JSDC: A Hybrid Approach for JavaScript Malware Detection and Classification

Junjie Wang, **Yinxing Xue**, Yang Liu,
Nanyang Technological University
Tian Huat Tan
Singapore University of Technology and Design
ASIACCS'15, Singapore

Problem

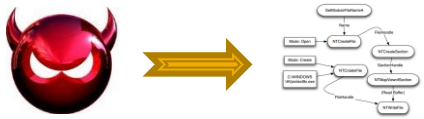
- The New Trend of Malware:

  JS: The largest number of malware in 2013H1 [MSIR:15].

- JavaScript based Malware Detection:
  - JSand [WWW'10] : static analysis + machine learning
  - Prophiler [WWW'11]: static analysis + machine learning
  - ZOZZLE [USS'11]: AST (static) analysis + Bayesian classificatior
  - Revolver [SEC'13]: AST (static) analysis + similarity calculation
  - Cujo [ACSAC'10] : static and dynamic analysis + SVM

- What is missed:

  A hybrid approach
  - to assure both accuracy and performance
  - to detect and also classify JS malware



Contribution

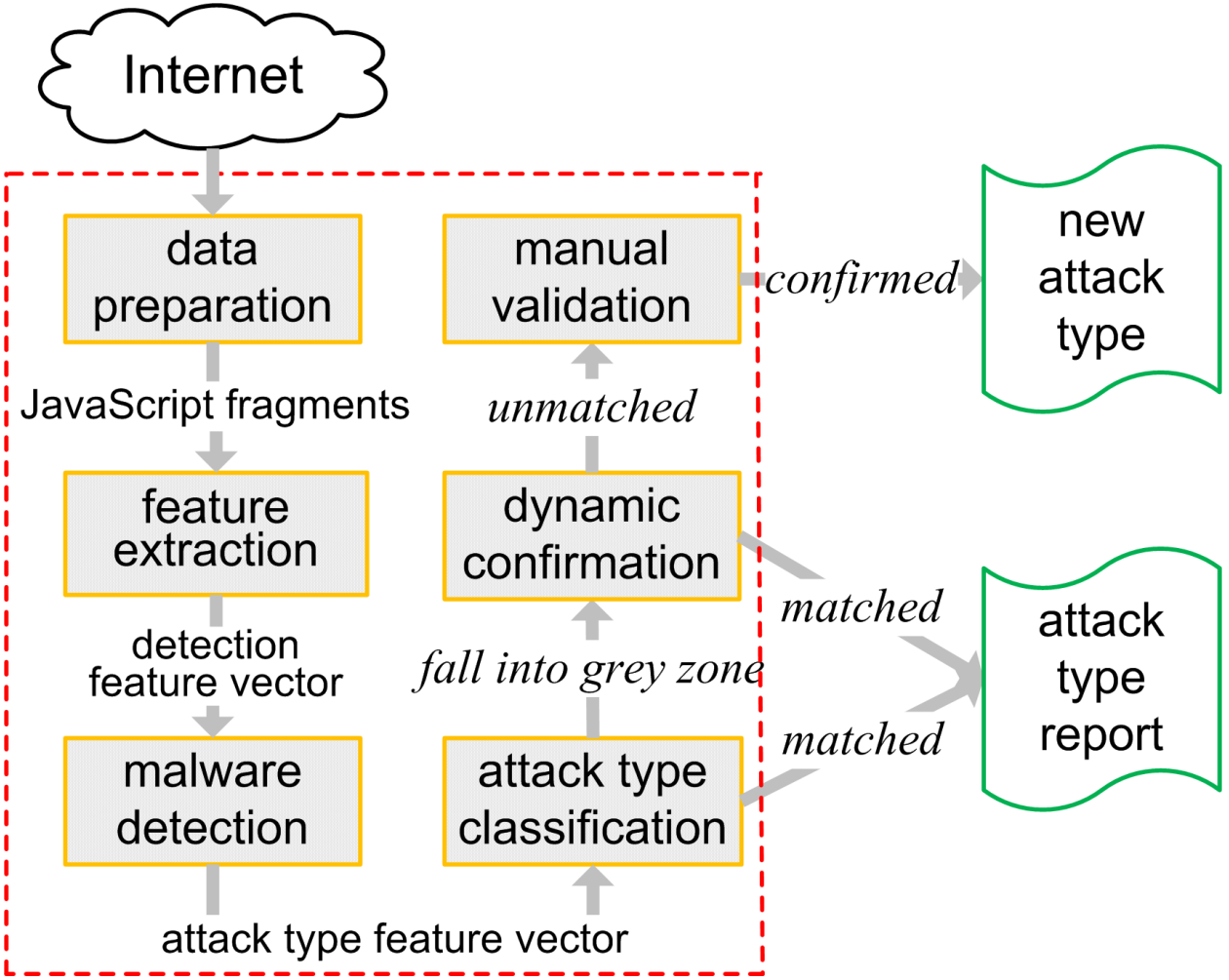
- Technical aspects:
  - Machine learning plus dynamic confirmation.
  - two phase machine learning: first detection, then classification
  - Features extracted from inner- and inter- script program analysis

- Evaluation
  - conduct large-scale evaluations to show its effectiveness.
  - low FP rate (0.2123%) and low FN rate (0.8492%).
  - 1,400,000 real-world JavaScript with over 1,500 malware reported, for which many anti-virus tools failed.

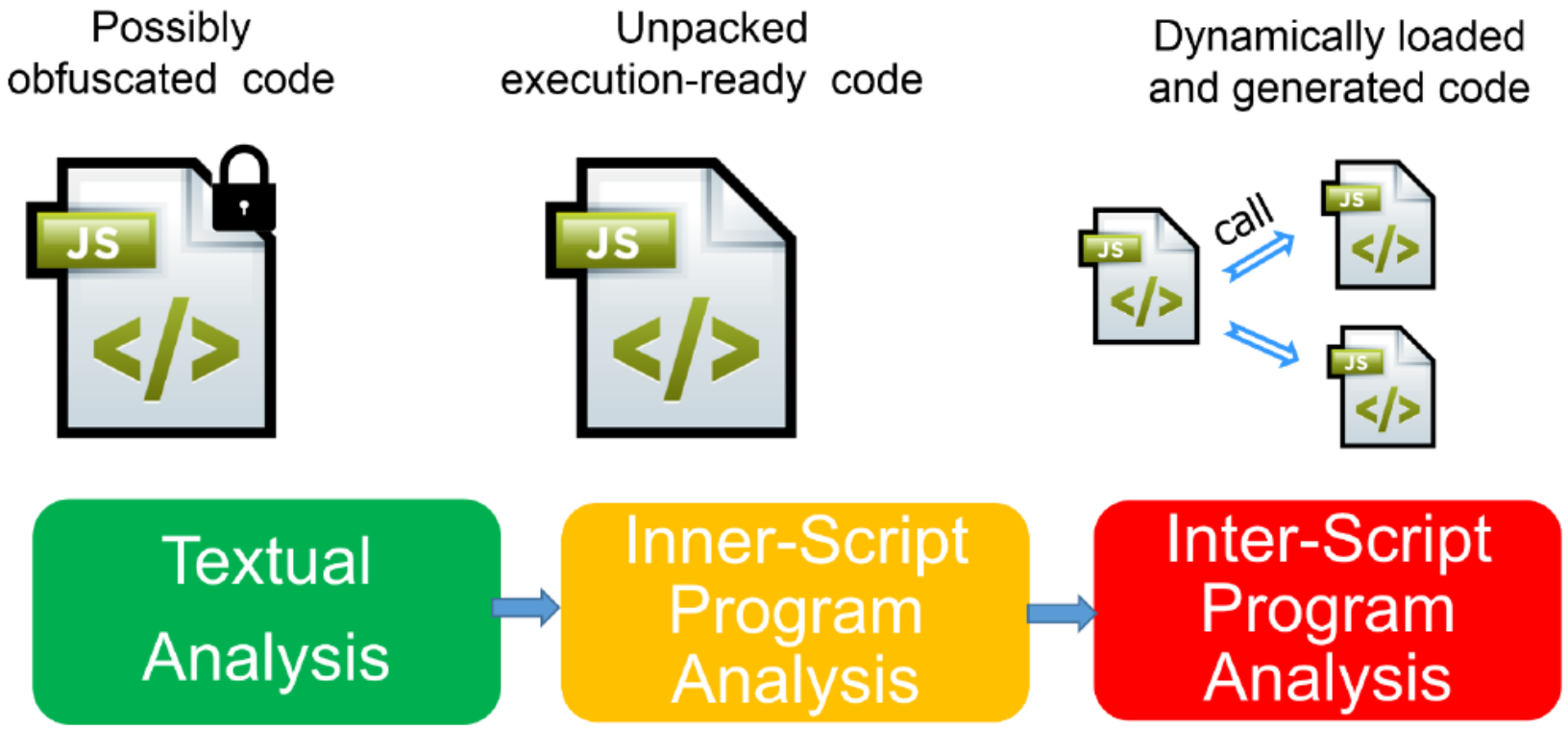
JavaScript Attack type

- A Vulnerability based classification
  - Type I: Attack targeting browser vulnerabilities.
  - Type II: Browser hijacking attack.
  - Type III: Attack targeting Adobe Flash.
  - Type IV: Attack targeting JRE.
  - Type V: Attack based on multimedia.
  - Type VI: Attack targeting Adobe PDF reader.
  - Type VII: Malicious redirecting attack.
  - Type VIII: Attack based on Web attack toolkits, e.g. Blacole.

System Overview of JSDC



Feature Extraction



Example

```
<!--
eval(String.fromCharCode(118,97,114,32,120,101,119,61,52,53,51,
56,48,48,53,52,51,59,118,97,114,32,103,104,103,52,53,61,34,110,
117,111,116,34,59,118,97,114,32,119,61,34,111,34,59,118,97,114,
32,114,101,54,61,34,108,108,46,34,59,118,97,114,32,104,50,104,
61,34,99,111,109,34,59,118,97,114,32,97,61,34,105,102,114,34,59,
118,97,114,32,115,61,34,104,116,116,34,59,100,111,99,117,109,
101,110,116,46,119,114,105,116,101,40,39,60,39,43,97,43,39,97,
109,101,32,115,114,39,43,39,99,61,34,39,43,115,43,39,112,58,47,
47,39,43,103,104,103,52,53,43,39,39,43,119,43,39,39,43,114,101,
54,43,39,39,43,104,50,104,43,39,47,39,43,39,34,32,119,105,100,
39,43,39,116,104,61,34,49,34,32,104,39,43,39,101,105,103,104,
116,61,34,51,34,62,60,47,105,102,39,43,39,114,97,109,101,62,39,
41,59,32,118,97,114,32,106,104,114,52,61,52,51,50,52,50,50,52));
//-->
```

(a) The original obfuscated version

```
var xew = 453800543;
var ghg45 = "nuot";
var w = "o";
var re6 = "ll.";
var h2h = "com";
var a = "ifr";
var s = "htt";
document.write("<"+a+"ame sr"+"c=\""+s+"p://"+ghg45+""+w+""+re6
+""+h2h+"/"+"\"wid"+"th=\"1\"h"+"eight=\"3\"></if"+"rame>");
var jhr4 = 4324224;
```

(b) The HtmlUnit unpacked version

Features for Detection

- Textual Analysis
  - Longest word size -- 814 as shown in figure (a)
  - Entropy -- obfuscated code is usually lower than 1.2, 1.1 as shown in figure (a)
  - Byte occurrence frequency of specific character -- 232 comma characters
  - Commenting style -- <!-- and //-->
- Inner-Script Program Analysis
  - Function calls with security risks
    - 7 types of 23 functions
  - AST features
    - e.g. the depth of the AST, the maximum breadth
  - Function call patterns
    - newActiveXObject() and createXMLHTTPRequest() are widely used by malware targeting vulnerability in ActiveX components

Features for Classification

- Inter-Script Program Analysis
  - we count external scripts from other domains.
  - Miscellaneous and derived features
    - feature *changeSRC* counts the number of changing of the src attribute (e.g., for <iframe src="..."/> tag)
    - *domAndDynamicUsageCnt* counts the number of invocation for APIs that change DOM structure or supporting dynamic execution of JavaScript code
    - *dynamicUsageContentLen* stores the length of contents that are passed as arguments to APIs that support dynamic execution of JavaScript;
    - .....

Example

- Calling external JavaScript

```
<script src="http://xxx.xxx.xxx/a.js"></script>
<script>
 new_element=document.createElement("script");
 new_element.setAttribute("type","text/javascript");
 new_element.setAttribute("src","a.js");//
 document.body.appendChild(new_element);
 function b() {
   a(); //a() is a function in a.js that contains malicious code
 }
</script>
```

Evaluation

- Data sets used in controlled experiments:

Benign data set	#samples
Alexa-top500 websites	20000
Malicious data sets	**#samples**
Attack targeting browser vulnerabilities (type I)	150
Browser hijacking attack (type II)	28
Attack targeting Flash (type III)	81
Attack targeting JRE (type IV)	191
Attack based on multimedia (type V)	190
Attack targeting PDF reader (type VI)	101
Malicious redirecting attack (type VII)	92
Attack based on Web attack toolkits (type VIII)	109
Total	942

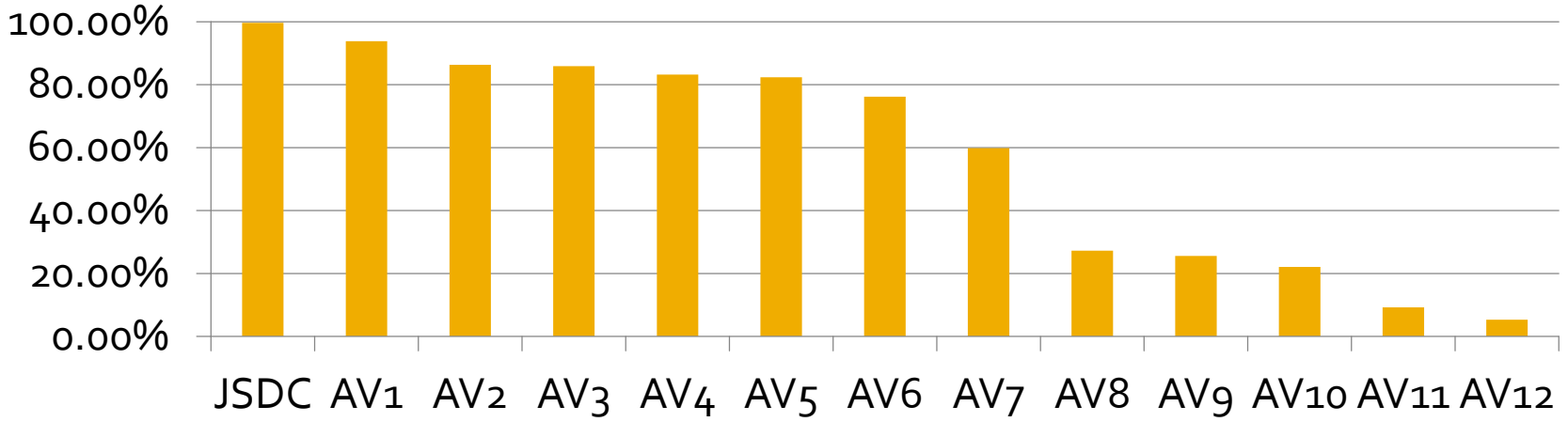
- Data sets used in wild predication:
  - 1,400,000 scripts crawled by Heritrix with randomly selected seeds,
  - from web sites of universities, governments, companies, discussion forums, etc.

Detection on the Labelled Data Sets

- Accuracies of different classifiers

ML classifier	Accuracy	FP rate	FN rate
RandomForest (RF)	99.9522%	0.2123%	0.8492%
J48	99.8615%	0.7431%	2.335%
Näive Bayes (NB)	98.2237%	1.127%	4.5280%
RandomTree (RT)	99.8758%	0.3609%	1.4862%

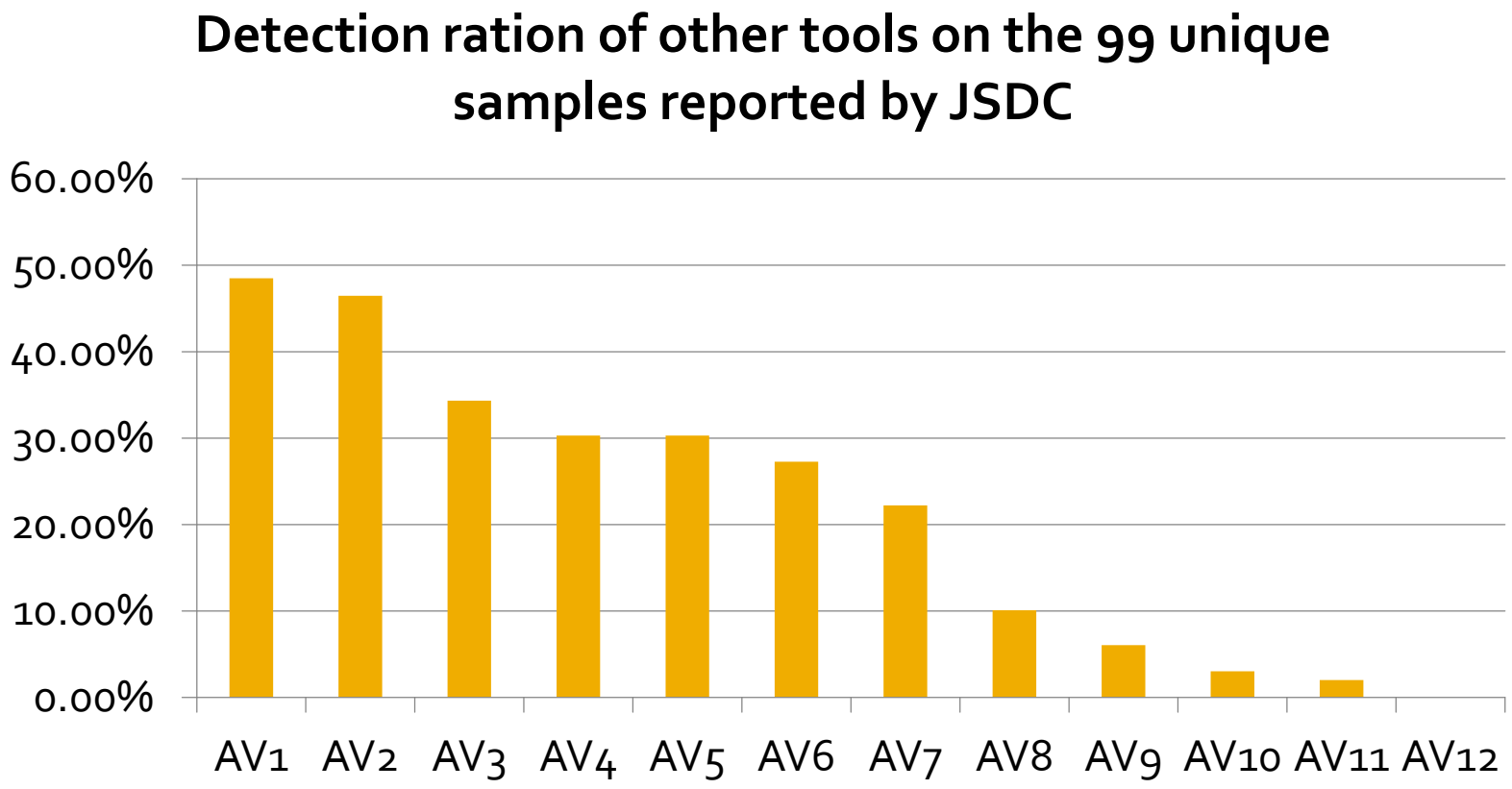
- Comparison with anti-virus tools



Detection on the Unlabelled Data Set

- 1,400,000 wildly crawled scripts, the best trained classifier RF predicates 1,530 snippets as malicious.
- manually inspect 100 cases (randomly selected).
- only 1 FP case.
- 11 out of 99 TP cases are missed by all the tools.

Detection on the Unlabelled Data Set



Evaluation of Attack Type classification

- The accuracy of the trained model on 942 known JS malware:

a	b	c	d	e	f	g	h	<–classified as
139	0	0	0	9	2	0	0	a = type I
0	23	4	0	0	0	0	1	b = type II
1	1	74	1	0	0	1	3	c = type III
0	0	2	179	9	0	1	0	d = type IV
1	0	0	0	179	10	0	0	e = type V
0	0	0	0	19	82	0	0	f = type VI
0	0	0	1	1	0	87	3	g = type VII
0	0	0	1	0	0	3	105	h = type VIII

- The accuracy in classification of 1530 wild JS malware

Type	type I	type II	type III	type IV
Num	113 (7.39%)	10 (0.65%)	75 (4.90%)	253 (16.54%)
Type	type V	type VI	type VII	type VIII
Num	202 (13.20%)	101 (6.60%)	350 (22.88%)	426 (27.84%)

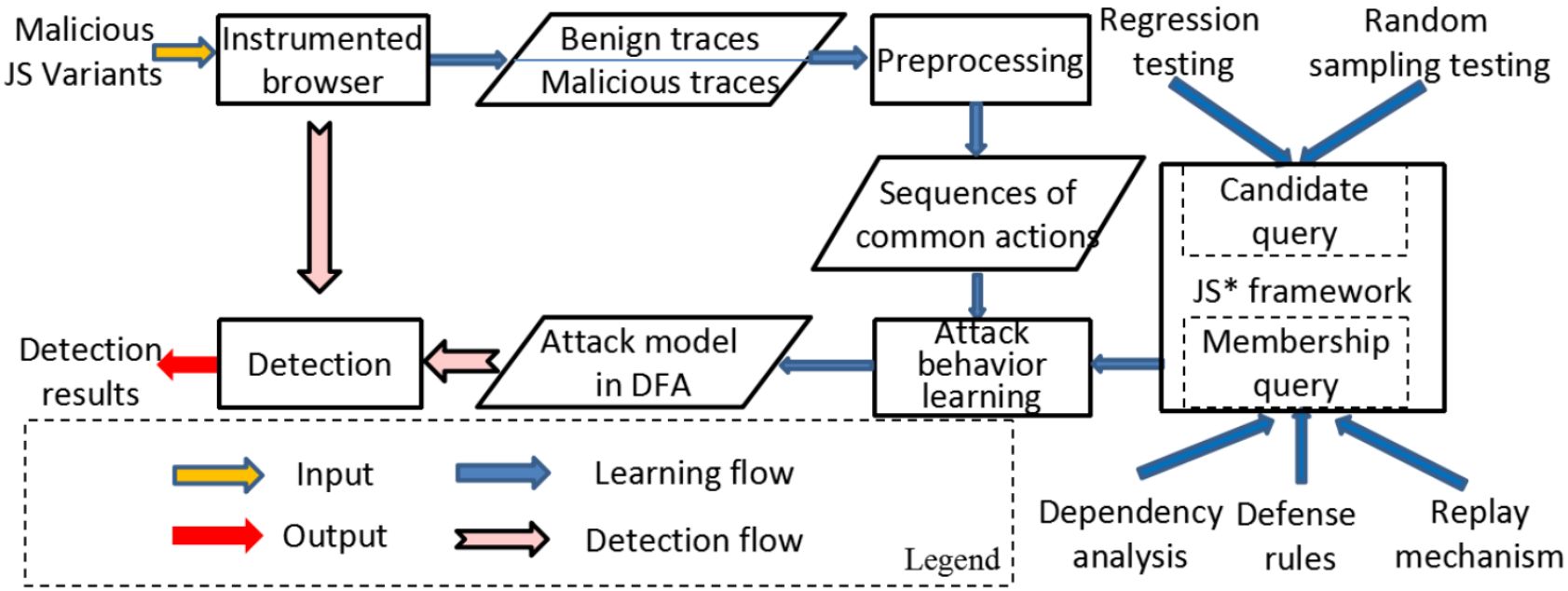
- we manually check 164 samples --- an accuracy of 87.8% (144/164)
  - Among the 20 error cases, 9 samples do not belong to any of the eight attack types and 11 samples are classified into the wrong types.

Certainty for Grey Zone

- The dynamic confirmation is applied on uncertain cases that fall into the grey zone during attack type classification.
- The certainty value and the number of samples that fall into grey zone

certainty	certain#	total	uncertain#	uncertain %
1	764	1530	766	51.31%
$\geqslant$0.9	854	1530	676	50.07%
$\geqslant$0.8	994	1530	536	44.18%
$\geqslant$0.7	1296	1530	234	15.29%
$\geqslant$0.6	1311	1530	219	14.31%

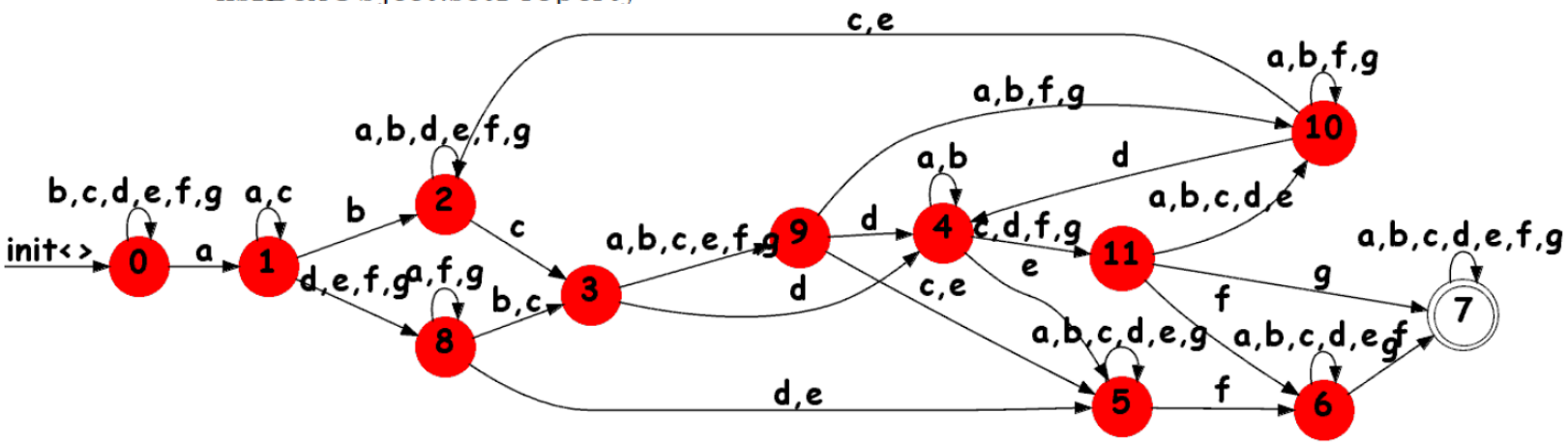
Dynamic Confirmation



- JS* framework [ISSTA'15]:
  - based on L* algorithm that learns a DFA from a set of strings

Example

System calls	Actions
nsISupportsString.data	
nsICommandLine.resolveURI	$\Rightarrow a$
nsIObserverService.notifyObservers	
nsIWebNavigation.loadURI	$\Rightarrow b$
nsIScriptableUnicodeConverter.convertToByteArray	$\Rightarrow c$
nsICryptoHash.update	$\Rightarrow d$
nsILocalFile.append	$\Rightarrow e$
nsITimer.initWithCallback	$\Rightarrow f$
nsIIOService2.newURI	$\Rightarrow g$
nsIScriptableUnicodeConverter.convertToInputStream	
nsIBoxObject.setProperty	



Performance of Malware Detection

- **Machine Learning**

Table 5: The running time for different classifiers

Operation	Num	Time(s)	Avg(ms)
Feature extraction	20942	1660.7	79.3
Training(RandomForest)	20942	0.785	0.037
Training(J48)	20942	0.364	0.017
Training(Näive Bayes)	20942	0.124	0.006
Training(RandomTree)	20942	0.275	0.013
Detection(RandomForest)	1,400,000	57.4	0.041
Detection(J48)	1,400,000	26.6	0.019
Detection(Näive Bayes)	1,400,000	8.4	0.006
Detection(RandomTree)	1,400,000	19.6	0.014

- **Dynamic Confirmation**

Discussion

- **Two- or one-phase machine learning classification?**
  - On the 20942 training samples, the accuracy of the 4 trained classifiers is 90.99% (RF), 85.74% (J48), 77.15% (NB) and 88.27% (RT), respectively.

- **Predicative features.**
  - 89% of Type I samples have feature changeSRC <1;
  - 52% of Type II have feature with a value>20 ;
  - 74% of Type III have feature eval with a value>1000;
  - 83% of Type IV have feature GetUserAgent > 2:5;
  - ….

Conclusions

- Our method not only learned features of maliciousness but also of attack type.
- We also demonstrated our effectiveness and efficiency by empirical wild prediction.
- Among over 1,400,000 scripts, we find over 1,500 malware with 8 attack types.
- Our detection speed is scalable with below 80 ms per script.

- Thanks! ☺

Function calls with security risks

Function Name	Function Type	Possible Threats
eval() window.setInterval() window.setTimeout()	Dynamic code execution	Dynamic code generation
location.replace() location.assign()	Change current URL	Redirect to malicious URL
getUserAgent() getAppName()	Check browser	Target specific browser
getCookie() setCookie()	Cookie access	Manipulate Cookie
document.addEventListener() element.addEventListener()	Intercepting Events	Block user's operation or emulating
document.write() element.changeAttribute() document.writeln() element.innerHTML() element.insertBefore() element.replaceChild() element.appendChild()	DOM operation	Embed malicious script, invisible java applets, invisible iframe, invisible silverlight, etc.
String.charAt() String.charCodeAt() String.fromCharCode() String.indexOf() String.split()	String operation	Hide intension, by encoding and encryption

Function call patterns

Function call pattern:	Intension:
unescape()	obfuscation to evade checking
eval()	dynamic generation
GetCookie()	check Cookie
dateObject.toGMTString()	generate time String used in cookie
SetCookie()	set cookie to mark
document.write()	generate dynamic document content
document.createElement()	create new document element
element.appendChild()	append new element to current one
newActiveXObject()	create new Active object
createXMLHTTPRequest()	download exploit file to local system