Pallas: Semantic-Aware Checking for Finding Deep Bugs in Fast Path

Jian Huang†  Michael Allen-Bond  Xuechen Zhang

†Georgia Institute of Technology  Washington State University
What is Fast Path?
What is Fast Path?

Slow path

Fast path

Allocate pages

Order-0 allocation?

Yes

No

Lockless fast path
What is Fast Path?

Fast path is derived from slow path or vice versa

They share the start and end entries in the workflow
Fast Path is Everywhere

Operating system

Memory allocation
Fast Path is Everywhere

- Operating system
- Web browser
- Memory allocation
- Web page loading
Fast Path is Everywhere

- Operating system
- Web browser
- Mobile OS

- Memory allocation
- Web page loading
- File system inode search
Fast Path is Everywhere

- Operating system
- Web browser
- Mobile OS
- Software-defined network

- Memory allocation
- Web page loading
- File system inode search
- Packet forwarding
Fast Path Introduces Semantic Bugs

<table>
<thead>
<tr>
<th></th>
<th>Memory manager</th>
<th>File system</th>
<th>Network</th>
<th>Device driver</th>
</tr>
</thead>
<tbody>
<tr>
<td># of fast paths</td>
<td>16</td>
<td>21</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td># of bugs per path</td>
<td>19</td>
<td>17</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Fix time (days)</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>
## Fast Path Introduces Semantic Bugs

<table>
<thead>
<tr>
<th></th>
<th>Memory manager</th>
<th>File system</th>
<th>Network</th>
<th>Device driver</th>
</tr>
</thead>
<tbody>
<tr>
<td># of fast paths</td>
<td>16</td>
<td>21</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td># of bugs per path</td>
<td>19</td>
<td>17</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Fix time (days)</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td><strong>12</strong></td>
</tr>
</tbody>
</table>

Fast-path bugs are related to software semantics and these bugs are hard to detect.
Detecting Fast-Path Bugs is Challenging

Program Verification

seL4[SOSP’09], Ironclad[OSDI’14], etc
But systems may not have verification frameworks
Detecting Fast-Path Bugs is Challenging

Program Verification
- seL4[SOSP’09], Ironclad[OSDI’14], etc
  But systems may not have verification frameworks

Model Checker
- SAMC[OSDI’14], FiSC[OSDI’04], etc.
  But they require models for specific systems
Detecting Fast-Path Bugs is Challenging

**Program Verification**
- seL4[SOSP’09], Ironclad[OSDI’14], etc
  - But systems may not have verification frameworks

**Model Checker**
- SAMC[OSDI’14], FiSC[OSDI’04], etc.
  - But they require models for specific systems

Can we use static analysis to detect fast-path bugs?
Our Study on Fast-Path Bugs in Linux

- Sampled 65 fast paths and 172 relevant patches
- Committed from 2009-2015
Fast-Path Bug Categorization

```
struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
}
```

Path state

Trigger condition

Fault handling

Path output

Assistant data structure

---

Flowchart:

- **S_in** → **S_1**
- **S_2** with conditions:
  - **No** → **S_3**
  - **Yes** → **Fast path**
- **S_4**
- **S_out**
- **S_fault**

Trigger conditions:

- **S_1**
- **S_2**
- **S_3**
- **S_4**
Fast-Path Bug Categorization

\[\text{struct inode}\{\]
  \hspace{1em} \text{kdev_t i_dev};
  \hspace{1em} \text{unsigned long i_ino};
  \hspace{1em} \ldots
\]

Path state

Trigger condition

Fault handling

Path output

Assistant data structure

Yes

No

Fast path

Slow path
Fast-Path Bug Categorization

```
struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
}
```

Path state
Trigger condition
Path output
Fault handling
Assistant data structure

Trigger condition
[Yes, No]

Fast path

Slow path

S_in

S_1

S_2

S_3

S_out

S_fault

S_4
Fast-Path Bug Categorization

```
struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...}
```

**Path state**

**Trigger condition**

**Path output**

**Fault handling**

**Assistant data structure**
Fast-Path Bug Categorization

Trigger condition

No

Yes

Path output

Fault handling

Assistant data structure

struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
}

Fast path

Slow path

Path state

Trigger condition
Fast-Path Bug Categorization

```
struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
}
```

Flowchart:
- **S_in**
- **S1**
- **S2**
- **S3**
- **S4**
- **S_out**
- **S_fault**

- **Trigger condition**
  - **No**
  - **Yes**

- **Path state**
- **Trigger condition**
- **Path output**
- **Fault handling**
- **Assistant data structure**

- **Fast path**
How does Path State Cause Bugs?

Semantics

Immutable variables
X
How does Path State Cause Bugs?

Semantics

Immutable variables $X$

Patterns

Overwriting immutable variables
How does Path State Cause Bugs?

Immutable variables

Overwriting immutable variables

+ if (order == 0) {
+   ...
+   gfp_mask = memalloc_noio_flags(gfp_mask);
+   ...
+ } else {

Lockless fast path

$S_{\text{in}}$  Order=0?  Yes  $S_{\text{output}}$

$S$  No  $S$

Patterns

Semantics

Immutable variables $X$
How does Path State Cause Bugs?

Semantics

Immutable variables $X$

Patterns

Overwriting immutable variables

+ if (order == 0) {
+   ...
+   gfp_mask = memalloc_noio_flags(gfp_mask);
+   ...
+ } else {

Incorrect page allocation!
How does Path State Cause Bugs?

**Semantics**

Immutable variables X

**Patterns**

Overwriting immutable variables

**Rules**

X should not be overwritten

```
+ if (order == 0) {
+   ...
+   gfp_mask = memalloc_noio_flags(gfp_mask);
+   ...
+ } else {
```

Incorrect page allocation!
How does Path State Cause Bugs?

Semantics

Immutable variables $X$

Patterns

Overwriting immutable variables

Rules

$X$ should not be overwritten

Uninitialized immutable variables

$X$ should be initialized
How does Path State Cause Bugs?

Semantics

- Immutable variables $X$

Patterns

- Overwriting immutable variables
- Uninitialized immutable variables
- Incomplete implementation

Rules

- $X$ should not be overwritten
- $X$ should be initialized
- The correlation of $Y$ and $Z$ should be detected
Fast-Path Bug Categorization

struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
};
How does Trigger Condition Cause Bugs?

Semantics

Trigger conditions
\{X\}
How does Trigger Condition Cause Bugs?

Semantics

Trigger conditions $\{X\}$

Patterns

Missing trigger condition checking
How does Trigger Condition Cause Bugs?

Semantics

Trigger conditions \{X\}

Patterns

Missing trigger condition checking

+ if (order == 0) {
+    ...
+    page = list_last_entry(list, struct page, lru);
+    ...
+ } else {

Lockless fast path

Yes

\(S_{in} \rightarrow S_{order=0?}\) Yes

\(S_{order=0?} \rightarrow S\) No

\(S \rightarrow S_{output}\)
How does Trigger Condition Cause Bugs?

Semantics

Trigger conditions \( \{X\} \)

Patterns

Missing trigger condition checking

\[
\begin{align*}
  + & \text{ if } (\text{order} == 0) \{ \\
  + & \quad \text{...} \\
  + & \quad \text{page} = \text{list_last_entry(list, struct page, lru);} \\
  + & \quad \text{...} \\
  + & \} \text{ else } \\
\end{align*}
\]

Lockless fast path

\[
\text{S}_\text{in} \rightarrow \text{Order=0?} \rightarrow \text{S} \rightarrow \text{S}_\text{output}
\]

Performance degradation!
How does Trigger Condition Cause Bugs?

**Semantics**

**Trigger conditions** \( \{X\} \)

**Patterns**

**Missing trigger condition checking**

**Rules**

**X should be detected in the path**

```c
+ if (order == 0) {
+   ...
+   page = list_last_entry(list, struct page, lru);
+   ...
+ } else {
```

- **Lockless fast path**

  - Performance degradation!

- **S**

  - **S_{in}**

  - **Order=0?**

  - **S**

  - **S_{output}**
How does Trigger Condition Cause Bugs?

**Semantics**

- Trigger conditions \( \{X\} \)

**Patterns**

- Missing trigger condition checking
- Incomplete implementation

**Rules**

- \( X \) should be detected in the path
- Any \( X \) in \( \{X\} \) should be detected
# How does Trigger Condition Cause Bugs?

## Semantics

| Trigger conditions \{X\} |

## Patterns

<table>
<thead>
<tr>
<th>Missing trigger condition checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete implementation</td>
</tr>
<tr>
<td>Incorrect order of condition checking</td>
</tr>
</tbody>
</table>

## Rules

<table>
<thead>
<tr>
<th>X should be detected in the path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any X in {X} should be detected</td>
</tr>
<tr>
<td>Order “X_i before X_j” should be detected</td>
</tr>
</tbody>
</table>

*The order of condition checking *\(X_i \text{ happens before } X_j\)
Fast-Path Bug Categorization

Trigger condition

- $S_{in}$
- $S_1$
- $S_2$
- $S_3$
- $S_{out}$
- $S_4$
- $S_{fault}$

**Path state**:
- Fast path

**Trigger condition**: 
- $S_{in}$ to $S_1$
- $S_1$ to $S_2$
- $S_2$ to $S_3$
- $S_3$ to $S_{out}$
- $S_{out}$ to $S_4$
- $S_4$ to $S_{fault}$

**Path output**: 
- No to $S_2$
- Yes to $S_1$
- $S_1$ to $S_3$
- $S_2$ to $S_4$

**Fault handling**: 
- Fault handling at $S_4$

**Assistant data structure**: 
- $\text{struct inode}\{
  \text{kdev_t i_dev;}
  \text{unsigned long i_ino;}
  \text{...}
\}$
How does Path Output Cause Bugs?

Semantics

Path output

$R_{\text{fast}}$ and $R_{\text{slow}}$
How does Path Output Cause Bugs?

Semantics

Path output
$R_{fast}$ and $R_{slow}$

Patterns

Mismatching output
How does Path Output Cause Bugs?

Path output $R_{\text{fast}}$ and $R_{\text{slow}}$

Mismatching output

```java
+ if (order == 0) {
+    ...
+    SetPageDirty(page)
+    ...
+ } else {
+    //Set page in dirty state
+    SetPageDirty(page);
}
```

Semantics

Patterns

Lockless fast path

Yes

$S_{\text{in}}$ → Order=0? → $S$ → $S_{\text{output}}$

No
How does Path Output Cause Bugs?

Path output $R_{\text{fast}}$ and $R_{\text{slow}}$

Mismatching output

```
if (order == 0) {
    ...
    SetPageDirty(page)
    ...
} else {
    //Set page in dirty state
    SetPageDirty(page);
```

Wrong page state!

Semantics

Patterns

Path output $R_{\text{fast}}$ and $R_{\text{slow}}$
How does Path Output Cause Bugs?

Semantics

Path output $R_{fast}$ and $R_{slow}$

Patterns

Mismatching output

Rules

$R_{fast} = R_{slow}$

+ if (order == 0) {
+   ...
+   SetPageDirty(page)
+   ...
+ } else {
  //Set page in dirty state
  SetPageDirty(page);

Wrong page state!

Lockless fast path

$S_{in} \rightarrow \text{Order=0?} \rightarrow S \rightarrow S_{output}$
How does Path Output Cause Bugs?

Semantics

Path output
$R_{\text{fast}}$ and $R_{\text{slow}}$

Patterns

Mismatching output

Missing output checking

Rules

$R_{\text{fast}} = R_{\text{slow}}$

$R_{\text{fast}}$ and $R_{\text{slow}}$ should be checked
How does Path Output Cause Bugs?

Semantics
- Path output \( R_{\text{fast}} \) and \( R_{\text{slow}} \)
- Predefined output states RS

Patterns
- Mismatching output
- Missing output checking
- Unexpected output

Rules
- \( R_{\text{fast}} = R_{\text{slow}} \)
- \( R_{\text{fast}} \) and \( R_{\text{slow}} \) should be checked
- \( R_{\text{fast}} \) and \( R_{\text{slow}} \) in RS
Fast-Path Bug Categorization

struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
}

Path state
Trigger condition
Path output
Fault handling
Assistant data structure
How does Fault Handler Cause Bugs?

Semantics

Fault state $S$
How does Fault Handler Cause Bugs?

Semantics

Fault state $S$

Patterns

Missing fault handler
How does Fault Handler Cause Bugs?

Semantics
Fault state $S$

Patterns
Missing fault handler

```
+ if (order == 0) {
+     page = list_last_entry(list, struct page, lru);
+     if(!page) {
+         //Fault handler
+     }
+ } else {
```

Lockless fast path

- Yes
- No

$S_{\text{in}}$ $\rightarrow$ Order=0? $\rightarrow$ $S$ $\rightarrow$ $S_{\text{fault}}$ $\rightarrow$ $S_{\text{output}}$
How does Fault Handler Cause Bugs?

Fault state $S$

- Lockless fast path
- Semantic Patterns
- Missing fault handler

```
+ if (order == 0) {
  +   page = list_last_entry(list, struct page, lru);
  +   if(!page) {
  +     //Fault handler
  +   }
  + } else {
```
How does Fault Handler Cause Bugs?

Semantics
Fault state $S$

Patterns
Missing fault handler

Rules
$S$ should be detected in the path

```
+ if (order == 0) {
+     page = list_last_entry(list, struct page, lru);
+     if(!page) {
+         //Fault handler
+     }
+ } else {
```

Lockless fast path

$S_{in}$

Order=0?

$S$

$S_{output}$

$S_{fault}$

Allocation failure!
Fast-Path Bug Categorization

struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    ...
}

Path state
Trigger condition
Path output
Fault handling
Assistant data structure

Trigger condition

Fast path

Slow path
How does Assistant Data Structure Cause Bugs?

Semantics

Assistant data structure DS
How does Assistant Data Structure Cause Bugs?

Semantics

Assistant data structure $DS$

Patterns

Suboptimal organization of data structure
How does Assistant Data Structure Cause Bugs?

Assistant data structure DS

Suboptimal organization of data structure

```
struct inode{
    kdev_t i_dev;
    unsigned long i_ino;
    int i_cindex;
    ...
}
```

Never used in fast path
How does Assistant Data Structure Cause Bugs?

Assistant data structure *DS*

Suboptimal organization of data structure

Unused variables should be not in *DS*

```
struct inode{
    kdev_t  i_dev;
    unsigned long i_ino;
    int i_cindex;
    ...
}
```

Never used in fast path
How does Assistant Data Structure Cause Bugs?

**Semantics**

- Assistant data structure $DS$
- $DS$ for caching path states

**Patterns**

- Suboptimal organization of data structure
- Stale value caused by uncoordinated updates

**Rules**

- Unused variables should be not in $DS$
- A path state update should be followed by a cache update
Pallas: A Semantic-Aware Static Checking Tool

Bug-finding rules → Pallas
Pallas: A Semantic-Aware Static Checking Tool

Bug-finding rules → Pallas

?
How to Generate Path Information?

A fast-path patch

Path information

```c
int * foo_get_page(gfp_mask)
{
+   //lockless fast path
+   if (order == 0) {
+       gfp_mask = noio_flag(gfp_mask);
+   }
+   return SUCCESS;
}
```
How to Generate Path Information?

int *foo_get_page(gfp_mask)
{
+   //lockless fast path
+   if (order == 0) {
+     gfp_mask = noio_flag(gfp_mask);
+   }
+   return SUCCESS;
}

A fast-path patch

Path information
How to Generate Path Information?

A fast-path patch

Path information

```c
int *foo_get_page(gfp_mask) {
+   //lockless fast path
+   if (order == 0) {
+       gfp_mask = noio_flag(gfp_mask);
+   }
+   return SUCCESS;
}
```
int * foo_get_page(gfp_mask)  
{
+ // lockless fast path
+ if (order == 0) {
+     gfp_mask = noio_flag(gfp_mask);
+ }
+ }
+return SUCCESS;
}

A fast-path patch

Signature  foo_get_page(gfp_mask)

Path information
How to Generate Path Information?

A fast-path patch

Path information

<table>
<thead>
<tr>
<th>Signature</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo_get_page(gfp_mask)</td>
<td>order == 0</td>
</tr>
</tbody>
</table>

```c
int * foo_get_page(gfp_mask) {
    // lockless fast path
    if (order == 0) {
        gfp_mask = noio_flag(gfp_mask);
    }
    return SUCCESS;
}
```
How to Generate Path Information?

```c
int * foo_get_page(gfp_mask)
{
    // lockless fast path
    if (order == 0) {
        gfp_mask = noio_flag(gfp_mask);
    }
    return SUCCESS;
}
```

<table>
<thead>
<tr>
<th>Signature</th>
<th>foo_get_page(gfp_mask)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>order == 0</td>
</tr>
</tbody>
</table>

A fast-path patch

Path information
int * foo_get_page(gfp_mask)
{
    // lockless fast path
    if (order == 0) {
        gfp_mask = noio_flag(gfp_mask);
    }
    return SUCCESS;
}

---

A fast-path patch

Path information

<table>
<thead>
<tr>
<th>Signature</th>
<th>foo_get_page(gfp_mask)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>order == 0</td>
</tr>
<tr>
<td>State</td>
<td>gfp_mask=noio_flag(gfp_mask)</td>
</tr>
</tbody>
</table>
How to Generate Path Information?

A fast-path patch

int * foo_get_page(gfp_mask)
{
+  //lockless fast path
+  if (order == 0) {
+    gfp_mask = noio_flag(gfp_mask);
+  }
+  return SUCCESS;
}

<table>
<thead>
<tr>
<th>Signature</th>
<th>foo_get_page(gfp_mask)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>order == 0</td>
</tr>
<tr>
<td>State</td>
<td>gfp_mask=noio_flag(gfp_mask)</td>
</tr>
</tbody>
</table>

Path information
### How to Generate Path Information?

```c
int * foo_get_page(gfp_mask)
{
    // lockless fast path
    if (order == 0) {
        gfp_mask = noio_flag(gfp_mask);
    }
    return SUCCESS;
}
```

<table>
<thead>
<tr>
<th>Signature</th>
<th>foo_get_page(gfp_mask)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>order == 0</td>
</tr>
<tr>
<td>State</td>
<td>gfp_mask=noio_flag(gfp_mask)</td>
</tr>
<tr>
<td>Output</td>
<td>SUCCESS</td>
</tr>
</tbody>
</table>

A fast-path patch

Path information
# How does Pallas Check Paths?

## Semantic input

<table>
<thead>
<tr>
<th>Immutable variable:</th>
<th>Condition variable:</th>
<th>Output variable:</th>
</tr>
</thead>
<tbody>
<tr>
<td>gfp_mask</td>
<td>order</td>
<td>SUCCESS</td>
</tr>
</tbody>
</table>

## Path information

<table>
<thead>
<tr>
<th>Signature</th>
<th>foo_get_page(gfp_mask)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>order == 0</td>
</tr>
<tr>
<td>Output</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>State</td>
<td>gfp_mask=noio_flag(gfp_mask)</td>
</tr>
</tbody>
</table>
# How does Pallas Check Paths?

## Semantic input

<table>
<thead>
<tr>
<th>Immutable variable:</th>
<th>Condition variable:</th>
<th>Output variable:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gfp_mask</code></td>
<td><code>order</code></td>
<td><code>SUCCESS</code></td>
</tr>
</tbody>
</table>

## Path information

<table>
<thead>
<tr>
<th>Signature</th>
<th>Condition</th>
<th>Output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>foo_get_page(gfp_mask)</code></td>
<td><code>order == 0</code></td>
<td><code>SUCCESS</code></td>
<td><code>gfp_mask=noio_flag(gfp_mask)</code></td>
</tr>
</tbody>
</table>

Rule violation: “`gfp_mask`” should not be overwritten.
How does Pallas Check Paths?

Semantic input

<table>
<thead>
<tr>
<th>Immutable variable:</th>
<th>Condition variable:</th>
<th>Output variable:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gfp_mask</code></td>
<td><code>order</code></td>
<td><code>SUCCESS</code></td>
</tr>
</tbody>
</table>

Path information

<table>
<thead>
<tr>
<th>Signature</th>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>foo_get_page(gfp_mask)</code></td>
<td><code>order == 0</code></td>
<td><code>SUCCESS</code></td>
</tr>
<tr>
<td>State</td>
<td><code>gfp_mask=noio_flag(gfp_mask)</code></td>
<td></td>
</tr>
</tbody>
</table>

Rule violation: “`gfp_mask`” should not be overwritten
Pallas Implementation

- Path state checker
- Trigger condition checker
- Path output checker
- Fault handling checker
- Assistant data structure checker

Path database
Evaluation with Real-World Systems

- Linux
- Chromium
- Android
- Open vSwitch
Evaluation with Real-World Systems

<table>
<thead>
<tr>
<th>Checkers</th>
<th>Bugs</th>
<th>False positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path state</td>
<td>29</td>
<td>18</td>
</tr>
<tr>
<td>Trigger condition</td>
<td>41</td>
<td>13</td>
</tr>
<tr>
<td>Path output</td>
<td>35</td>
<td>16</td>
</tr>
<tr>
<td>Fault handling</td>
<td>27</td>
<td>10</td>
</tr>
<tr>
<td>Assistant DS</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>155</strong></td>
<td><strong>69</strong></td>
</tr>
</tbody>
</table>
The bugs already exist for 3.1 years on average
An Example of New Bugs in Chromium

Void PPBNaClPrivate::DownloadNexe(...) 
{
    // Try the fast path for retrieving the file
    PP_FileHandle handle = OpenNaClExecutable(instance, url, &out_file_info)
    if(handle != PP_kInvalidFileHandle) {
        DownloadNexeCompletion(request, out_file_info, FileDownloader::SUCCESS);
        return;
    }

    // The fast path didn’t work, try slow path
    ...
}
An Example of New Bugs in Chromium

```cpp
Void PPBNaClPrivate::DownloadNexe(...) {
  // Try the fast path for retrieving the file
  PP_FileHandle handle = OpenNaClExecutable(instance, url, &out_file_info);
  if(handle != PP_kInvalidFileHandle) {
    DownloadNexeCompletion(request, out_file_info, FileDownloader::SUCCESS);
    return;
  }

  // The fast path didn’t work, try slow path
  ...
}
```

No fault handling when the function fails
Conclusions

• Fast path introduces semantic bugs

• General bug patterns exist in fast-path bugs

• Pallas: a semantic-aware static checking tool

*We will release bug database soon
Thanks!

Jian Huang†
jian.huang@gatech.edu

Michael Allen-Bond

Xuechen Zhang

Q&A