IBM Zurich Research Laboratory

Preventing Injection Vulnerabilities through Context-Sensitive String Evaluation (CSSE)

Tadeusz Pietraszek <pie@zurich.ibm.com>
Chris Vanden Berghe <vbc@zurich.ibm.com>

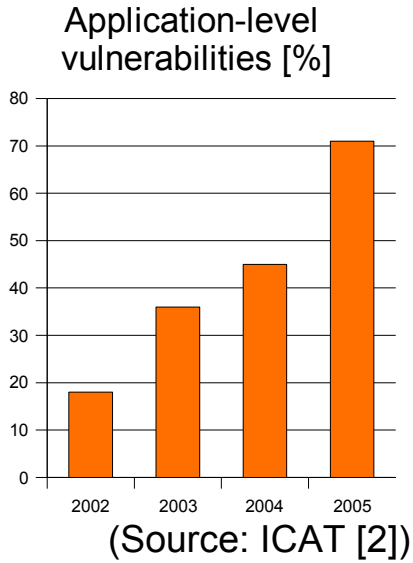
RAID 2005 Symposium
Seattle, Sept 07, 2005

Outline

1. **Motivation**
2. **Injection Vulnerabilities**
3. **Context-Sensitive String Evaluation (CSSE)**
   - Metadata Assignment
   - Context-Preserving String Operations
   - Context-Sensitive String Evaluation
4. **Implementation**
   - Implementation Details
   - Experiments & Results
5. **Discussion and Conclusions**

Motivation: application-level security

- **Shift towards application-level vulnerabilities:**
  - lack of security training:
    - "everybody can program web applications in PHP"
  - lack of security-friendly tooling:
    - use of weakly typed scripting languages



(Source: ICAT [2])

- **Three in OWASP Top Ten [1] are related to injections**
  - A1: Unvalidated input
  - A4: Cross-site scripting (XSS) Flaws
  - A6: Injection flaws

Motivation: our goal

Shift the burden of application-level security from the many application developers to a small team of security-savvy platform developers.

Injection Vulnerabilities: SQL example

- **Programming flaws allowing attacker to manipulate semantics of SQL query by providing input with syntactic content.**
- **Example of SQL injection vulnerability:**

```
$query = "SELECT * FROM users WHERE email ='" .
         $email . "' AND pincode =" . $pincode;
$result = mysql_query($query);
```

- **If pincode =** `"1234 OR 1=1"`
  - Query result becomes independent of pincode and thus circumvents authentication logic
- **Defense can be nontrivial and error prone**
  - complex syntax: e.g., comments, multiple queries,
  - database dependent: MySQL vs. MSSQL vs. DB2
  - context-dependent: strings vs. numeric constants

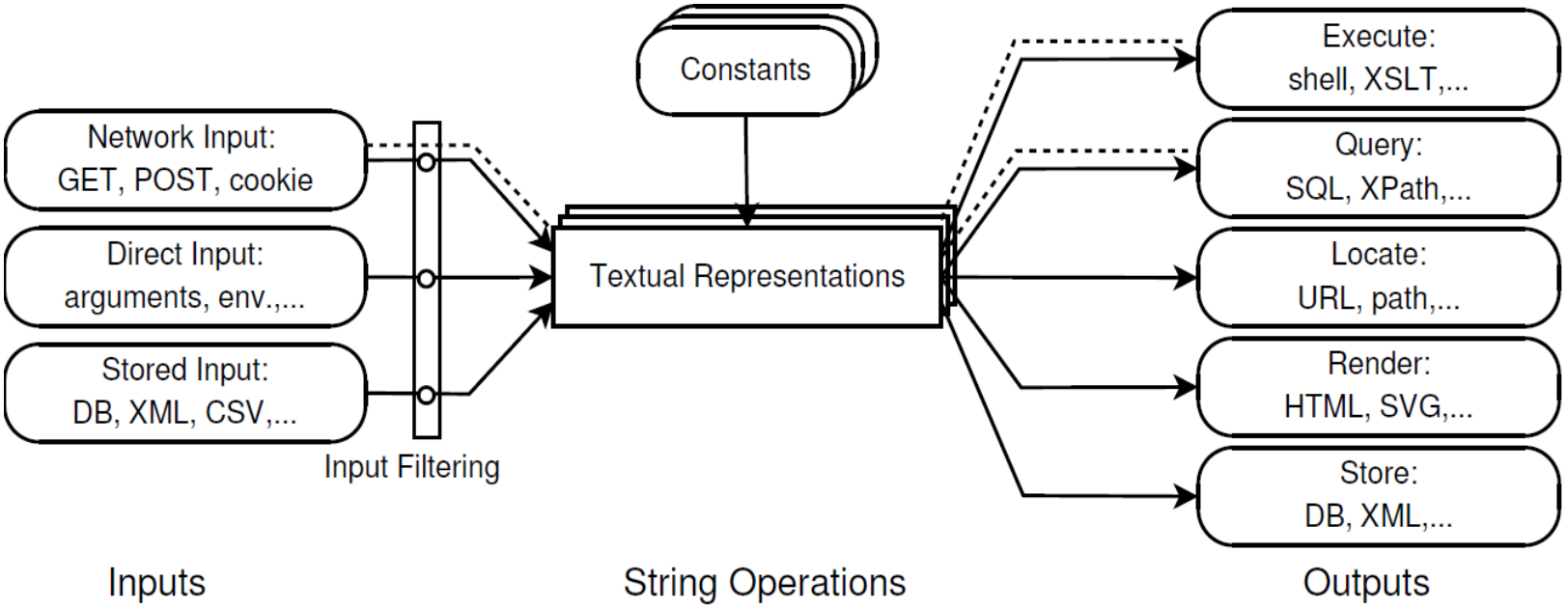
Injection Vulnerabilities: types

- **Injections are the major class of application-level vulnerabilities**

Input \ Output	Execute (e.g., shell, XSLT)	Query (e.g., SQL, XPath)	Locate (e.g., URL, path)	Render (e.g., HTML, SVG)	Store (e.g., DB, XML)
Network input (GET/POST)	**shell inj.** (CAN-2003-0990)	**SQL inj.** (CVE-2004-0035)	**path traversal** (CAN-2004-1227)	**"phishing" through XSS** (CAN-2004-0359)	**preparation for n^{th}-order inj.**
Direct input (arguments)	**command inj.** (CAN-2001-0084)	regexp inj.	local path traversal	PostScript inj. (CAN-2003-0204)	
Stored input (DB, XML)		n^{th}-order SQL inj.		**XSS** (CAN-2002-1493)	preparation for $(n+1)^{th}$-ord. inj.

- **Common varieties include:**
  - SQL injection
  - Cross-site scripting (XSS)
  - Shell injection
- **But there are also:**
  - XPath injection
  - Path traversal
  - Server-side includes (SSI)
  - ...

Injection Vulnerabilities: root cause



- **Prerequisites for injection vulnerabilities:**
  - textual representation of output expressions (channel mixing)
  - ad-hoc serialization of output expressions
  - user-originating data used in the output expressions

Injection Vulnerabilities: defense techniques

- **Safe ad-hoc serialization:**
  - Manual input validation (has to be actively used)
  - Automated input validation (filters) [3] (makes assumptions about the usage of expressions e.g., SQL statement)
  - Perl variable tainting [4] (the programmer has to write a filtering regexp)
  - Static analysis [5] (difficult to apply to languages such as PHP)
- **Serialization APIs:**
  - Prepared statements (SQL)
  - Document Object Model (XML) [6]
  - Xen (XML and SQL) [7]

CSSE: design goals

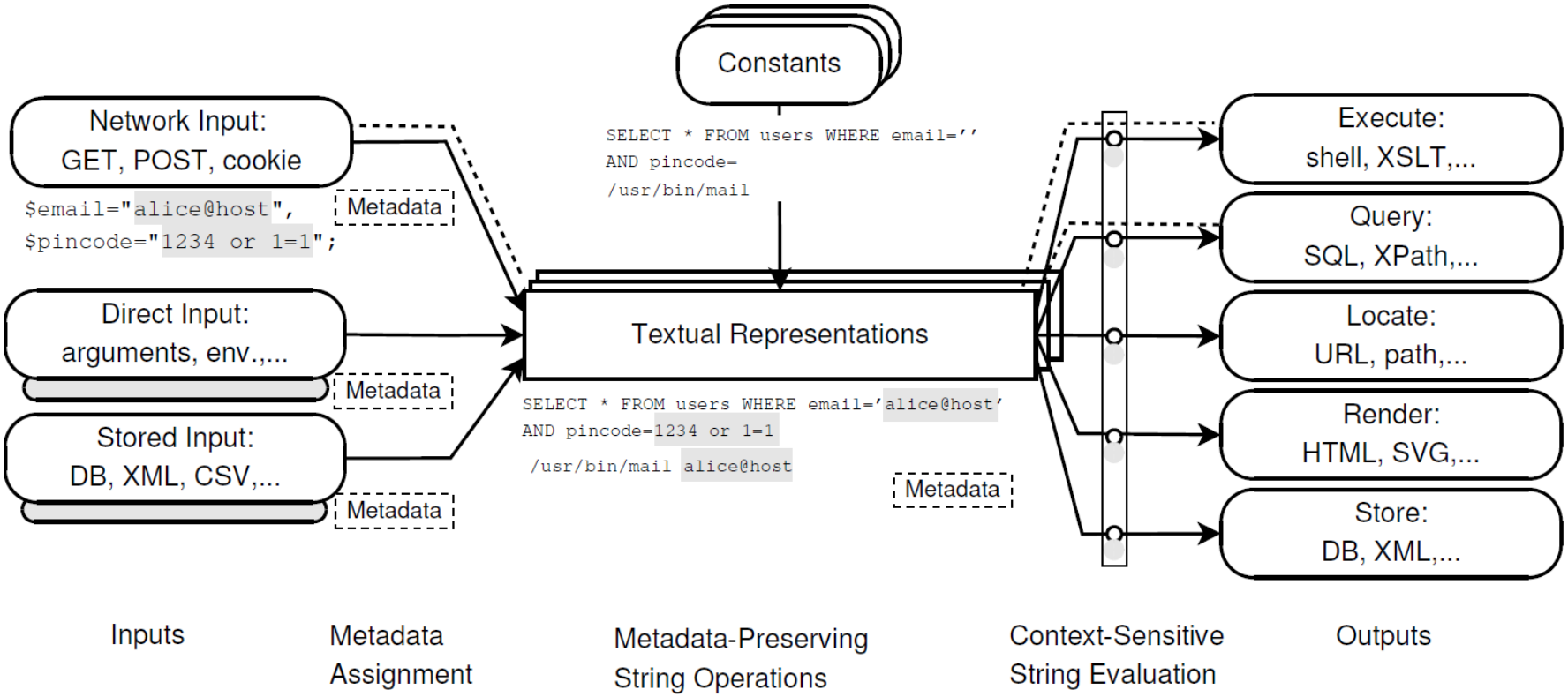
Requirements	Safe ad-hoc serialization	Serializ. APIs
Allow for ad-hoc serialization	✓	✘
Require no developer interaction	✘	✘
Work with legacy code	✓	✘
Non error-prone	✘	✓
Performance overhead	✓	✓

CSSE: what

- CSSE automatically applies the appropriate checks for syntactic content in user-provided input

- Therefore it needs to be able to:
  - distinguish between user- and developer-provided parts of the output expressions
    (1) Metadata assignment to user-provided input
    (2) Metadata-preserving string operations
  - determine the appropriate checks on the user-provided parts before the command execution (e.g. SQL command)
    (3) Context-sensitive string evaluation

CSSE: example



CSSE: metadata assignment

(1) Metadata Assignment

- metadata describes which string fragments are user-provided and which developer-provided
  - basic assumption: user-provided input is untrusted
  - more fine-grained than Perl tainting (~1 bit/variable)

- interception of all input vectors:
  - network input: e.g., HTTP headers
  - direct input: e.g., environment variables
  - stored input: e.g., DB, XML

CSSE: metadata-preserving string operations

(2) Metadata-Preserving String Operations

- all string operations should be intercepted:
  - preserve and update metadata
  - complexity:
    - quite often trivial: copy metadata (e.g., tolower())
    - often easy: merge metadata (e.g., concat())
    - sometimes more difficult (e.g., regexp matching)

(1) + (2) ability to distinguish between user- and developer-provided fragments at any point in creation of the output expressions

CSSE: context-sensitive string evaluation

(3) Context-Sensitive String Evaluation

- all output vectors are intercepted:
  - appropriate validation function for output vector is called
  - validation function will:
    - leverage the metadata to determine untrusted fragments
    - perform limited syntactic analysis of output expression
    - *detect* syntactic content in untrusted fragments (not trivial!)
    - *prevent* by escaping, blocking, removing, ... syntactic content

(1) + (2) + (3) prevention of injection attacks

Implementations: PHP prototype

- **Modification of the original PHP platform v 5.0.2**
- **Focus on the network and MySQL DB input and output vectors**
  - metadata framework:
    - configuration options and metadata repository
  - input vectors:
    - network: GET, POST, COOKIE
    - MySQL database query
  - string operations:
    - most commonly used string operations (e.g., concat, expansion)
  - output vectors:
    - echo, print
    - MySQL database query
- **Total lines of source code added: 587**

Implementations: validation

- **Tested on phpBB, a popular bulletin-board application:**
  - v2.0.x had 12 SQL injection vulnerabilities in BugTraq [8] ...
  - ... of which we managed to reproduce seven (Bugtraq 6634, 9122, 9314, 9942, 9896, 9883, 10722)

- **Example vulnerability (BugTraq 9112):**

```
1 { code }
2 $search_id = (isset($HTTP_GET_VARS['search_id'])) ? $HTTP_GET_VARS[ '
      search_id'] : '';
3 { code }
4   if ( intval($search_id) )
5   {
6     $sql = "SELECT search_array FROM " . SEARCH_TABLE . " WHERE search_id =
          $search_id AND session_id = '". $userdata['session_id'] . "'";
7       if (!($result = $db->sql_query($sql)))
8         { code }
9 { code }
```

Implementations: experimental results

- **Results:**
  - Self-tests passed (as with unpatched)
  - Correct operation of phpBB
  - All SQL injections detected successfully
- **Overhead:**
  - Performance overhead (CGI): ~2-10%
    - the number of affected strings is relatively small
    - efficient lookup
  - Memory overhead: ~2%
    - storing metadata only for strings containing only one unsafe part

Conclusions?

Requirements	Safe ad-hoc serialization	Serializ. APIs	CSSE
Allow for ad-hoc serialization	✓	✘	✓
Require no developer interaction	✘	✘	✓
Work with legacy code	✓	✘	✓
Non error-prone	✘	✓	✓
Performance overhead	✓	✓	✓

Discussion: false positives/negatives

- **CSSE is an efficient intrusion defense method**
- **We identified three scenarios in which FP/FN can occur:**
  - Incomplete implementations
    - not a trivial task, but needs to be done only once
  - Incorrect implementations
    - not a trivial task, but can be done by security savvy experts
  - Invalid basic assumptions
    - modifying the internal representation of string directly
    - second-order injections with database storage may need special care to handle correctly

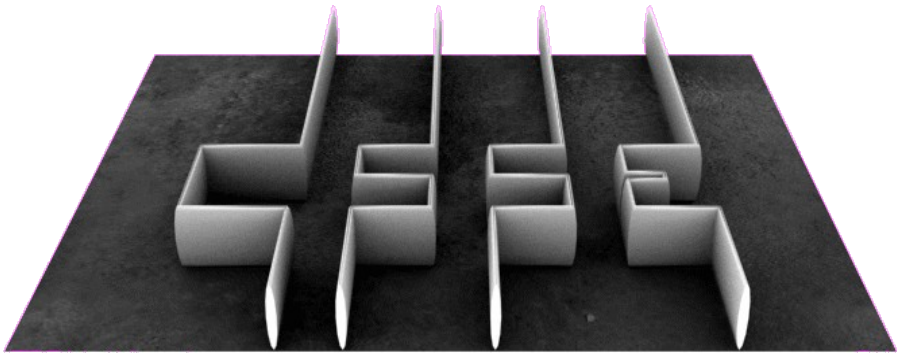
Conclusions

- **A lot of "real ugly" code out there, with a lot of security vulnerabilities - application level security.**
- **Many different security vulnerabilities (SQL injection, XSS, directory traversal,...) are injection vulnerabilities**
- **Need something which addresses injection vulnerabilities holistically and works with legacy code.**
- **CSSE adds metadata to variables, defining its origin, which is subsequently updated throughout their lifetime...**
  - metadata assignment
  - context preserving string operations
- ...and evaluated before passing data to a subsystem
  - context sensitive string evaluation

Conclusions

- **Designed a prototype system for PHP and tested with with a well known application - phpBB**
  - prevented all injections with reasonable performance and memory overhead
- **CSSE is not platform-specific – we have a Java implementation with AspectJ.**
- **You can write CSSE for your favorite platform.**
- **There are cases, where it won't work, but it is just another layer in defense in depth.**

Thank you!



- **Questions?**

- **Further info:**
  - Global Security Analysis Lab @ IBM Research Zurich
    http://www.zurich.ibm.com/csc/infosec/gsal/projects/csse

- **Contact:**

Tadeusz Pietraszek (pie@zurich.ibm.com)
Chris Vanden Berghe (vbc@zurich.ibm.com)

References

[1] OWASP, http://www.owasp.org/documentation/topten.html

[2] ICAT, http://icat.nist.gov/

[3] PHP MagicQuotes, http://www.php.net/

[4] Perl tainting, http://www.perl.com/

[5] Finding Security Vulnerabilities in Java Applications with Static Analysis, V. Livshits and M. Lam, Usenix Security '05

[6] DOM, http://www.w3.org/DOM/

[7] Unifying tables, objects and documents, E. Meijer, W. Schulte, G. Bierman, DP-COOL'03

[8] BugTraq, Security Focus, http://www.securityfocus.com/bid/