Rebootless kernel updates

Jeff Arnold
jbarnewold@ksplice.com

http://www.ksplice.com
What is Ksplice?
What is Ksplice?

Running kernel with bug
What is Ksplice?

Running kernel with bug

Ksplice

Running kernel without bug

Update the kernel without disruption
What is Ksplice?

Update the kernel without disruption
Why should you care?
Why should you care?

• Get the benefits of patching...
  • Security improvements
  • Reliability improvements

• ... without the disruption of rebooting
Why should you care?

• Get the benefits of patching...
  • Security improvements
  • Reliability improvements

• ... without the disruption of rebooting

• Debugging/instrumentation
Why is avoiding reboots important?
Why is avoiding reboots important?

- Downtime
Why is avoiding reboots important?

- Downtime
- Lose software state
Why is avoiding reboots important?

- Downtime
- Lose software state
- Reboots can cause unexpected problems
Why is patching important?
Why is patching important?

- 70-140 bug-fixes per month
Why is patching important?

- 70-140 bug-fixes per month
- New privilege escalation CVE every month
Why is patching important?

- 70-140 bug-fixes per month
- New privilege escalation CVE every month
- > 90% of attacks exploit known vulnerabilities
Features
Features

• Any kernel since 2.6.8
Features

- Any kernel since 2.6.8
- Modules and assembly code
Features

• Any kernel since 2.6.8
• Modules and assembly code
• Negligible performance impact
Features

• Any kernel since 2.6.8
• Modules and assembly code
• Negligible performance impact
• So far: x86-32, x86-64, ARM
What is its status?
What is its status?

- Initial release: April 2008 (GPLv2)
What is its status?

• Initial release: April 2008 (GPLv2)
• Production use at MIT for 1 year
What is its status?

- Initial release: April 2008 (GPLv2)
- Production use at MIT for 1 year
- Tools in Debian sid, Ubuntu Jaunty, Fedora 8-10
What is its status?

• Initial release: April 2008 (GPLv2)
• Production use at MIT for 1 year
• Tools in Debian sid, Ubuntu Jaunty, Fedora 8-10
• Proposed for mainline
What is its status?

- Initial release: April 2008 (GPLv2)
- Production use at MIT for 1 year
- Tools in Debian sid, Ubuntu Jaunty, Fedora 8-10
- Proposed for mainline
- 5 engineers working on Ksplice full-time
CVE-2008-0600

fs/splice.c:

    if (unlikely(!len))
        break;
    error = -EFAULT;
- if (unlikely(!base))
+ if (!access_ok(VERIFY_READ, base, len))
    break;

/*
$ ksplice-create --patch=splice ~/src
$ ksplice-create --patch=splice ~/src
Update written to ksplice-8c4.tar.gz
$ ksplice-create --patch=splice ~/src
Update written to ksplice-8c4.tar.gz

user then becomes the superuser
$ ksplice-create --patch=splice ~/src
Update written to ksplice-8c4.tar.gz

user then becomes the superuser

# ksplice-apply ./ksplice-8c4.tar.gz
Done!

#
The Challenge

patch:
- if(aa) {bb}
+ if(cc) {dd}

Kernel

457f46
4c0102
000100
000002
00e300
The rest of this talk

• How Ksplice works
• Evaluation: 2005-2008 CVEs
• Using Ksplice for debugging
• Demos
  • Protecting against x86 exploit
  • Debugging on Android
• Future plans
Design Outline
Design Outline

- Identify which functions are modified by the source code patch
Design Outline

• Identify which functions are modified by the source code patch

• Generate a “replacement function” for every to-be-replaced function
Design Outline

• Identify which functions are modified by the source code patch

• Generate a “replacement function” for every to-be-replaced function

• Start redirecting execution to the replacement functions
pre-post differencing

pre source

post source
pre-post differencing

pre source

\[\text{gcc} \rightarrow \text{compiled pre}\]

post source

\[\text{gcc} \rightarrow \text{compiled post}\]
pre-post differencing

pre source

\[\text{gcc}\]

compiled pre

Find what differs

compiled post

post source

\[\text{gcc}\]
pre-post differencing

pre source

\[
\text{gcc}
\]

compiled pre

Find what differs

post source

\[
\text{gcc}
\]

compiled post

replacement functions
Redirect execution

Replacement function

foo'

55e8f0000001

Kernel

foo
Redirect execution

Replacement function

Kernel

foo'

55e8f0000001

foo

jmp
Handling symbolic references

Symbol table not sufficient
Matching pre code to running kernel
Matching pre code to running kernel

- Byte-by-byte comparison
Matching pre code to running kernel

- Byte-by-byte comparison
- When pre code refers to symbol, discover symbol value based on running kernel
Matching pre code to running kernel

- Byte-by-byte comparison
- When pre code refers to symbol, discover symbol value based on running kernel
- Discovered symbol values used to resolve symbols in replacement functions
replacement foo:
...
[bar]
...
...
replacement foo':
...
[bar]
...
...

Any pre function X from same scope:
...
[bar]
...
...
replacement foo':
...
[bar]
...

Any pre function X from same scope:
...
[bar]
...

Kernel's running code:
replacement foo':
  ...
  [bar]
  ...

Any pre function X from same scope:
  ...
  [bar]
  ...

Kernel's running code:

[addr f00000000]
function X:
  ...
  00 11 11 00
  ...
  ...
replacement foo':
...
[bar]
...

Any pre function X from same scope:
...
[bar]
...

Kernel's running code:

[addr f00000000]
function X:
...
00 11 11 00
...

bar = 00111100 + f00000002 - (-4)
replacement foo':
...
[bar]
...

Any pre function X from same scope:
...
[bar]
...

Kernel's running code:

```
[addr f00000000]
function X:
...
00 11 11 00
...
```

```
bar = 00111100 + f0000002 - (-4)
= f0111106
```
pre function X:
...

Kernel's running code:

[addr f00000000]
function X:
...

Also serves as extensive safety check
When to switch to new version

Kernel

foo

jmp

foo'

55e8f0000001

Should not while foo is running
When to switch to new version

Should not while foo is running

Kernel

foo'
55e8f0000001

foo
jmp
Safely redirect execution
Safely redirect execution

• Temporarily grab all CPUs
Safely redirect execution

- Temporarily grab all CPUs

- For every thread, check that the thread is not in the middle of executing any replaced function
Safely redirect execution

- Temporarily grab all CPUs
- For every thread, check that the thread is not in the middle of executing any replaced function
- If necessary, abort (rare)
Safely redirect execution

- Temporarily grab all CPUs

- For every thread, check that the thread is not in the middle of executing any replaced function

- If necessary, abort (rare)

- Paused less than 0.7ms
Data structure changes

- Design so far only changes code—not data
Data structure changes

- Design so far only changes code—not data
- Sometimes need to walk existing data structures, updating them:
  - Add a field to a struct
  - Change how a data structure is initialized
Ksplice support for data structure changes

- Simply modify the patch or add code to the patch
- Can use macros to run code when the update is applied
  - ksplice_pre_apply(func)
  - ksplice_apply(func)
  (and others...)
CVE-2006-1056 patch

--- a/arch/i386/kernel/cpu/amd.c
+++ b/arch/i386/kernel/cpu/amd.c
@@ -207,6 +207,9 @@ static void __init
    init_amd(struct cpuinfo_x86 *c)
...
+ if (c->x86 >= 6)
+   set_bit(X86_FEATURE_FXSAVE_LEAK,
+           c->x86_capability);
...

(and other changes)
static void set_fxsave_leak_bit(int id)
{
  int i;
  for (i = 0; i < NR_CPUS; i++) {
    struct cpuinfo_x86 *c = cpu_data + i;
    if (c->x86 >= 6 && c->x86_vendor == X86_VENDOR_AMD)
      set_bit(X86_FEATURE_FXSAVE_LEAK, c->x86_capability);
  }
}

ksplice_apply(set_fxsave_leak_bit);
Adding fields to structs

```c
struct foo {
    int a;
    + int b;
};
struct foo x[3];
```
Adding fields to structs

```c
struct foo {
    int a;
+    int b;
};
struct foo x[3];
```

Old layout

```
<table>
<thead>
<tr>
<th>a</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4 bytes</td>
</tr>
<tr>
<td>a</td>
<td>4 bytes</td>
</tr>
<tr>
<td>a</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>
```
Adding fields to structs

```c
struct foo {
    int a;
    + int b;
};
struct foo x[3];
```

**Old layout**

```
  a
  a
  a
```

4 bytes

```
  a
  a
  a
```

4 bytes

```
  a
  a
  a
```

4 bytes

**New layout**

```
  a
  b
```

8 bytes

```
  a
  b
```

8 bytes

```
  a
```

8 bytes

```
  b
```

8 bytes

```
  a
  b
```

8 bytes
Adding fields to structs

```c
struct foo {
    int a;
    int b;
};
```

```c
struct foo x[3];
```

Old layout

```
a
a
a
4 bytes
```

New layout

```
a
b
4 bytes

a
a
b
4 bytes

a
a
b
4 bytes
```

8 bytes

8 bytes

8 bytes

Shadow hashing

• “shadow” field(s) off to side
Shadow hashing

- “shadow” field(s) off to side
- Lookup shadows by hashing the address of the structure instance (O(1) time)

Old instance of struct foo at address 0xbeef

b_hashtable{0xbeef}
Hypothesis

- Most Linux security patches can be hot-applied without writing much new code

- Interested in:
  - How many patches can be applied without any new code?
  - How much new code is needed to apply the other patches?
Methodology

• Matched all 'significant' CVEs against Linux patch commit logs
Methodology

- Matched all 'significant' CVEs against Linux patch commit logs

- Generated a hot update for each CVE patch, confirming that:
  - Update applies cleanly
  - Still passes POSIX stress test
  - For available exploits: the exploit stops working
Summary of Results

• Hot-apply most security patches (88%) without any patch changes
Summary of Results

- Hot-apply most security patches (88%) without any patch changes.
- Hot-apply 100% with modest programmer effort (~17 lines of new code per patch).
CVEs that do not require any new code

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
## CVEs needing new code

<table>
<thead>
<tr>
<th>CVE #</th>
<th>Logical Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-0007</td>
<td>34</td>
</tr>
<tr>
<td>2007-4571</td>
<td>10</td>
</tr>
<tr>
<td>2007-3851</td>
<td>1</td>
</tr>
<tr>
<td>2006-5753</td>
<td>1</td>
</tr>
<tr>
<td>2006-1056</td>
<td>4</td>
</tr>
<tr>
<td>2006-2071</td>
<td>14</td>
</tr>
<tr>
<td>2005-3179</td>
<td>20</td>
</tr>
<tr>
<td>2005-2709</td>
<td>48</td>
</tr>
</tbody>
</table>
Debugging or Instrumenting
Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
- kgdb
- SystemTap
Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice
Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice
  - Real C
Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice
  - Real C
  - Insert code almost anywhere
Debugging or Instrumenting

• Sometimes, looking inside running system is invaluable
  • kgdb
  • SystemTap
• Advantages of Ksplice
  • Real C
  • Insert code almost anywhere
  • Discover any symbol value
Demos

- Protecting against x86 exploit
- Debugging on Android
Future plans

• Deliver existing technology

  Ksplice, Inc. starting to provide rebootless update service

• Continue advancing hot updates

  One goal: apply almost entire stable tree using Ksplice
Acknowledgments

Frans Kaashoek
Tim Abbott
Anders Kaseorg
Waseem Daher
MIT SIPB
http://www.ksplice.com

Subscribe to ksplice-announce:
http://lists.ksplice.com

Jeff Arnold
jbarnold@ksplice.com

Ksplice®