White-Box Testing of Behavioral Web Service Contracts with Pex

Tool Demo

Nikolai Tillmann
nikolait@microsoft.com

Jonathan de Halleux
jhalleux@microsoft.com

Microsoft Research
One Microsoft Way, Redmond WA 98052, USA

ABSTRACT

A web service exposes a public API that can be accessed by potentially hostile clients over the internet. Pex, a white-box test generation tool for .NET, can automatically create test inputs that cover corner cases of a web service implemented in .NET, simulating a malicous attacker.

Categories and Subject Descriptors

D2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms

Testing, Security

Keywords

testing, unit testing, symbolic execution, web service

1. OVERVIEW

Manual testing of web-services is costly. Tools can automate the required work. Recently, white-box test input generation tools have become popular [3, 2, 4, 9]. In this tool demo, we show how Pex [10, 8], a dynamic symbolic execution tool for .NET, can be used to generate test cases for a web service.

We apply Pex in the context of the Windows Communication Foundation (WCF) [7] library, a programming model for building service-oriented applications, including web services.

2. WEB SERVICE EXAMPLE

A WCF service contract is defined by a .NET interface, whose methods represent the operations that service provides. The contract can be customized using .NET attributes. Each operation may have parameters.

The following is an example of a WCF service contract. This contract describes the signature of the service. Some attributes, e.g. `FaultContract` may indicate exceptional behavior. (We omit visibility modifiers throughout this article.)

```
[ServiceContract]
interface IProperNamesService
{
    [OperationContract]
    [FaultContract(typeof(ProperNameRecord))]
    void AddProperName(
        String properName,
        int ownerID);
    [OperationContract]
    bool IsProperName(String properName);
    [OperationContract]
    int GetTotalByOwner(Int32 ownerID);
    ...
}
```

A WCF service implementation realizes a service contract. The service can then be hosted by a web server.

Web services defined by WCF service contracts have several properties which make them well-suited for automated test-input generation tools:

- By default, all operations performed on a service are serialized by its host. Thus, the test-input generation tool only has to analyze a single-threaded program.
- All parameter types of an operation must be serializable. The consequence is that most operation parameter types are either primitive, or collections of primitive types.
- WCF service contracts hide the complexity of the underlying network protocols. The WCF host takes care of marshalling the data. As a result, a white-box test generation tool like Pex can be directly applied on the service implementation, without having to deal with the involved network protocols.

In this tool demo, we will show how Pex can be used to test the above `IProperNamesService` WCF web service.

3. PARAMETERIZED UNIT TESTING

Pex generates test inputs for *parameterized unit tests* (PUTs) [11, 13]. In the context of a web service, a PUT is simply a method that takes parameters, invokes a sequence of web service operations, and asserts properties of the expected behavior of the operations.

For example, the following PUT describes how the two operations `AddProperName` and `IsProperName` of a web service should relate:

```
[PexMethod]
void AddAndCheckProperName(
    string properName, int ownerID)
{
    ProperNamesService service =
        new ProperNamesService();
    service.AddProperName(properName, ownerID);
    Assert.IsTrue(service.IsProperName(properName));
}
```

Such a PUT is in fact a *behavioral contract* that the web service must fulfill for all possible values of `properName` and `ownerID`.

While Pex has a wizard that can automatically generate basic PUTs for all operations of a WCF web service, meaningful assertions and non-trivial behavioral contracts must be written by a human being.

4. DYNAMIC SYMBOLIC EXECUTION

Pex explores the reachable statements of a PUT using *dynamic symbolic execution* [3, 2]. This technique consists in executing the program, starting with very simple inputs, while performing a symbolic execution in parallel to collect symbolic constraints over inputs, obtained from conditional branches along the execution. Then Pex uses a constraint solver to compute variations of the previous inputs in order to steer future program executions along different execution paths. In this way, all execution paths will be exercised eventually.

Dynamic symbolic execution extends conventional static symbolic execution [5] with additional information that is collected at runtime, which makes the analysis more precise [3].

5. UNIT TEST GENERATION

Pex is a test input generator, but it persists the generated data as executable code in the form of traditional (parameterless) unit tests.

For example, for an implementation of the above service contract, Pex may generate test cases such as the following.

```
[TestMethod]
void AddAndCheckProperName1() {
    this.AddAndCheckProperName("a", 2);
}

[TestMethod]
[PexRaisedException(
    typeof(FaultException<ProperNameRecord>))]
void AddAndCheckProperName2() {
    this.AddAndCheckProperName("a", -214748348);
}
```

Each generated test case calls a PUT with certain arguments. When Pex generated these tests, the first one passed, and the second one raised an exception. Pex' white box analysis found the corner case caused by the argument `-214748348`. At this point, the user can either change the PUT to reflect the actual implementation behavior, or the user can correct the implementation to reflect the specified behavior.

6. LIMITATIONS

Pex can only explore deterministic, single-threaded service implementations, whose compiled code can be instrumented.

7. RELATED WORK

Behavioral contracts for web services have been proposed before, see e.g. [1]. In this tool demo, we wrote behavioral contracts in the same language as the web service implementation, which avoids the potential semantical gap between a specification and an implementation language.

Automatic test generation for web services was previously done by random black-box test generation, see e.g. [6].

The distributed nature of web services makes white-box testing difficult to apply. Pex enables exhaustive testing of each service in isolation by automatic generation of mock object behavior for other web services [12].

The idea of symbolic execution was pioneered by [5]. Dynamic symbolic execution was first suggested in DART [3]. Several related approaches followed [2, 4, 9]. They differ between each other in their target platforms (C programs, Java programs, machine code), and in the extent of their symbolic reasoning capabilities.

8. REFERENCES

[1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web service semantics - wsdl-s version 1.0. `http://www.w3.org/Submission/WSDL-S/`, November 2005.

[2] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. Exe: automatically generating inputs of death. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 322–335, New York, NY, USA, 2006. ACM Press.

[3] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. *SIGPLAN Notices*, 40(6):213–223, 2005.

[4] P. Godefroid, M. Y. Levin, and D. Molnar. Automated whitebox fuzz testing. In *Proceedings of NDSS'08 (Network and Distributed Systems Security)*, pages 151–166, 2008.

[5] J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.

[6] E. Martin, S. Basu, and T. Xie. Automated testing and response analysis of web services. In *Proc. the IEEE International Conference on Web Services (ICWS 2007), Application Services and Industry Track*, pages 647–654, July 2007.

[7] Microsoft Corporation. Windows Communication Foundation. `http://msdn.microsoft.com/en-us/netframework/aa663324.aspx`. [accessed 05-June-2008].

[8] Pex development team. Pex. `http://research.microsoft.com/Pex`, 2007.

[9] K. Sen and G. Agha. CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools. In *CAV*, pages 419–423, 2006.

[10] N. Tillmann and J. de Halleux. Pex – white box test generation for .NET. In *Proc. of Tests and Proofs (TAP'08)*, volume 4966 of *LNCS*, pages 134–153, Prato, Italy, April 2008. Springer.

[11] N. Tillmann and W. Schulte. Parameterized unit tests. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering,*, pages 253–262. ACM, 2005.

[12] N. Tillmann and W. Schulte. Mock-object generation with behavior. In *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 365–368, Washington, DC, USA, 2006. IEEE Computer Society.

[13] N. Tillmann and W. Schulte. Unit tests reloaded: Parameterized unit testing with symbolic execution. *IEEE Software*, 23(4):38–47, 2006.