SPARC vermeidet "Buffer Overflows" mit "Software in Silicon"

Martin Müller
Oracle Deutschland

Schlüsselworte

Sicherheit, Security , SPARC M7, Software development, Application data integrity, ADI

Einleitung

Mit der SPARC M7 CPU wird Oracle komplexere Software Funktionen in die CPU integrieren, die zum einen die Leistung der Oracle Datenbank bei speicherintensiven Operationen erheblich verbessert, sowie erstmals ermöglicht im laufenden Betrieb eine bestimmte Klasse von *Softwarefehlern* oder *logischen Fehlern* zu erkennen. Hier soll es um genau diese Klasse von Fehlern gehen, zu denen auch die sog. "buffer overflows" gehören.

Ein weitverbreitete Klasse von Sicherheitslücken oder Fehlerquellen von Softwareprodukten sind Referenzierungsfehler. Hier wird von einem Referenzierungsfehler gesprochen, wenn eine Zeigervariable weiter genutzt wird, obwohl der zugehörige Speicher nicht mehr verwendet sollte. Im Falle einer Shared Memory Anwendung könnte ein Thread einen Speicherbereich nutzen den ein anderer Thread angelegt hat, ohne daß der Programmierer das vorgesehen hat. Ein „Buffer overflow“ entsteht, wenn absichtlich versucht wird, über einen Ein- und Ausgabepuffer hinaus zu lesen und in dem Programm keine Sicherheitsvorkehrungen getroffen werden.

„Silicon Secured Memory“

Mit der SPARC M7 CPU wird es erstmals möglich transparent für gewisse Applikationen die eingangs beschriebenen Referenzierungsfehler zu verhindern. Durch „Application Data Integrity“ ist es erstmals möglich diese Klasse von Fehlern auch im produktiven Einsatz in Echtzeit zu überwachen und zu erkennen.

Bisher konnte diese Klasse von Softwarefehlern nur im Rahmen von Qualitätssicherungsmaßnahmen mit Hilfe von Emulatoren gefunden werden. Diese Emulatoren bewirken aber eine erhebliche Verlangsamung der Verarbeitungsgeschwindigkeit und sind daher keine Option für den produktiven Einsatz

Mithilfe einer neuen Funktionalität der M7 CPU kann jeder Hauptspeicherlokation 64byte-weise eine Markierung (oder „Farbe“) zugewiesen werde, sowie jedem Instruktionsstrom. Die CPU überprüft nun transparent ob diese beiden Markierungen übereinstimmen

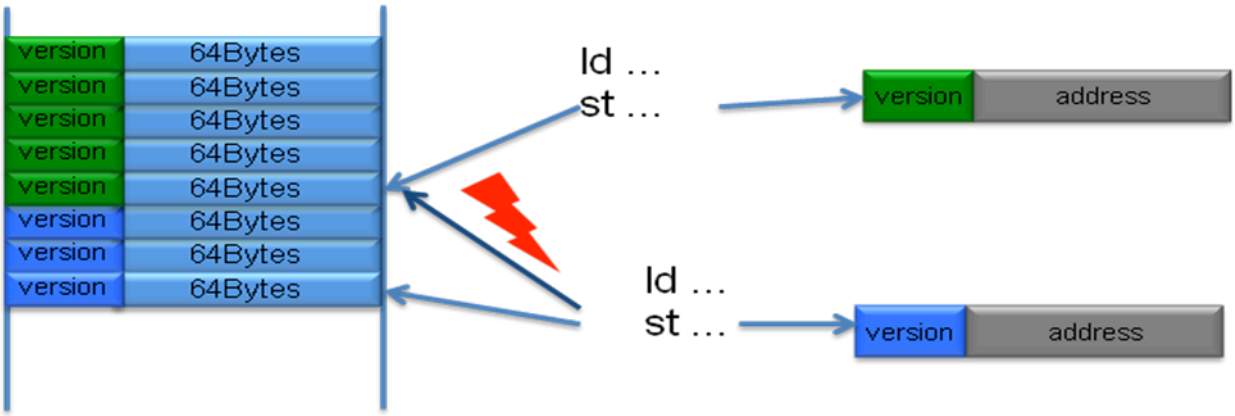

Abb. 1: „Einfärben“ von Hauptspeicher und Threads

Solange die „Farbe“ des Threads/Instruktionsstromes und der zugegriffenen Hauptspeicherlokation übereinstimmen, greift die CPU nicht in die Ausführung ein. Sollten aber die beiden Farben voneinander abweichen, wird ein Trap ausgelöst. Dieser Trap kann dann entweder nur eine Fehlermeldung ausgeben, oder eine andere Aktion auslösen.

Die Farben entsprechen im wirklichen Leben einer 4bit breiten Zahl, die sowohl der 64byte breiten Hauptspeicherzeile als auch der Zeigervariablen zugewiesen wird. Die Farbe einer Zeigervariablen sind die obersten 4bits derselben, also bit 63 bis bit 60.

Die Nutzung dieser Funktion ist natürlich optional, kein Programm **muß** davon Gebrauch machen bzw. auf M7 angepasst werden.

Nutzung von Application Data Integrity

Prinzipiell gibt es drei verschiedene Wege um von "Application Data Integrity" zu profitieren:

1. Der Erwerb eines Software Produktes (nicht notwendigerweise von Oracle), das ADI nutzt. (Die Oracle Datenbank 12c wäre ein Beispiel)

2. Die Anpassung einer vorhandenen Software deren Sourcecode im Zugriff ist. Diese Möglichkeit bietet die größte Flexibilität, erfordert aber auch umfangreiche Änderungen an vorhandener Software

3. Die Verwendung der libadimalloc Bibliothek bei Software, die nur als Binärcode verfügbar ist und die die libC Standardroutine malloc() zur Speicheranforderung nutzt.

Möglichkeit 1 besteht generell für alle der sog. "Software on Silicon" Neuerungen, und soll hier nicht beleuchtet werden. Die Punkte 2 und 3 werden im folgenden detailliert behandelt.

Direkte Verwendung von "Application Data Integrity"

Das Solaris Betriebssystem wird erweiterte Standardbiblitotheken bereitstellen, die die Nutzung von Application Data Integrity ermöglichen werden. Die Routinen werden u. a. erlauben, ADI sowohl für einen LWP als auch für bestimmte Hauptspeicherbereiche ein- und auszuschalten. Auf LWP oder Thread Seite geschieht dies durch Manipulation des zugehörigen PSTATE Registers, auf Hauptspeicherseite wird ADI pro memory page aktiviert. Das Ein- oder Ausschalten alleine ändert noch nicht das Verhalten eines Programms bei einem illegalen oder ungewollten Speicherzugriff, erst das Einschalten und die Zuwendung und die Verwendung geeigneter "Farben" im Sinne der Einleitung

würde einen Effekt hervorrufen. Daher wird in Zukunft ADI immer aktiviert sein, aber da keine "Farben" zugewiesen wurden würde sich das Ablaufverhalten nicht ändern. (Es kann natürlich auch ausgeschaltete werden, falls gewünscht.

Der anspruchsvolle Schritt vor der Verwendung von ADI ist die Entscheidung wie und welche "Farben" verwendet werden, und die Kodierung dieser Entscheidung. Diese Arbeit geht über das einfache Neuübersetzen einer vorhandenen Software hinaus, z. B. muß entschieden ob einzelne Speicherbereiche nur durch einen kleinen anders-farbigen Bereich getrennt werden (ein versehentlichen "Überlesen" eines Bereiches würde dann eine Unterbrechung des normalen Programmablaufs verursachen) oder verschiedene Speicherbereiche erhalten verschiedene Farben.

Mögliche Nutzung von ADI mit Hilfe von "libadimalloc"

Für viele bestehende Programme ließe sich ADI ohne große Eingriffe einsetzen, wenn diese Programme die Standardsystemroutine malloc() zur Allokation von Hauptspeicher einsetzen und die damit initialisierten Zeigervariablen nur "konservativ" genutzt werden.

Durch Überladen der normalen Implementierung von malloc() durch setzen der LD_PRELOAD Umgebungsvariable[1] wird das "normale" malloc() durch eine Implementierung ersetzt, die zum einen die angeforderten Speicherbereiche mit geeigneten "Farben" versieht und einen Zeiger in der passenden Farbe zurück liefert.

Die "Farben" werden als 4bit Zahlen in den obersten 4bit der virtuellen Adressen kodiert, daraus leiten sich die zwei Voraussetzungen für den erfolgreichen Einsatz von libadimalloc ab:

1. Die vier höchstwertigsten Bits dürfen nicht schon für andere Zwecke verwendet werden

2. Dynamisch zugewiesene Zeigervariablen dürfen nicht miteinander verknüpft werden, d. h. sie dürfen nicht addiert oder subtrahiert werden. Eine derartige Operation würde ja auch auf die "Farben" angewendet werden und das Resultat mit einer undefinierten Farbe versehen

In vielen Fällen dienen solche Zeigervariablen nur zur Aufnahme von Daten, deren Menge zum Zeitpunkt der Übersetzung nicht bekannt ist. Derartige Programme könnten von ADI profitieren ohne daß eine Änderung an ihnen notwendig wäre:

Ohne ADI	Mit ADI

```
#include <stdio.h>
#include <stdlib.h>

int
main(void){
  char* public = (char*)malloc(sizeof(char)*100);
  char* secret = (char*)malloc(sizeof(char)*100);

  printf("public text -> ");  scanf("%s", public);
  printf("secret text -> ");  scanf("%s", secret);
```

[1] Der dynamische Linker von Solaris erlaubt das Überladen von Systemaufrufen mit alternativen Implementierungen: der malloc() Aufruf ist ein häufiges Ziel dieses Überladens, es stehen Versionen zur Verfügung, die besonders im Hinblick auf "multi-threaded codes" optimiert wurden, oder auch Versionen die die Überprüfung der Zuweisung von Speicher erleichtern.

```
  for(int ii = 0; ii < 150; ii++)
    printf("%c\n", public[ii]);

  printf("\n");
  return 0;
}
```

```
martimue@SPARC-M7,ADI>./malloc
public text -> hello
secret text -> secret
h
e
l
l
o
---snip---

s
e
c
r
e
t
---snip---
martimue@SPARC-M7,ADI>
```

```
martimue@SPARC-M7,ADI>
  LD_PRELOAD=libadimalloc.so ./malloc
public text -> hello
secret text -> secret
h
e
l
l
o
---snip---

Segmentation Fault (core dumped)
martimue@SPARC-M7,ADI>
```

Dieses einfache Beispiel zeigt wie der Versuch, über den "öffentlichen" Bereich hinaus zu lesen von ADI unterbunden wird. Mit Hilfe des Solaris Studio Debuggers ließe sich untersuchen an welcher Stelle im Programm und bei welchem Stand der Schleifenvariable der SEGV Fehler auftrat. Daraus ließe sich leicht ableiten, welcher Fehler in dem Beispielprogramm gemacht wurde. Im Beispiel ist der Fehler offensichtlich: es werden 150 Zeichen aus einem Bereich gelesen, der nur 100 Zeichen lang ist. Da sich aber an diesen Bereich ein zweiter anschließt, wird das Überlesen nicht registriert und der Inhalt des "geheimen" Bereiches ausgegeben.

Kontaktadresse:
Martin Müller
Oracle Deutschland
Hamborner Str. 51
40472 Düsseldorf
Telefon: +49 211 74839-853
E-Mail: martin.x.mueller@oracle.com
Internet: www.oracle.com