntddpar.h parport.c dispatchredirect.c fdoclose.c utils.c

1775: P4CompleteRequest{
1776:     IN PIRP Irp,
1777:     IN NTSTATUS Status,
1778:     IN ULONG_PTR Information
1779: }
1780: {
1781:     P5TraceIrpCompletion( Irp );
1782:     Irp->IoStatus.Status = Status;
1783:     Irp->IoStatus.Information = Information;
1784:     IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:     return Status;
1786: }
1787: }
1788: }
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791:     IN PIRP Irp,
1792:     IN NTSTATUS Status,
1793:     IN ULONG_PTR Information,
1794:     IN PIO_REMOVE_LOCK RemLock
1795: }
1796: {
1797:     P4CompleteRequest( Irp, Status, Information );
1798:     PptReleaseRemoveLock( RemLock, Irp );
1799:     return Status;
1800: }
1801: 
1802: 1803: // pcutil.c follows:

File: ../../../../util.c, Line: 1782, Function 'P4CompleteRequest'

Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.
1775: P4CompleteRequest{
1776:     IN PIRP    Irp,
1777:     IN NTSTATUS Status,
1778:     IN ULONG_PTR Information
1779: }
1780: {
1781:     E5TraceIrpCompletion( Irp );
1782:     Irp->IoStatus.Status = Status;
1783:     Irp->IoStatus.Information = Information;
1784:     IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:     return Status;
1786: }
1787: }
1788: }
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791:     IN PIRP    Irp,
1792:     IN NTSTATUS Status,
1793:     IN ULONG_PTR Information,
1794:     IN PIO_REMOVE_LOCK RemLock
1795: )
1796: {
1797:     P4CompleteRequest( Irp, Status, Information );
1798:     PptReleaseRemoveLock( RemLock, Irp );
1799:     return Status;
1800: }
1801: 
1802: // pcutil.c follows:
Call #1

```c
1775: F4CompleteRequest{
1776:     IN PIRP Irp,
1777:     IN NTSTATUS Status,
1778:     IN ULONG_PTR Information
1779: }
1780: {
1781:     P5TraceIrpCompletion( Irp );
1782:     Irp->IoStatus.Status = Status;
1783:     Irp->IoStatus.Information = Information;
1784:     IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:     return Status;
1786: }
1787:
1788: 
1789: NTSTATUS
1790: F4CompleteRequestReleaseRemLock(
1791:     IN PIRP Irp,
1792:     IN NTSTATUS Status,
1793:     IN ULONG_PTR Information,
1794:     IN PIO_REMOVE RemLock,
1795: }
1796: {
1797:     F4CompleteRequestReleaseRemLock( Information );
1798:     PptReleaseRemoveLock( RemLock, Irp );
1799:     return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
```
DoubleCompletion.slic| parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c |
| ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

1775: P4CompleteRequest{
1776:  IN PIRP Irp,
1777:  IN NTSTATUS Status,
1778:  IN ULONG_PTR Information
1779: }
1780: {
1781:  P5TraceIrpCompletion( Irp );
1782:  Irp->IoStatus.Status = Status;
1783:  Irp->IoStatus.Information = Information;
1784:  IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:  return Status;
1786: }
1787: }
1788: };
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791:  IN PIRP Irp,
1792:  IN NTSTATUS Status,
1793:  IN ULONG_PTR Information,
1794:  IN PIO_REMOVE_LOCK RemLock
1795: }
1796: {
1797:  P4CompleteRequest( Irp, Status, Information );
1798:  PptReleaseRemoveLock( RemLock, Irp );
1799:  return Status;
1800: }
1801: 
1802: // pcutil.c follows:
1803: // pcutil.c follows:

Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.
24: status = P4CompleteRequest( Irp, STATUS_SUCCESS, 0 );
25:
26: goto target_exit;
27:
28:
29:
30: //
31: // Try to acquire RemoveLock to prevent the device object from going
32: // away while we're using it.
33: //
34: status = PptAcquireRemoveLock(&fdx->RemoveLock, Irp);
35: if( !NT_SUCCESS( status ) ) {
36: // Our device has been removed, but since this is only a CLOSE, SUCCEED anyway.
37: status = STATUS_SUCCESS;
38: goto target_exit;
39: }
40:
41: //
42: // We have the RemoveLock
43: //
44:
45: ExAcquireFastMutex(&fdx->OpenCloseMutex);
46: if( fdx->OpenCloseRefCount > 0 ) {
47: //
48: // prevent rollover - strange as it may seem, it is perfectly
49: // legal for us to receive more closes than creates - this
50: // info came directly from Mr. PnP himself
51: //
52: if( ((LONG)InterlockedDecrement(&fdx->OpenCloseRefCount)) < 0 ) {
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

38:             goto target_exit;
39: }
40: }
41: //
42: // We have the RemoveLock
43: //
44: ExAcquireFastMutex(&fdx->OpenCloseMutex);
45: if( fdx->OpenCloseRefCount > 0 ) {
46:     //
47:     // prevent rollover - strange as it may seem, it is perfectly
48:     // legal for us to receive more closes than creates - this
49:     // info came directly from Mr. PnP himself
50:     //
51:     if( ((LONG)InterlockedDecrement(&fdx->OpenCloseRefCount) < 0 ) {  
52:         // handle underflow
53:         InterlockedIncrement(&fdx->OpenCloseRefCount);
54:     }
55: }
56: ExReleaseFastMutex(&fdx->OpenCloseMutex);
57: }
58: target_exit:
59: DD((PCE)fdx,DDT,"PptFdoClose - OpenCloseRefCount after close = %d\n",fdx->OpenCloseRe
60: 63: return P4CompleteRequestReleaseRemLock( Irp, STATUS_SUCCESS, 0, &fdx->RemoveLock );
64: }
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c |
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

1775: P4CompleteRequest{
1776:     IN PIRP Irp,
1777:     IN NTSTATUS Status,
1778:     IN ULONG_PTR Information
1779: }
1780: {
1781:     P5TraceIrpCompletion( Irp );
1782:     Irp->IoStatus.Status = Status;
1783:     Irp->IoStatus.Information = Information;
1784:     IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:     return Status;
1786: }
1787: }
1788: 
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791:     IN PIRP Irp,
1792:     IN NTSTATUS Status,
1793:     IN ULONG_PTR Information,
1794:     IN PIO_REMOVE_LOCK RemLock
1795: }
1796: {
1797:     P4CompleteRequest( Irp, Status, Information );
1798:     PptReleaseRemoveLock( RemLock, Irp );
1799:     return Status;
1800: }
1801: 
1802: // pcutil.c follows:

File: ..//..//..//..//utils.c, Line: 1797, Function 'P4CompleteRequestReleaseRemLock'

Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.
DoubleCompletion.slic|parallel.h|pdopnp.c|datalink.c|debug.c|sdv-harness.c|fd0wmi.c
ieee1284.c|fdopnp.c|wdmguid.h|ntddpar.h|parport.c|dispatchredirect.c|fd0close.c|utils.c

1775: P4CompleteRequest{
1776:    IN PIRP Irp,
1777:    IN NTSTATUS Status,
1778:    IN ULONG_PTR Information
1779: }
1780: {
1781:    P5TraceIrpCompletion( Irp );
1782:    Irp->IoStatus.Status = Status;
1783:    Irp->IoStatus.Information = Information;
1784:    IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:    return Status;
1786: }
1787:
1788: }
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791:    IN PIRP Irp,
1792:    IN NTSTATUS Status,
1793:    IN ULONG_PTR Information,
1794:    IN PIO_REMOVE_LOCK RemLock
1795: }
1796: {
1797:    P4CompleteRequest( Irp, Status, Information );
1798:    PptReleaseRemoveLock( RemLock, Irp );
1799:    return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:

File: ../../..../..../utils.c, Line: 1782, Function 'P4CompleteRequest'

Driver: src\kernel\parport   Rule: DoubleCompletion   Defect: The driver is calling IoCompleteRequest twice.
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
| ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

1775: P4CompleteRequest{
1776:     IN PIRP    Irp,
1777:     IN NTSTATUS Status,
1778:     IN ULONG_PTR Information
1779: }
1780: {
1781:     E5TraceIrpCompletion( Irp );
1782:     Irp->IoStatus.Status = Status;
1783:     Irp->IoStatus.Information = Information;
1784:     IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:     return Status;
1786: }
1787: }
1788: 
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791:     IN PIRP    Irp,
1792:     IN NTSTATUS Status,
1793:     IN ULONG_PTR Information,
1794:     IN PIO_REMOVE_LOCK RemLock
1795: )
1796: {
1797:     P4CompleteRequest( Irp, Status, Information );
1798:     PptReleaseRemoveLock( RemLock, Irp );
1799:     return Status;
1800: }
1801: 
1802: // pcutil.c follows:

File: ../../../../utils.c, Line: 1783, Function 'P4CompleteRequest'

Driver: src\kernel\parport  Rule: DoubleCompletion  Defect: The driver is calling IoCompleteRequest twice.
DoubleCompletion(slic|parallel.h|pdopnp.c|datalink.c|debug.c|sdv-harness.c|fdowmi.c|ieee1284.c|fdoopnp.c|wmguid.h|ntddpar.h|parport.c|dispatchredirect.c|fdoclose.c|utils.c)

1775: F4CompleteRequest(
1776:     IN PIRP       Irp,
1777:     IN NTSTATUS  Status,
1778:     IN ULONG_PTR Information
1779: );
1780: {
1781:     P5TraceIrpCompletion( Irp );
1782:     Irp->IoStatus.Status = Status;
1783:     Irp->IoStatus.Information = Information;
1784:     IoCompleteRequest( Irp, IO_NO_INCREMENT );
1785:     return Status;
1786: }
1787:
1788:  }
1789: NTSTATUS
1790: F4CompleteRequestReleaseRemLock(
1791:     IN PIRP       Irp,
1792:     IN NTSTATUS  Status,
1793:     IN ULONG_PTR Information,
1794:     IN PIRP       RemLock
1795: );
1796: {
1797:     F4CompleteRequestReleaseRemLock( RemLock, Irp );
1798:     return Status;
1799: }
1800: }
1801:
1802:
1803: // pcutil.c follows:

Driver: src\kernel\parport  Rule: DoubleCompletion  Defect: The driver is calling IoCompleteRequest twice.

File: ../.../../..//utils.c, Line: 1784, Function 'F4CompleteRequest'
Further reading

Reading assignment

• David Hovemeyer, Jaime Spacco, William Pugh: Evaluating and tuning a static analysis to find null pointer bugs. *PASTE 2005*: 13-19