Web Application Security: from Static Analysis to Dynamic Protections and Recovery

Miguel Correia

joint work with Ibéria Medeiros, Nuno Neves, Miguel Beatriz, Dário Nascimento,...


ULisboa / IST / INESC-ID

• Universidade de Lisboa – Portugal
  – largest univ. in Portugal; ~50K students; ~460 programs; 18 schools
• Instituto Superior Técnico
  – largest engineering school in Portugal; ~12K students; 80 programs
• INESC-ID
  – large lab in computer science and electrical engineering; 100+ PhDs (most IST faculty); ~250 PhD/MSc students; many research groups
• Distributed Systems Group (GSD)
  – 12 IST faculty, ~30 PhD students, ~40 MSc students, 3 EC projects
Research overview (1)  
Intrusion Tolerance

- To apply the Fault Tolerance paradigm in the domain of Security
- *Do the best we know to protect systems ...but vulnerabilities still remain... so tolerate intrusions that still occur*

![Diagram showing prevention, intrusion, and tolerance.]
Research overview (3)
MinBFT

- First efficient BFT SMR protocol: PBFT (1999)
  - 3f+1 replicas
  - 5 communic. steps
- MinBFT (2009-13)
  - requires local secure component: monotonic counter (simpler than TPM)
  - 2f+1 replicas
  - 4 communic. steps


Research overview (4)
DepSky

- Service: intrusion-tolerant cloud storage
  - Client-side software
  - Server-side are cloud storage services (diversity!)
- Byzantine quorum protocol (consistency) + erasure codes (space) + symmetric cripto (confidentiality)
- Wide-area experiments:
  + availability
  + read speed
  - write speed

A. N. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa,
Overview of my research (5)
Software Security

- **Diversity** is a means to get different vulnerabilities in replicas, mostly in software, but how? This motivated me to understand [software vulnerabilities](#).
- Also **reducing vulnerabilities** is crucial so auditing, static analysis, dynamic protection, secure coding...
- => **Software Security** that is the major topic of this presentation

---

Overview of my research (6)
Software Security

- Older work:
  - Attack injection / fuzzing
  - Vulnerabilities in software ported from 32 to 64-bit CPUs
  - Anomaly-based intrusion detection in web apps
- Teaching a course since 2004
OVERVIEW OF THE PRESENTATION

Outline

1. **WAP**: vulnerability detection with static analysis using taint analysis + classifier

2. **DEKANT**: vulnerability detection with static analysis using a sequence model

3. **SEPTIC**: blocking attacks in the DBMS

4. **SHUTTLE**: intrusion recovery in the cloud
Papers


WAP: ___. Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. IEEE Transactions on Reliability 2016

WAP: ___. Equipping WAP with WEAPONS to Detect Vulnerabilities. DSN 2016

DEKANT: ___. DEKANT: A Static Analysis Tool that Learns to Detect Web Application Vulnerabilities. ISSTA 2016


1

WAP: VULNERABILITY DETECTION WITH STATIC ANALYSIS USING TAINT ANALYSIS + CLASSIFIER
Motivation

- Web applications are exposed to malicious user inputs; if vulnerable, they can be attacked successfully
- “So why do developers keep making the same mistakes? (...) Instead of relying on programmers’ memories, we should strive to produce tools that codify what is known about common security vulnerabilities and integrate it directly into the development process.”
  — David Evans and David Larochelle, Improving Security Using Extensible Lightweight Static Analysis, 2002

Static (source) code analysis

- **Objective:** to find vulnerabilities in the applications’ (source) code automatically  
  - Similar to compiler’s error checking but for vulnerabilities  
  - Similar to manual code reviewing but automatically  
- **Static** because the code is not executed
Generic static analysis tool

WAP: outline

- **Overview**
- Taint analysis
- False positive classification
- Code correction
- The WAP tool
- Results
Vulnerability example (SQLI)

**PHP code:**

```php
$u = $_POST['user'];
$p = $_POST['password'];
$q = "SELECT * FROM users WHERE user='$u' AND pass='$p';"
$r = mysql_query($q);

$q = "SELECT * FROM users WHERE user='" or 1=1 -- ' AND pass='any';"
$r = mysql_query($q);
```
Mechanism 1: Taint Analysis

- Analyses the source code, starting at every **entry point**, propagating **taintedness**, checking if a **sensitive sink** is fed with tainted data

```php
$u = $_POST['user'];
$p = $_POST['password'];
$q = "SELECT * FROM users WHERE user='\$u' AND pass='\$p';"
$r = mysql_query($q);
```

**Vulnerability!**

```php
$uu = mysql_real_escape_string($u);
$pp = mysql_real_escape_string($p);
$q = "SELECT * FROM users WHERE user='$uu' AND pass='$pp';"
$r = mysql_query($q);
```

**OK!**

---

**Challenge: False Positives**

- **False positive:** the analyzer says there’s a vulnerability, but that’s false
  - **Cause:** sanitization function(s) missing from list
  - **Obvious solution:** add missing info to the analyzer
- **How do we know which functions untaint data?**
  - Some are obvious, like `mysql_real_escape_string`
  - Some aren’t, like `substr` or `trim`
Programming

• How do computers “know” how to do something?
• Humans create programs, i.e., sequences of instructions
  – Knowledge is the program plus data (config., DBs)
  – Our case: program = analyser; data = sanitization functions, etc.
• Drawback: humans have first to synthetize this knowledge in a precise way

Machine Learning

• Programs learn automatically from data
  – No need to express knowledge precisely!
  – Human effort can be much smaller
• “We can think of machine learning as the inverse of programming” (Pedro Domingos)
• Extensively used today to solve complex problems
  – voice recognition, natural language translation, playing Jeopardy...
Mechanism 2: Classification

- **Key idea:**
  - for less obvious sanitization functions (or combinations) don’t ask experts, let the tool learn
  - we let the taint analyzer produce false positives, but use a classifier to distinguish true from false
- Classifier works based on a set of examples
  - a user can add more examples to make the tool more precise; no need to program knowledge
  - other tools: user learns function X sanitizes, then codes X
  - our tool: user sees example Y not vulnerable, then adds Y

Mechanism 3: Code Correction

- Correcting vulnerabilities is tiresome and they can be removed mostly automatically using fixes
- Let the tool to do it when it detects a vulnerability
WAP: outline

- Overview
- Taint analysis
- False positive classification
- Code correction
- The WAP tool
- Results

Scheme

```
+-----------------------+  +-----------------+
| PHP source code      |  | parsing code    |
|                      |  | AST             |
|                      |  | vulnerability   |
|                      |  | detector module|
|                      |  | candidate       |
|                      |  | vulnerabilities |
|                      |  +-----------------+
```

ep: entry points
ss: sensitive sinks
san: sanitization functions
WAP: outline

- Overview
- Taint analysis
- **False positive classification**
- Code correction
- The WAP tool
- Results

Key idea

- **Code slice**: sequence of all instructions from an entry point to a sensitive sink that affect data flow
- Key idea: given a code slice in which the taint analyzer detected a vulnerability, **classify** it as vulnerable or not
  - confirming the conclusion of the taint analyzer
  - or saying it was a false positive
- How to distinguish vulnerable from non-vulnerable slices? Using symptoms / **features**
Features for FP classification

- What are the features of the possible existence of a false positive? A symptom exists when the user input is (examples):
  - changed
    - string manipulation functions (e.g., `substr`)
    - concatenation operations
  - validated
    - type checking functions (e.g., `isset`, `is_string`
    - white and black listing
- Features are binary: presence or not of one of these

FP classification: other ingredients

- What do we need for classification?
- A set of features to characterize false positives
- Classification classes; we use two:
  - is a FP (Y); is not a FP (N = real Vulnerability)
- Learning data set of slices annotated as Y or N
  - original set: 76 instances (32 Y, 44 N)
  - obtained manually, tedious
- A classification algorithm: we didn’t select one but defined a process to do the selection
Original learning data set

- 76 instances: 32 false positives + 44 real vulnerabilities
- 15 features, corresponding to 24 symptoms (functions)

<table>
<thead>
<tr>
<th>Potential vulnerability</th>
<th>String manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Webapp</td>
</tr>
<tr>
<td>SQLi</td>
<td>Current Cost</td>
</tr>
<tr>
<td>SQLi</td>
<td>Current Cost</td>
</tr>
<tr>
<td>XSS</td>
<td>Crosssite</td>
</tr>
<tr>
<td>XSS</td>
<td>XSS</td>
</tr>
<tr>
<td>XSS</td>
<td>ZIPREC 0.42</td>
</tr>
<tr>
<td>RFI</td>
<td>DAVKA 1.67</td>
</tr>
<tr>
<td>RFI</td>
<td>SBD</td>
</tr>
<tr>
<td>CoC</td>
<td>DAVKA 1.67</td>
</tr>
<tr>
<td>XSS</td>
<td>avFoundation 15</td>
</tr>
<tr>
<td>XSS</td>
<td>HTTP-0.13</td>
</tr>
</tbody>
</table>

Evaluation of classifiers

- With the WEKA tool we:
  - evaluated 10 machine learning classifiers
    - ID3, C4.5/J48, Random Forest, Random Tree, K-NN, Naive Bayes, Bayes Net, MLP, SVM, and Logistic Regression
  - tested the classifiers with 10-fold cross validation
    - data set divided into 10 buckets, train the classifier with 9 of them and test it with the 10th; repeat the process with every combination (10 times)
  - used 10 metrics to evaluate the classifiers performance
Evaluation of classifiers

• Results for Logistic Regression (the best):

<table>
<thead>
<tr>
<th></th>
<th>Observed</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes (FP)</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>No (not FP)</td>
<td>5</td>
<td>43</td>
</tr>
</tbody>
</table>

– Accuracy = (TP+TN)/(P+N) = 92.1% (instances well classified)
– Precision = TP/(TP+FP) = 96.4% (FP instances well classified)

• Later we repeated this with much more data

Classifiers implemented

• First version: we first implemented LR
• Second version: we implemented a combination of the top 3 classifiers (LR, RT, SVM) (same data set)
WAP: outline

- Overview
- Taint analysis
- False positive classification
- **Code correction**
- The WAP tool
- Results

---

**Code correction**

- **Idea:** when a vulnerability is found, insert a *fix* that does sanitization or validation of the data
  - A fix is just a call to a function that does it
  - Sanitization: escaping metacharacters / metadata
  - Validation: checking the data and executing the sensitive sink or not depending on this verification

- **SQLI example:**
  - fix calls a PHP sanitization function that depends on the DBMS (e.g., *pg_escape_string*)
  - fix inserted in the last write in the query string
Correction of code correction (!)

- We never observed fixes breaking an application functioning, but it’s not impossible
- Solution: regression testing
  - consists in running the same tests before and after program modifications
  - to check if what was working correctly still does
- We did some simple experiments with Selenium

WAP: outline

- Overview
- Taint analysis
- False positive classification
- Code correction
- The WAP tool
- Results
WAP - Web Application Protection

• Does what we saw for PHP: analysis, classification, correction
• Gives feedback:
  – reports vulnerabilities detected and how were corrected
  – outputs a corrected version of the web application
  – reports the false positives identified
• Available online: ~9000 downloads!

WAP

[Diagram showing the process of code analysis, false positive prediction, and code correction.]
Vulnerabilities considered

- Most exploited:
  - SQL Injection
  - Cross Site Scripting (XSS)
- Others:
  - Remote file inclusion
  - Local file inclusion
  - Directory traversal / path traversal
  - Source code disclosure
  - OS command injection
  - PHP code injection

Challenges of implementing WAP

- PHP syntax uncertainty: PHP is not formally specified and poorly documented features are used often
- Environment variables: resolve name of the included files
- Interprocedural, global, context-sensitive, class analysis
WAPe

- Extending static analysis tools to find new vulnerability classes requires programming, its complex and takes time
- Solution: modify WAP to deal with new vulnerability classes defined by the users without programming
- “Equipping WAP with WEAPONS” (WAP extensions)
WAPe: Classifier and data set

- We increased the data set and redone the classifier study:

<table>
<thead>
<tr>
<th>WAP</th>
<th>WAPe</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 features</td>
<td>60 features</td>
</tr>
<tr>
<td>24 symptoms (functions)</td>
<td>60 symptoms (functions)</td>
</tr>
<tr>
<td>data set with 76 instances</td>
<td>data set with 256 instances</td>
</tr>
</tbody>
</table>

Classifiers:
- Support Vector Machine
- Logistic Regression
- Random Tree

Classifiers:
- Support Vector Machine
- Logistic Regression
- Random Forest

WAPe: new vulnerabilities

- LDAP injection (LDAPi)
- XPath injection (XPathI)
- NoSQL injection (NoSQLi)
- Comment spamming (CS)
- Session fixation (SF)
- Header injection / HTTP response splitting (HI)
- Email injection (EI)
- SQLI for WordPress
WAP: outline

- Overview
- Taint analysis
- False positive classification
- Code correction
- The WAP tool
- Results

WAP vs Pixy

- Pixy does taint analysis to detect SQLI and XSS vulnerabilities
WAP vs PhpMinerII

- PhpMinerII predicts the presence of SQLI/XSS vulnerabilities in PHP code (in slices) using a ML classifier
- unlike WAP, it does not identify where vulnerabilities are
- also only SQLI and XSS

<table>
<thead>
<tr>
<th>Observed</th>
<th>Yes (Vuln.)</th>
<th>No (not Vuln.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes (Vuln.)</td>
<td>48</td>
<td>5</td>
</tr>
<tr>
<td>No (not Vuln.)</td>
<td>5</td>
<td>20</td>
</tr>
</tbody>
</table>

**Logistic Regression**
- Accuracy = 87.2%
- Precision = 85.2%

Summary

<table>
<thead>
<tr>
<th>Metric</th>
<th>WAP</th>
<th>Pxy</th>
<th>PhpMinerII</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>92.1%</td>
<td>44.0%</td>
<td>87.2%</td>
</tr>
<tr>
<td>precision</td>
<td>92.5%</td>
<td>50.0%</td>
<td>85.2%</td>
</tr>
</tbody>
</table>
## WAP with all vulnerability classes

<table>
<thead>
<tr>
<th>Webapp</th>
<th>Detected taint analysis</th>
<th>Detected data mining</th>
<th>Corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQLI</td>
<td>RFI, LFI</td>
<td>SC</td>
</tr>
<tr>
<td>currentcost</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DVWA 1.0.7</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>emoncms</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MeasureIt 1.14</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mfn 0.13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mutillidae 2.3.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OWASP Vicnun</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SRD(1)</td>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Wackopico</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ZIPEC 0.32</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>22</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

## WAP totals

- **1.38 MLOCs**
- **388 vulnerabilities**

### Webapp Line of Code (LOC) by Vulnerability

<table>
<thead>
<tr>
<th>Web application</th>
<th>Files</th>
<th>Lines of code</th>
<th>Analyze time (s)</th>
<th>Val files</th>
<th>Val found</th>
<th>FP</th>
<th>Real val</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminer-1.1.0</td>
<td>45</td>
<td>3,435</td>
<td>27</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Butterfly Installer 0.6</td>
<td>36</td>
<td>2,946</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Checkmark</td>
<td>18</td>
<td>2,244</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>codebox</td>
<td>3</td>
<td>270</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>coremetrics</td>
<td>5</td>
<td>1,099</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DAVWA 1.0.5</td>
<td>310</td>
<td>31,405</td>
<td>15</td>
<td>12</td>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>emoncms</td>
<td>70</td>
<td>6,876</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>gallery2</td>
<td>694</td>
<td>124,414</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ghost</td>
<td>199</td>
<td>42,126</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>gitrew-pip</td>
<td>14</td>
<td>328</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GooPHP</td>
<td>82</td>
<td>4,835</td>
<td>10</td>
<td>3</td>
<td>3</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>Ifconfig 0.4</td>
<td>11</td>
<td>1,648</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Horde-0.7</td>
<td>447</td>
<td>38,704</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Listbuilder-1.0.25+1.6</td>
<td>150</td>
<td>3,700</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MeasureIt 1.14</td>
<td>2</td>
<td>967</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Mfn 0.13</td>
<td>7</td>
<td>8,859</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Mutillidae 1.3</td>
<td>18</td>
<td>1,623</td>
<td>6</td>
<td>10</td>
<td>10</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Mutillidae 2.3.3</td>
<td>578</td>
<td>105,367</td>
<td>65</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>NetBiRD-alpha</td>
<td>620</td>
<td>100,139</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>ovproxy 0.2</td>
<td>4</td>
<td>243</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OWASP Vicnun</td>
<td>22</td>
<td>814</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>phpMDB 0.3</td>
<td>100</td>
<td>11,079</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pringae</td>
<td>10</td>
<td>398</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>PHP X-Tempalte 0.4</td>
<td>10</td>
<td>2,009</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPB 1.4.6</td>
<td>62</td>
<td>20,796</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Physio 1.2.2</td>
<td>6</td>
<td>221</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHD</td>
<td>6</td>
<td>621</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PhPy     1.0.1</td>
<td>9</td>
<td>618</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPFusion</td>
<td>603</td>
<td>27,000</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>phpsqlania-1.2.3</td>
<td>97</td>
<td>24,001</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>phpList 1.4</td>
<td>73</td>
<td>13,262</td>
<td>30</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>PHPMA/Alpha 2.0.5</td>
<td>40</td>
<td>6,730</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPM/Alpha 2.0.5</td>
<td>34</td>
<td>6,830</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPM/Alpha 2.1.6+1</td>
<td>287</td>
<td>163,171</td>
<td>105</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Phantomial 1.33</td>
<td>13</td>
<td>2,463</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DAMATE</td>
<td>22</td>
<td>332</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Thwki 1.6</td>
<td>1,363</td>
<td>490,313</td>
<td>14</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>webcrawler</td>
<td>43</td>
<td>3,883</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WachPacker</td>
<td>57</td>
<td>4,156</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>WebCredite</td>
<td>129</td>
<td>36,525</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Webcheck 1.0</td>
<td>27</td>
<td>5,596</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Webscripts</td>
<td>50</td>
<td>391</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>WordPress 2.0</td>
<td>215</td>
<td>44,254</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>ZIPEC 0.32</td>
<td>30</td>
<td>765</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>7,860</td>
<td>1,262,636</td>
<td>357</td>
<td>174</td>
<td>174</td>
<td>431</td>
<td>431</td>
</tr>
</tbody>
</table>
**WAPe totals**

<table>
<thead>
<tr>
<th>Web application</th>
<th>Version</th>
<th>Files</th>
<th>Lines of code</th>
<th>Analysis time (s)</th>
<th>Vuln. files</th>
<th>Vuln. found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin Control Panel Lite 2</td>
<td>0.10.2</td>
<td>14</td>
<td>1,984</td>
<td>1</td>
<td>9</td>
<td>81</td>
</tr>
<tr>
<td>Anywhere Board Games</td>
<td>0.150/25</td>
<td>3</td>
<td>501</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Clip Bucket</td>
<td>2.7.0.4</td>
<td>597</td>
<td>148,129</td>
<td>11</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>Clip Bucket</td>
<td>2.8</td>
<td>606</td>
<td>149,830</td>
<td>12</td>
<td>18</td>
<td>26</td>
</tr>
<tr>
<td>Community Mobile Channels</td>
<td>0.2.0</td>
<td>372</td>
<td>119,890</td>
<td>8</td>
<td>116</td>
<td>47</td>
</tr>
<tr>
<td>divine</td>
<td>0.1.3a</td>
<td>5</td>
<td>706</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>ldap address book</td>
<td>0.22</td>
<td>18</td>
<td>4,615</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Minutes</td>
<td>0.42</td>
<td>19</td>
<td>2,670</td>
<td>1</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Mle Moodle</td>
<td>0.8.8.5</td>
<td>235</td>
<td>59,723</td>
<td>18</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Flip Open Chat</td>
<td>3.0.2</td>
<td>249</td>
<td>83,899</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>pivocx</td>
<td>2.3.10</td>
<td>254</td>
<td>108,893</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Play sms</td>
<td>1.3.1</td>
<td>1,420</td>
<td>248,875</td>
<td>19</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>RCR AEir</td>
<td>0.11a</td>
<td>8</td>
<td>396</td>
<td>1</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>refbase</td>
<td>0.8.6</td>
<td>171</td>
<td>109,600</td>
<td>10</td>
<td>18</td>
<td>48</td>
</tr>
<tr>
<td>SAE</td>
<td>1.1</td>
<td>150</td>
<td>47,207</td>
<td>7</td>
<td>39</td>
<td>48</td>
</tr>
<tr>
<td>Tomahawk Mail</td>
<td>2.0</td>
<td>155</td>
<td>16,742</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>vfront</td>
<td>0.99.3</td>
<td>438</td>
<td>93,042</td>
<td>15</td>
<td>25</td>
<td>77</td>
</tr>
<tr>
<td>Total</td>
<td>4,714</td>
<td>1,196,702</td>
<td>123</td>
<td>280</td>
<td>413</td>
<td></td>
</tr>
</tbody>
</table>

**WAPe: 0-day vulnerabilities**

- WordPress is the most popular CMS; many plugins
- 115 WordPress plugins analyzed
  - some have more than 1M downloads
  - some are installed in more than 10K websites
- 23 were found vulnerable
  - 153 zero-day vulnerabilities
  - 16 known vulnerabilities
  - 55 SQLI, 71 XSS, 31 DT/RFI/LFI, etc.
WAP wrap-up

• An approach and a tool (WAP)
  – to automatically identify and correct these vulnerabilities
  – and to predict false positives using data mining
  – leveraging the idea of learning instead of programming knowledge
• Millions of LOCs analyzed, many 0-days found

WAP: better input validation

![Image of a phone with a sticker indicating only 911 can be dialed]
DEKANT: VULNERABILITY DETECTION WITH STATIC ANALYSIS USING A SEQUENCE MODEL

Motivation

• Typical static analysis tools:
  – detect vulnerabilities they are programmed to
  – learning would be interesting, as seen already

• WAP: limited capacity to learn
  – does classification of FPs based on symptoms
  – does not take into account the order of elements that appear in the code

• Is it possible to have a tool that learns “everything”? 
**DEKANT: outline**

- **Overview**
- Intermediate slice language
- Sequence model
- The DEKANT tool
- Results

**DEKANT**

- No vulnerability knowledge is programmed in the tool
  - *not 100% true:* slicing is programmed; expert assigns functions to classes
- The tool extracts knowledge (learns) from a corpus, i.e., a set of annotated source code samples
- This knowledge is modeled using a sequence model (a Hidden Markov Model – HMM)
Natural language processing

- Example: part-of-speech (POS) tagging
  - Nelson Évora is expected to win tomorrow
  - Nelson Évora/NNP is/VBZ expected/VBN to/TO win/VB tomorrow/NN

- POS classifies each word (observation) of a sentence (sequence) with a tag
  - taking into account the context of the word (i.e., its place in the sentence, order)

- context/order are modeled using a HMM
- knowledge about tags is learned from a corpus

Hidden Markov Model

- States are hidden and emit observations
- For a sequence of observations, the HMM allows discovering the sequence of states that emits that sequence
Hidden Markov Model

- Goal: calculate which state emits $\text{obs}_n$
- How: by calculating the probability that each state emits $\text{obs}_n$ given the previous states
- Winner: the sequence with highest probability

```
input: obs1 obs2 obs3 obs4
output: ?? sequence of states ??
```

---

Static analysis vs HMM

- Putting the two together we have SAT that learns to detect vulnerabilities using a HMM
Knowledge and learning

• Create the corpus:
  – collect slices (vulnerable and otherwise)
  – translate slices into ISL (*Intermediate Slice Language*)
  – annotate the slices with states (Vul and N-Vul)
  – remove duplicates

• Learn vulnerability characteristics:
  – generate matrices of probabilities
  – train the HMM

DEKANT: outline

• Overview
• Intermediate slice language
• Sequence model
• The DEKANT tool
• Results
Intermediate slice language (ISL)

- A language that represents abstractly the source code elements
- Composed by tokens and a grammar

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
<th>PHP Func.</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>entry point</td>
<td>$_GET</td>
</tr>
<tr>
<td>var</td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td>sanit_f</td>
<td>sanitization function</td>
<td>htmlspecialchars</td>
</tr>
<tr>
<td>ss</td>
<td>sensitive sink</td>
<td></td>
</tr>
<tr>
<td>typechk.str</td>
<td>type checking string function</td>
<td></td>
</tr>
<tr>
<td>typechk.num</td>
<td>type checking numeric function</td>
<td></td>
</tr>
<tr>
<td>contentchk</td>
<td>content checking function</td>
<td></td>
</tr>
<tr>
<td>fillchk</td>
<td>fill checking function</td>
<td></td>
</tr>
<tr>
<td>cond</td>
<td>if instruction presence</td>
<td></td>
</tr>
<tr>
<td>join_str</td>
<td>join string function</td>
<td></td>
</tr>
<tr>
<td>erase_str</td>
<td>erase string function</td>
<td></td>
</tr>
<tr>
<td>replace_str</td>
<td>replace string function</td>
<td></td>
</tr>
</tbody>
</table>

Translating a slice into ISL

```php
$u = $_POST['username'];
$sql = "SELECT pass FROM users WHERE user='{\$u}';
$result = mysql_query($sql);
```
Translating a slice into ISL

```php
$u = $_POST['username'];
$sql = "SELECT pass FROM users WHERE user='\$u';"
$result = mysql_query($sql);
```
Translating a slice into ISL

```php
$u = \$_POST['username'];
$q = "SELECT pass FROM users WHERE user='\"$.u\"';
$result = mysql_query($q);
```

<table>
<thead>
<tr>
<th>slice-isl</th>
<th>variable map</th>
</tr>
</thead>
<tbody>
<tr>
<td>input var</td>
<td>1 - u</td>
</tr>
<tr>
<td>var</td>
<td>1 u q</td>
</tr>
<tr>
<td>ss var var</td>
<td>1 - q result</td>
</tr>
</tbody>
</table>

1.0 : is an assignment instruction or not
- : is not a variable
u : the name of the variable in the slice

DEKANT: outline

- Overview
- Intermediate slice language
- **Sequence model**
- The DEKANT tool
- Results
Sequence Model

- The **model** is the HMM model already presented
- an **ISL instruction**
  - is a **sequence of observations** for the HMM
  - is classified as **taint** or **n-taint**
- the **last observation** from last instruction carries the classification of the whole slice-isl: **taint** or **n-taint**, i.e., **vulnerable** or **not**

---

**Sequence Model**

**Vocabulary**
- 20 tokens from ISL
- 1 special token (var_vv)

**States**
- Taint
- N-taint
- San(itation)
- Val(idation)
- Chg_str(ing)

**Probability matrices**
- Initial (1 x 5)
- Transition (5 x 5)
- Emission (21 x 5)

**Decoder**
- Viterbi algorithm modified, interacting with the VM and the TL, CTL and SL lists

**Model graph**
Classification example

```
\$u = \$_POST['username'];
\$q = "SELECT pass FROM users WHERE user='".$u.'";"
\$result = mysql_query($q);
```

<table>
<thead>
<tr>
<th>sliceisl</th>
<th>variable map</th>
</tr>
</thead>
<tbody>
<tr>
<td>input var</td>
<td>1 - u</td>
</tr>
<tr>
<td>var var</td>
<td>1 u q</td>
</tr>
<tr>
<td>ss var var</td>
<td>1 - q result</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sequence</th>
<th>before</th>
<th>Viterbi</th>
</tr>
</thead>
<tbody>
<tr>
<td>input var</td>
<td>----</td>
<td>&lt;input,taint&gt; &lt;var_vv_u,taint&gt;</td>
</tr>
<tr>
<td>var var</td>
<td>var_vv var</td>
<td>&lt;var_vv_u,taint&gt; &lt;var_vv_q,taint&gt;</td>
</tr>
<tr>
<td>ss var var</td>
<td>ss var_vv var</td>
<td>&lt;ss,N-taint&gt; &lt;var_vv_q,taint&gt; &lt;var_vv_result,taint&gt;</td>
</tr>
</tbody>
</table>

Vulnerability!

DEKANT: outline

- Overview
- Intermediate slice language
- Sequence model
- The DEKANT tool
- Results
The DEKANT Tool

• Implements the learning phase and the sequence model
• Corpus with 510 slices extracted from real web applications (414 vulnerable, 96 non-vulnerable)
• Detects 8 vulnerability classes: SQLI, XSS, RFI, LFI, DT SCD, OSCI, PHPCI
• Composed by 4 modules:
  – knowledge extractor
  – slice extractor
  – slice translator
  – vulnerability detector

DEKANT: outline

• Overview
• Intermediate slice language
• Sequence model
• The DEKANT tool
• Results
Evaluation: WordPress plugins

10 WordPress plugins analyzed

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Slices</th>
<th>Real vulnerabilities</th>
<th>N-Vul</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>appointment-booking-calendar 1.1.7</td>
<td>12</td>
<td>CVE-2015-3419, CVE-2015-7220</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>contact-form-generator 2.0.1</td>
<td>3</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>easy2asp 1.2</td>
<td>6</td>
<td>CVE-2015-7068, CVE-2015-7069</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>event-calendar-sp 1.0.5</td>
<td>8</td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>payment-form-for-paypal-pro 1.0.3</td>
<td>11</td>
<td>CVE-2015-70306</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>resale 1.0.1</td>
<td>2</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>simple-support-ticket-system 1.2*</td>
<td>29</td>
<td></td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>wordfence 6.2.17</td>
<td>6</td>
<td>CVE-2015-7070</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>wp-widget-master 1.2</td>
<td>9</td>
<td></td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>86</td>
<td></td>
<td>6</td>
<td>19</td>
</tr>
</tbody>
</table>

0-day vvs:
- confirmed and fixed by developers
- registered in CVE

5 vulnerable
16 0-day vvs

Evaluation: real web applications

10 web applications with known vulnerabilities

<table>
<thead>
<tr>
<th>Web application</th>
<th>Slices</th>
<th>VUL</th>
<th>SUS</th>
<th>YC</th>
<th>Total</th>
<th>N-VUL</th>
<th>FP</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext-0.8.8</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>10</td>
<td></td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>communityEdition</td>
<td>16</td>
<td>36</td>
<td>8</td>
<td>60</td>
<td></td>
<td>16</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>epeni-1.6.0-20140710</td>
<td>25</td>
<td>1</td>
<td>8</td>
<td>34</td>
<td></td>
<td>25</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>NeoBill0.9-alpha</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td></td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>phpMyAdmin-4.2.2.6-en</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>14</td>
<td></td>
<td>1</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>refbase-0.9.6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>12</td>
<td></td>
<td>5</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Schoolmate-1.5.4</td>
<td>120</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td></td>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VideosTube</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Webchess 1.0</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td></td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Zero-CMS-1.0</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>18</td>
<td></td>
<td>2</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>211</td>
<td>92</td>
<td>47</td>
<td>310</td>
<td></td>
<td>211</td>
<td>87</td>
<td>12</td>
</tr>
</tbody>
</table>

10 vulnerable > 4200 files > 1.5 M Loc
223 vulnerabilities found
211 Vul 12 FP 0 FN

classified manually
211 Vul
99 N-Vul
Evaluation: real web applications

<table>
<thead>
<tr>
<th>Metric</th>
<th>DEKANT</th>
<th>WAP</th>
<th>PhpMinerII</th>
<th>Pixy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>accuracy</td>
<td>96%</td>
<td>90%</td>
<td>89%</td>
</tr>
<tr>
<td></td>
<td>precision</td>
<td>95%</td>
<td>88%</td>
<td>83%</td>
</tr>
<tr>
<td></td>
<td>false positive</td>
<td>12%</td>
<td>27%</td>
<td>4%</td>
</tr>
<tr>
<td></td>
<td>false negative</td>
<td>0%</td>
<td>2%</td>
<td>32%</td>
</tr>
</tbody>
</table>

DEKANT wrap-up

- New approach inspired in NLP to detect web application vulnerabilities
- Knowledge is learned (except...)
  - first learn about vulnerabilities from corpus
  - then detect vulnerabilities taking the order of instructions into consideration
- Nice results in comparison with other tools
- *Just a first step in a promising research direction*
Motivation: dynamic protection

- Widely successful in the binary application world
- Today buffer overflows automatically blocked by:
  - canaries in the stack – detect return address modification
  - heap hardening – detects heap meta-data modification
  - non-executable pages – jumps into injected code make program crash
  - address space layout randomization – makes addresses hard to guess
  - and many more, e.g., https://wiki.debian.org/Hardening
Motivation: dynamic protection

- **Idea**: block attacks that may exploit existing vulnerabilities
- **Benefit**: can be deployed transparently (operating system, compiler, virtual machine), independently of vulnerabilities existing or not
- *Successful with binary applications, why not with web applications?*

SEPTIC

- **Problem**:  
  - SQLI injection attacks retrieve/store data in DB  
  - Sometimes they circumvent sanitization functions  
  - *Semantic mismatch* between server-side language and DBMS
- **Our solution**:  
  - DBMS self-protected against injection attacks  
  - Detect and block injection attacks inside the DBMS
- **How**:  
  - “hacking” the DBMS → SEPTIC mechanism
Semantic mismatch example

- Input sanitized with `mysql_real_escape_string`
  - username `admin'` → ' is escaped
  - username `admin%27` → `%27` not escaped but MySQL interprets `%27` as a prime and executes `SELECT name FROM users WHERE user='admin'

- Semantic mismatch
  - different views from PHP and MySQL
  - PHP programmers don’t know this attack works

Semantic mismatch cases

<table>
<thead>
<tr>
<th>Encoded characters</th>
<th>do nothing</th>
<th>decodes and executes</th>
</tr>
</thead>
<tbody>
<tr>
<td>%27, 0x027</td>
<td>%27, 0x027</td>
<td>'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unicode characters</th>
<th>do nothing</th>
<th>translates and executes</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0027, U+02BC</td>
<td>U+0027, U+02BC</td>
<td>'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Space character evasion</th>
<th>do nothing</th>
<th>removes and executes</th>
</tr>
</thead>
<tbody>
<tr>
<td>char(39)<strong>/OR/</strong>/1=1 --</td>
<td>char(39)<strong>/OR/</strong>/1=1 --</td>
<td>' OR 1=1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INSERT query</th>
<th>sanitize</th>
<th>unsanitizes and inserts data</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin' --</td>
<td>admin' --</td>
<td>admin'</td>
</tr>
</tbody>
</table>

Server-side language interprets in one way

DBMS interprets in another way
SEPTIC: outline

- Attack detection in SEPTIC
- Running SEPTIC
- Results

Attacks handled by SEPTIC

- SQLI
  - Syntax structure: alter the structure of the query
  - Syntax mimicry: mimic the structure of the query
  - Obfuscation:
    - Encoded characters
    - Unicode characters
    - Dynamic SQL
    - Numeric fields
    - Space character evasion
  - Stored procedures
  - Blind SQLI

- Stored injection
  - Second order SQLI
  - Stored XSS
  - Stored RCI, RFI, LFI
  - Stored OSCI
Query processing vs SEPTIC

Server-side language engine

query (Q)

Web application

DBMS

pares

validates

SEPTIC

executes

detection: query is compared to model(s); no mismatch as mechanism runs just before query is executed!

SElf-Protecting daTabases preventing attaCks

SEPTIC: creating query models

SELECT name FROM users WHERE user = 'alice' AND pass = 'foo'

Query parse tree

DBMS parser

Query Structure (QS)

Query Model (QM)

each query should have its own identifier (ID)
Query ID creation: SSLE IDs

- SSLE best place to create IDs
  - programmer not involved
  - lot’s of info about the code
- Basic ID:
  - file:line – file pathname and line number where DBMS is called (e.g., `mysql_query`)
  - problem: single function used for different queries
- Full ID:
  - file:line | ... | file:line – 1st pair has same meaning
  - other pairs: lines where query is passed as argument to a function
Query ID creation: DBMS IDs

SQLi detection: step 1- structurally

- compare the number of nodes of QS with its QM
- if #nodes is different, then SQLi attack detected
  - otherwise goto step 2
  - quick and covers many attacks, e.g., admin’ --
SQLI detection: step 2 - syntactically

- compare the content of nodes of QS with its QM
- if a pair does not match, a SQLI attack is detected

Example: second order SQLI

Second order SQLI attack
- malicious code is injected in an INSERT or UPDATE query
- malicious code is retrieved from the DB and used in a second query. The attack is performed.
**Example: syntax mimicry**

```
SELECT name
FROM users
WHERE user = ?
AND pass = ?
```

**Syntax mimicry**
- malicious code does not alter the structure of the query

```
Malicious code:
admin' AND 1=1 --
```

```
Initial query
SELECT name
FROM users
WHERE user = 'admin' AND 1=1 --
AND pass = 'any'
```

```
Final query (validated)
SELECT name
FROM users
WHERE user = 'admin'
AND 1 = 1
```

---

**Stored injection detection**

- **Stored injection attack**
  - Malicious data: JavaScript (stored XSS), shell commands, PHP code
  - 1st step: malicious data inserted in the DB
  - 2nd step: malicious data retrieved from DB and used
- **Detection using code detectors (plugins)**
  - inputs from INSERT/UPDATE queries are checked looking for malicious data
  - we didn’t go much deep in this (only XSS, basic)
SEPTIC: outline

- Attack detection in SEPTIC
- Running SEPTIC
- Results

SEPTIC operation modes

**Training**
- build query models

**Normal**
- detect & block attacks

- **training phase**: a pure training phase to learn QMs
  - unit tests of the application
  - septic_training module

- **Incremental**: there is no training phase; QM is built for the first query with each ID

- **prevention**: detect attacks; log attacks; drop the queries; and DBMS stops the query processing

- **detection**: detect attacks; log attacks; and DBMS processes the query
Creating/storing query model

![Diagram of query model creation and storage]

Training mode | training phase
Normal mode | incremental

Detecting/blocking SQLI

![Diagram of SQL injection detection and blocking]

Normal mode | prevention or detection
Detecting/blocking stored injection

Normal mode | prevention or detection

SEPTIC full architecture
SEPTIC: outline

- Attack detection in SEPTIC
- Running SEPTIC
- Results

SEPTIC implementation (#changes)

- MySQL DBMS – SEPTIC itself
  - 1 file: 14 loc
  - SEPTIC detector
  - SEPTIC setup
  - septic_training module
- PHP / Zend engine – insertion of IDs in the SSLE
  - 3 files: 27 loc
  - SEPTIC identifier
- Java/Spring framework – to show it’s not specific to PHP
  - 1 file: 16 loc
  - SEPTIC identifier
- Also analyzed cases of MariaDB and PostgreSQL
SEPTIC detection w/code samples

- SQLI unrelated to semantic mismatch
  - 23 from the sqlmap project
  - 11 by Ray & Ligatti (4 are not attacks/vuln.)
  - 7 other samples (for other SQLI attacks)
- SQLI related to semantic mismatch
  - 17 code samples
- Stored injection
  - 5 code samples
- Total: 59 attacks/vuln., 4 non-attacks/vuln.

Comparison with other tools

![Comparison with other tools diagram](attachment:comparison_diagram.png)
SEPTIC: real open source software

- Vulnerabilities detectedblocked in real webapps
- Zero CMS
  - CVE-2014-4194
  - CVE-2014-4034
  - OSVDB ID 108025
- WebChess
  - 13 vulnerabilities
- measureit
  - 1 stored XSS

SEPTIC: performance

Apache & Zend Web applications

BenchLab

MySQL & SEPTIC

each 1 to 5 browsers

<table>
<thead>
<tr>
<th>SEPTIC combinations</th>
<th>SQLI detector</th>
<th>Stored inj. det.</th>
<th>0.82%</th>
<th>2.24%</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>0.82%</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
<td>2.24%</td>
<td></td>
</tr>
</tbody>
</table>
SEPTIC wrap-up

• Putting protection in the DBMS allows detecting / blocking attacks efficiently
  – Subtle attacks related to semantic mismatch
• (Mostly) transparent protection for web applications
• Low performance overhead
• *May have practical impact in webapp security?*

4

SHUTTLE: INTRUSION RECOVERY IN THE CLOUD
Cloud computing (public cloud)

- Cloud provider vs consumers
- Fundamental ideas
  - Computing as a utility
  - Pay-as-you-go
  - Resource pooling
  - Elasticity
- Large-scale datacenters

Cloud computing service models

- Infrastructure as a Service (IaaS)
  - virtual machines, storage (e.g., Amazon EC2, Amazon S3)
- Platform as a Service (PaaS)
  - programming and execution (e.g., Google AppEngine, Force.com, Windows Azure)
- Software as a Service (SaaS)
  - mostly web applications (e.g., Yahoo! Mail, Google Docs, Facebook,...)
Platform as a Service (PaaS)

• PaaS services allow running applications
• Consumer develops application to run in that environment, using
  – Supported languages, e.g., Java, Python, Go, PHP
  – Supported components, e.g., SQL/NoSQL databases, load balancers
  – Examples: Google App Engine, Windows Azure Cloud Services, Salesforce Force.com,...

Motivation

• Intrusions in PaaS applications may happen due to
  – Software vulnerabilities (e.g., Shellshock)
  – Configuration and usage mistakes
  – Corrupted legitimate requests (e.g., SQLI)
• Attacker can run commands in the application and delete, add, and modify data
• Legitimate users can then do commands on corrupted data...
Motivation

Shuttle: outline

- Shuttle
- Evaluation
Shuttle

- Recovers the state integrity of PaaS applications when there are intrusions
- Isn’t it what backups do?
  - Backups: remove both bad and good operations
  - Shuttle: removes bad operations but keeps good ones

State of the art

- Previous works
  - Operating systems: Taser, Retro
  - Databases: ITDB, Phoenix
  - Web applications: Goel et. al, Warp, Aire
  - Others (Email): Undo for Operators
- Limitations
  - Max. complexity: 1 app server, 1 database instance
  - All require setup and configuration
  - Cause application downtime during recovery
Shuttle

- **Supported by the cloud**: available without consumer setup
- Supports applications deployed in various instances
- Avoids application **downtime** as no need to stop the application during recovery
- Leverage elasticity to make recovery faster

---

**PaaS applications architecture**

[Diagram of PaaS applications architecture showing User Request, Proxy, Load Balancer, Application Server, and Database Instance connections]
Shuttle architecture

User Request
Proxy
Load Balancer
Interceptor
Application Server
DB Proxy
Database Instance

Shuttle during recovery

User Request
Proxy
Load Balancer
Interceptor
Application Server
DB Proxy
Database Instance

Storage
Manager

Replay Instances
Recovery process

1. Detect/identify the malicious operations (not Shuttle)
2. Start new instances of the application and database
3. Load a snapshot previous to intrusion instant; create a new branch (application stays running in previous branch)
4. Replay requests in new branch
5. Block incoming requests; replay last requests
6. Change to new branch; shutdown unnecessary instances

Recovery modes

- Full-Replay: Replay every operation after snapshot
- Selective-Replay: Replay only affected (tainted) operations
- Serial: Replay all dependency graph sequentially
- Clustered: Replay independent clusters concurrently; allowed by the cloud elasticity

<table>
<thead>
<tr>
<th>Modes supported:</th>
<th>Full-Replay</th>
<th>Selective-Replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cluster (Serial)</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Clustered</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>
Shuttle: outline

- Shuttle
- Evaluation

Evaluation environment

- Amazon EC2, c3.xlarge instances, Gb Ethernet
- WildFly application server (formely JBoss)
- Voldemort database
- Ask Q&A application; data from Stack Exchange
Accuracy

- Intrusion Scenarios:
  - 1. Malicious requests
  - 2. Software vulnerabilities
  - 3. External channels (e.g. SSH due to Shellshock)

<table>
<thead>
<tr>
<th>Scenario</th>
<th># data items affected</th>
<th># requests tainted</th>
<th># requests replayed (Selective Replay)</th>
<th># requests replayed (Full Replay)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>106</td>
<td>0</td>
<td>&lt; 605</td>
<td>38 620</td>
</tr>
<tr>
<td>1b</td>
<td>58</td>
<td>14</td>
<td>&lt; 379</td>
<td>38 620</td>
</tr>
<tr>
<td>1c</td>
<td>48</td>
<td>52</td>
<td>&lt; 253</td>
<td>38 620</td>
</tr>
<tr>
<td>2a</td>
<td>4 338</td>
<td>0</td>
<td>-</td>
<td>38 620</td>
</tr>
<tr>
<td>2b</td>
<td>18 286</td>
<td>1 278</td>
<td>-</td>
<td>38 620</td>
</tr>
<tr>
<td>3</td>
<td>&gt; 2 000</td>
<td>-</td>
<td>-</td>
<td>38 620</td>
</tr>
</tbody>
</table>

Performance overhead

- in normal execution

<table>
<thead>
<tr>
<th>Scenario</th>
<th>50% Reads 50% Inserts</th>
<th>95% Reads 5% Inserts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>opsec latency (ms)</td>
<td>opsec latency (ms)</td>
</tr>
<tr>
<td>Shuttle</td>
<td>6325 5.78</td>
<td>15 346 3.62</td>
</tr>
<tr>
<td>No Shuttle</td>
<td>7148 5.07</td>
<td>17 821 3.01</td>
</tr>
<tr>
<td>overhead</td>
<td>13% 14%</td>
<td>16% 20%</td>
</tr>
</tbody>
</table>

Overhead seems acceptable; penalty mostly due to single proxy
Recovery time

- for 1 million requests

Clustered replay greatly reduces recovery time.

Restrain duration

**Restrain**: 46 seconds
Storage overhead

- for 1 million requests

<table>
<thead>
<tr>
<th># objects</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shuttle Storage:</td>
<td></td>
</tr>
<tr>
<td>Requests</td>
<td>1 million</td>
</tr>
<tr>
<td>Response</td>
<td>1 million</td>
</tr>
<tr>
<td>Start/End timestamps</td>
<td>2 million</td>
</tr>
<tr>
<td>Keys</td>
<td>137 million</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

| Database node:     |           |
| Version List       | 14 593    | 1.4 |
| Operation List     | 9 million | 277 |
| Total              |           | 282 |
| Manager Graph      | 1 million | 718 |

Storage is considerable but mostly due to storing full responses
$47 per month if 20 Million requests per day (without responses)

SHUTTLE wrap-up

- New intrusion recovery service for PaaS offerings
- Supports applications running in various instances, backed by distributed databases
- Leverages the resource elasticity and pay-per-use model to reduce the recovery time and costs
- Provides intrusion recovery without service downtime using a branching mechanism
Outline

1. **WAP**: vulnerability detection with static analysis using taint analysis + classifier

2. **DEKANT**: vulnerability detection with static analysis using a sequence model

3. **SEPTIC**: blocking attacks in the DBMS

4. **SHUTTLE**: intrusion recovery in the cloud

---

Papers


**WAP**: ___. Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. IEEE Transactions on Reliability 2016

**WAP**: ___. Equipping WAP with WEAPONS to Detect Vulnerabilities. DSN 2016

**DEKANT**: ___. **DEKANT**: A Static Analysis Tool that Learns to Detect Web Application Vulnerabilities. ISTTA 2016


Thank you

Miguel Pupo Correia
miguel.p.correia@tecnico.ulisboa.pt
http://www.gsd.inesc-id.pt/~mpc/