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

Miguel Correia

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

Building Trust in the Information Age – Summer School on Computer Security and Privacy – Cagliari, Sep. 2016

TÉCNICO LISBOA

inesc id lisboa

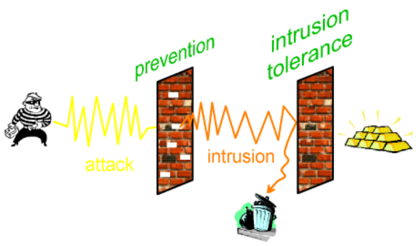
ULisboa / IST / INESC-ID

- 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

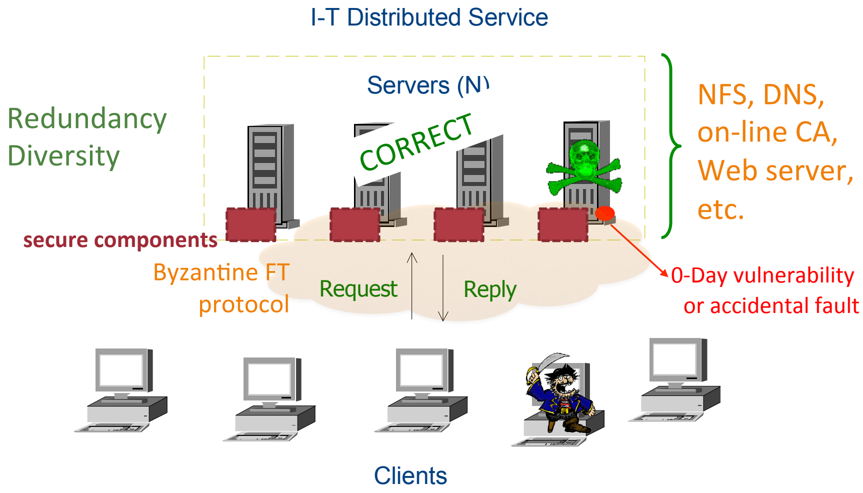
2

Research overview (1)
Intrusion Tolerance

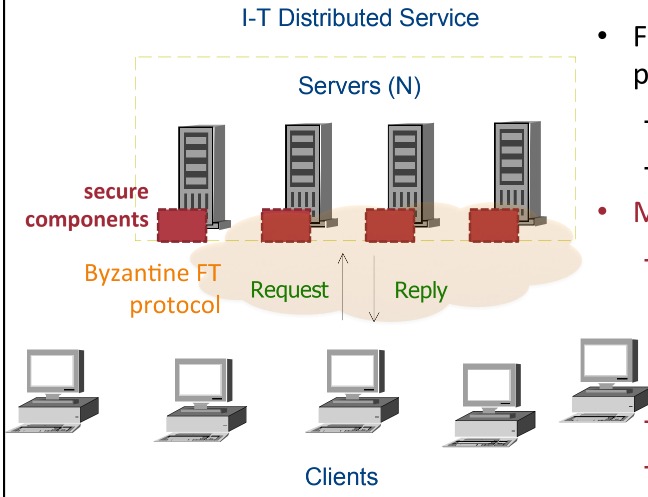
- To apply the Fault Tolerance paradigm in the domain of Security
- *Do the best we know to protect systems …but vulnerabilities still remain… so tolerate intrusions that still occur*



Research overview (2)
Intrusion-Tolerant Services



Research overview (3)
MinBFT



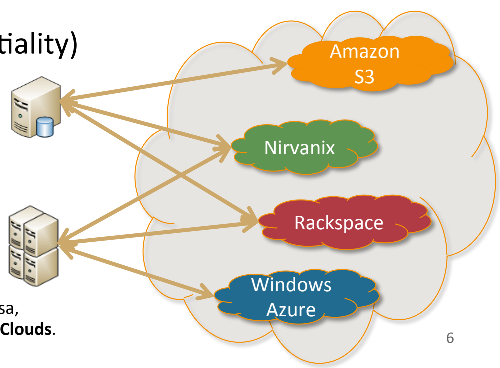
- First efficient **BFT SMR** protocol: PBFT (1999)
  - 3f+1 replicas
  - 5 communic. steps
- MinBFT (2009-13)
  - requires local secure component: monotonic counter (simpler than TPM)
  - 2f+1 replicas
  - 4 communic. steps

G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, P. Verissimo.
Efficient Byzantine Fault Tolerance. IEEE Transactions on Computers 2013.

5

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,
DepSky: Dependable and Secure Storage in a Cloud-of-Clouds.
EuroSys 2011 and ACM Transactions on Storage 2013.

6

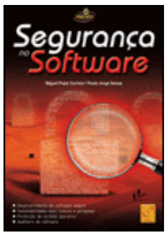
Overview of my research (5) Software Security

- 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

7

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



8

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: I. Medeiros, N. F. Neves, M. Correia. **Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives.** WWW 2014

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

SEPTIC: I. Medeiros, M. Beatriz, N. Neves and M. Correia. **Hacking the DBMS to Prevent Injection Attacks**. CODASPY 2016

SHUTTLE: D. Nascimento, M. Correia. **Shuttle: Intrusion Recovery for PaaS.** ICDCS 2015.

11

wap

1

WAP: VULNERABILITY DETECTION WITH STATIC ANALYSIS

USING TAINT ANALYSIS + CLASSIFIER

12

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

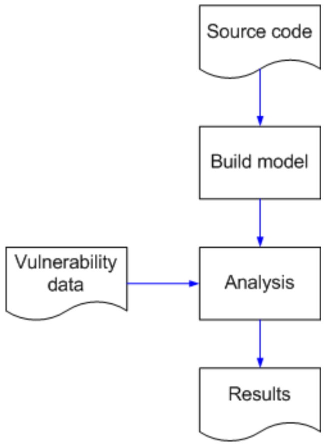
13

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

14

Generic static analysis tool



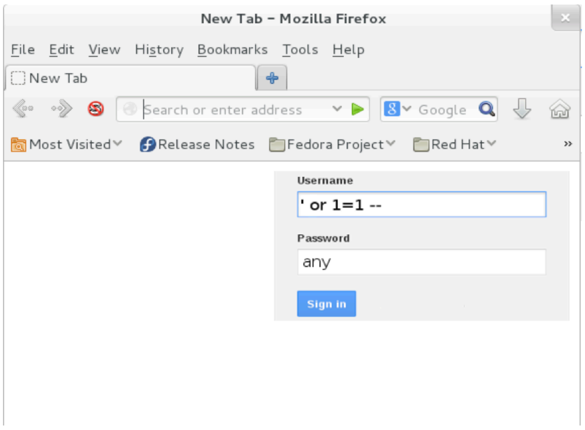
15

WAP: outline

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

16

Vulnerability example (SQLI)

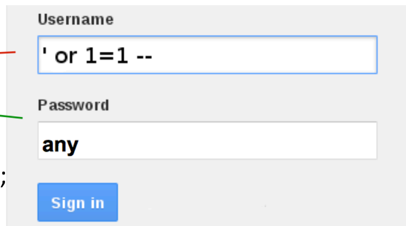


17

Vulnerability example (SQLI)

PHP code:

$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**)**;

18

Mechanism 1: Taint Analysis

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

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

SQL Injection detected

Vulnerability!

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

some functions sanitizes, so "untaints", the data flow

OK!

19

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*

20

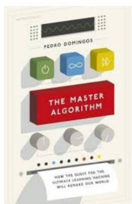
Programming

- 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

21

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...

22

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

23

Mechanism 3: Code Correction

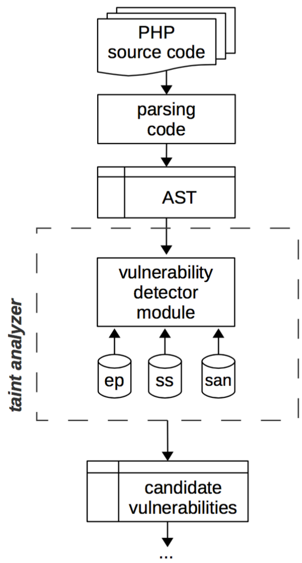
- Correcting vulnerabilities is tiresome and they can be removed mostly automatically using fixes
- Let the tool to do it when it detects a vulnerability

24

WAP: outline

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

Scheme



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

31

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

32

Original learning data set

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

Potential vulnerability		String manipulation				
Type	Webapp	Extract substring	String concat.	Add char	Replace string	Remove whitesp.
SQLI	CurrentCost	Y	Y	Y	N	N
SQLI	CurrentCost	Y	Y	Y	N	N
SQLI	CurrentCost	N	N	N	N	N
XSS	emoncms	N	Y	N	Y	N
XSS	Mfm-0.13	N	Y	N	Y	Y
XSS St.	ZiPEC 0.32	N	Y	N	N	N
RFI	DVWA 1.0.7	N	N	N	N	N
RFI	SRD	N	N	N	Y	N
RFI	SRD	N	N	N	Y	N
OSCI	DVWA 1.0.7	N	Y	N	Y	N
XSS St.	vicnum15	Y	N	N	N	N
XSS	Mfm-0.13	N	N	N	N	N

(...)

Validation						SQL query manipulation				
Type checking	IsSet entry point	Pattern control	While list	Black list	Error / exit	Aggreg. function	FROM clause	Numeric entry point	Complex query	Class
N	N	N	N	N	N	Y	N	N	N	Yes
N	N	N	N	N	N	N	N	N	N	Yes
N	N	N	N	N	N	N	N	N	N	No
N	N	N	N	N	N	NA	NA	NA	NA	Yes
N	N	N	N	N	N	NA	NA	NA	NA	Yes
N	N	N	N	N	N	NA	NA	NA	NA	No
N	N	N	Y	N	Y	NA	NA	NA	NA	Yes
N	Y	N	N	N	N	NA	NA	NA	NA	No
N	Y	Y	N	N	N	NA	NA	NA	NA	No
N	N	N	N	Y	N	NA	NA	NA	NA	Yes
N	N	Y	N	N	N	NA	NA	NA	NA	Yes
N	N	N	N	Y	N	NA	NA	NA	NA	Yes

33

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

34

Evaluation of classifiers

- Results for Logistic Regression (the best):

Predicted		Observed: Yes (FP)	Observed: No (not FP)
	Yes (FP)	27 TP	1 FP
	No (not FP)	5 FN	43 TN

  - 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

35

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)

36

WAP: outline

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

37

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

38

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

39

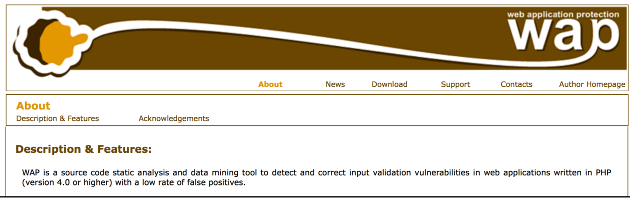
WAP: outline

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

40

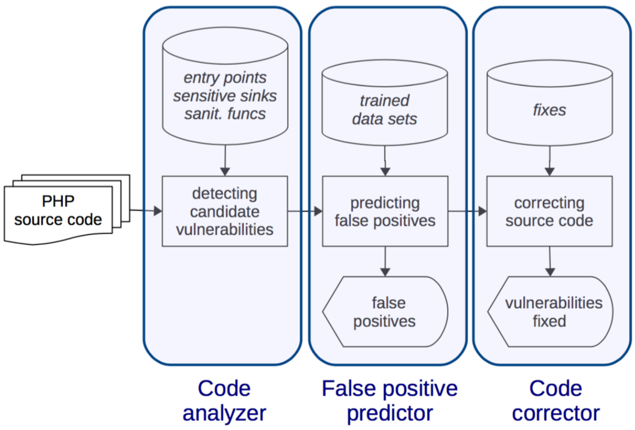
WAP - Web Application Protection

- 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!
  - http://awap.sourceforge.net/ and at OWASP



41

WAP



42

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

43

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

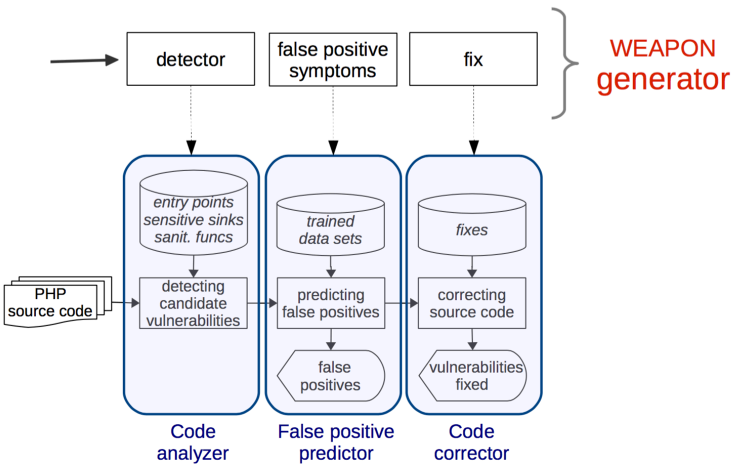
44

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)

45

WAPe: Basic scheme



46

WAPe: Classifier and data set

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

WAP	WAPe
15 features	60 features
24 symptoms (functions)	60 symptoms (functions)
data set with 76 instances	data set with 256 instances
Classifiers: Support Vector Machine Logistic Regression Random Tree	Classifiers: Support Vector Machine Logistic Regression Random Forest

48

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

52

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

Webapp	WAP-TA				Pixy				WAP (complete)				
	SQLI	XSS	FP	FN	SQLI	XSS	FP	FN	SQLI	XSS	FP	FN	Corrected
CurrentCost	3	4	2	0	3	5	3	0	1	4	2	0	5
DVWA 1.0.7	4	2	2	0	4	0	2	2	2	2	2	0	4
emoncms	2	6	3	0	2	3	0	0	2	3	3	0	5
Measureit 1.14	1	7	7	0	1	16	16	0	1	0	7	0	1
Mfm-0.13	0	8	3	0	0	10	8	3	0	5	3	0	5
Multilidae 2.3.5	0	2	0	0	-	-	-	-	0	2	0	0	2
SAMATE	3	11	0	0	4	11	1	0	3	11	0	0	14
Vicnum15	3	1	3	0	3	1	3	0	0	1	3	0	1
Wackopicko	3	5	0	0	-	-	-	-	3	5	0	0	8
ZiPEC 0.32	3	0	1	0	3	7	8	0	2	0	1	0	2
Total	22	46	21	0	20	53	41	5	14	33	21	0	47

WAP-TA:
68 vuln.: 21 are FP
0 false negatives
Same 11 FP than Pixy

Pixy:
73 vuln.: 41 are FP
5 false negatives
Same 11 FP than WAP
+ 30 FP than WAP

$\underbrace{\text{WAP-TA and Pixy}}_{\text{Without data mining}}$

WAP (complete):
47 real vulnerabilities
21 predicted false positives
0 false negatives
47 vulnerabilities corrected

$\underbrace{\text{WAP (complete)}}_{\text{With data mining}}$

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

		Observed	
		Yes (Vuln.)	No (not Vuln.)
Predicted	Yes (Vuln.)	48	5
	No (not Vuln.)	5	20

Logistic Regression

Accuracy = 87.2%
Precision = 85.2%

Summary

Metric	WAP	Pixy	PhpMinerII
accuracy	92.1%	44.0%	87.2%
precision	92.5%	50.0%	85.2%

WAP with all vulnerability classes

Webapp	Detected taint analysis							Detected data mining	Corrected
	SQLI	RFI, LFI DT/PT	SCD	OCSI	XSS	Total	FP		
currentcost	3	0	0	0	4	7	2	5	5
DVWA 1.0.7	4	3	0	6	4	17	8	9	9
emoncms	2	0	0	0	13	15	3	12	12
Measureit 1.14	1	0	0	0	11	12	7	5	5
Mfm 0.13	0	0	0	0	8	8	3	5	5
Mutillidae 2.3.5	0	0	0	2	8	10	0	10	10
OWASP Vicnum	3	0	0	0	1	4	3	1	1
SRD(1)	3	6	0	0	11	20	1	19	19
Wackopico	3	2	0	1	5	11	0	11	11
ZiPEC 0.32	3	0	0	0	4	7	1	6	6
Total	22	11	0	9	69	111	28	83	83

57

WAP totals

Web application	Files	Lines of code	Analysis time (s)	Vul files	Vul found	FP	Real vul
adminer-1.11.0	45	5,434	27	3	3	0	3
Butterfly insecure	16	2,364	3	5	10	0	10
Butterfly secure	15	2,678	3	3	4	0	4
currentcost	3	270	1	2	4	2	2
dmoz2mysql	6	1,000	2	0	0	0	0
DVWA 1.0.7	310	31,407	15	12	15	8	7
emoncms	76	6,876	6	6	15	3	12
gallery2	644	124,414	27	0	0	0	0
getboo	199	42,123	17	30	64	9	55
Ghost	16	398	2	2	3	0	3
gilbitron-PIP	14	328	1	0	0	0	0
GTD-PHP	62	4,853	10	33	111	0	111
Hexjector 1.0.6	11	1,640	3	0	0	0	0
Hotelmis 0.7	447	76,754	9	2	7	5	2
Lithuanian-7.02.05-v1.6	132	3,790	24	0	0	0	0
Measureit 1.14	2	967	2	1	12	7	5
Mfm 0.13	7	5,859	6	1	8	3	5
Mutillidae 1.3	18	1,623	6	10	19	0	19
Mutillidae 2.3.5	578	102,567	63	7	10	0	10
NeoBill0.9-alpha	620	100,139	6	5	19	0	19
ocsvg-0.2	4	243	1	0	0	0	0
OWASP Vicnum	22	814	2	7	4	3	1
paCRUD 0.7	100	11,079	11	0	0	0	0
Peruggia	10	988	2	6	22	0	22
PHP X Template 0.4	10	3,009	5	0	0	0	0
PhpBB 1.4.4	62	20,743	25	0	0	0	0
Phpcms 1.2.2	6	227	2	3	5	0	5
PhpCrud	6	612	3	0	0	0	0
PhpDiary-0.1	9	618	2	0	0	0	0
PHPFusion	633	27,000	40	0	0	0	0
phpldapadmin-1.2.3	97	28,601	9	0	0	0	0
PHPLib 7.4	73	13,383	35	3	14	0	14
PHPMyAdmin 2.0.5	40	4,730	18	0	0	0	0
PHPMyAdmin 2.2.0	34	9,430	12	0	0	0	0
PHPMyAdmin 2.6.3-pl1	287	143,171	105	0	0	0	0
Phpweather 1.52	13	2,465	9	0	0	0	0
SAMATE	22	353	1	10	20	1	19
Tikiwiki 1.6	1,563	499,315	1	4	4	0	4
volkszaehler	43	5,883	1	0	0	0	0
WackoPicko	57	4,156	3	4	11	0	11
WebCalendar	129	36,525	20	0	0	0	0
Webchess 1.0	37	7,704	1	5	13	0	13
WebScripts	5	391	4	2	14	0	14
Wordpress 2.0	215	44,254	10	7	13	1	12
ZiPEC 0.32	10	765	2	1	7	1	6
Total	6,708	1,381,943	557	174	431	43	388

1.38 MLOCs
388 vulnerabilities

WAPe totals

Web application	Version	Files	Lines of code	Analysis time (s)	Vuln. files	Vuln. found
Admin Control Panel Lite 2	0.10.2	14	1,984	1	9	81
Anywhere Board Games	0.150215	3	501	1	1	3
Clip Bucket	2.7.0.4	597	148,129	11	16	22
Clip Bucket	2.8	606	149,830	12	18	26
Community Mobile Channels	0.2.0	372	119,890	8	116	47
divine	0.1.3a	5	706	1	2	9
Ldap address book	0.22	18	4,615	2	4	1
Minutes	0.42	19	2,670	1	2	10
Mle Moodle	0.8.8.5	235	59,723	18	4	7
Php Open Chat	3.0.2	249	83,899	7	9	11
Pivotx	2.3.10	254	108,893	6	1	1
Play sms	1.3.1	1,420	248,875	19	7	6
RCR AEsir	0.11a	8	396	1	6	13
refbase	0.9.6	171	109,600	10	18	48
SAE	1.1	150	47,207	7	39	48
Tomahawk Mail	2.0	155	16,742	3	3	3
vfront	0.99.3	438	93,042	15	25	77
	Total	4,714	1,196,702	123	280	413

59

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.

60

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 programing knowledge
- Millions of LOCs analyzed, many 0-days found

61

WAP: better input validation

62

dekant

2

DEKANT: VULNERABILITY DETECTION WITH STATIC ANALYSIS USING A SEQUENCE MODEL

63

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"?

64

DEKANT: outline

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

65

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)

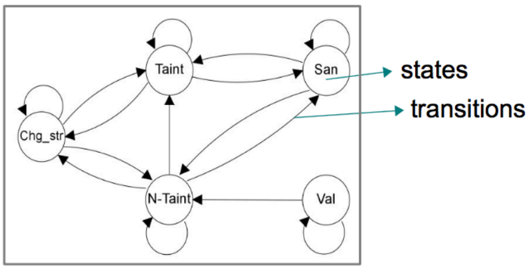
66

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

67

Hidden Markov Model

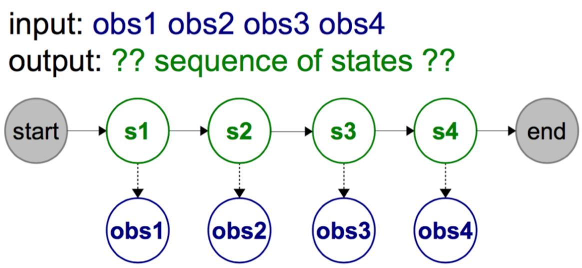


- States are hidden and emit observations
- For a sequence of observations, the HMM allows discovering the sequence of states that emits that sequence

68

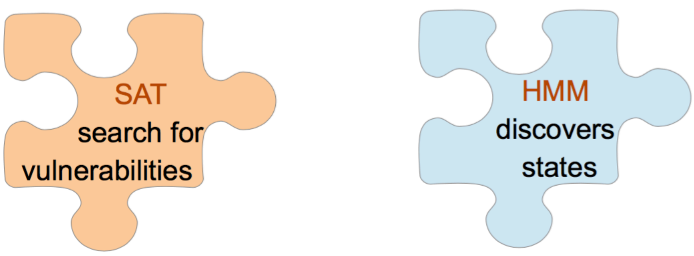
Hidden Markov Model

- Goal: calculate which state emits obs_n
- How: by calculating the probability that each state emits obs_n given the previous states
- Winner: the sequence with highest probability



69

Static analysis vs HMM

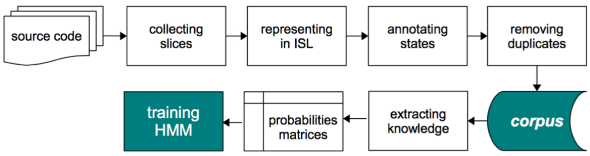


- Putting the two together we have SAT that learns to detect vulnerabilities using a HMM

70

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



71

DEKANT: outline

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

72

Intermediate slice language (ISL)

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

Token	Description	PHP Func.
input	entry point	$_GET
var	variable	–
sanit_f	sanitization function	htmlentities
ss	sensitive sink	mysql_query
typechk_str	type checking string function	is_string
typechk_num	type checking numeric function	is_int
contentchk	content checking function	preg_match
fillchk	fill checking function	isset, is_null
cond	*if* instruction presence	if
join_str	join string function	implode, join
erase_str	erase string function	trim
replace_str	replace string function	preg_replace

...

Translating a slice into ISL

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

input var

Translating a slice into ISL

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

input var
var var

Translating a slice into ISL

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

input var
var var
ss var var

Translating a slice into ISL

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

slice-isl		
input	var	
var	var	
ss	var	var

variable map			
1	-	u	
1	u	q	
1	-	q	result

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

79

Sequence Model

Vocabulary

20 tokens from ISL
1 special token (var_vv)

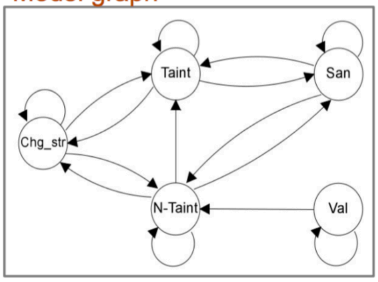
Decoder

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

States

Taint
N-taint
San(itization)
Val(idation)
Chg_str(ing)

Model graph



Probability matrices

Initial (1 x 5)
Transition (5 x 5)
Emission (21 x 5)

80

Classification example

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

slice_isl
input var
var var
ss var var

variable map
1 - u
1 u q
1 - q result

sequence	before	Viterbi
input var	----	<input,taint> <var_vv_u,taint>
var var	var_vv var	<var_vv_u,taint> < var_vv_q,taint>
ss var var	ss var_vv var	<ss,N-taint> <var_vv_q,taint> <var_vv_result,taint>

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

83

DEKANT: outline

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

84

Evaluation: WordPress plugins

10 WordPress plugins analyzed

5 vulnerable

16 0-day vvs

Plugin	Slices	Real vulnerabilities SQLI	XSS	DT & LFI	N-Vul	FP
appointment-booking-calendar 1.1.7*	12	1	3	–	6	2
		CVE-2015-7319, CVE-2015-7320				
calculated-fields-form 1.0.60	3	–	–	–	2	1
contact-form-generator 2.0.1	5	–	–	–	4	1
easy2map 1.2.9*	6	–	1	2	3	0
		CVE-2015-7668, CVE-2015-7669				
event-calendar-wp 1.0.0	6	–	–	–	6	0
payment-form-for-paypal-pro 1.0.1*	11	–	2	–	8	1
		CVE-2015-7666				
resads 1.0.1*	2	–	2	–	0	0
		CVE-2015-7667				
simple-support-ticket-system 1.2*	20	5	–	–	15	0
		CVE-2015-7670				
wordfence 6.0.17	6	–	–	–	6	0
wp-widget-master 1.2	9	–	–	–	6	3
Total	80	6	8	2	56	8

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

Evaluation: real web applications

10 web applications with known vulnerabilities

10 vulnerable
> 4200 files
> 1,5 M Loc

223 vulnerabilities found

211 Vul
12 FP
0 FN

Web application	Slices Vul	San	VC	Total	DEKANT Vul	N-Vul	FP	FN
cacti-0.8.8b	2	0	8	10	2	6	2	0
communityEdition	16	36	8	60	16	44	0	0
epesi-1.6.0-20140710	25	1	8	34	25	5	4	0
NeoBill0.9-alpha	19	0	0	19	19	0	0	0
phpMyAdmin-4.2.6-en	1	6	7	14	1	13	0	0
refbase-0.9.6	5	4	3	12	5	1	6	0
Schoolmate-1.5.4	120	0	0	120	120	0	0	0
VideosTube	1	0	2	3	1	2	0	0
Webchess 1.0	20	0	0	20	20	0	0	0
Zero-CMS.1.0	2	5	11	18	2	16	0	0
Total	211	52	47	310	211	87	12	0

classified manually

211 Vul

99 N-Vul

Evaluation: real web applications

Metric	DEKANT	WAP	PhpMinerII		Pixy
			original	analyzed	
acurracy	96%	90%	89%	71%	18%
precision	95%	88%	83%	19%	13%
false positive	12%	27%	4%	23%	87%
false negative	0%	2%	32%	69%	24%

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*

3

SEPTIC: BLOCKING ATTACKS IN THE DBMS

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?*

93

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

94

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

95

Semantic mismatch cases

Encoded characters	do nothing	decodes and executes
%27, 0x027	%27, 0x027	'
Unicode characters	do nothing	translates and executes
U+0027, U+02BC	U+0027, U+02BC	'
Space character evasion	do nothing	removes and executes
char(39)/**/OR/**/1=1 --	char(39)/**/OR/**/1=1 --	' OR 1=1
INSERT query	sanitize	unsanitizes and inserts data
admin' --	admin\' --	admin'
	Server-side language interprets in one way	DBMS interprets in another way

96

SEPTIC: outline

- **Attack detection in SEPTIC**
- Running SEPTIC
- Results

Attacks handled by SEPTIC

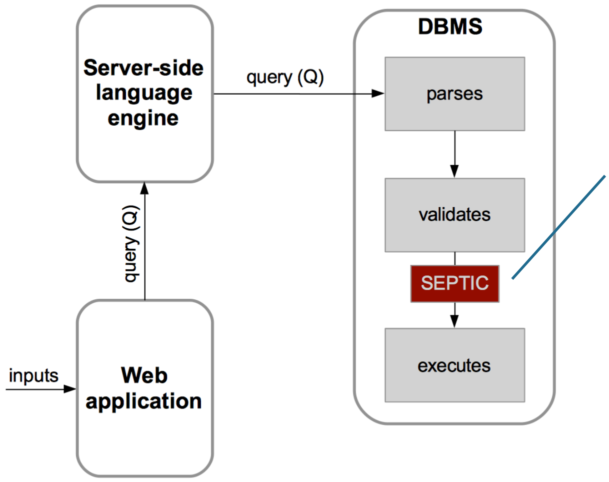
SQLI

- 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



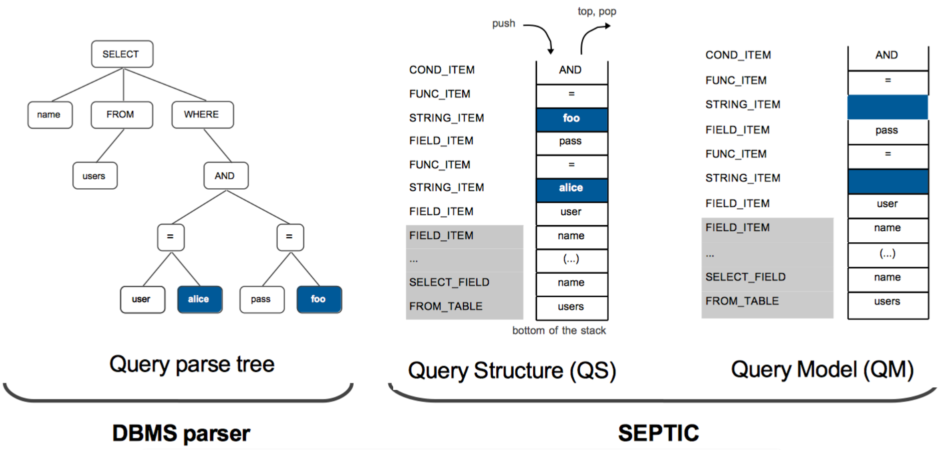
detection: query is compared to model(s); no mismatch as mechanism runs just before query is executed!

SElf-**P**rotecting da**T**abases prevent**I**ng atta**C**ks

99

SEPTIC: creating query models

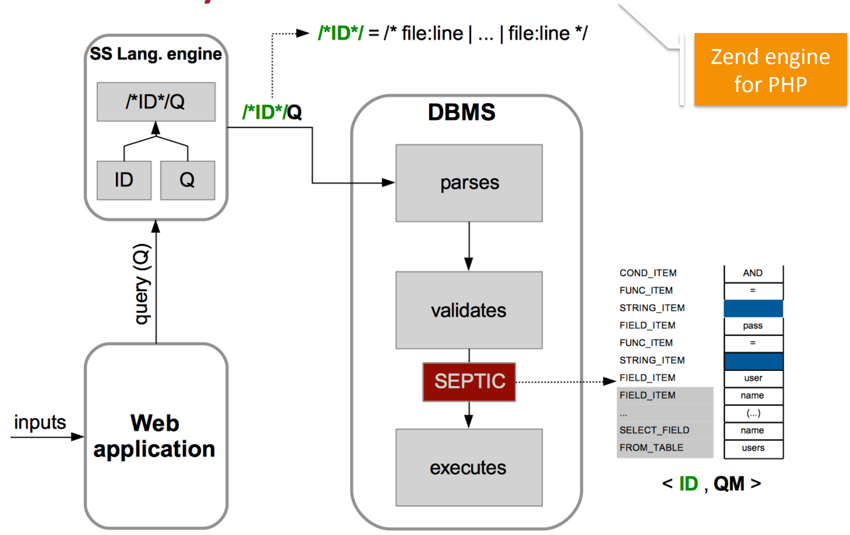
SELECT name FROM users WHERE user = '**alice**' AND pass = '**foo**'



each query should have its own identifier (ID)

100

Query ID creation: SSLE IDs



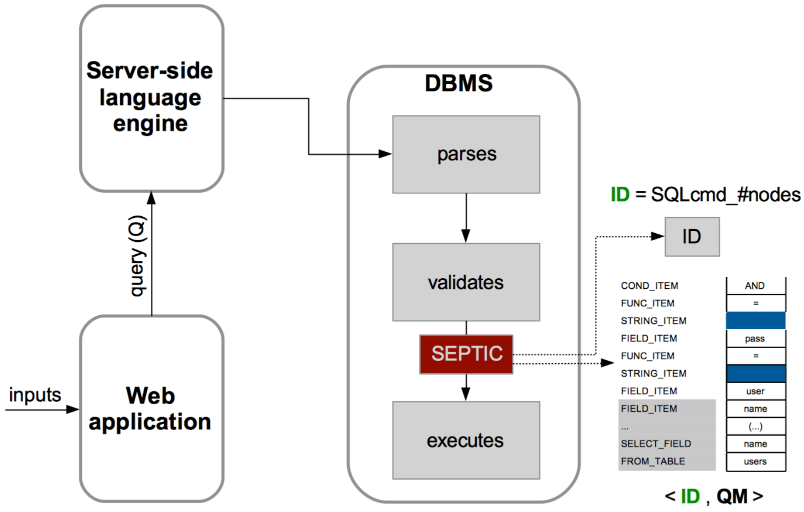
101

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

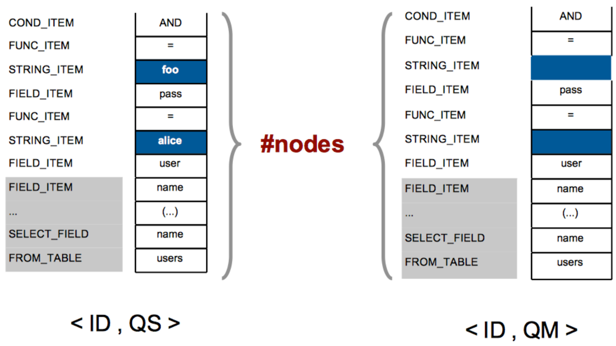
102

Query ID creation: DBMS IDs



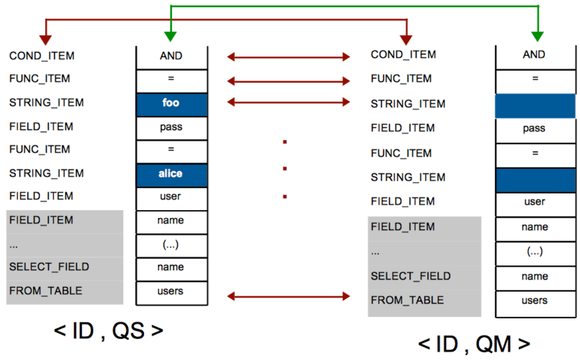
SQLI detection: step 1- structurally

- 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

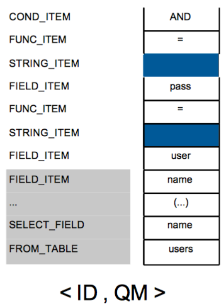
- compare the content of nodes of QS with its QM
- if a pair does not match, a SQLI attack is detected



105

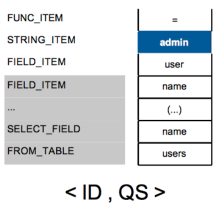
Example: second order SQLI

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



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.



Attack detected
1st step

Malicious code: admin' --

Initial query

SELECT name
FROM users
WHERE user = **'admin' --**
AND pass = **'any'**

Final query (validated)

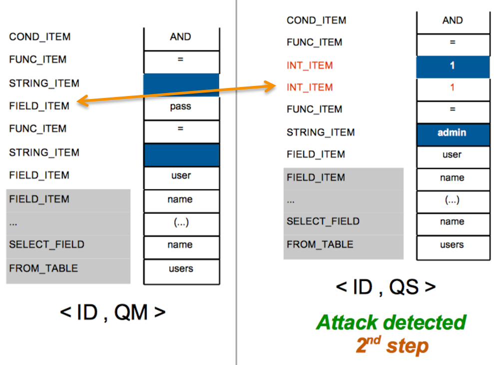
SELECT name
FROM users
WHERE user = **'admin'**

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

107

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)

108

SEPTIC: outline

- Attack detection in SEPTIC
- **Running SEPTIC**
- Results

SEPTIC operation modes

Training
build
query models

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

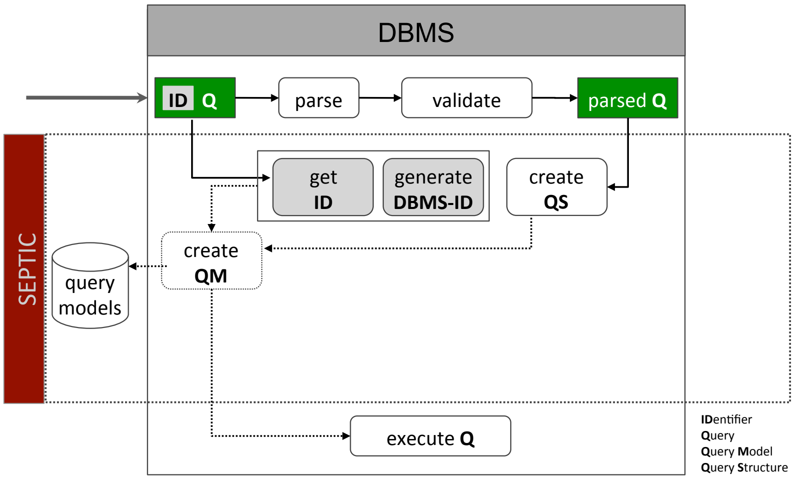
Incremental: there is no training phase;
QM is built for the first query with each ID

Normal
detect & block
attacks

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

detection: detect attacks; log attacks;
and DBMS processes the query

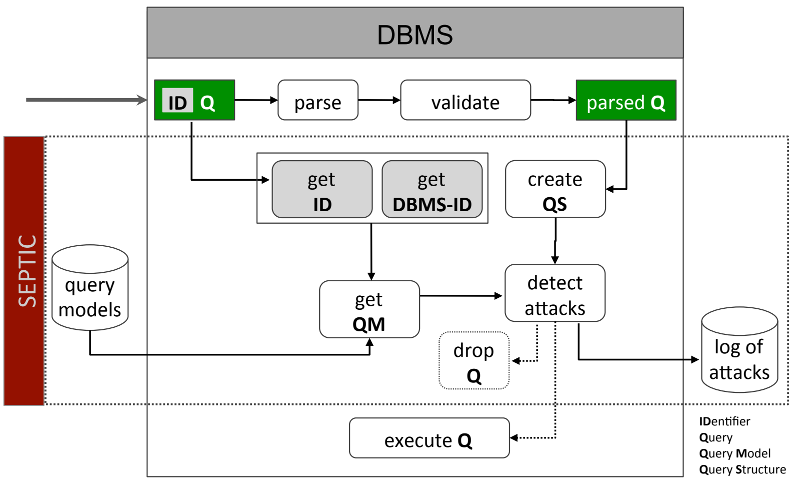
Creating/storing query model



Training mode | **training phase**
Normal mode | **incremental**

111

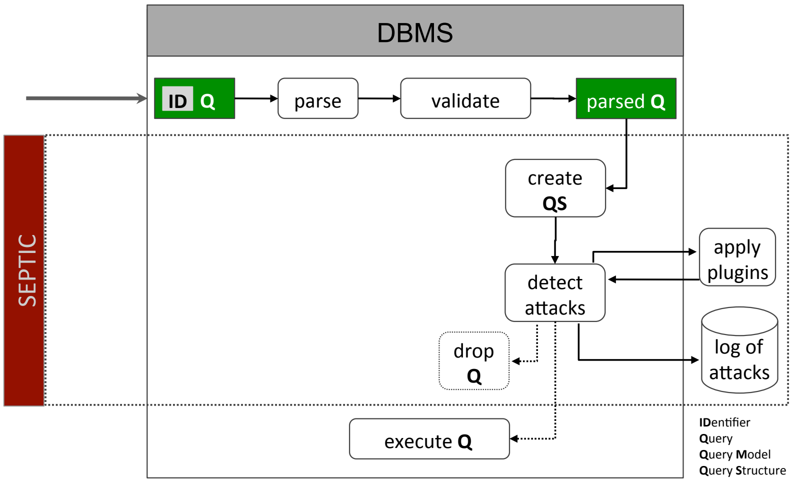
Detecting/blocking SQLI



Normal mode | **prevention or detection**

112

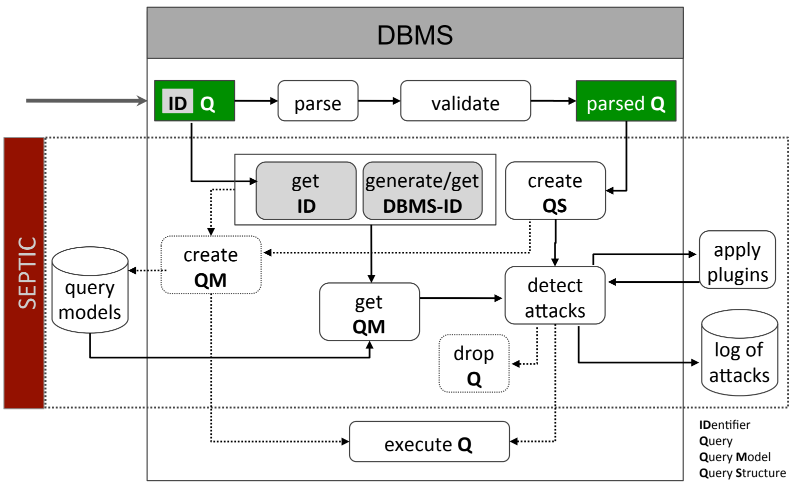
Detecting/blocking stored injection



Normal mode | **prevention or detection**

113

SEPTIC full architecture



114

SEPTIC: outline

- Attack detection in SEPTIC
- Running SEPTIC
- **Results**

115

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

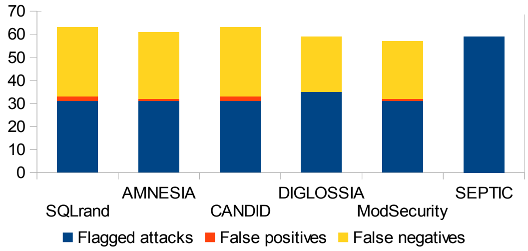
116

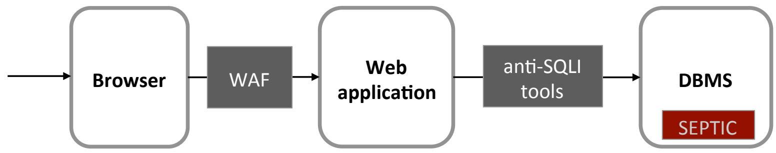
SEPTIC detection w/code samples

- SQLI unrelated to semantic mismatch
  - 23 from the *sqlmap* project
  - 11 by Ray & Ligatti (4 are not attacks/vulnerab.)
  - 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.

117

Comparison with other tools





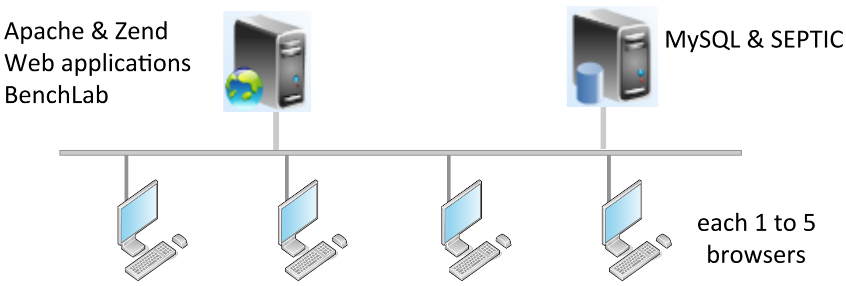
120

SEPTIC: real open source software

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

121

SEPTIC: performance



SEPTIC combinations	
SQLI detector	**Stored inj. det.**
off	off
on	off
off	***on***
on	***on***

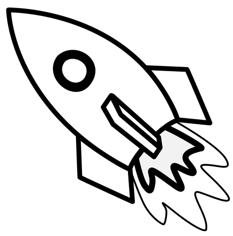
0.82%
↓
2.24%

122

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?*

123



4

SHUTTLE: INTRUSION RECOVERY IN THE CLOUD

124

Cloud computing (public cloud)

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

125

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,…)

126

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,...

127

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...

128

Motivation

129

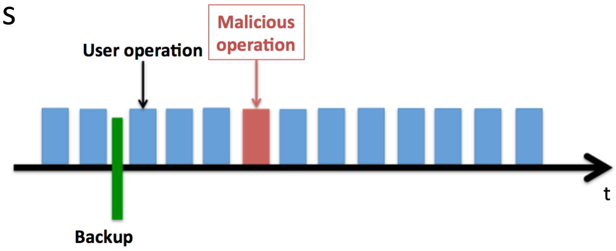
Shuttle: outline

- **Shuttle**
- Evaluation

130

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



131

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

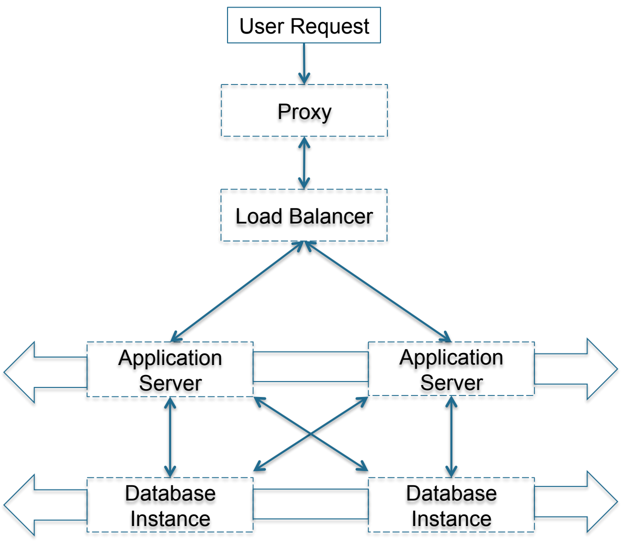
132

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

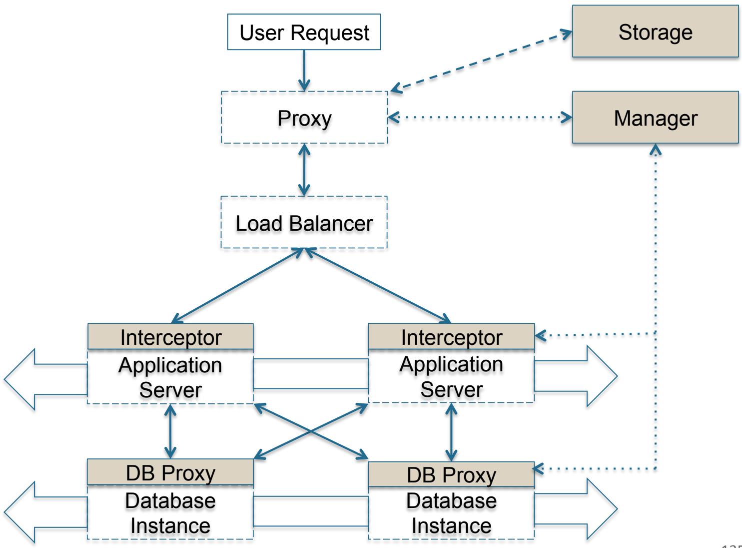
133

PaaS applications architecture



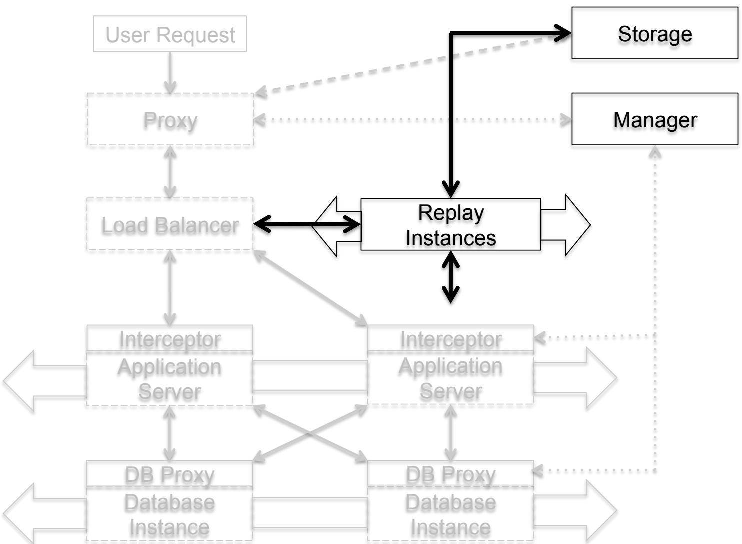
134

Shuttle architecture



135

Shuttle during recovery



136

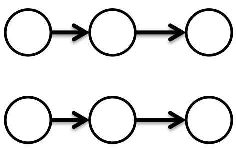
Recovery process

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



- Modes supported:

	Full-Replay	Selective-Replay
1 Cluster (Serial)	✔	✔
Clustered	✔	✗

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)

	# data items affected	# requests tainted	# requests replayed – Selective Replay	# requests replayed – Full Replay
1a	106	0	< 605	38 620
1b	58	14	< 379	38 620
1c	48	52	< 253	38 620
2a	4 338	0	-	38 620
2b	18 286	1 278	-	38 620
3	> 2 000	-	-	38 620

141

Performance overhead

- in normal execution

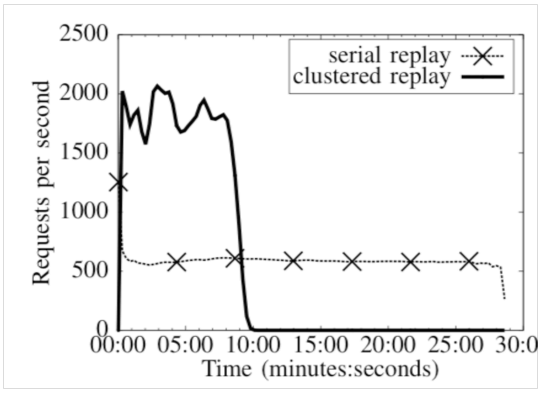
	50% Reads 50% Inserts		95% Reads 5% Inserts	
	ops/sec	latency (ms)	ops/sec	latency (ms)
Shuttle	6325	5.78	15 346	3.62
No Shuttle	7148	5.07	17 821	3.01
overhead	**13%**	**14%**	**16%**	**20%**

Overhead seems acceptable; penalty mostly due to single proxy

142

Recovery time

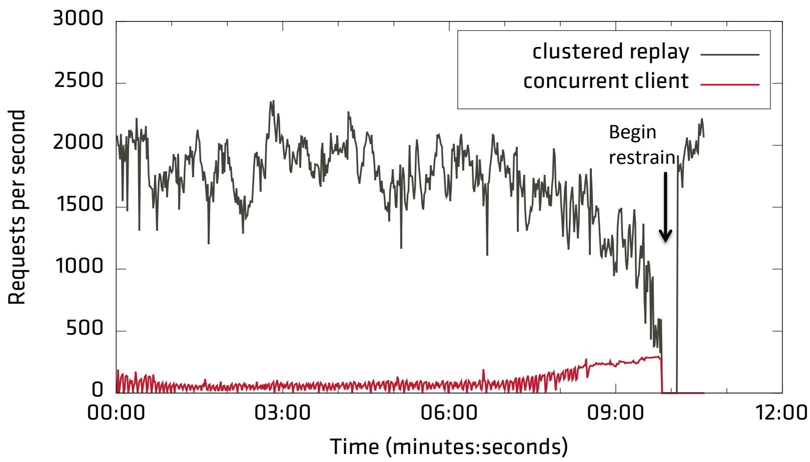
- for 1 million requests



Clustering greatly reduces recovery time

143

Restrain duration



Restrain: 46 seconds

144

Storage overhead

- for 1 million requests

	# objects	Size (MB)
Shuttle Storage:		
Requests	1 million	212
Response	1 million	8 767
Start/End timestamps	2 million	16
Keys	137 million	488
Total		9 648
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

146

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

147

Papers

WAP: I. Medeiros, N. F. Neves, M. Correia. **Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives.** WWW 2014

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

SEPTIC: I. Medeiros, M. Beatriz, N. Neves and M. Correia. **Hacking the DBMS to Prevent Injection Attacks**. CODASPY 2016

SHUTTLE: D. Nascimento, M. Correia. **Shuttle: Intrusion Recovery for PaaS.** ICDCS 2015.

G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, P. Verissimo. **Efficient Byzantine Fault Tolerance**. IEEE Transactions on Computers 2013.

A. N. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa, **DepSky: Dependable and Secure Storage in a Cloud-of-Clouds**. EuroSys 2011 and ACM Transactions on Storage 2013.

148

Thank you

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