Ksplice: Rebootless Kernel Updates

Waseem Daher (wdaher@ksplice.com) – LinuxTag 2009

"You must reboot to install these updates." We have all seen this message, and we all hate it. Since rebooting is so inconvenient, users and system administrators often postpone installing updates, despite the increased security risk--more than 90% of attacks exploit known vulnerabilities. Ksplice is new software for eliminating the disruption caused by updates to the Linux kernel; with Ksplice, all Linux security updates (and many other updates) can be applied seamlessly, without a reboot.

Why Ksplice?

Currently, installing any software update is a disruptive process: an update to the operating system requires a reboot, and an update to an application requires an application restart. This represents an inconvenience for end-users—but it represents a serious pain for system administrators.

The fact of the matter is that software updates require downtime, and this downtime can be expensive, in the form of lost productivity, and, depending on the deployment, missed sales and irate customers. In general, this downtime takes the form of a scheduled multi-hour outage at a time when systems are believed to be less commonly in use (e.g. Sunday morning at 3 am). Of course, performing an update at off-peak hours means that a system administrator has to be paid extra to come in at off-peak hours—so there is also real operational expense associated with this process.

Reboots also cause a loss of software state: any open connections are closed, and any running computations are interrupted. Finally, reboots can commonly cause unexpected problems—incorrect boot-time configuration, hardware failures, and lengthy disk checks are all common occurrences.

So if rebooting for software updates is so painful, why do people do it at all? Because updates often fix important security problems. For context, 70-140 bugfixes are committed to the Linux stable tree each month, and in general, one or two tend to fix privilege escalation vulnerabilities. In other words: Linux (just like any other large software project) is rapidly changing, and new security vulnerabilities are constantly being discovered and fixed. It is important to stay up to date with these fixes.

Why? Because more than 90% of exploits take advantage of known vulnerabilities. So patching right away can dramatically improve your system's security, and any delay can have serious consequences. For example, within just six days after the announcement of the Linux sys_prctl vulnerability, many servers had been compromised, including an important server for the Debian distribution.

Waiting days or weeks to install updates is simply just too long.
The core technical challenge

The idea behind Ksplice is as follows: all software, no matter how good it is, has bugs. We have a running application or operating system that has a bug in it, and we want to get to a version without the bug, without any disruption.

One key insight is that software developers are already describing how to fix these bugs, in the traditional source code patches that they create. However, these patches are written under the assumption that the program or operating system will be recompiled and started from scratch. We instead need to determine how to change the running memory of the program or OS, to create a rebootless update.

So how can we go from a traditional source code patch to a rebootless update, in a manner that is as automated as possible? This is the core Ksplice technical challenge.

Features

Ksplice is currently implemented for the Linux kernel, though the design applies equally well to any piece of software. It does not require any advance kernel modification, which means that it can be used to update a machine without requiring an initial reboot.

Ksplice is also not limited to the core kernel or C—it can handle patches to kernel modules and kernel assembly code, as well as patches containing symbols that are not in the kernel symbol table.

As of this writing, it is implemented for x86, x86-64, and ARM, and works even in virtualized hosts or guests (VMware, Xen, KVM) and with custom kernel patches (e.g. OpenVZ, grsecurity).
How it works

There are two main steps involved in Ksplice's rebootless update process: creating a rebootless update from a traditional source code patch, and actually applying the update to a running system. These processes are separated, which means that an update could be created on a powerful computer with lots of CPU and RAM and then installed on a tiny embedded device with a small memory footprint.

Constructing an update

The update construction process requires a few inputs: the source code and associated configuration corresponding to the running kernel, and the traditional source code patch that we wish to apply using Ksplice. We refer to the source code of the running kernel as the “pre source,” because it is the source code before applying the patch.

Naturally, the first step is to copy the pre source and apply the traditional source code patch to it—resulting in what we call the “post source”. This is the source code for the kernel we would like to have.

Once we have these two kernel trees, we compile them both, ideally with the original kernel and assembler used to compile the running kernel. We now have the pre and post object code. We then compare the object code, with the goal of identifying which functions have changed as a result of the source code patch.

Once these functions are identified, both the old and new versions are extracted and turned into a set of kernel modules, along with kernel code that will actually install the update. It is this collection of kernel modules (packaged in a tarball) that can be shipped to the machine that we are updating.

The update creation process.
Applying an update

To apply an update, the prepared kernel modules are loaded into memory. This means that the new, corrected versions of the buggy functions are now in the kernel. However, no one is calling them—all of the callers are accessing the old, buggy versions of these functions.

So we need to redirect execution from the old functions to the new functions. This is accomplished as follows: first, we find the buggy copy of a given function in kernel memory. Once we have located it, we verify that it matches our old copy of the function (recall that we have both the old and new copy of every to-be-replaced function in our module). This process, known as run-pre matching, serves as a safety check by verifying that the kernel is as we expect it to be, but also gathers some potentially important information.

Once this check is done, we would like to redirect execution from the old, buggy versions of the functions to the new, corrected versions. We will accomplish this by writing a “trampoline” (jump instruction) at the top of the old function—now, all new callers will be immediately redirected to the new version of the function.

However, if this step is performed while the function being replaced is in use, we risk running into problems. For example, if the locking semantics change between versions of a function, we may deadlock. So the question is: how can we perform this update in a way that ensures that the functions being updated are not currently in use?

The answer: a Linux function called stop_machine, which temporarily grabs all of the CPUs on a system. Once we have grabbed all of the CPUs, we check that no kernel thread is in the middle of executing any replaced function—a check.
done by examining every memory address on each thread's stack, and verifying that it is not a reference into a to-be-replaced function.

If this check passes, we insert the trampolines and release the CPUs. The process takes less than 700 microseconds, and the system is now updated. In the rare event that the check fails, we abort the process (and can try again later). In general, a strategy of multiple retries succeeds fairly readily, since few functions in the kernel are constantly in use.

**Reversing an update**

Reversing an update follows the same process as applying an update—except instead of adding the trampolines, we remove them and restore the original instructions, taking care to perform the procedure at a safe time as above. After this procedure, all new callers of a function get the old version of the function, just as before the update.

**Changes that require additional effort**

The design outlined above only describes changes to kernel code, not data. However, certain patches may add a field to a struct or change how a data structure is initialized. In these cases, we may need to walk existing data structures to update them.

Ksplice accomplishes this by having a programmer modify the patch to add code to the traditional source code patch to assist in the process. In particular, Ksplice supports macros that allow a programmer to run code as an update is applied. This code can be used to transition the state of kernel data structures appropriately.

**Evaluation**

So just how practical is Ksplice? We set out to test this by examining important security vulnerabilities (privilege escalation, information disclosure, etc.) in Linux from May 2005 to May 2008. In particular, we are interested in knowing a few things: Can we apply them all? How many of them can be applied without writing any new code? Of the ones that do require new code, how much new code is required?

**Methodology**

We began by matching each important vulnerability from the MITRE CVE database against the Linux commit logs. There are 64 such patches in this interval. Given each traditional source code patch, we used Ksplice to generate a rebootless update for it, confirming that the update applies cleanly, that an updated machine still passes a POSIX correctness-checking stress test, and, if exploit code is available, that the exploit stops working.
Results

100% of the patches can be applied using Ksplice. Most of them (88%) can be applied without any patch changes, and the remaining 12% can be applied with modest programmer effort—an average of 17 lines of new code per patch. More detailed results appear below:

```
```

The patches that fix the above CVEs do not require any new code to be applied as Ksplice updates.

```
<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>2006-2071</td>
<td>14</td>
</tr>
<tr>
<td>2006-1056</td>
<td>4</td>
</tr>
<tr>
<td>2005-3179</td>
<td>20</td>
</tr>
<tr>
<td>2005-2709</td>
<td>48</td>
</tr>
</tbody>
</table>
```

CVEs that do require new code, and the number of additional lines of code that a programmer must write to apply the patches using Ksplice.

In short, Ksplice is quite practical, as demonstrated by its ability to keep a Linux system up to date and secure against three years' worth of important security vulnerabilities.

FAQ

What is Ksplice's current status?

Ksplice was initially released under the GPLv2 in April 2008, and has been in production use at MIT for over a year. As of this writing, the tools are also available in Debian sid, Ubuntu Jaunty, and Fedora 8-10.
Is Ksplice actively maintained?

Yes—in June 2008, we formed Ksplice, Inc., a company dedicated to making software systems more reliable and easier to maintain, based on this technology. As of this writing, we currently have five full-time and two part-time engineers actively working on the project.

Are you working to get this in mainline?

Yes, definitely. While Ksplice can work on an unmodified kernel, and thus does not require advance modifications to Linux, we understand the value and benefit of participating in the Linux community, and we are eager to have Ksplice embraced by the kernel community as the technology for Linux hot updates.

Does using Ksplice make me vulnerable to rootkits?

No. The Ksplice technology does not require any advance modification to your system, and does not require you to enable any special features—this is a fundamental capability of any program that can read and write kernel memory. In fact, Ksplice improves your security by allowing you to apply security updates more rapidly, without the disruption of rebooting.

Where did you get the idea for Ksplice?

Jeff Arnold, Ksplice’s lead inventor, first had the idea three years ago, when he was a student administering servers at MIT. A new kernel update was announced in the middle of the week, and, as is common practice, he decided to wait until the weekend to install it, to avoid disrupting the system’s many active users until a scheduled downtime window. Unfortunately, the delay was costly—an attacker took advantage of the kernel vulnerability and broke in, forcing Arnold and others to reinstall the system.

The incident got him thinking about why rebooting was a fundamental requirement for software updates, and he went on to tackle the problem of rebootless software updates in his award-winning MIT master’s thesis, which forms the basis for the Ksplice technology.

Conclusions

Rebooting represents a huge inconvenience for end-users, and a costly and painful disruption for system administrators. Fortunately, there is an answer: Ksplice is new software that can apply software updates to a Linux kernel, without requiring a reboot. A study of Linux security vulnerabilities over a three year interval confirms the effectiveness and practicality of this technology: 100% of the important security vulnerabilities in this interval can be corrected without rebooting.