Security Testing

Eileen Donlon

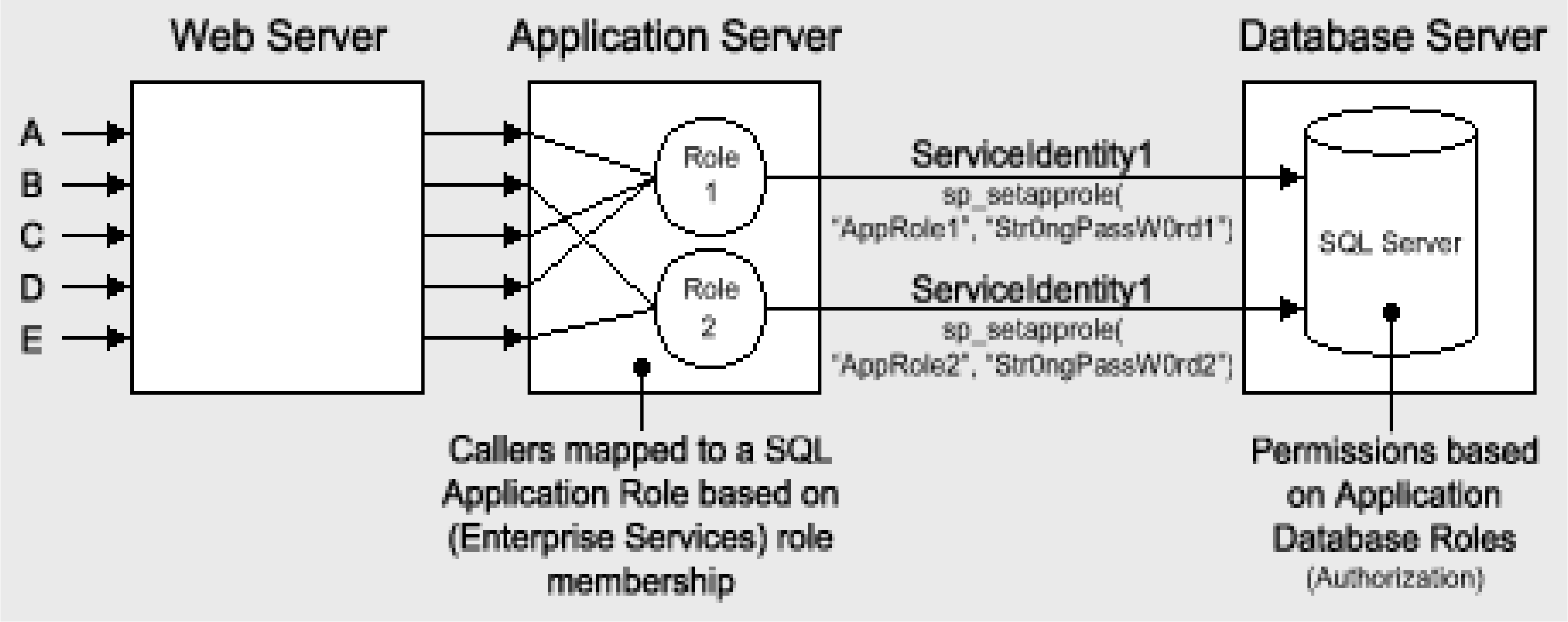
CMSC 737 Spring 2008

Testing for Security

- Functional tests
  - Testing that role based security functions correctly

- Vulnerability scanning and penetration tests
  - Testing whether there are any flaws in the application or configuration that leave the system vulnerable to attack

Role Based Security


Web Server
Application Server
Database Server
A
B
C
D
E
Role 1
Role 2
ServiceIdentity1
sp_setapprole("AppRole1", "Str0ngPassW0rd1")
ServiceIdentity1
sp_setapprole("AppRole2", "Str0ngPassW0rd2")
SQL Server
Callers mapped to a SQL Application Role based on (Enterprise Services) role membership
Permissions based on Application Database Roles
(Authorization)

Web Vulnerability Scanners Compared Fonseca, Vieira and Madeira; 2007

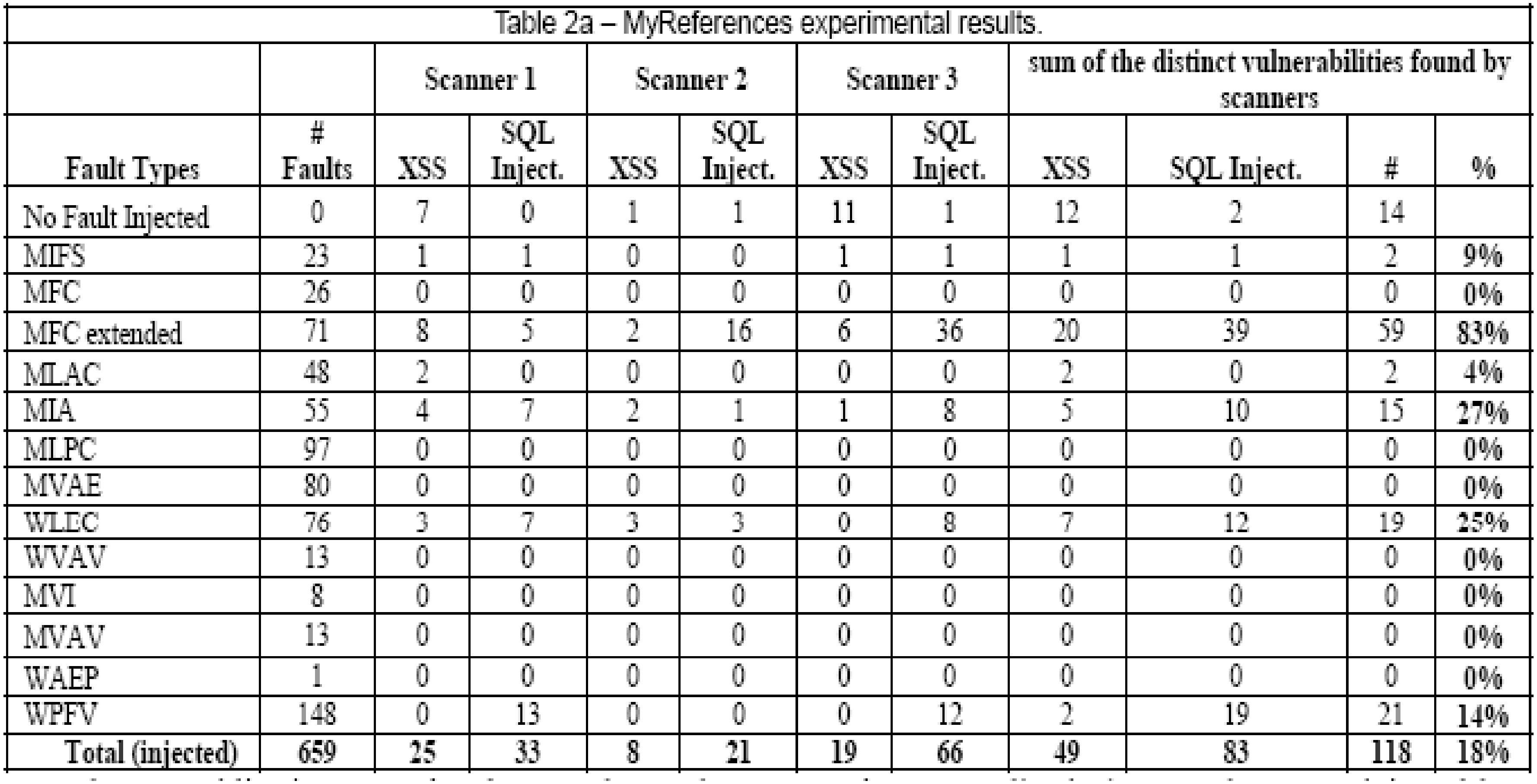


Table 2a – MyReferences experimental results.

		Scanner 1		Scanner 2		Scanner 3		sum of the distinct vulnerabilities found by scanners			
Fault Types	# Faults	XSS	SQL Inject.	XSS	SQL Inject.	XSS	SQL Inject.	XSS	SQL Inject.	#	%
No Fault Injected	0	7	0	1	1	11	1	12	2	14	
MIFS	23	1	1	0	0	1	1	1	1	2	9%
MFC	26	0	0	0	0	0	0	0	0	0	0%
MFC extended	71	8	5	2	16	6	36	20	39	59	83%
MLAC	48	2	0	0	0	0	0	2	0	2	4%
MIA	55	4	7	2	1	1	8	5	10	15	27%
MLPC	97	0	0	0	0	0	0	0	0	0	0%
MVAE	80	0	0	0	0	0	0	0	0	0	0%
WLEC	76	3	7	3	3	0	8	7	12	19	25%
WVAV	13	0	0	0	0	0	0	0	0	0	0%
MVI	8	0	0	0	0	0	0	0	0	0	0%
MVAV	13	0	0	0	0	0	0	0	0	0	0%
WAEP	1	0	0	0	0	0	0	0	0	0	0%
WPFV	148	0	13	0	0	0	12	2	19	21	14%
Total (injected)	**659**	**25**	**33**	**8**	**21**	**19**	**66**	**49**	**83**	**118**	**18%**

Bypass Testing of Web Applications

Offutt, Wu, Du, and Huang

ISSRE, Nov 2004

Bypass Testing

Bypass client side input validation in order to create tests for web application robustness and security

- Allows automated test execution

- Provides access to hidden form fields

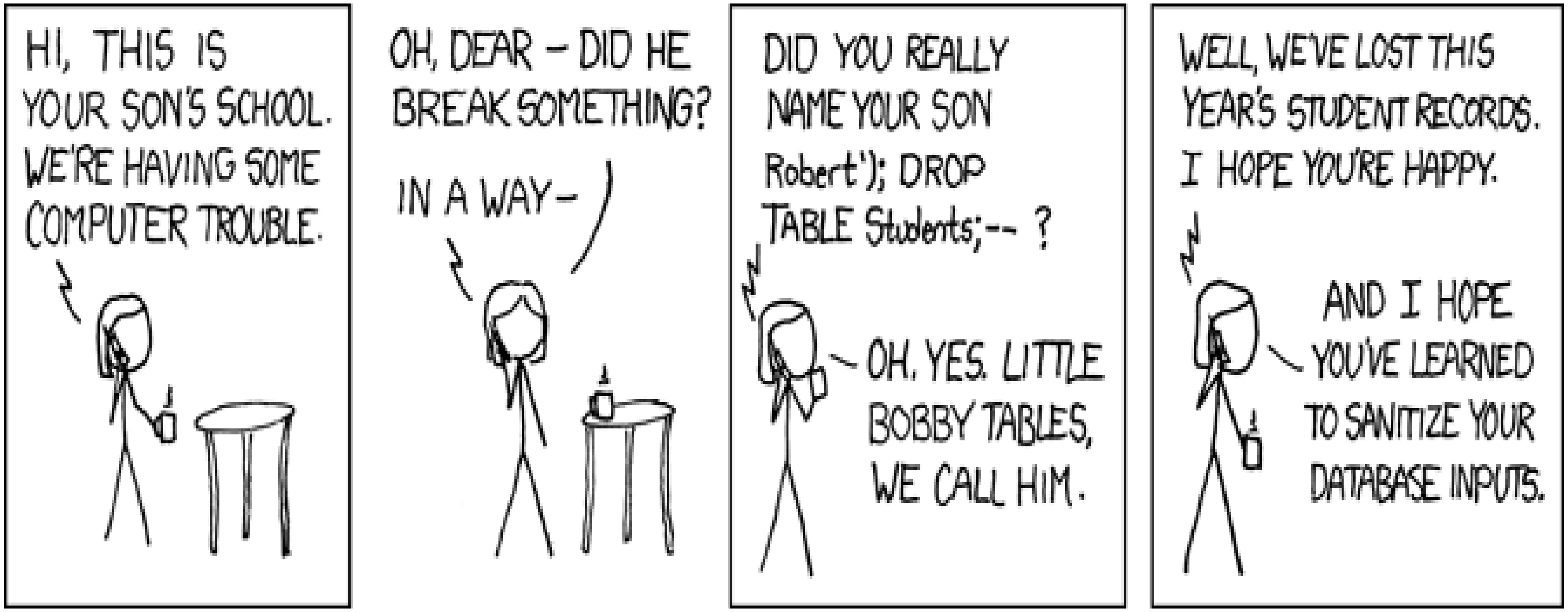
SQL Injection Attack

- Insertion of SQL statements into web applications in order to force a database to modify the database in an unintended way, or to return inappropriate data or to produce an error that reveals database access information.
  - Web forms
  - Web services
- Two factors required:
  - The SQL statement is run in the context of a user with sufficient privileges to execute the attack.
  - Dynamic SQL

Database Security

- Stored Procedures and views can be used to enhance security because permissions to access a view can be granted, denied, or revoked, regardless of the set of permissions to access the underlying table(s).
- Stored procedures and views can be used to conceal the underlying data objects.
- By using stored procedures and view, you can limit the data that is available to a user to a restricted set of the columns and rows instead of querying the entire table.
- ***This does not apply when you use dynamic SQL! Dynamic SQL involves checking permissions on all data objects used in the query.***

SQL Injection Attack


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Types of Client Side Validation

- Semantic Validation

- Syntactic Validation

Semantic Input Validation

- Data type conversion
  - Convert strings to integers
- Data format validation
  - Phone numbers, currency, email addresses
- Inter-value constraints
  - Credit card number and expiration date

Syntactic Input Validation

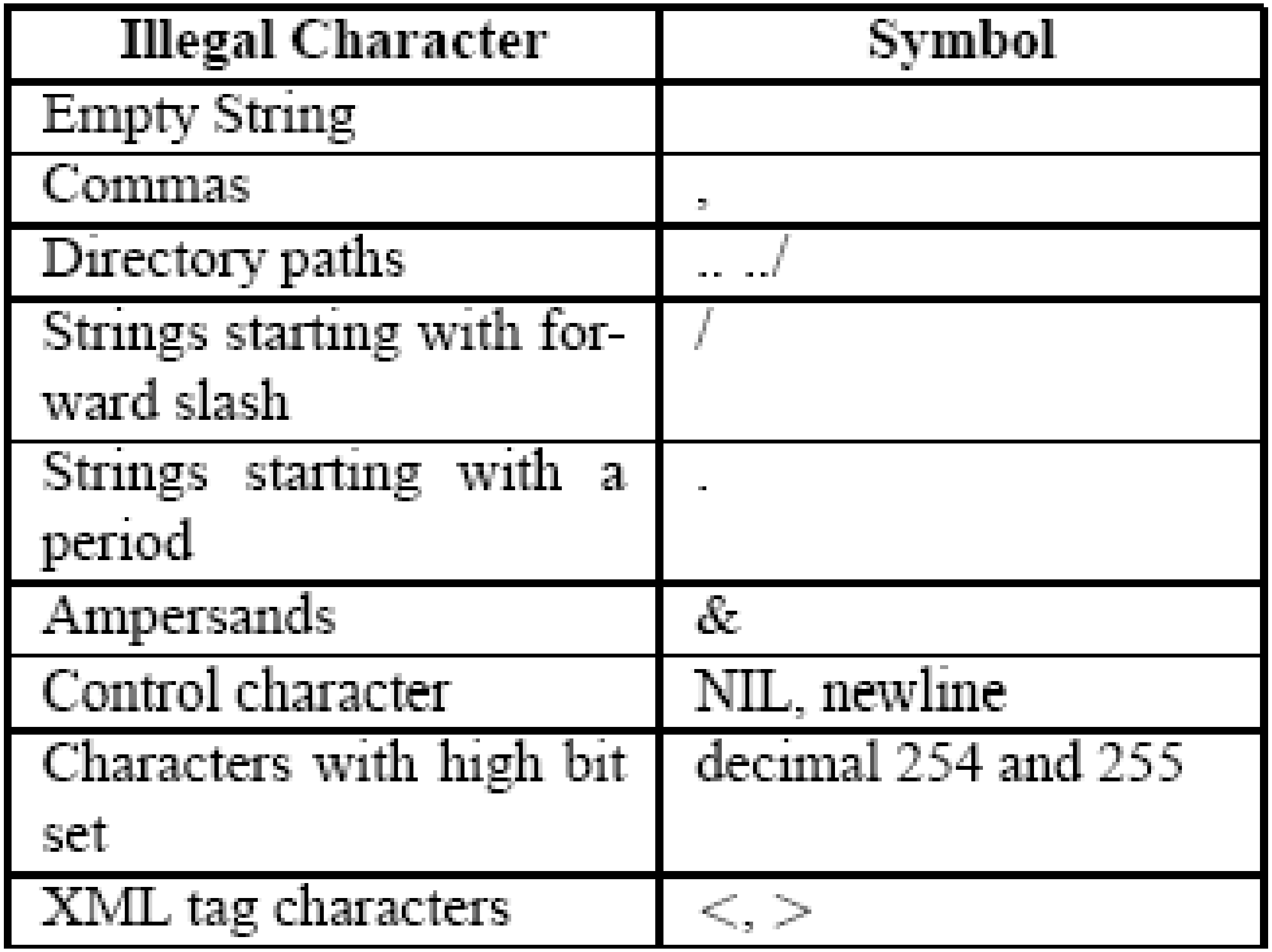
- Built-in length restriction
- Built-in value restriction
  - Pick lists
- Built-in transfer mode
  - HTTP GET or POST
- Built-in data access
  - Hidden Form Fields
  - Cookies

Syntactic Input Validation

- Built-in field selection
  - Pre-defined fields, enabled/disabled
- Built-in control flow restriction
  - Action attributes in FORM tags, links

Server Input Validation

- Numeric limits
- Email addresses
  - Username and valid domain
- URLs
  - Valid form, exist
- Character Patterns
  - Regular expressions
- Character filters



Illegal Character	Symbol
Empty String	
Commas	,
Directory paths	.. ../
Strings starting with forward slash	/
Strings starting with a period	.
Ampersands	&
Control character	NIL, newline
Characters with high bit set	decimal 254 and 255
XML tag characters	<, >

Table 1. Characters that sometimes cause problems for Web applications

Feasibility Study

Can bypass testing be used successfully to test real web applications?

- Cyber Chair, paper submission and reviewing open source web application used by ISSRE
- Black box approach
- Valid user id and access code to enter, saved web pages and modified for bypass testing

Feasibility Study Results

- Submission without authentication
  - Changed action from relative url to complete url
- Unsafe use of hidden field
  - Changed hidden user id field
- Disclosing information
  - Error messages on removing hidden user id field
- No validation for parameter constraint
  - Mismatch between actual and specified file types
- No data type or value validation
  - Negative values, non-integers, etc. as page count

How to do Bypass Testing?

- Static or dynamic web pages
- Possibly multiple forms per page
  - Amazon’s web page had 20 forms and 169 hyperlinks

- Bottom line:
  - Automated input validation needed
  - Facilitated by formal model for html inputs

Model of HTML Input

Input Unit IU = (S, D, T)

- S = Server
- D = set of ordered pairs (n, v), where n is the name and v is the set of values that can be assigned to n
- T = Transfer mode (HTTP GET or POST)

Model of HTML Input

Types of IU

- Form
    - S = Action attribute of Form tag
    - D = Form fields
    - T = Method attribute of Form tag
- Link
    - An anchor <a href="prog?val=1">
        - S = Static html or server program
        - D = Query string
        - T = GET

Composing Input Units

Redundancy on dynamic pages is eliminated through 3 composition rules:

1. Identical IU composition:

- Two IUs $iu_1 = (S_1, D_1, T_1)$, $iu_2 = (S_2, D_2, T_2)$, are identical IFF $S_1 = S_2$, $D_1 = D_2$, and $T_1 = T_{2.}$
- Two identical IUs are merged to form one IU $iu = (S_1, D_1, T_1)$.

Composing Input Units

2. Optional input element composition:

- Two IUs $iu_1 = (S_1, D_1, T_1)$, $iu_2 = (S_2, D_2, T_2)$, have optional elements if $S_1 = S_2$, $T_1 = T_2$, and one input has an element name that is not in the other.
- The two IUs are merged to form one IU

  $iu = (S_1, D', T_1)$, where $D' = \{D_1 \text{ U } D_2\}$

Composing Input Units

3. Optional input value composition:

- Two IUs $iu_1 = (S_1, D_1, T_1)$, $iu_2 = (S_2, D_2, T_2)$, have optional elements if $S_1 = S_2$, $T_1 = T_2$, and there exists $(n_1, v_1) \in D_1$, $(n_2, v_2) \in D_2$ such that $n_1 = n_2$, but $v_1 \neq v_2$
- The two IUs are merged to form one IU

  $iu = (S_1, D', T_1)$, where

  $D' = \{D_1 - (n_1, v_1)\} \cup \{D_1 - (n_2, v_2)\} \cup \{(n_1, (v_1 \cup v_2)\}$

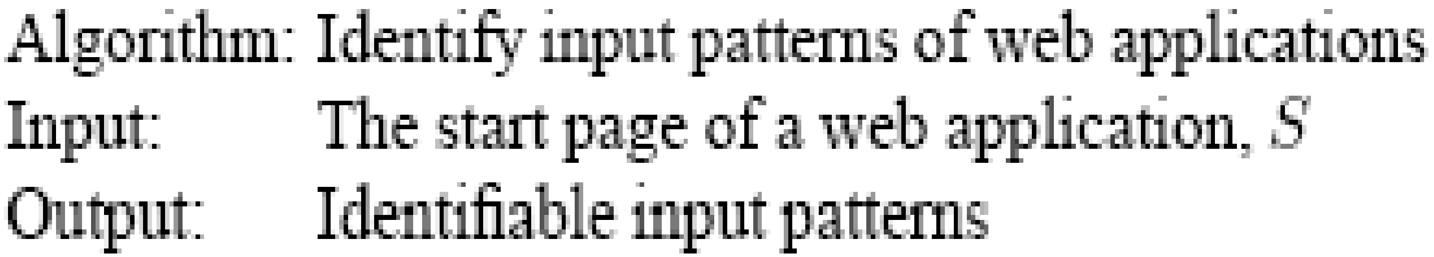
Bypass Testing

- Value Level
  - Addresses data type conversion, data value validation, and built-in value restriction
  - For each input, generate invalid values according to the 14 types of input validation (client + server)
- Examples
  - Modify select to return undefined values
  - Violate value length restriction

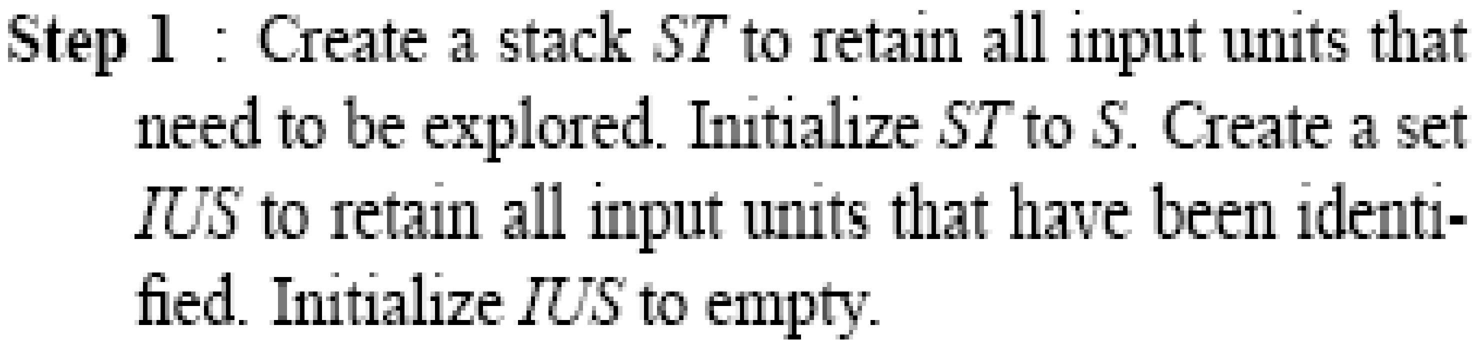
Bypass Testing

- Parameter Level
  - Addresses built-in parameter selection, built-in data access, and inter-value constraints
  - Execute test cases that violate restrictive relationships among parameters
  - Parameter relationships are hard to identify
    - Invalid pair
    - Required pair

Parameter Level Bypass Testing



Algorithm: Identify input patterns of web applications
Input: The start page of a web application, *S*
Output: Identifiable input patterns



Step 1 : Create a stack *ST* to retain all input units that need to be explored. Initialize *ST* to *S*. Create a set *IUS* to retain all input units that have been identified. Initialize *IUS* to empty.

Step 2 : While *ST* is not empty, pop an *input unit* (defined in Section 3) from *ST*, generate data for the input unit and send it to the server. When a reply is returned, analyze the HTML content. For each input unit iu:

- if iu is a link input unit, and iu does not belong to a different server, do **not** push iu onto the stack.
- if $iu \in IUS$ (it has already been found), do **not** push iu onto the stack.
- if there exists an input unit $iu^{'} \in IUS$ such that iu and $iu^{'}$ have optional input elements, update the possible value of iu. Do **not** push iu onto the stack.
- Otherwise, a new input pattern has been identified; add iu to *IUS* as an optional input unit, and then push iu onto *ST*.

Parameter Level Bypass Testing

Results of applying the algorithm are:

- Collection of IUs where D = $\{P_1, P_2, \ldots, P_k\}$ and $P_i = \{(n_1,v_1)_i,(n_2,v_2)_i,\ldots(n_a,v_a)_i\}$. Each P_i is a valid input pattern for the IU.
- Generate invalid input patterns using values from the set of valid values
- Goal is testing relationships among parameters

Parameter Level Bypass Testing

Three types of invalid input patterns:

- Empty input pattern
  - Submits no data
  - Violates all required pairs
- Universal input pattern
  - Submits values for all known parameters
  - Violates all invalid pairs
- Differential input pattern
  - Appropriate values for all parameters in an input pattern + a value for one parameter not in the input pattern

Bypass Testing

Third level is Control Flow Bypass Testing

- Execute test cases that break the normal execution sequence
  - Backward and forward control flow alteration
    - Reverse the order of a transition between 2 UIs
  - Arbitrary control flow alteration

Evaluation

- Small Text Information System (STIS)
  - Mysql database
  - 17 Java server pages, 8 of which process parameterized requests
- 3 Response Types:
  - Invalid inputs recognized and handled
  - Invalid inputs not recognized, abnormal server behavior handled
  - Invalid inputs not recognized, abnormal server behavior exposed to users

Table 2. Failures found for each dynamic component

I: Value Level, No Parameter or Control
II: Parameter Level, No Control Level
III: Control Level, No Parameter Level
IV: Parameter Level and Control Level
T = number of tests, F = number of failures

Component	I		II		III		IV		Total	
	T	F	T	F	T	F	T	F	T	F
login	15	0	2	2	n/a		n/a		17	2
browse	7	4	1	0	1	1	1	1	10	6
record_edit	17	9	5	2	1	1	5	5	28	17
record_delete	5	0	2	0	1	1	2	2	10	3
record_insert	13	9	3	1	1	1	3	3	20	14
categories	12	2	2	0	1	0	2	0	17	2
category_edit	13	2	2	0	1	0	2	0	18	2
register_save	25	11	6	3	1	0	6	6	38	19
Total (#tests & #failures)	107	37	23	8	7	4	21	17	158	66

Results

- Only 55 of 158 tests could have been executed without using bypass testing
    - 9 failures (of 66 total) from these 55 tests

Contributions

- Introduces Bypass testing
- Detailed model for choosing inputs to server side components
- Model supports general input validation testing, and rules are defined for bypass and input validation
- Empirical results from open source conference management system and home grown web apps

Conclusions

- Bypass testing is a novel technique for web application test case generation
- Approach requires no back end source code, only what’s received by a browser
- Complexity of inputs on dynamically generated web forms was handled by the algorithm presented
- Future work: automated form analysis and generation of bypass tests