

SECURITY TESTING OF WSDL-BASED WEB SERVICES WITH FUZZING

Martin A. Schneider
Leon Bornemann

STV, Octobre 19th 2015



MOTIVATION

Web services

1. provide a **permanent attack surface** to the public through the Internet.
2. provide **a lot of information** about their interface via formalized descriptions, e.g. WSDL and XML Schema.
3. are often part of a complex service infrastructures and may constitute a **gateway to the physical world**, e.g. in the logistics domain.
4. process and store **sensitive data**, in particular health data.

Therefore, security testing of web services is a inevitable.

OUTLINE

1. Fuzzing

- Introduction
- Data Fuzzing
- Behavioral Fuzzing

2. Scheduling for Security Testing

- Negative Input Space Complexity Metric

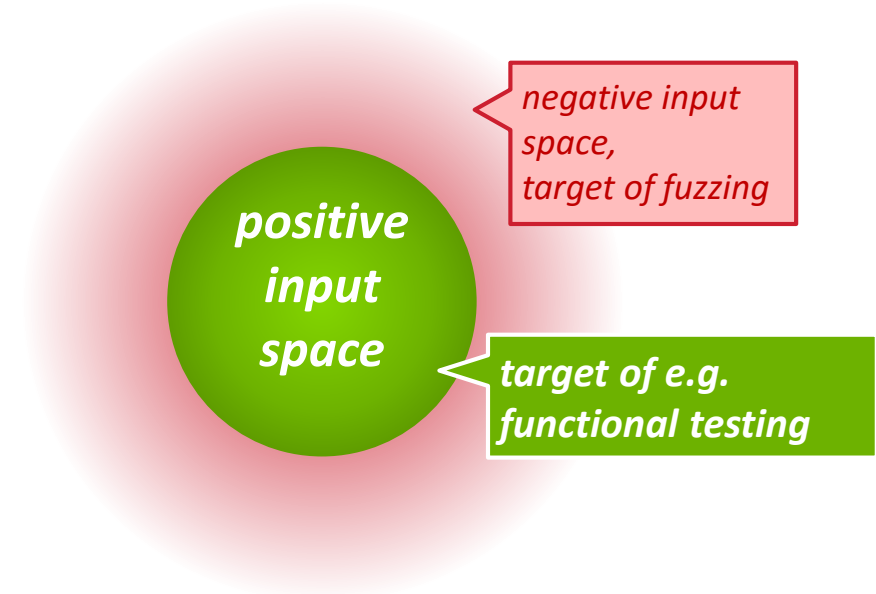
3. Verdict arbitration for Security Testing

FUZZING

FUZZING

Introduction

- Fuzzing is about **injecting invalid or unexpected inputs**
 - to obtain **unexpected behaviour**
 - to identify **errors** and potential **vulnerabilities**
- Interface robustness testing
- Fuzzing is able **to find (0day-) vulnerabilities**, e.g.
 - crashes
 - denial of service
 - security exposures
 - performance degradation
- highly-automated black box approach



INTRODUCTION TO FUZZING: CATEGORIZATION

dumb

- **Random-based fuzzers** generate randomly input data. They don't know nearly anything about the SUT's protocol.
fuzzed input: `HdmxH&k dd#**&%`
- **Template-based fuzzers** uses existing traces (files, ...) and fuzzes some data.
template: `GET /index.html`
fuzzed input: `GE? /index.html, GET /inde?.html`
- **Block-based fuzzers** break individual protocol messages down in static (grey) and variable (white) parts and fuzz only the variable part.

GET	/index.html
-----	-------------

only the (white) part gets fuzzed

fuzzed input: `GET /inde?.html, GET /index.&%ml`

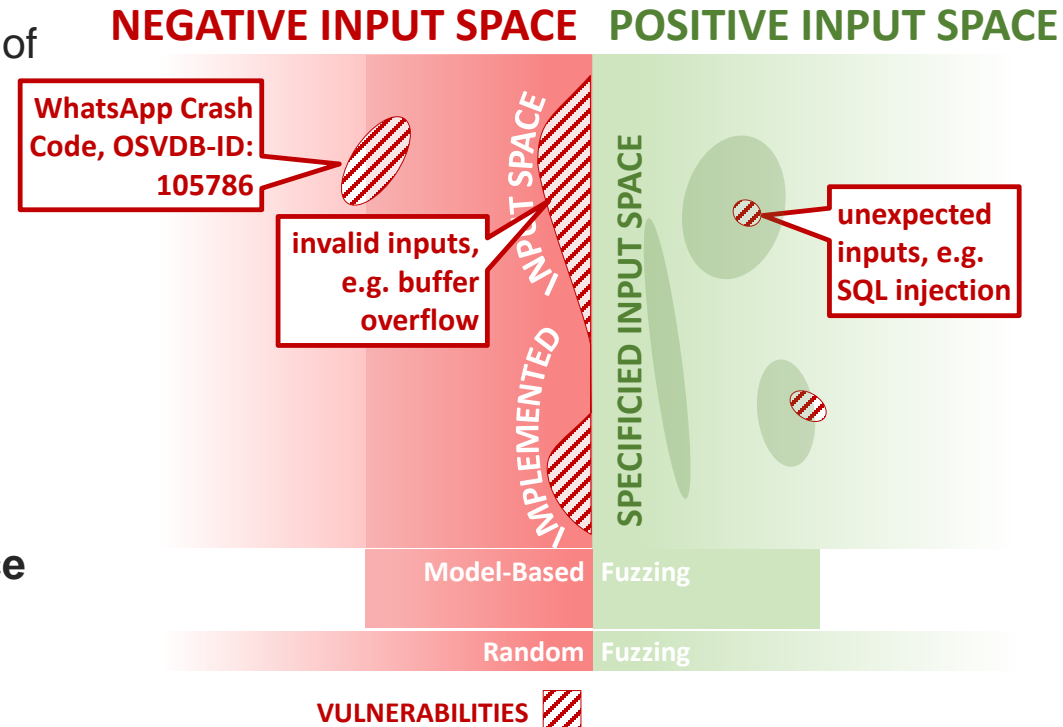
- **Dynamic Generation/Evolution-based fuzzers** learn the protocol of the SUT from feeding the SUT with data and interpreting its responses, for example using evolutionary algorithms.
- **Model-based Fuzzers**

smart

MODEL-BASED FUZZING

Model-Based Fuzzers

- **Model-based fuzzers** uses models of the input domain (protocol models, e.g. context free grammars), for generating systematic non-random test cases
- The model is used to **generate complex interaction** with the SUT.
- Employ fuzzing **heuristics to reduce the input space** of invalid and unexpected inputs
- Model-based fuzzers finds defects which human testers would fail to find.



DATA FUZZING

Fuzzing Library Fuzzino



FUZZINO

- make traditional data **fuzzing widely available**
 - allow an **easy integration into existing tools**
 - **without deep knowledge** about fuzz data generation
- allow data fuzzing **without the need for**
 - **making familiar** with a specific fuzzing tool
 - **integrating further fuzzing tools** into the test process
- approach: didn't reinvent the wheel, **used the potential of existing fuzzing tools**



Peach



Sulley

XSD Type Descriptions



FUZZINO

```
<xsd:simpleType name="String1000Type">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="1000"/>
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="GIAIType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[- !&quot;%&amp;'()*+,. /0-9:;&lt;=&gt;?A-Z_a-z]{4,30}"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="GRAIType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{14}[- !&quot;%&amp;'()*+,. /0-9:;&lt;=&gt;?A-Z_a-z]{0,16}"/>
  </xsd:restriction>
</xsd:simpleType>
```

DATA FUZZING

Fuzzing Library Fuzzino



FUZZINO

- **test case generation on model level**
 - UML profile for data fuzzing
 - automated selection of heuristics
- **test data generation on TTCN-3 level**
 - primitive types with simple constraints, e.g. length
 - based on regular expressions (transformation to grammar)
- **integration with the test execution on TTCN-3 level**
 - external functions constitute the interface to Fuzzino

```
external function initStringRequest(StringRequest stringRequest) return boolean;
```

```
external function hasNextString(charstring requestName) return boolean;
```

```
external function getNextString(charstring requestName) return universal charstring;
```

BEHAVIORAL FUZZING

Data & Behavioural Fuzzing

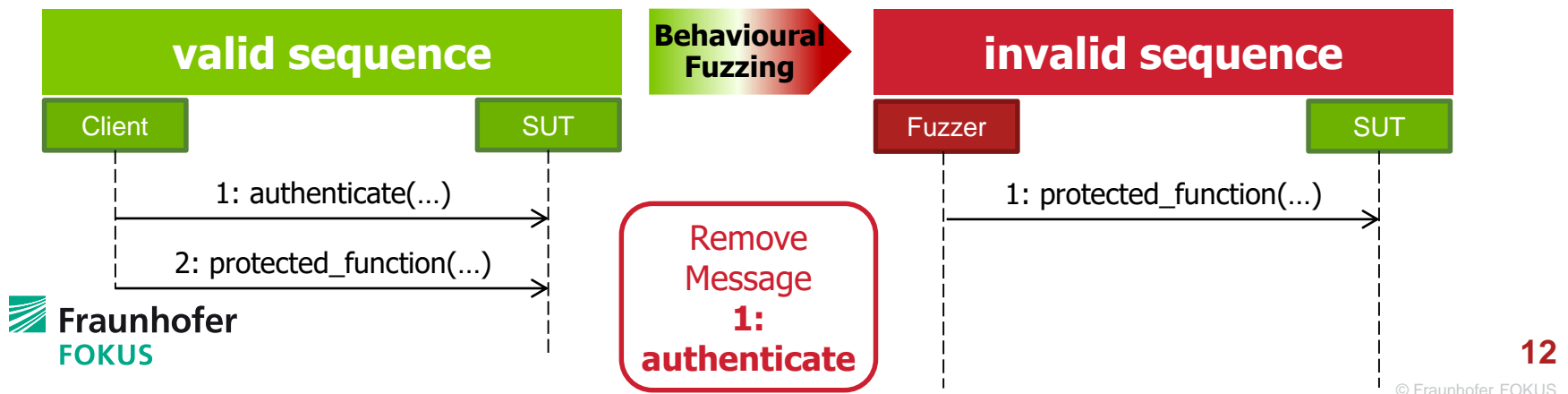
- Traditional **data fuzzing** generates invalid input data to find vulnerabilities in the SUT.
- **Behavioural fuzzing** complements traditional fuzzing by not fuzzing only input data of messages but **changing the appearance and order of messages**, too.
- The **motivation** for the idea of behavioural fuzzing is that vulnerabilities cannot only be revealed when invalid input data is accepted and processed but also when invalid sequences of messages are accepted and processed.
 - *A real-world example is given in [1] where a vulnerability in Apache web server was found by repeating the host message for an HTTP request.*

[1] Takahisa, K.; Miyuki, H.; Kenji, K.: "AspFuzz: A state-aware protocol fuzzer based on application-layer protocols," Computers and Communications (ISCC), 2010 IEEE Symposium on , vol., no., pp.202-208, 22-25 June 2010

MODEL-BASED FUZZING

Model-Based Behavioural Fuzzing

- Test cases are generated by **fuzzing valid sequences**, e.g. functional test cases.
- Behavioural fuzzing is realized by changing the order and appearance of messages in two ways
 - **By rearranging messages directly.** This enables straight-lined sequences to be fuzzed.
 - **By modifying control structures** of UML 2.x sequence diagrams
- **Invalid sequences** are generated by applying fuzzing operators to a valid sequence.



CLASSIFICATION OF FUZZING OPERATORS

	one deviation	a few deviations	many deviations
random	<ul style="list-style-type: none">• remove message• repeat message• change type of message• insert message	<ul style="list-style-type: none">• move message• swap messages	<ul style="list-style-type: none">• permute messages regarding single SUT lifeline• permute messages regarding several SUT lifelines
smart	<ul style="list-style-type: none">• negate interaction constraint• change bounds of loop• change time/duration constraint	<ul style="list-style-type: none">• interchange interaction constraints• disintegrate combined fragment• change interaction operator• move combined fragment	<ul style="list-style-type: none">• remove combined fragment• repeat combined fragment

SCHEDULING FOR SECURITY TESTING

A Negative Input Space Complexity Metric [1]

[1] Schneider, M. A., Wendland, M.-F., Hoffmann, A.: A Negative Input Space Complexity Metric as Selection Criterion for Fuzz Testing. To appear in: 27th IFIP WG 6.1 International Conference ICTSS 2015 Proceeding, ser. LNCS, K. El-Fakih, G. Barlas, N. Yevtushenko, Eds., vol. 9447. Springer, 2015, pp. 1–6 © Fraunhofer FOKUS

NEGATIVE INPUT SPACE COMPLEXITY METRIC

Challenge

- Generally, fuzz test case generation leads to a large number of test cases.
- **How to select the relevant test cases?**

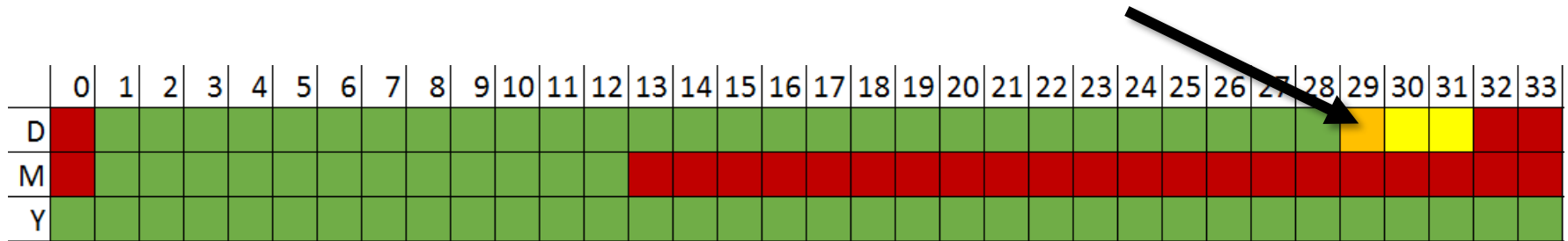
Solution

- **By restricting fuzzing to most error prone parts**, the number of test cases can be reduced to a reasonable set without missing too many security-relevant weaknesses.
- Cataldo et al. [Ca10] investigated **interface complexity, operation argument complexity and error proneness** and found a statistically significant correlation.
- We suppose this correlation holds true for security-relevant errors as well.
- By adapting their metrics to the **negative input space**, an error proneness metric for data fuzzing can be established.

NEGATIVE INPUT SPACE COMPLEXITY METRIC

- Calendar date example

leap year condition: $year \bmod 4 = 0 \vee (year \bmod 100 \neq 0 \vee year \bmod 400 = 0)$



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
D	Red	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green
M	Red	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
Y	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green

- A negative input space complexity metric has to consider the constraints that distinguish valid and invalid input data.

NEGATIVE INPUT SPACE COMPLEXITY METRIC

- A negative input space complexity metric has to consider the constraints that distinguish valid and invalid input data.
- a **static boundary** is defined by an expression that does not contain any variable despite that one whose value shall be decided whether it is valid or not,

e.g. $month > 0 \wedge month < 13$

- a **dynamic boundary** depends on other variables, e.g. parts of a given input data, in order to determine if a provided data is valid

e.g. $day < 29 \wedge month = 2$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
D																																			
M																																			
Y																																			

NEGATIVE INPUT SPACE COMPLEXITY METRIC

- **Dynamic boundary**
 - number of **involved variables**
 - complexity of the constraints measured in terms of **height of the abstract syntax tree**
- examples
 - 1 for the expression $day < 29 \wedge month = 2$
 - 2 for the expression $day < 31 \wedge (month = 4 \vee month = 6 \vee month = 9 \vee month =$

$$m = |b_{stat}| + \sum_{i=1}^{|b_{dyn}|} |vars_i| \cdot height_{AST}$$

NEGATIVE INPUT SPACE COMPLEXITY METRIC: EXAMPLE

$$m = |b_{stat}| + \sum_{i=1}^{|b_{dyn}|} |vars_i| \cdot height_{AST}$$

$$\begin{aligned} & (day < 29 \wedge month = 2) \vee \\ & (day < 31 \wedge (month = 4 \vee month = 6 \vee month = 9 \vee month = 11)) \vee \\ & (day < 32 \wedge (month = 1 \vee month = 3 \vee month = 5 \vee month = 7 \vee \dots)) \end{aligned}$$

leap year: $year \bmod 4 = 0 \vee (year \bmod 100 \neq 0 \vee year \bmod 400 = 0)$

$$|b_{stat}| = 1_{day_lower} + 1_{month_lower} + 1_{month_upper} = 3$$

$$|b_{dyn}| = 1_{day_upper} = 1$$

$$|vars_1| = 1_{month} + 1_{year} = 2$$

$$height_{AST} = 3_{without_leap_years} + 2_{leap_years} = 5$$

$$m = 3 + 2 \cdot 5 = 13$$

EXAMPLES FROM LOGISTICS PILOT

Type name	Expression	Complexity score
String80Type	$length \geq 1 \wedge length \leq 80$	2
GIAIType	$[-!''\&'()*+,\./0-9;,<=>?A-Z_a-z]4,30$	27
GRAIType	$\backslash d\{14\}[-!''\&'()*+,\./0-9;,<=>?A-Z_a-z]4,30$	102

- difference GIAIType and GRAIType results from the **ld**
- interesting starting point for fuzz testing
- **Scheduling of fuzz test cases according to the metric results**

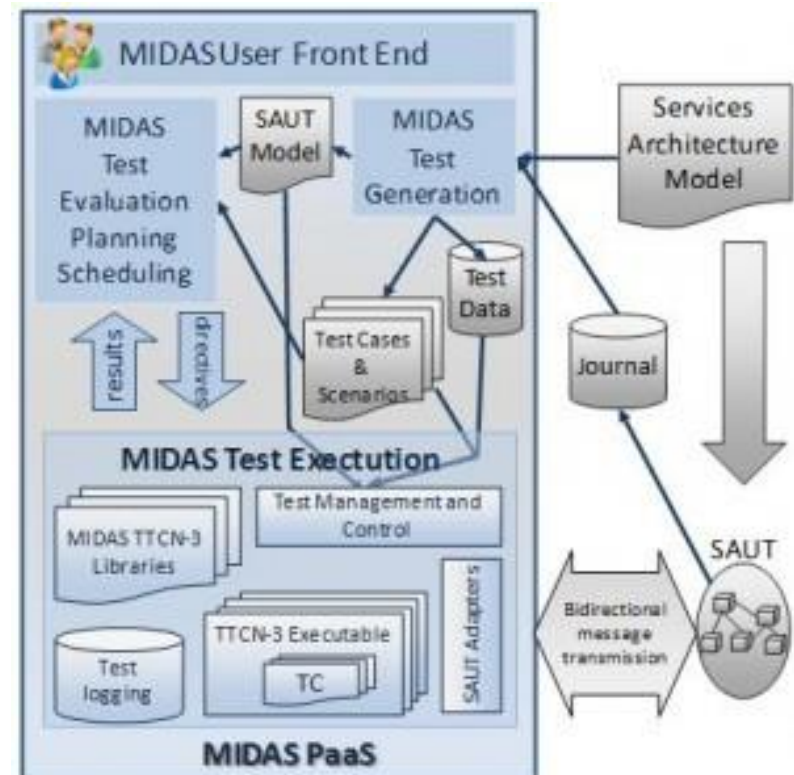
VERDICT ARBITRATION SECURITY TESTING

VERDICT ARBITRATION

- different as done for functional testing
- cannot rely on the response of the SUT: What response can be expected to a malicious stimulus
 - pass: ignored, error message
 - fail: depends on the vulnerability
- partial solution: valid case instrumentation
 - execute functional test case(s) after each fuzz test case
 - select functional test case(s) carefully (false negatives/positives)

EVALUATION ON USE CASES WITHIN MIDAS PROJECT

- The goal of the MIDAS project is to design and build a test automation facility that targets SOA implementations
- Test methods are implemented as services
 - functional testing
 - usage-based testing
 - security testing
- TTCN-3
 - Application and evaluation on case studies
 - Logistics
 - Healthcare



CONTACT

Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
www.fokus.fraunhofer.de

Martin A. Schneider
Scientist
martin.schneider@fokus.fraunhofer.de
Phone +49 (0)30 3463-7383