LEHRSTUHL 14
SOFTWARE ENGINEERING

Willkommen zur Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2011

Prof. Dr. Jan Jürjens

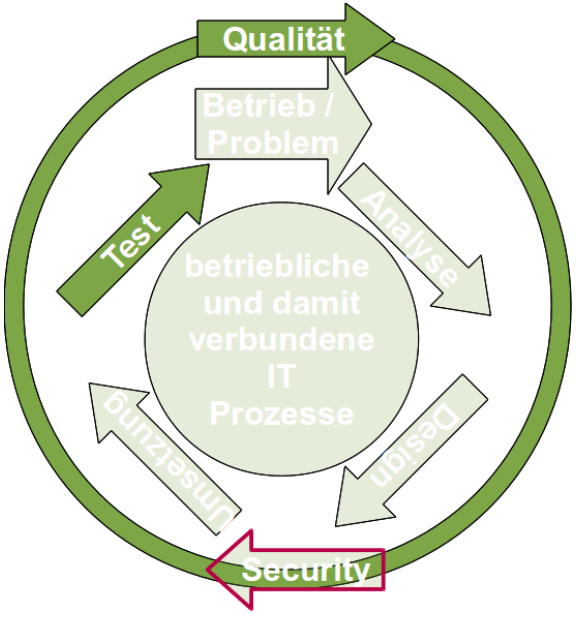
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

14. JUnit, Fuzzing, Spike

[inkl. Beiträge von Massimo Felici, The University of Edinburgh und Mike Zusman, New York University]

Einordnung
Testmethoden aus der Praxis

- JUnit
- Fuzzing
  - Verschiedene Typen
- Spike



Einordnung
Grundlagen Testen

- Business Prozesse
- Qualitätsmanagement
- Testen
  - Grundlagen
  - Blackbox / Whitebox Testing
  - Beispiele: Junit, Fuzzing (Spike)
- Sicherheit
- Sicheres Software Design

Themen

- **JUnit**
- Fuzzing
- Spike

JUnit

JUnit ist ein Framework, um Tests zu schreiben.

- Geschrieben von Erich Gamma (Design Patterns) und Kent Beck (eXtreme Programming).
- JUnit benutzt die Reflexions-Fähigkeit von Java (Java Programme können ihren eigenen Code prüfen) und (ab Version 4) Assertions.
- JUnit erlaubt uns...
  - Tests und Testfolgen zu definieren und auszuführen
  - Tests zu benutzen, um effektiv die Spezifikation zu beurteilen
  - überprüften Code in den Build zu integrieren
- JUnit ist in einigen IDEs verfügbar, z.B. BlueJ, JBuilder und Eclipse haben JUnit bis zu einem gewissen Grad eingebaut.

Für mehr Infos über JUnit

- Die JUnit Seite bietet eine Fülle von nützlichen Informationen über JUnit und einer Menge von JUnit-basierten Anwendungen:

  http://www.junit.org

JUnits Terminologie

- Ein **test runner** ist eine Software, die Tests ausführt und Ereignisse darstellt.

  Es gibt viele Implementationen: alleinstehende GUI, Kommandozeile, integriert in die IDE

- Eine **test suite** ist eine Sammlung von Testfällen.
- Ein **test case** testet eine einzelne Methode mit einer bestimmten Menge von Eingaben.
- Ein **unit test** ist ein Test eines kleinsten Codeelements, das sich noch überprüfen lässt. Normalerweise ist das eine einzelne Klasse.

JUnits Terminologie

- Eine **test fixture** ist eine Umgebung, in der ein Test ausgeführt wird. Eine neue **fixture** wird aufgesetzt, bevor die Testfälle ausgeführt werden und wird danach wieder abgestellt.

  Beispiel: Wenn man einen Datenbank-Client testet, stellt die **fixture** den Datenbank-Server auf den Initialstatus. Der Client kann sich dann verbinden.

- Ein **integration test** ist ein Test, wie gut die Klassen zusammenarbeiten.

  *JUnit bietet begrenzte Unterstützung für **integration tests.***

- *Ordnungsgemäße* Komponententests würden **mock objects** einbeziehen – das sind zu diesem Zweck erstellte Versionen der anderen Klassen, mit denen die Klasse im Test interagiert.

  JUnit hilft dabei nicht. Es ist wichtig, den Begriff der mock objects zu kennen, aber nicht immer notwendig sie einzusetzen.

Struktur einer JUnit (4) Testklasse

Wir wollen eine Klasse namens `Triangle` testen

- Das ist ein *unit test* für die `Triangle` Klasse. Er definiert Objekte, die durch einen oder mehrere Tests benutzt werden.

```
public class TriangleTestJ4{

}
```

- Das ist der *default constructor*.

```
public TriangleTest() { }
```

Struktur einer JUnit (4) Testklasse

- `@Before public void int()`

  Erstellt ein *text fixture,* indem es Objekte und Werte erstellt und initialisiert.

- `@After public void cleanUp()`

  Gibt alle Systemresourcen, die durch den *text fixture* benutzt wurden, frei. Java macht das normalerweise selber, aber Dateien, Netzwerkverbindungen, etc … werden nicht immer automatisch „aufgeräumt“.

- `@Test public void noBadTriangles(), @Test public void scaleneOk(), etc.`

  Diese Methoden beinhalten Tests für den `Triangle` Konstruktor und die `isScalene()` Methode.

Tests erstellen: Assertions

- Innerhalb eines Tests:
  - Rufe die Methode auf, die getestet wird, und erhalte die aktuellen Resultate.
  - Stelle eine Behauptung (Assertion) bzgl. einer Eigenschaft auf, die im Ergebnis enthalten ist.
  - Jede Behauptung ist eine Anforderung an das Testergebnis.
- Wenn die Eigenschaft nicht die Behauptung erfüllt, wirft sie ein `AssertionFailedError`:
  - JUnit fängt diesen Error, zeichnet die Ergebnisse des Tests auf und stellt diese dar.

Tests erstellen: Assertions

- `static void assertTrue(boolean test)`

  `static void assertTrue(String message, boolean test)`

  Wirft ein `AssertionFailedError` wenn der Test fehlschlägt. Die optionale *`message`* ist im Error enthalten.

- `static void assertFalse(boolean test)`

  `static void assertFalse(String message, boolean test)`

  Wirft ein `AssertionFailedError` wenn der Test erfolgreich endet.

Nebenbei: Throwable

- `java.lang.Error`: Ein Problem, das die Applikation normalerweise versucht zu behandeln – braucht in keiner `throws` Klausel deklariert werden.

  z.B.: Ein Kommandozeilen Programm bekommt unpassende Parameter vom Benutzer.

- `java.lang.Exception`: Ein Problem, mit dem die Applikation angemessen umgehen muss – muss in einer `throws` Klausel deklariert werden.

  z.B.: Die Netzwerkverbindung überschreitet die Maximalzeit während des Verbindungsaufbaus.

- `java.lang.RuntimeException`: Die Applikation könnte damit umgehen, tut dies aber selten - braucht in keiner `throws` Klausel deklariert werden.

  z.B.: I/O buffer overflow.

Triangle Klasse

Als Beispiel werden wir eine triviale `Triangle` Klasse definieren und testen.

- Der Konstruktor erstellt ein `Triangle` Objekt, in dem nur die Längen der Seite abgespeichert werden, wobei die private Variable `p` die längste Seite ist.
- Die `isScalene` Methode gibt `true` zurück, wenn das Dreieck ungleichseitig ist.
- Die `isEquilateral` Methode gibt `true` zurück, wenn das Dreieck gleichseitig ist.
- Wir können die Testmethoden vor dem Code schreiben. Das hat Vorteile bei der Trennung vom Code und den Tests.

  Eclipse hilft mehr, wenn die Klassen im Test zuerst erstellt werden. Es erstellt Stubs (Methoden mit leeren Körper) für alle Methoden und Konstruktoren.

Anmerkungen zur Test-Erstellung

- **Größe:** Oft übertrifft die Menge an (normalem) Testcode die Größe des Codes bei kleineren Projekten.
- **Komplexität:** Komplexen Code zu testen, kann zu sehr komplexen Tests führen.
- **Aufwand:** Der Aufwand, den es kostet, Test-Code zu schreiben, zahlt sich durch reduzierte Entwicklungszeit wieder aus.
- **Verhalten:** Tests zu erstellen hilft dabei, klar zu machen wie eine Methode sich verhalten soll (besonders in Ausnahmeumständen).

Ein JUnit 3 Test für Triangle


```
import junit.framework.TestCase;
public class TriangleTest extends TestCase {
    private Triangle t;
    // Any method named setUp will be executed before each test.
    protected void setUp( ) {
        t = new Triangle(5,4,3);
    }
    protected void tearDown( ) { } // tearDown will be executed afterwards
    public void testIsScalene ( ) { // All tests are named test[Something]
        assertTrue(t.isScalene( ));
    }
    public void testIsEquilateral( ) {
        assertFalse(t.isEquilateral ( ));
    }
}
```

Ein JUnit 4 Test für Triangle

Mehr Importierung notwendig →

Nicht notwendig von Testcase zu erben →

Annotationen verwenden →

… anstatt besondere Namen →

```
Package st;
import static.org.junit.Asser.*;
import org.junit.Before;
import org.junit.Test;
public class TestTriangle {
    private Triangle t;
    @Before public void setUp( ) throws Exception {
        t = new Triangle(3, 4, 5);
    }
    @Test public void scaleneOk( ) {
        assertTrue(t.isScalene( ));
    }
}
```

Die Triangle Klasse

- Ist JUnit zu viel für kleine Programme?
- Nicht wenn man denkt, es könne Fehler reduzieren.
- Tests bei dieser Programmgröße finden häufig Fehler und Auslassungen – konstruieren Sie den Test anhand der Spezifikation.
- Manchmal kann man bei besonders einfachen Teilen des Systems auf Tests verzichten.

Die Triangle Klasse


```
public class Triangle {
private int p; // Longest edge
private int q;
private int r;
public Triangle(int s1, int s2,
                        int s3) {
    if (s1>s2) {
        p = s1; q = s2;
    } else {
        p = s2; q = s1;
    }
    if (s3>p) {
        r = p; p = s3;
    } else {
        r = s3;
    }
}
```

```
public boolean isScalene() {
    return ((r>0) && (q>0) && (p>0) &&
    (p<(q+r))&& ((q>r) || (r>q)));
}

public boolean isEquilateral() {
    return p == q && q == r;
}
}
```

Assert Methoden II

- `assertEquals(expected, actual)`

  `assertEquals(String message, expected, actual)`

  Diese Methode ist stark überlagert: Expected und actual müssen beide Objekte oder beide vom selben primitiven Typ sein. Für Objekte benutzt es die Equal Methode, wenn sie ordnungsgemäßg definert wurde, wie `public boolean equals(Object o)` – andernfalls wird folgendes benutzt:

- `assertSame(Object expected, Object actual)`

  `assertSame(String message, Object expected, Object actual)`

  Behauptet, dass zwei Objekte auf dasselbe Objekt verweisen (unter Verwendung von ==).

- `assertNotSame(Object expected, Object actual)`

  `assertNotSame(String message, Object expected, Object actual)`

  Behauptet, dass zwei Objekte nicht auf dasselbe Objekt verweisen.

Assert Methoden III

- `assertNull(Object object)`
- `assertNull(String message, Object object)`
  - Stellt fest, dass ein Objekt null ist.
- `assertNotNull(Object object)`
- `assertNotNull(String message, Objectobject)`
  - Stellt fest, dass ein Objekt nicht null ist.
- `fail()`
- `fail(String message)`
  - Lässt den Test fehlschlagen und wirft eine AssertionFailedError Exception.

Die Assert-Anweisung in Java

- Frühere Versionen von JUnit hatten eine `assert` Methode anstatt einer `assertTrue` Methode. Der Name musste geändert werden, als mit Java 1.4 die assert-Anweisung hinzukam.

- Es gibt zwei Arten der assert-Anweisung:
  - `assert boolean_condition;`
  - `assert boolean_condition : error_message ;`

  Beide Arten werfen einen `AssertionFailedError` wenn die `boolean_condition false` ist. Die zweite Art, mit der expliziten Fehlermeldung, ist selten notwendig.

Die Assert-Anweisung in Java

Wann man assert-Anweisungen benutzen sollte:

- Man benutzt sie, um Bedingungen zu dokumentieren, wo man weiß, das sie `true` sind.
- Man benutzt `assert false` im Code für Stellen, die wissentlich nicht erreicht werden (wie z.B. der `default case` in einer Switch-Anweisung).
- Man benutzt `assert false` nicht, um die Korrektheit der Eingabeparameter zu überprüfen. Generell sollte man es nicht benutzen, wo der Wurf einer Exception angebrachter wäre.

JUnit in Eclipse

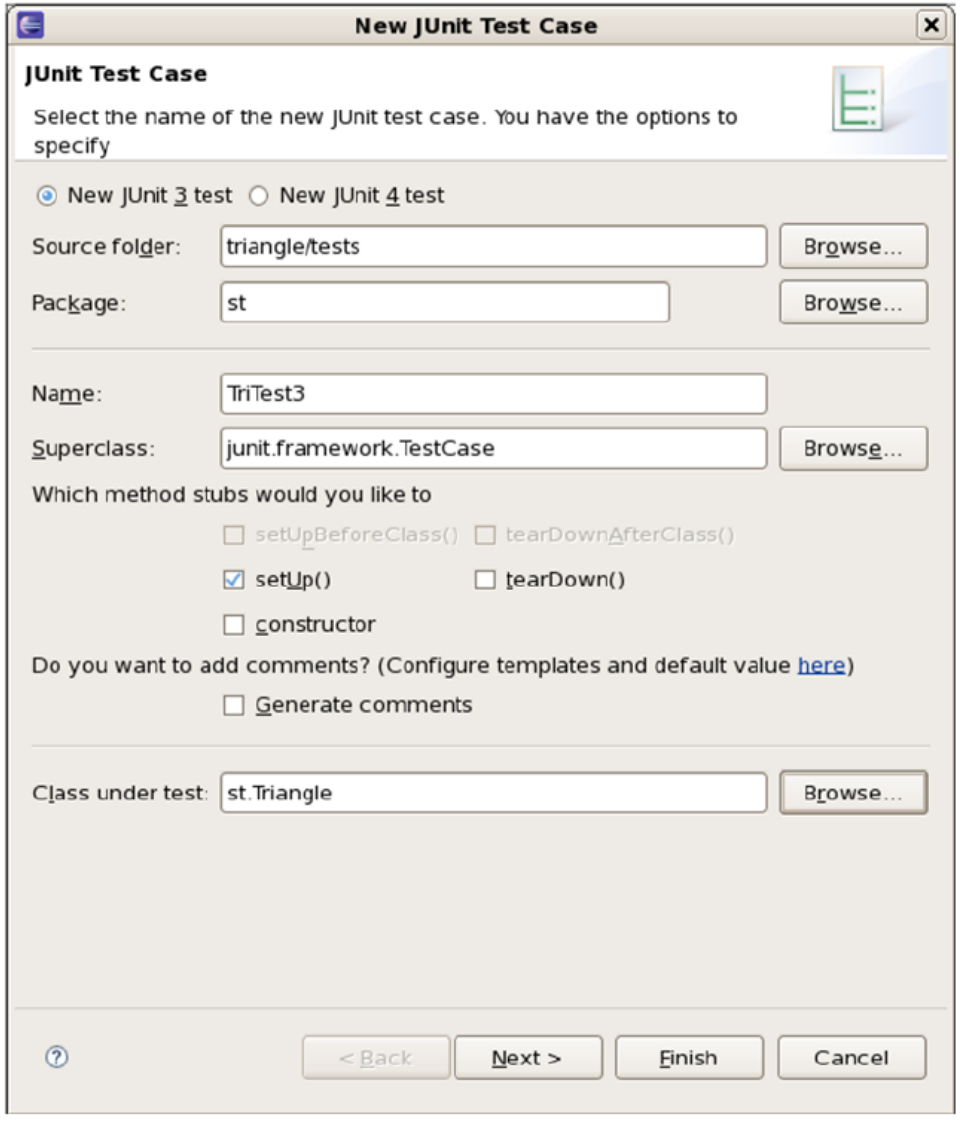
Um eine Testklasse zu erstellen, wählen Sie File → New → JUnit Test Case und wählen einen Namen für den Testfall

Package →

Test Klasse →

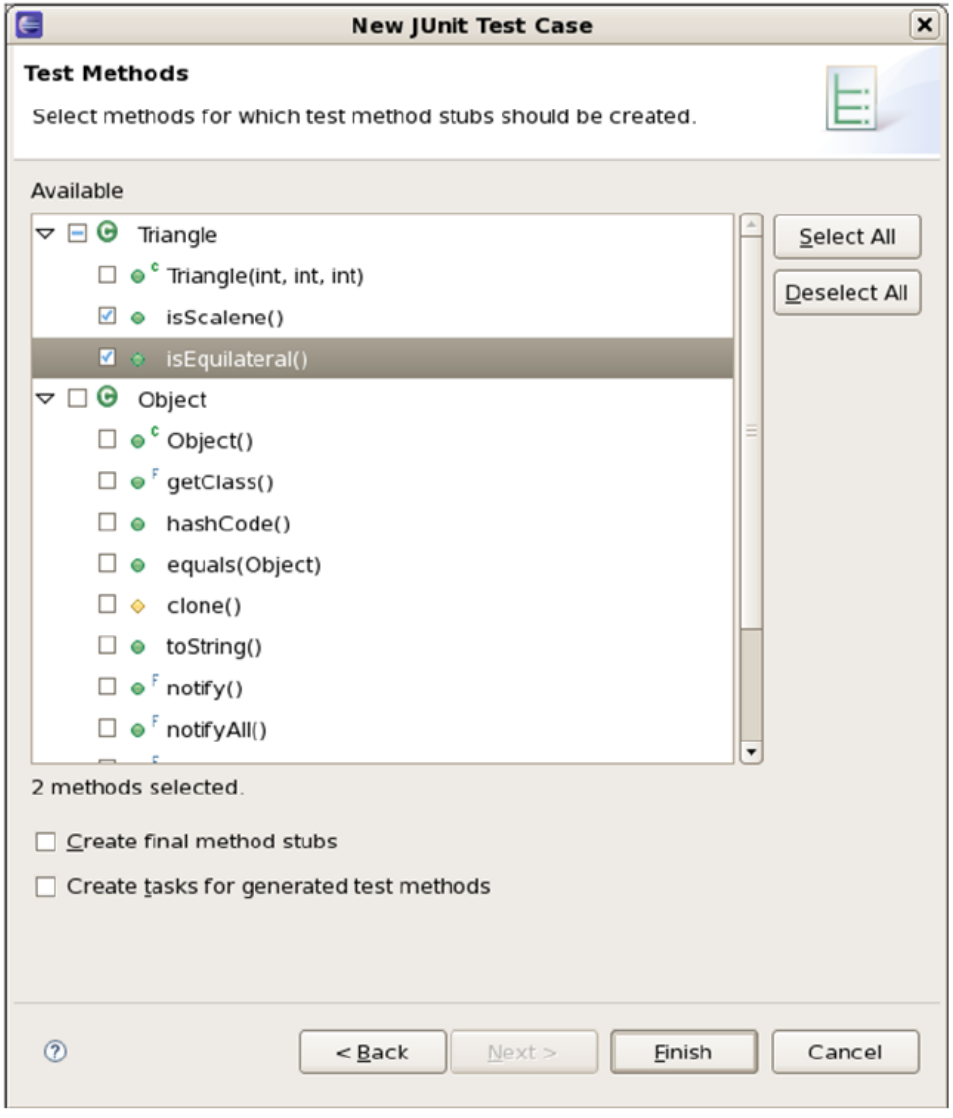
Welche Stubs wollen Sie erstellen →

Identifiziert die Klasse im Test →



Einen Test erstellen

Entscheiden Sie was sie testen wollen →



Vorlage für einen neuen Test

Java - TriangleTest2.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | JUnit

Finished after 0.01

Runs: 2/2 Errors: 0 Failures: 0

Failures | Hierarchy

Failure Trace

Triangle.java | TriangleTest2.java

```
import junit.framework.TestCase;

public class TriangleTest2 extends TestCase {

    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /*
     * Test method for 'Triangle.Scalene()'
     */
    public void testScalene() {

    }

    /*
     * Test method for 'Triangle.Equilateral()'
     */
    public void testEquilateral() {

    }
```

Outline

- import declarations
- TriangleTest2
  - setUp()
  - tearDown()
  - testScalene()
  - testEquilateral()

Javadoc | Declaration

TriangleTest.tearDown()

Tears down the fixture, for example, close a network connection. This method is called after a test is executed.

Writable | Smart Insert | 17 : 1

JUnit ausführen

Java - TriangleTest.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

Finished after 0.01

Runs: 2/2 Error

Failures Hierar

1 TriangleTest
2 TriangleTest.setUp
3 TestThatWeGetHelloWorldPrompt
Run As
Run...
Organize Favorites...
1 JUnit Test Alt+Shift+X, T

TriangleTest2.java

```
amework.TestCase;

setUp(){ // executed before each t
angle(5,4,3);
}

protected void tearDown(){} // executed after eac

public void testScalene(){
    assertTrue(t.Scalene());
}

public void testEquilateral(){
    assertFalse(t.Equilateral());
}
}
```

Outline

import declarations
TriangleTest
t : Triangle
setUp()
tearDown()
testScalene()
testEquilateral()

Failure Trace

Javadoc Declaration

TriangleTest.tearDown()

Tears down the fixture, for example, close a network connection. This method is called after a test is executed.

Writable Smart Insert 16 : 28

Ergebnis

Ergebnis steht hier →

Java - TriangleTest.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | JUnit

Finished after 0.01

Runs: 2/2 Errors: 0 Failures: 0

Failures | Hierarchy

Failure Trace

TriangleTest.java | TriangleTest2.java

```
import junit.framework.TestCase;

public class TriangleTest extends TestCase {
private Triangle t;

protected void setUp(){ // executed before each t
    t = new Triangle(5,4,3);
}

protected void tearDown(){} // executed after eac

public void testScalene(){
    assertTrue(t.Scalene());
}

public void testEquilateral(){
    assertFalse(t.Equilateral());
}
}
```

Outline

- import declarations
- TriangleTest
  - t : Triangle
  - setUp()
  - tearDown()
  - testScalene()
  - testEquilateral()

Javadoc | Declaration

TriangleTest.tearDown()

Tears down the fixture, for example, close a network connection. This method is called after a test is executed.

Writable | Smart Insert | 16 : 28

Probleme mit JUnit

JUnit hat ein Modell aus aufrufenden Methoden und dem Überprüfen von Ergebnissen anhand von erwarteten Ergebnissen. Probleme sind:

- Zustand: Objekte mit signifikantem internen Zustand (z.B. Kollektionen mit erweiterter Struktur) sind schwerer zu testen, weil es viele Methodenaufrufe benötigt, um das Objekt in den gewünschten Testzustand zu bringen. Lösungen:
  - Schreibe lange Tests, die einige Methoden oft aufrufen.
  - Füge zusätzliche Methoden hinzu, die es erlauben, den Zustand des Objekts zu überwachen (oder mache private Variablen public?).
  - Füge zusätzliche Methoden hinzu, die es erlauben, den internen Zustand auf einen bestimmten Wert zu setzen.
  - „Heisenbugs“ können ein Problem sein (ändere die Observationsmethode und es ändert sich das Beobachtete).

Probleme mit JUnit

- Andere Effekte, z.B. dass die Ausgabe manchmal schwer zu ermitteln ist.
- JUnit GUI-Tests sind nicht sonderlich hilfreich.

Positives

- Die Benutzung von JUnit regt zu einem *testbaren* Stil an, in dem die Ergebnisse einer Methodenanfrage leicht anhand der Spezifikation zu prüfen sind:
  - Die kontrollierte Nutzung von Zustand
  - Zusätzliche Überwachung des Zustands (Test Oberfläche)
  - Zusätzliche Komponenten im Ergebnis, die das Überprüfen vereinfachen
- Es ist in viele IDEs gut integriert (z.B. Eclipse)
- Tests sind dort leicht zu definieren und anzuwenden.
- JUnit regt zu fortlaufenden Tests während der Entwicklung an – z.B. XP (eXtreme Programming) *„test as specification“.*
- Junit neigt dazu, den Code leicht testbar zu halten.
- Junit unterstützt eine Reihe von Erweiterungen, die Strukturtests unterstützen (z.B. die Analye der Abdeckung).

Ein anderes Framework fürs Testen

- Framework for Integrated Test (FIT), von Ward Cunningham (Erfinder von Wiki), http://fit.c2.com
- Ist ein Tool, das die Kommunikation zwischen Entwicklern und Benutzern unterstützt, indem es dem Benutzer ermöglicht Tests in Form von strukturierten (HTML) Tabellen zu erstellen.
  - Benutzt HTML Tabellen mit erwarteten Verhalten vom Kunden
  - Verwandelt diese Tabelle in Testdaten: Eingabe, Aktivitäten und Behauptungen zu erwarteten Ergebnissen
  - Führt den Test aus und erstellt Tabellen mit einer Zusammenfassung des Testlaufs.
- Nur ein paar Jahre alt, aber relativ weit verbreitet.

Ein anderes Framework fürs Testen

- FitNesse schließt FIT an ein Wiki an, welches gemeinsames Entwickeln von Tests noch einfacher macht.

  http://fitnesse.org/

Themen

- JUnit
- **Fuzzing**
  - **Historie**
  - Fuzzer-Arten
  - Fuzzing
- Spike

Was genau ist fuzzing?

- *„Fuzzing is the process of sending intentionally invalid data to a product in the hopes of triggering an error condition or fault. These error conditions can lead to exploitable vulnerabilities.“*

  HD Moore (in dem Buch „Fuzzing“)

- *„Throw sh!t at the wall and see what sticks!“*

  b3nn

Was genau ist fuzzing?

- *„There are no rules of fuzzing.“*

  aus dem Buch Fuzzing

- *„I haven't really seen it get to the enterprise. Today, for the most part, if you want to be doing fuzzing you have to develop your own apps for that.“*

  Michael Sutton, Autor und Sicherheits Evangelist, SPI Dynamics

- *„There are no guarantees in fuzzing.“*

  Mike Zusman

Historie

- Fuzzing ist nicht neu
  - Es ist bereits seit 20 Jahren bekannt
- Professor Barton Miller
  - Vater des Fuzzing
  - Entwickelte 1988/89 das Fuzz-Testen mit seinen Studenten an der Universität von Wisconsin-Madison
  - ZIEL: UNIX-Anwendungen verbessern

Historie

- Millers Fuzzer was sehr grundlegend
- Er sendete zufällige Stringdaten an die Anwendung
- If (CRASH||HAND) {Finding(fuzzStr);}
- Bessere fuzzer sollten folgen ...

Historie

- 1999 brachte die Universität von Oulu PROTOS heraus.
- PROTOS begann, indem es PROTOkoll Spezifikationen analysierte.
- Pakete wurden modelliert, die die Spezifikation verletzten.
- Test-Suiten wurden entworfen, die gegen verschiedene Produkte des Herstellers angewendet werden konnten.

Historie

- 2002
  - Microsoft investierte in PROTOS
- Einige PROTOS Mitglieder starteten Codenomicon
  - Der erste kommerzielle Fuzzer

Historie

- SPIKE fuzzer wurde ebenfalls 2002 veröffentlicht
  - Dave Aitel schrieb ihn
- Wo Millers Fuzzer dumm war, war SPIKE ein Genie
  - Fähigkeit, Daten zu beschreiben
  - Eingebaute Bibliotheken für bekannte Protokolle (*RPC)
  - Fuzz Strings wurden so entworfen, dass es die Softwareausführung fehlschlagen ließ.

Historie

- 2004: Browser-Fuzzing
- MangleMe von Michal Zalewski
  - Fuzz'te HTML, um Browser Fehler zu finden.

Attachments		
input element crash (25 bytes, text/html) 2004-10-18 13:23 PDT, Daniel Veditz	no flags	Details
attack of the marquees (361 bytes, text/html) 2004-10-18 13:23 PDT, Daniel Veditz	no flags	Details
col span demo (non-crashing) (58 bytes, text/html) 2004-10-18 13:24 PDT, Daniel Veditz	no flags	Details
crasher not viewing as a file:/// (62.88 KB, text/html) 2004-10-23 21:41 PDT, Keith Gable	no flags	Details
Add an attachment (proposed patch, testcase, etc.)		View All

Description From Daniel Veditz 2004-10-18 13:22:03 PDT

http://securityfocus.com/archive/1/378632/2004-10-15/2004-10-21/0

extract:

```
A gallery of quick examples I examined to locate the offending tag
(total time to find and extract them - circa 1 hour):
```

Historie

- 2004: Dateiformat-Fuzzing
- Microsoft Security Bulletin MS04-028
  - Buffer Overun beim JPEG Processing (GDI+) könnte Remote Code Execution erlauben.

Historie

- 2005: Dateiformat-Fuzzer werden veröffentlicht.
- FileFuzz, SPIKEfile, notSPIKEfile
  - Michael Sutton und sein Team
- Dann kam die Ernüchterung.
  - „When Office 2003 shipped, we thought, we´d done some good work and that it would be a secure product," sagte David LeBlanc, ein Senior Softwareentwicklungs Ingenieur mit seinem Office-Team. „For the first two years after release, it held up really well, only two bulletins. [But] then people shifted their tactics and started finding problems in fairly large numbers." -

http://www.infoworld.com/article/07/09/21/Microsoft-developer-Fuzzing-key-to-Office-security_1.html?DESKTOP%20SECURITY

Historie

- 2005: Mehr Browser Fuzzer
- Hamachie
  - HD Moore, Aviv Raff
  - Fuzz'te Dynamic HTML.
- CSSDIE
  - HD Moore und sein Team
  - Fuzz'te CSS Style Sheets.

Historie

- 2006: Monat der Browser Bugs
  - HD Moore und sein Team haben jeden Tag einen Browser Bug veröffentlicht

    http://www.foxnews.com/story/0,2933,202547,00.html

Historie

- 2006 ActiveX Fuzzing
- COMRaider
  - GUI basiert
- AxMan
  - Komplizierter in der Benutzung als COMRaider
- Warum so einfach?
  - ActiveX/Com Objekte haben exportierbare Typbibliotheken, die alle Methoden, Interfaces und Eigenschaften beschreiben

Historie

- 2007: Mehr Browser-Fuzzing

Advisory: a specially crafted JavaScript can make Opera execute arbitrary code

A specially crafted JavaScript can make Opera execute arbitrary code.

Severity:

Highly severe

Problem description

A virtual function call on an invalid pointer that may reference data crafted by the attacker can be used to execute arbitrary code.

Opera's response

Opera Software has released Opera 9.23, where this issue has been fixed.

Credits

Thanks to Mozilla.org for providing their JavaScript fuzzer.

Themen

- JUnit
- **Fuzzing**
  - Historie
  - **Fuzzer-Arten**
  - Fuzzing
- Spike

Fuzzing-Methoden

- Sende zufällige Daten
  - Am wenigstens effektiv
  - Leider ist der Code manchmal so schlecht, dass diese Methode ausreicht, um Schwachstellen zu finden.
- Manuelle Protokoll Mutation
  - Du bist der Fuzzer.
  - Zeitaufwändig, kann aber recht genau sein, wenn man eine Ahnung hat.
  - Penetrations-Testen von Web-Anwendungen

Fuzzing Methoden

- Mutation oder Brute Force Testing
  - Beginnt mit einem validen Beispiel
  - Fuzze jeden, aber wirklich jeden Byte des Beispiels
- Automatic Protocol Generation Testing
  - Benutzer muss das Protokoll verstehen.
  - Code wird geschrieben, um das Protokoll zu beschreiben (eine „Grammatik“).
  - Der Fuzzer weiß dann, welche Teile zu fuzzen sind und welche unverändert gelassen werden (Spike).

Fuzzer Arten

- Lokaler Fuzzer
  - Damit lassen sich Programme über die Kommandozeile fuzzen.
  - Zu welchem Zweck?
    - Stellt sicher, dass das Ziel einen Wert hat (setuid)
- Umgebungsvariablen Fuzzer
  - Beispiel:

```
#include <string.h>

Int main (int argc, char **argv)

{

    Char buffer[10];

    strcpy(buffer, getenv(„Home“));

}
```

Fuzzer Arten

- Dateiformat-Fuzzer
  - Fuzz't valide Dateien.
  - Gibt sie an eine ausführbare Datei.
- Remote Fuzzer
  - Hört die Verbindungen eines Netzwerks ab.
  - Wenn ein Klienten sich verbindet, wird er gefuzzt!

Fuzzer Arten

- Netzwerkprotokoll-Fuzzer
  - Der Fuzzer ist der Klient.
  - Er muss das Protokoll verstehen.
    - Simple Protokolle
      - Text basiert
      - Telnet, FTP, POP, HTTP
    - Komplexe Protokolle
      - Binäre Daten (ASCII)
      - Komplexe Authentifizierung, Verschlüsselung, etc.
      - MSRPC (durch Spike unterstützt).

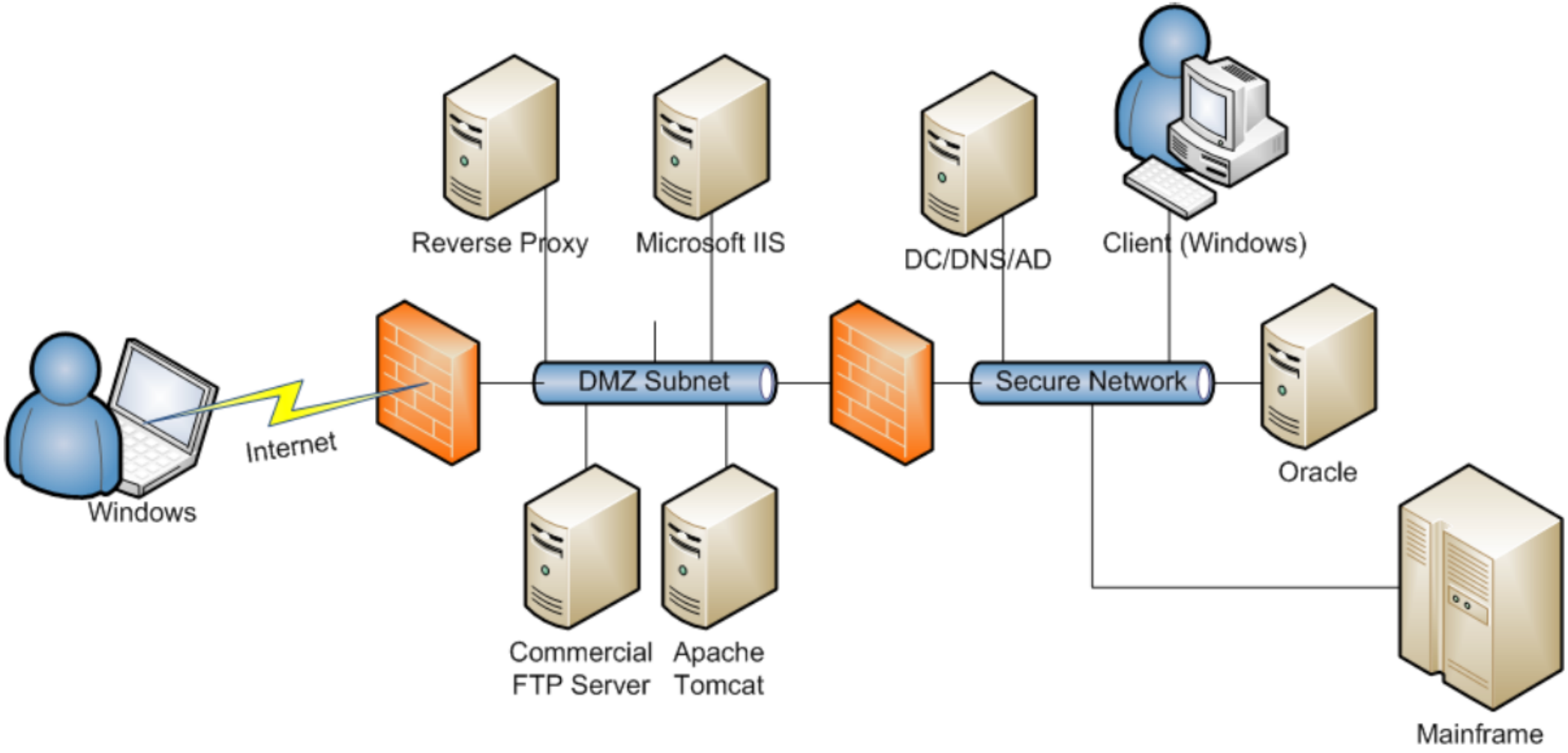
Themen

- JUnit
- **Fuzzing**
  - Historie
  - Fuzzer-Arten
  - **Fuzzing**
- Spike

Der Prozess des Fuzzing

1. Ziele identifizieren
2. Eingaben identifizieren
3. Fuzz-Daten generieren
4. Fuzz-Daten ausführen
5. Auf Ausnahmen überwachen
6. Verwertbarkeit bestimmen

Ziele identifizieren



Prozess des Fuzzing: Beispiel

1. Ziel: Kommerzieller FTP-Server (WARFTP)
2. **Eingaben identifizieren**
3. Fuzz-Daten generieren
4. Fuzz-Daten ausführen
5. Auf Ausnahmen überwachen
6. Verwertbarkeit bestimmen

Prozess des Fuzzing: Beispiel

1. Ziel: Kommerzieller FTP-Server (WARFTP)
2. **Eingaben identifizieren**

TCP Port 21, p1, p2 (PASV or active?)

Commands: USER, PASS, CWD, DELE, etc

Spezielle Chars: \r \n, <space>

Prozess des Fuzzing: Beispiel

1. Ziel: Kommerzieller FTP-Server (WARFTP)
2. Eingaben: TCP21, FTP Kommandos, Spezielle Chars, binäre Dateien
3. **Fuzz-Daten generieren**

USER <username>

Generation one: 1024 x A

Generation two: 2048 x A

Generation three: 4096 x String (random)

PASS <password>

Generation one: 1024 x A

Generation two: 2048 x A

Prozess des Fuzzing: Beispiel

1. Ziel: Kommerzieller FTP-Server (WARFTP)
2. Eingaben: TCP21, FTP Kommandos, Spezielle Chars, binäre Dateien
3. Fuzz-Daten: <username>, <password>
4. **Fuzz-Daten ausführen**

```
for (int w=0; w<maxIterations; w++) {
    openhost();
    for (int commandIndex=0; commandIndex < commandCount; commandIndex++)
    {
            userInput = commands[commandIndex].command;
            if (commands[commandIndex].argument.equals("%f")){
                // FUZZ IT
                if (AttackID != 4) {
                    UserInput = userInput + stringAttacks[w].replace(„\r", „").replace("\n", "");
                } else {
                    UserInput = userInput + stringAttacks[w];
                }
            … vollständig: nächste Folie
```

Prozess des Fuzzing: Beispiel


```
for (int w=0; w<maxIterations; w++) {
    openhost();
    for (int commandIndex=0; commandIndex < commandCount; commandIndex++)
    {
            userInput = commands[commandIndex].command;
            if (commands[commandIndex].argument.equals("%f")){
                // FUZZ IT
                if (AttackID != 4) {
                    UserInput = userInput + stringAttacks[w].replace("\r", "").replace("\n",
                } else {
                    UserInput = userInput + stringAttacks[w];
                }
            } else {
                UserInput = userInput + " " + commands[commandIndex].argument;
            }
            UserInput = userInput + "\r\n";
            Try {
                toServer.write(userInput.getBytes(),0,userInput.getBytes().length);
            } catch (Exception e){
                System.out.println("Connection dropped on write");
            }
```

Prozess des Fuzzing: Beispiel

1. Ziel: Kommerzieller FTP-Server (WARFTP)
2. Eingaben: TCP21, FTP Kommandos, Spezielle Chars, binäre Dateien
3. Fuzz-Daten: <username>, <password>
4. Fuzz-Daten an Ziel gesendet (Code)
5. Den Socket auf Ausnahmen überwachen.

```
try {
     response = in.readLine();
     if(!response.equals("")){
          // do nothing
     }
} catch (IOEcception e) {
     System.out.println("ANOMALY: Connection was dropped.");
     if(AttackID != 4) System.out.println("ANOMALY: String length was " + stringAttacks[w].replace("\
", "").replace("\n", "").length());
} catch (Exception e) ...
```

Prozess des Fuzzing: Beispiel

1. Ziel: Kommerzieller FTP-Server (WARFTP)
2. Eingaben: TCP21, FTP Kommandos, Spezielle Chars, binäre Dateien
3. Fuzz-Daten: <username>, <password>
4. Fuzz-Daten an Ziel gesendet (Code)
5. Den Socket auf Ausnahmen überwachen
6. **Verwertbarkeit bestimmen**

   ist Abhängig von...

Prozess des Fuzzing: Beispiel

- Verwertbarkeit bestimmen – über das Netzwerk
  - Man muss wissen, welche Daten man gesendet hat.
    - Es sollten alle Fuzz-Strings gespeichert und Ausnahmen registriert werden.
    - Network Captures (Wireshark)
  - Szenario reproduzieren
  - Ist es eine Speichermanipulations-Schwachstelle)„memory corruption bug”) ?
  - Ist es eine Schwachstelle der Anwendungslogik („application logic flaw”) ?
- Verwertbarkeit bestimmen – Lokal
  - Einen Debugger einsetzen

Fuzzing-Logistik

- „A good fuzzer needs to allow a user to quickly narrow down the iteration that caused the crash."

  stryde_hax

  - Protokolliere alle Fuzz-Versuche.
  - Der letzte vor eine Anomalie (Ausnahme) ist der beste um zu starten.

Fuzzing-Logistik

- Die Herausforderung der Reproduzierbarkeit
  - Was passiert, wenn man seit zwei Tagen eine Fuzzing Übung ausführt und dann einen Fehler findet.
  - Wie kann man schnell das Szenario reproduzieren, das den Crash verursachte?

Themen

- JUnit
- Fuzzing
- **Spike**

Protokoll-Fuzzing

- Finde so viele Daten wie möglich über die Anwendung.
  - Google ist dein Freund.
  - Vielleicht hat es schon jemand gefuzz't.
  - Vielleicht benutzt es ein Standardprotokoll.

Protokoll-Fuzzing

- Welches ist die Transportschicht?
  - TCP oder UDP?
    - Auswirkung auf die Anomalieerkennung

Protokoll-Fuzzing

- Welcher Protokolltyp
  - SIMPLE
    - Textbasiert
  - COMPLEX
    - Binär

Protokoll-Fuzzing

- Müssen wir uns authentisieren?
  - Welches Authentisierungsprotokoll?
- Eingrenzung der Untersuchung
  - Z.B. nur den Protokoll-Teil vor der Authentisierung

Protokoll-Fuzzing

- Das Protokoll re-engineeren
  - Generiere und überwache Protokoll-Kommunikation.
  - Benutze wireshark
  - Stelle Syntax auf (authentisiere zuerst, dann command1, gefolgt von command2)
  - Stelle eine Liste von Befehlen auf
  - Stelle eine Liste von Argumenten auf

Protokoll-Fuzzing

- Das Protokoll re-engineeren
  - Konstruiere Befehlsprototypen

    `<argument> : erforderlich`

    `[argument] : optional`

    `{CONSTANT1|CONSTANT2 …}: Erfordert constant argument`
- Beispiel:
  - PASS {SYS | USER <Username>} <Password>

Protokoll-Fuzzing

- Wenn man einmal verstanden hat, wie man mit einem Service kommuniziert, kann man Pakete zum Service senden.
- Einfache Protokolle
  - Benutze telnet, nc.exe, openssl
- Komplexe Protokolle
  - Schreibe den Code

Protokoll-Fuzzing

- Jetzt wo Sie mit dem Protokoll kommunizieren können...
- Fuzzing Strategie
  - Wie würden Sie es fuzzen?

- Was können Sie in diesem Prototyp fuzzen?
  - PASS {SYS | USER <Username>} <Password>

Protokoll-Fuzzing

- Fuzzing ist repetitiv:
  - Öffne/Schließe Verbindungen zum Host
  - Konstruiere ein UDP Packet
  - Schreibe Daten auf ein Socket
  - Lese Daten von einem Socket
  - Wiederhole durch eine Sequenz
  - Fuzze jeden Parameter
  - etc.

SPIKE

- SPIKE ist ein Fuzzingframework/API, das die Rahmenfunktionalitäten von Netzwerkprotokoll-Fuzzern anbietet.
  - Geschrieben in C von Dave Aitel

SPIKE

- Einfaches textbasiertes Protokollfuzzing
  - line_send_tcp.c
    - Akzeptiert ein „script“ von SPIKE Befehlen
    - Beispiel:

```
s_string_variable("PASS");
s_string(" ");
s_string_variable("USER");
s_string(" ");
s_string_variable("devel_user");
s_string(" ");
s_string_variable("secretpassword");
s_string("\r\n");
```

SPIKE

- Einfaches textbasiertes Protokollfuzzing
  - line_send_tcp.c
    - ./line_send_tcp <IP> <PORT> script.spk 00

SPIKE

- SPIKEs wahrer Wert:
  - Komplexe Protokolle haben Längen- und Datenfelder
  - Längenfelder zu verfolgen während man Datenfelder fuzzed ist kompliziert
  - SPIKE übernimmt dies
  - Block-basierte Protokollrepräsentation

SPIKE

- Was ist ein SPIKE?
  - „Ein SPIKE ist eine einfache Liste von Strukturen, die Informationen zu Blockgrößen und eine Byte-Queue besitzt.“

```
s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);
```

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- Schiebe 4 NULLs in die Byte-Queue (Platzhalter für Länge)
- Dann wird ein neuer Block-Listener mit dem Namen „somepacketdate“ zugeteilt.

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- Das Script durchsucht die Block-Listener nach einem mit dem Namen „somepacketdata“.
- Die Blockzeiger „start“ werden aktualisiert, um die Blockpositionen in der Queue wiederzuspiegeln.

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- 4 Bytes von Daten werden ins Queue geladen

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- Der Block ist beendet und die Größen sind definiert.
- Die orginalen 4 NULL-Bytes werden mit dem dazugehörenden Größenwert aktualisiert.

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

								1008								1000
	41	41	41	41	00	00	00	04	04	03	02	01	00	00	00	00

Block2
morepacketdata

Big Endian Word

Start Pointer: 1008

Block1
somepacketdata

Big Endian Word

Start Pointer: 1000

Zusammenfassung

- Lesson Learned
  - JUnit 3 & 4
  - Fuzzing-Grundlagen
  - Fuzzing-Arten
  - Spike