Combining Static Analysis and Machine Learning for Industrial-quality Information-flow-security Enforcement

Marco Pistoia

IBM T. J. Watson Research Center
Yorktown Heights, New York

Workshop on Software Correctness and Reliability
ETH Zurich, October 2015

Joint Work with…

- Omer Tripp
- Patrick Cousot
- Radhia Cousot
- Salvatore Guarnieri
- Aleksandr Aravkin

Part 1

MOTIVATIONS

Top Web/Mobile Vulnerabilities

Injection	Broken Authentication and Session Management	Cross-site Scripting	Insecure Direct Object Reference
Security Misconfiguration	Sensitive Data Exposure	Missing Function Level Access Control	Cross-site Request Forgery
Using Components with Known Vulnerabilities	Unvalidated Redirects and Forwards	Unintended Data Leakage	Broken Cryptography

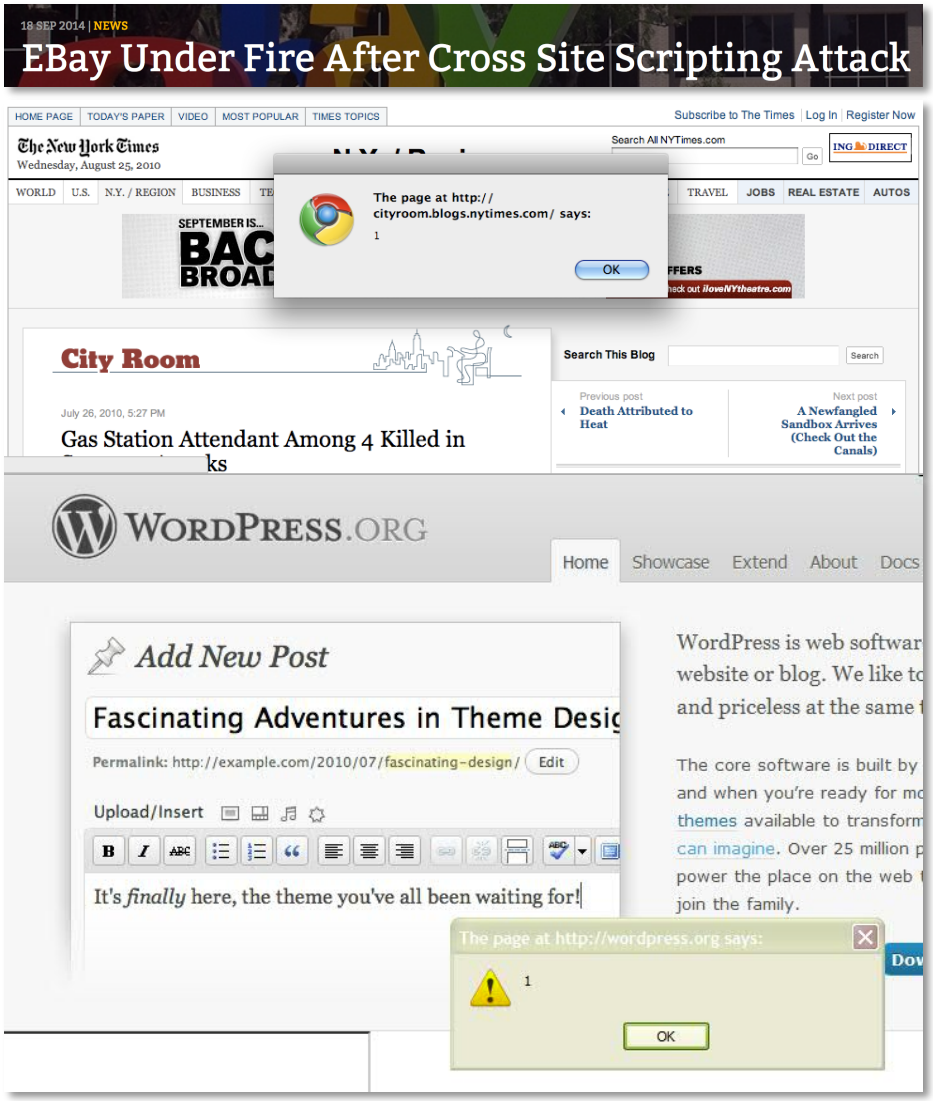
Information Flow Access Control Configuration

Injection


```
String query = "SELECT *
  FROM users WHERE name='"
    + userName + "' AND
    pwd='" + pwd + "'";
```

- **Date:** April 20, 2011
- **Target:** SONY's PlayStation Network
- **Impact:**
  - 77 million PlayStation Network accounts hacked
  - Attackers gained access to full names, passwords, e-mails, home addresses, purchase history, credit card numbers, and PSN/Qriocity logins and passwords
  - SONY is said to have lost millions while the site was down for a month

Cross-site Scripting (XSS)



- **Date:** September 18, 2014
- **Target:** eBay
- **Impact:**
  - A XSS vulnerability allowed attackers to redirect users to a phishing page.
  - The hackers had apparently exploited the common vulnerability to inject malicious Javascript into several listings
  - Users were taken to what appeared to be an eBay log-in page, which was actually hosted elsewhere and had been designed to harvest user log-ins for the hackers

Sensitive Data Exposure

- **Date:** September 2, 2014
- **Target:** The Home Depot
- **Impact:**
  - Customers' credit card information was stored unencrypted
  - 56 million credit cards were compromised
  - $62 million estimated damage

Cross-site Request Forgery (CSRF)

- **Date:** May 19, 2014
- **Target:** Facebook
- **Impact:**
  - CSRF attacks are used by hackers to gain access to online accounts to which the victim has signed in
  - Facebook disclosed that its system was vulnerable to CSRF
  - Attackers were able to impersonate other users and steal information
  - The CSRF token contained a truncated SHA-2 hash that incorporated the account ID and current date
  - A person with 3 Facebook sessions within a single day would have received an identical CSRF token each time," Facebook engineers Chad Parry and Christophe Van Gysel said in a statement. "Now our system replaces the token with a new one every time it is requested."

Unvalidated Redirects and Forwards

ZDNet

300+ Bank homepages hacked and redirected!

Summary: *A little more than half of the 600 hosted bank sites were modified to redirect traffic which puts the total number of Banks affected at over 300. The homepages of those banks were modified so that they would direct all online banking traffic to a malicious site in Madrid Spain to collect login credentials from unsuspecting customers.*

- **Date:** June 1, 2006
- **Target:** Goldleaf Financial Solutions
- **Impact:**
  - >300 bank home pages hacked and redirected to a malicious site in Madrid, Spain
  - User login information collected from unsuspecting customers

cnet

Serious security flaw in OAuth, OpenID discovered

Attackers can use the "Covert Redirect" vulnerability in both open-source log-in systems to steal your data and redirect you to unsafe sites.

- **Date:** May 2, 2014
- **Target:** Oauth, OpenID
- **Impact:**
  - Attackers can use this vulnerability in both open-so log-in systems to steal user data and redirect user to u sites
  - The log-in tools OAuth and OpenID are used by many Web sites and tech titans including **Google, Facebook, Microsoft, and LinkedIn**

Unintended Data Leakage

PCWorld

Hackers claim to expose phone information of 4.6 million Snapchat users

NETWORKWORLD

Snapchat says sorry for the hack, with a tweak to its app

The company's mobile app now lets users de-link their phone numbers from their usernames

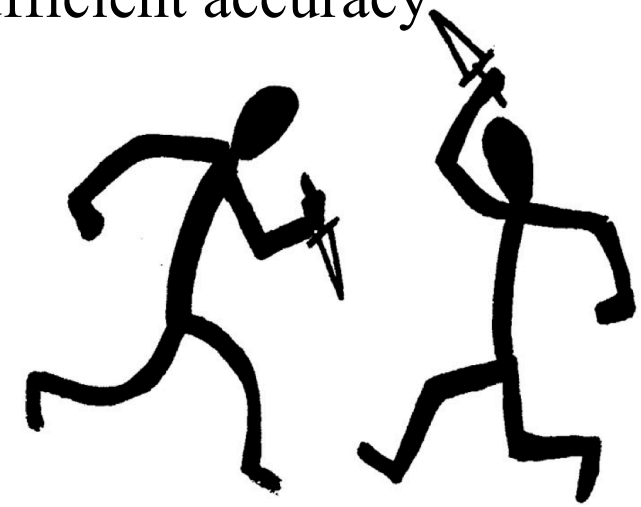
- **Date:** January 1, 2014
- **Target:** Snapchat
- **Impact:**
  - 4.6 million private users' information exposed
  - Company had to allow phone numbers to be de-linked from mobile app

Part 2

PROGRAM ANALYSIS FOR INFORMATION-FLOW-SECURITY ENFORCEMENT

Existing Static-Analysis Solutions

- Type systems:
  - Complex, conservative, require code annotations
- Classic slicing:
  - Has not been shown to scale to large applications while maintaining sufficient accuracy



Precision **Scalability**

TAJ (PLDI 2009)

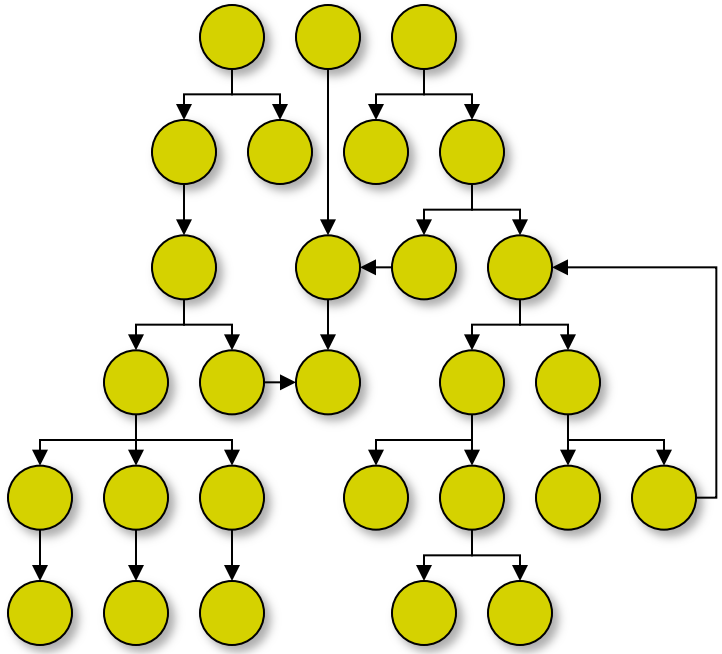
- Pointer analysis is a variant of Andersen's analysis
- Analysis is field sensitive
- Analysis is intraprocedurally flow sensitive and interprocedurally flow insensitive (accounting for multithreaded code)
- Custom context-sensitivity policy:
  - Unlimited-depth object sensitivity for Java collections (up to recursion)
  - One level of call-string context for factory methods
  - One level of call-string context for taint APIs
  - One-level receiver-object context-sensitivity as default

Andromeda (FASE 2013)

- Web applications are large and complex
- Sound analyses
  - If too precise, do not scale well
  - If too imprecise, have too many false positives
- Unsound analyses
  - Have false negatives
  - Are often unstable (extra-sensitivity to program changes)

Intuition behind Andromeda

- Taint analysis can be treated as a demand-driven problem
- This enables lazy computation of vulnerable information flows, instead of eagerly computing a complete data-flow solution



Publications on Andromeda

- FASE 2013 – Andromeda algorithm
  - Omer Tripp, Marco Pistoia, Patrick Cousot, Radhia Cousot, Salvatore Guarnieri, "Andromeda: Accurate and Scalable Security Analysis of Web Applications"
- ISSTA 2014 – Andromeda for JavaScript and String Analysis (ACM SIGSOFT Distinguished Paper Award)
  - Omer Tripp, Pietro Ferrara, Marco Pistoia, "Hybrid Security Analysis of Web JavaScript Code via Dynamic Partial Evaluation"
- OOPSLA 2011 – Integration with Framework for Frameworks (F4F)
  - Manu Sridharan, Shay Artzi, Marco Pistoia, Salvatore Guarnieri, Omer Tripp, Ryan Berg, "F4F: Taint Analysis of Framework-based Web Applications"
- ISSTA 2011 (1) – Andromeda for JavaScript
  - Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolby, Stephen Teilhet, Ryan Berg, "Saving the World Wide Web from Vulnerable JavaScript"
- ISSTA 2011 (2) – Andromeda as the basis for String Analysis (ACM SIGSOFT Distinguished Paper Award)
  - Takaaki Tateishi, Marco Pistoia, Omer Tripp, "Path- and Index-sensitive String Analysis based on Monadic Second-order Logic"
- IBM Journal on Research and Development 2013 – Permission analysis for Android applications
  - Dragoș Sbîrlea, Michael G. Burke, Salvatore Guarnieri, Marco Pistoia, Vivek Sarkar, "Automatic Detection of Inter-application Permission Leaks in Android Applications"

Contributions of Andromeda

- Scalable and sound demand-driven taint analysis
- Modular analysis
- Incremental analysis
- Framework and library support
- Multiple language support (Java, .NET, JavaScript, Android)
- Inclusion in an IBM product: IBM Security AppScan Source

Motivating Example

```
public class Aliasing5 extends HttpServlet {
   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
         throws ServletException, IOException {
      StringBuffer buf = new StringBuffer("abc");
      foo(buf, buf, resp, req);
   }

   void foo(StringBuffer buf, StringBuffer buf2, ServletResponse resp,
         ServletRequest req) throws IOException {
      String name = req.getParameter("name");
      buf.append(name);
      PrintWriter writer = resp.getWriter();
      writer.println(buf2.toString()); /* BAD */
   }
}
```

High-level Algorithm

- Input: Web application plus supporting rules
  - {(*Sources*, *Sinks*, *Sanitizers*)}
- Build class hierarchy
- Construct CHA-based call graph with intra-procedural type-inference optimization
- Perform data-flow analysis (explained next)
- Report any flow from a source to a sink not intercepted by a sanitizer in the same rule

Abstract Domain

- Consists of triplets:
  - Method where Static Single Assignment (SSA) variable is defined
  - SSA variable ID
  - Access path
- Inputs form a lattice according to subsumption relation defined on access paths, *e.g.*:

  `o.* ≥ o.f.* ≥ o.f.g`

  - The `*` symbol represents any feasible sub-path
  - Array load/store semantics is applied to arrays, maps, session objects, etc.

Modularity of the Analysis

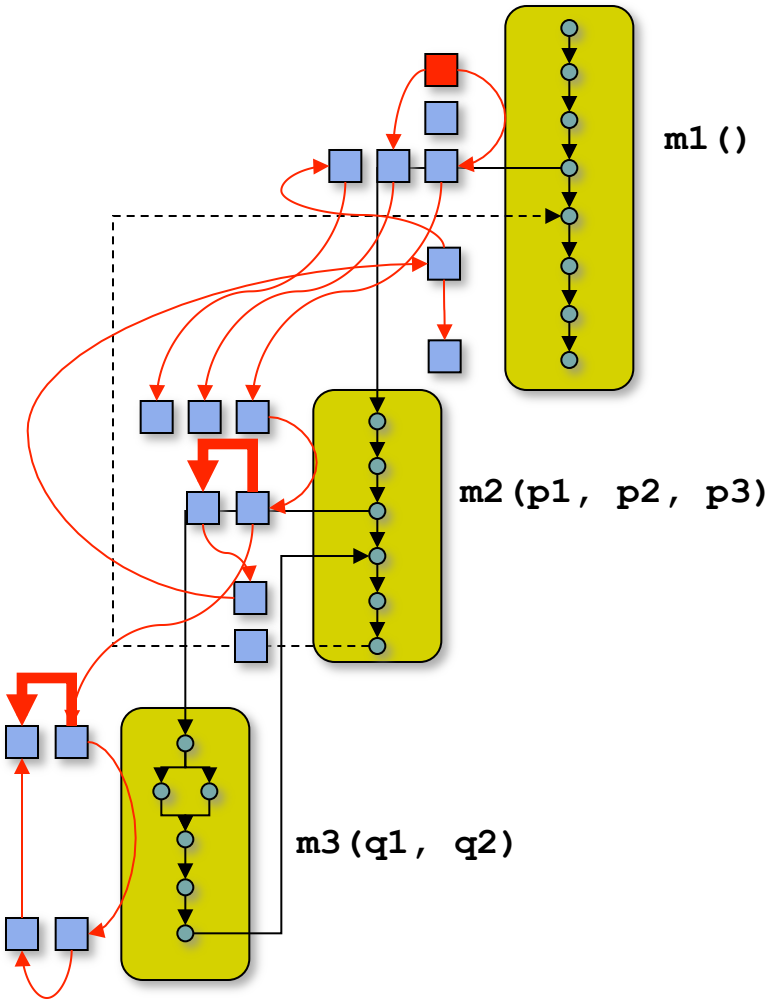
- Runs on data flow (def-to-use)
- Produces and uses pre-compiled models
  - Format:

    `<method, entry>` → `<method, exit>`
  - Example:

    `<m, v2.f.g>` → `<m, v1.h>`

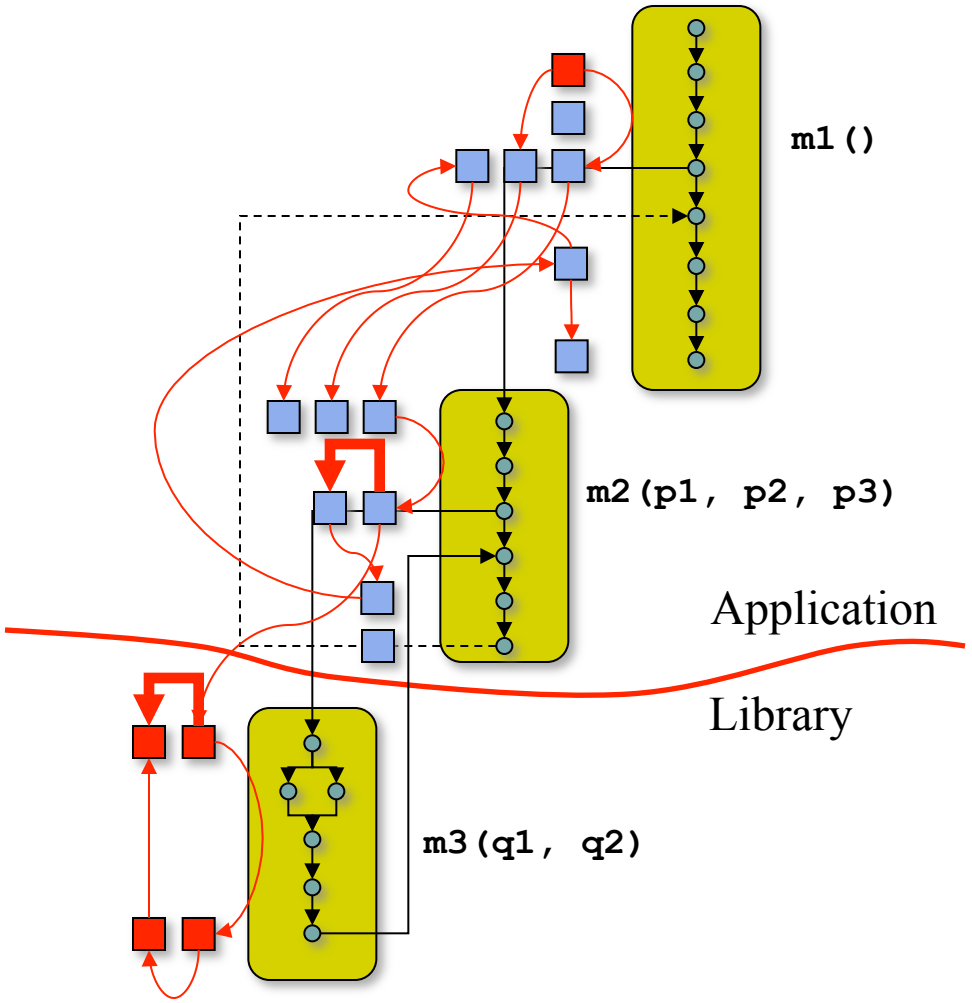
A Novel Approach to Taint Analysis

- Start from taint sources
- Propagate taint intra-procedurally through def-to-use
- Inter-procedurally propagate taint forward and record constraints in callees
- Record constraints on call sites, recursively (allows for polymorphism)
- Resolve aliasing by going back to allocation sites
- In the final *constraint-propagation graph*, detect paths between sources and sinks not intercepted by sanitizers



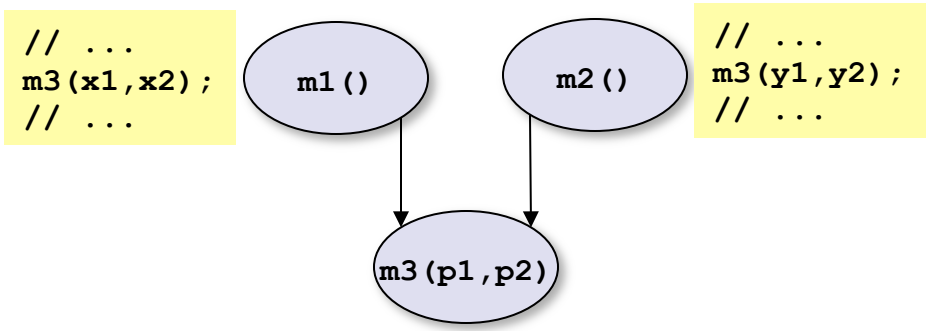
Modular Analysis

- Persist constraint edges at library entrypoints
- Constraint edges are mapped to contexts
- During analysis time, the constraint edges specific to a particular context are used
- Summaries are source-, sink- and sanitizer-specific



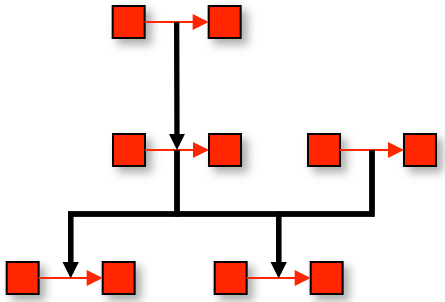
Backward Propagation

- Pushes constraints back to callers
- The constraint `p1.f.g` → `p2.h` in `m3` is propagated to `m1` and `m2` (and, recursively, to their callers)
  - `x1.f.g` → `x2.h`
  - `y1.f.g` → `y2.h`



Incremental Analysis

- A *taint constraint* is an edge in the constraint-propagation graph 
- The *support graph* records how constraints were learned (*i.e.*, based on which other constraints) 
- Facts learned in a scope that underwent change are transitively invalidated
- Preconditions recomputed
- Fixed-point analysis recommenced



Integration with F4F

- F4F (OOPSLA 2011) analyzes code and metadata of frameworks and represents them in artifacts written in an XML-like language
- Andromeda translates those artifacts into legal Java code that – from a data-flow perspective – is equivalent to the original framework code
- New code is human-readable and reusable by other analyzers
- New code is compiled and added to the analysis scope

Integration with Monadic Second-order Logic String Analysis (Best Paper at ISSTA'11)

- *String analysis* is a static analysis that, given a String variable in a program, produces the grammar of the language of all the possible values that that variable can take at run time
- We designed and implemented a novel string analysis based on Monadic Second-order Logic
- This analysis reduces:
  - False negatives, by detecting incorrect sanitizers and validators
  - False positives, by detecting unknown sanitizers and validators

Custom Sanitizers

```
static final String PUNCTUATION_CHARS_ALLOWED = " ()&+,-=._$";
static String cleanLink(String link){
  return cleanLink(link, PUNCTUATION_CHARS_ALLOWED); }

static String cleanLink(String link,
                        String allowedChars){
  if (link == null) return null;
  link = link.trim();
  StringBuffer clean=new StringBuffer(link.length());
  boolean isWord = true; boolean wasSpace = false;
  for (int i = 0; i < link.length(); i++){
    char ch = link.charAt(i);
    if (Character.isWhitespace(ch)) {
      if (wasSpace) continue;
      wasSpace = true;
    } else { wasSpace = false; }
    if (Character.isLetterOrDigit(ch)||
        allowedChars.indexOf(ch) != -1) {
      if (isWord) ch = Character.toUpperCase(ch);
      clean.append(ch); isWord = false;
    } else { isWord = true; }
  }
  return clean.toString(); }
```

Substring extraction

String validation

String replacement

Experimental Results*

	ANDROMEDA	TAJ
Average TPs	82%	68%
Average FPs	12%	30%
Average Unknowns	6%	2%

Change Type	**Response Time (s)**			
	AltoroJ		Webgoat	
	Deletion	**Addition**	**Deletion**	**Addition**
Taint-propagator statement	2	2.2	1.9	2.2
Security sink	0.5	2	1.9	2.5
Security source	2.1	2.1	1.8	3.2
Irrelevant statement	1.9	2	2.5	2.8
Relevant method	2.2	1.9	1.8	2.7
Irrelevant method	2.2	1.7	1.7	1.7

* More details in paper

What We Learned

- The notorious scalability barrier finally lifted without compromising soundness
- Incremental analysis is a great promise for developers
- Production summaries already generated
- Industrial-level analysis must include support for:
  - Frameworks
  - String analysis
  - Multiple languages

Part 3

INTEGRATION WITH MACHINE LEARNING

Dimensions of Precision

[redacted].scan

Issue #1 (jsDOMXSSandOpenRedirect)

[redacted]/Default?Openpage

```
13      if (protocol == "https://" & window.location.protocol == "http:") {
14          var host = window.location.hostname;
15          var pathname = window.location.pathname;
16  1       var search = window.location.search;
17  2       var url = protocol + host + pathname + search;
18  3       location.replace(url);
19      }
20
21      function setFormFocus() {
```

Flow insensitivity

```
x.f = read();
x.f = "";
write(x.f);
```

Path insensitivity

```
x.f = "";
if (b)
  x.f = read();
if (!b)
  write(x.f);
```

Context insensitivity

```
y1 = id(x);
y2 = id(read());
write(y1);
```

Main Problem

- B. Johnson, Y.Song, E. Murphy-Hill, and R. Bowdidge: *Why don't software developers use static analysis tools to find bugs?* In ICSE 2013
- Answer: Too many false positives

Aletheia (CCS 2014)

- Aletheia is a Machine Learning system that acts on the output of *any* static security analyzer
- To evaluate Aletheia, we ran a commercial static JavaScript security checker on a set of:
  - 1,700 HTML pages
  - Taken from the 675 top-popular Web sites
  - Which resulted in a total of 3,758 warnings
  - Classified warnings: 200
- Policy *preserving of true positives*
  - Precision improvement: ×2.868
  - Recall degradation: ×1.006
- Policy *reducing false alarms*
  - Precision improvement: ×9,014
  - Recall degradation: ×2.212

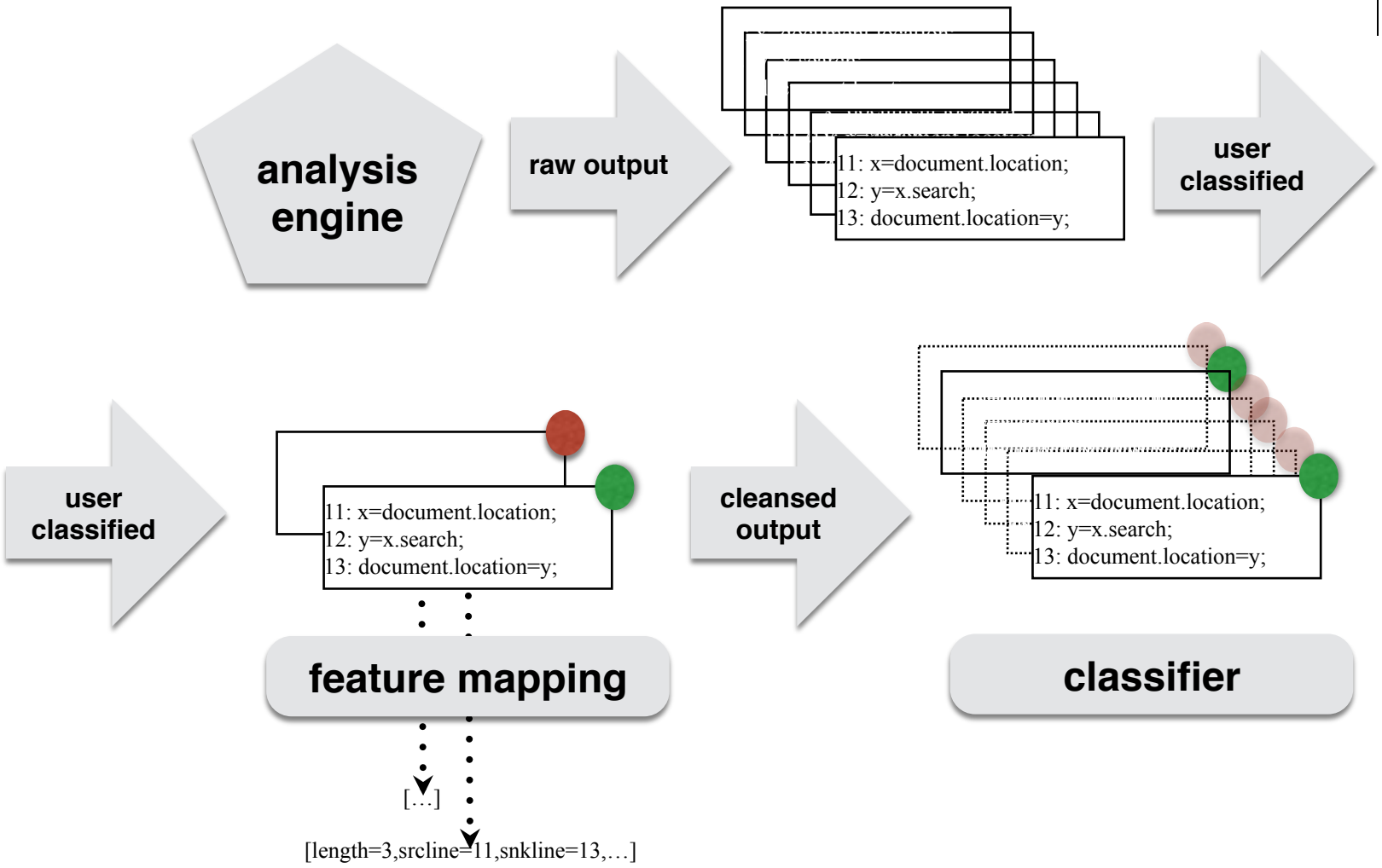
$$p = \frac{tp}{tp + fp} \quad (precision)$$

$$r = \frac{tp}{tp + fn} \quad (recall)$$

The Aletheia System

- Input to Aletheia:
  - Raw warnings $W = \{w_1, \ldots, w_k\}$
  - Classified warnings $\{(w_{i1}, b_{i1}) \ldots, (w_{ik}, b_{ik})\}$, where
    - $w_{i1}, \ldots, w_{ik}$ are randomly chosen in R
    - $b_{i1}, \ldots, b_{ik}$ are Boolean values indicating whether the corresponding warning is a true or false positive
- Aletheia outputs a classified subset of W

The Architecture of Aletheia



Feature Mapping

- Feature mapping derives simple-structured features from the warnings
- Features are complex objects that cannot be learned directly
- A given warning is abstracted as a set of attributes:

$$
\begin{array}{rcl}
[\mathit{length} = 14, \mathit{time} = 2.5, \mathit{srcline} = 10, \ldots] & \mapsto & \mathit{false} \\
[\mathit{length} = 6, \mathit{time} = 1.1, \mathit{srcline} = 38, \ldots] & \mapsto & \mathit{true} \\
[\mathit{length} = 18, \mathit{time} = 3.6, \mathit{srcline} = 26, \ldots] & \mapsto & \mathit{false} \\
\ldots & &
\end{array}
$$

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
                window.location.search; // SOURCE
2: var idx =
                search.indexOf("redirect=") +
                "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search = // swfobject.js
                window.location.search; // SOURCE
2: var idx =
                search.indexOf("redirect=") +
                "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK // client.html
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objcets (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace("mailto:" + url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|=1	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|=3	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time=2.1s	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conds|=0	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name= DOM-based XSS
source/sink line numbers	|steps|	severity
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Learning Features

lexical	*quantitative*	*security*
source/sink identifiers	|results|	rule name
source/sink line numbers	|steps|	severity=1
source/sink URLs	time	
external objects (`mailto`, `embed`, etc)	|path conditions|	

```
1: var search =
              window.location.search; // SOURCE
2: var idx =
              search.indexOf("redirect=") +
              "redirect=".length;
3: var url = search.substring(idx);
4: location.replace(url); // SINK
```

Classifier

- Aletheia generates a set of candidate filters by training different classification algorithms on the training set
- Each of the candidate filters is applied to the testing set, and the resulting classifications are reduced to a **score** based on the rate of true positives, false positives and false negatives
- The filter that achieves the highest score is applied to the remaining warnings
- The user is presented with the findings surviving the filter

Learning Algorithms

functional	*clustering (or instance based)*	*tree and rule based*	*bayesian*
neural network	Kstar	decision tree	naive bayes
SVM		OneR	bayesian network
logistic regression			

Learning Algorithms

functional	*clustering (or instance based)*	*tree and rule based*	*bayesian*
neural network	Kstar	decision tree	naive bayes
SVM		OneR	bayesian network
logistic regression			

-compute boundary in feature/derived space

- e.g. hyperplane
- geometric interpretation mistreats discrete features (like line no)

Learning Algorithms

functional	*clustering (or instance based)*	*tree and rule based*	*bayesian*
neural network	Kstar	decision tree	naive bayes
SVM			
logistic regression			

measure distance between incoming and labeled datapoints (again geometry)

Learning Algorithms

functional	*clustering (or instance based)*	*tree and rule based*	*bayesian*
neural network	Kstar	decision tree	naive bayes
SVM		OneR	bayesian network
logistic regression			

-divide-and-conquer approaches:

- decision trees: maximize 'information gain'
- rule-based methods: covering rules describing each class exclusively

Learning Algorithms

functional	*clustering (or instance based)*	*tree and rule based*	*bayesian*
neural network	Kstar	decision tree	naive bayes
SVM	O		bayesian network
logistic regression			

$$P(C=c|X=x) = \frac{P(X=x|C=c)P(C=c)}{P(X=x)}$$

(*C*: class ; *X*: attributes)

Learning Algorithms

functional	*clustering (or instance based)*	*tree and rule based*	*bayesian*
neural network	Kstar	decision tree	naive bayes
SVM			bayesian network
logistic regression			

so which one???

Policy

precision = tp / (tp + fp) [0 w 1] recall = tp / (tp + fn)

discretized as

w in {0,0.33,0.5,0.66,1}

such that

score = w x recall + (1-w) x precision

Evaluation

- Back to 3,758 classified warnings…
- Random sampling to simulate user classification
- Average score across 10 runs

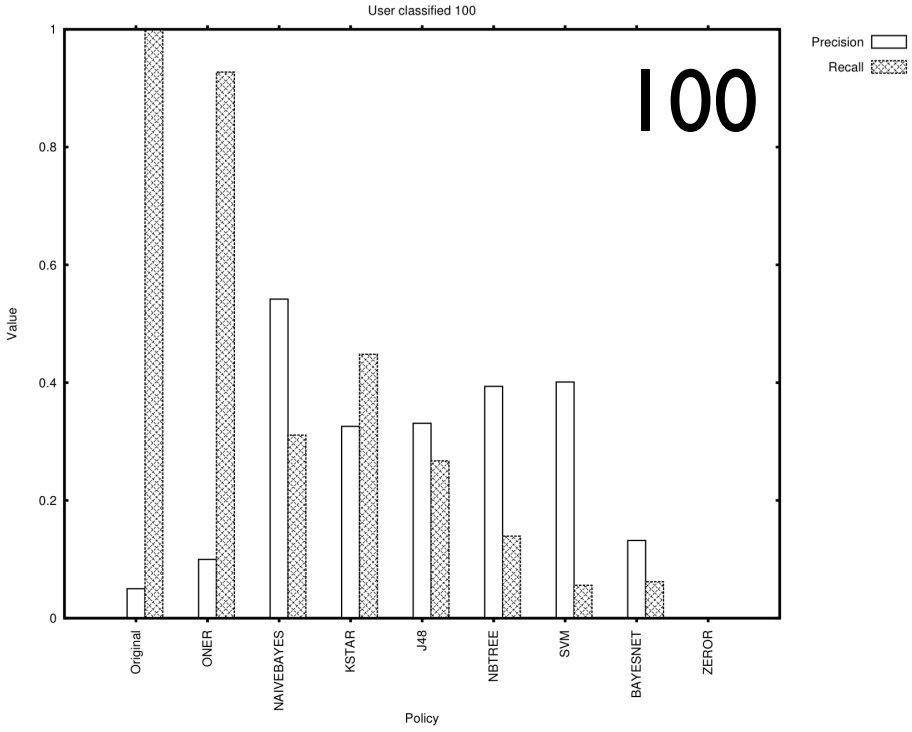
Conclusion 1: Feasibility

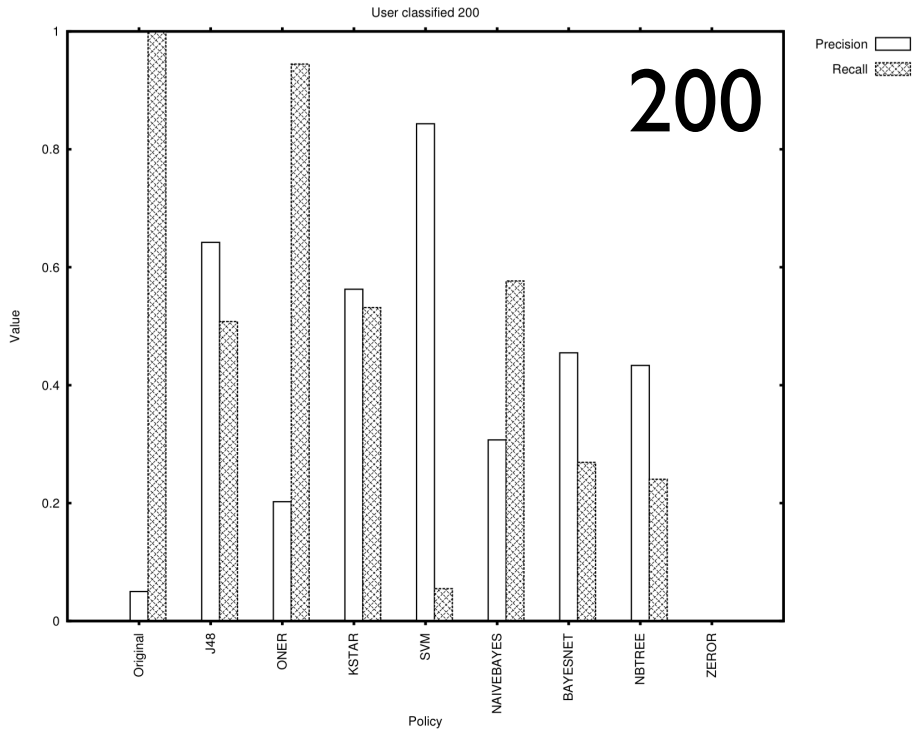
Based on tolerable* user effort,
it is possible to filter the
remaining warnings effectively*
w.r.t. the specified policy

We consider manual classification of up to 200 warnings as *tolerable* and a filter that achieves at least 95% accuracy as *effective*

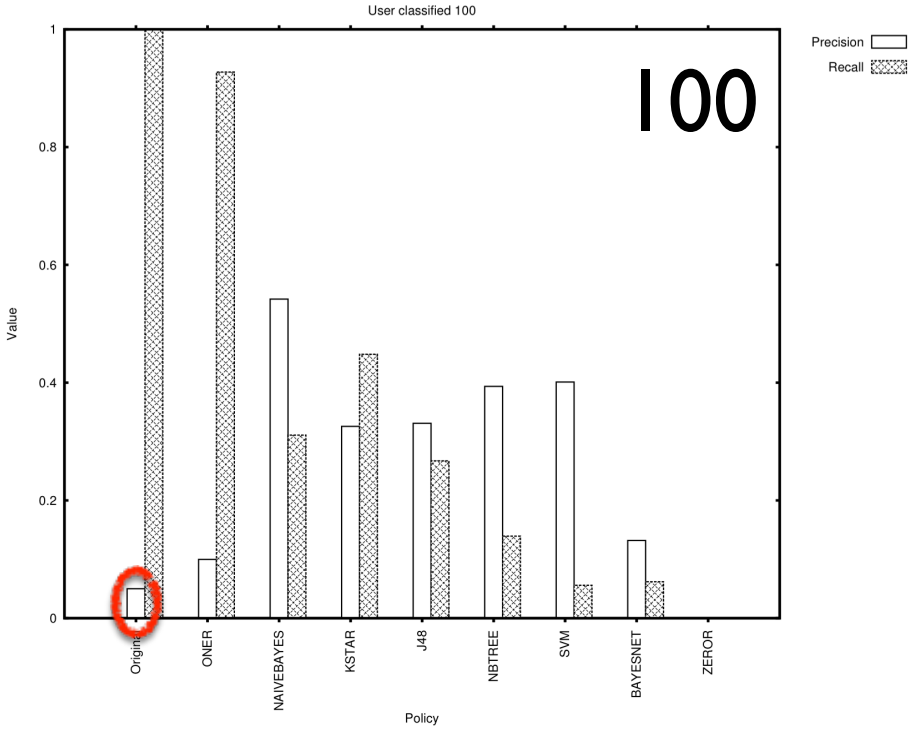
$$\text{accuracy} = \frac{tp+tn}{tp+tn+fp+fn}$$

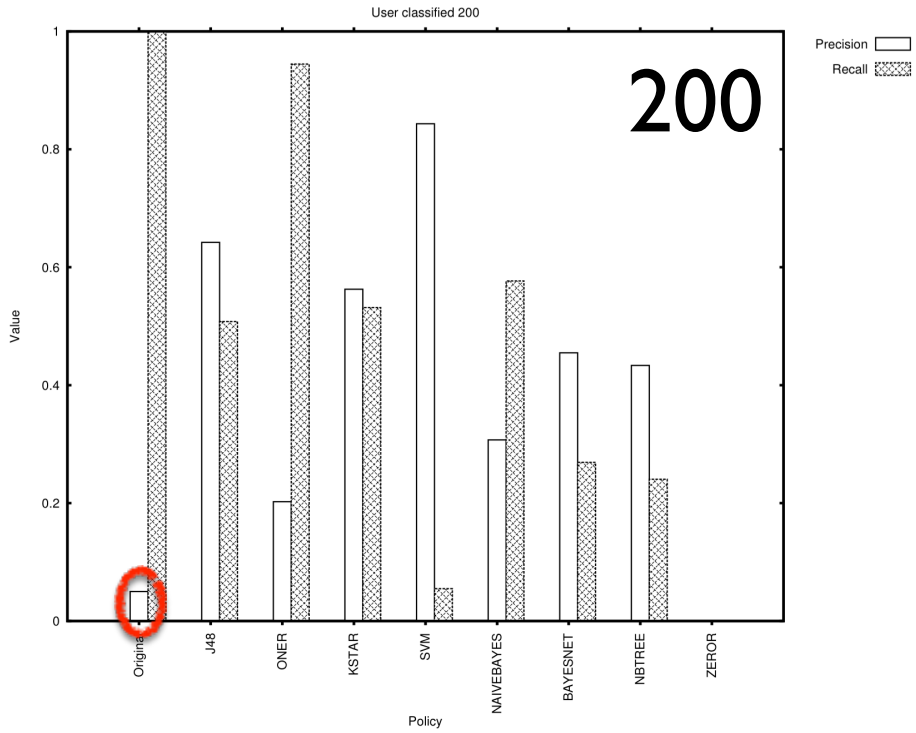
precision/recall by classifier



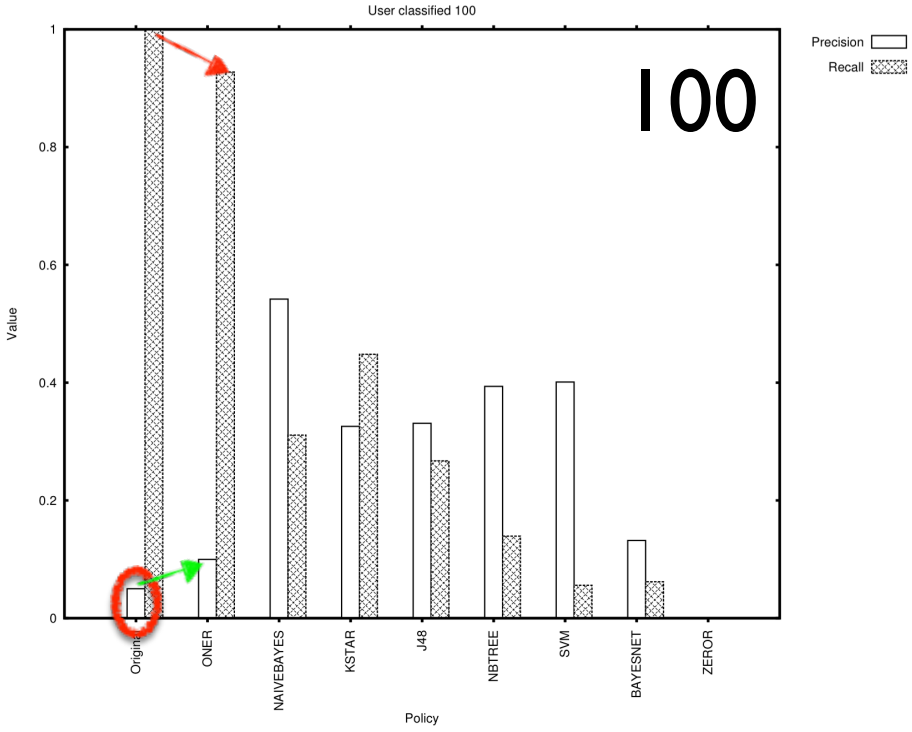


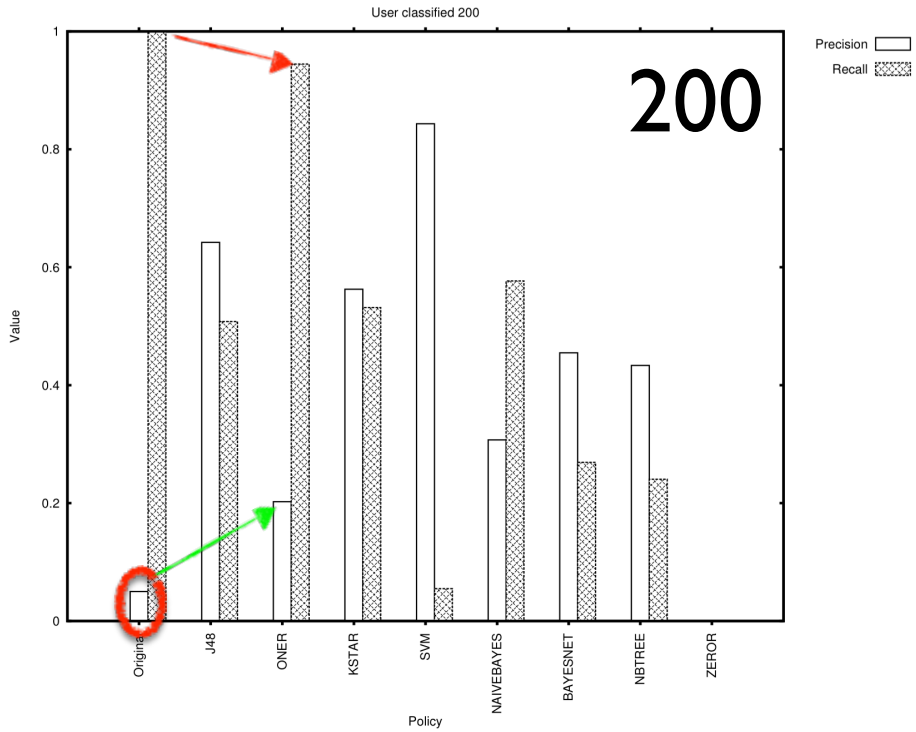
precision/recall by classifier





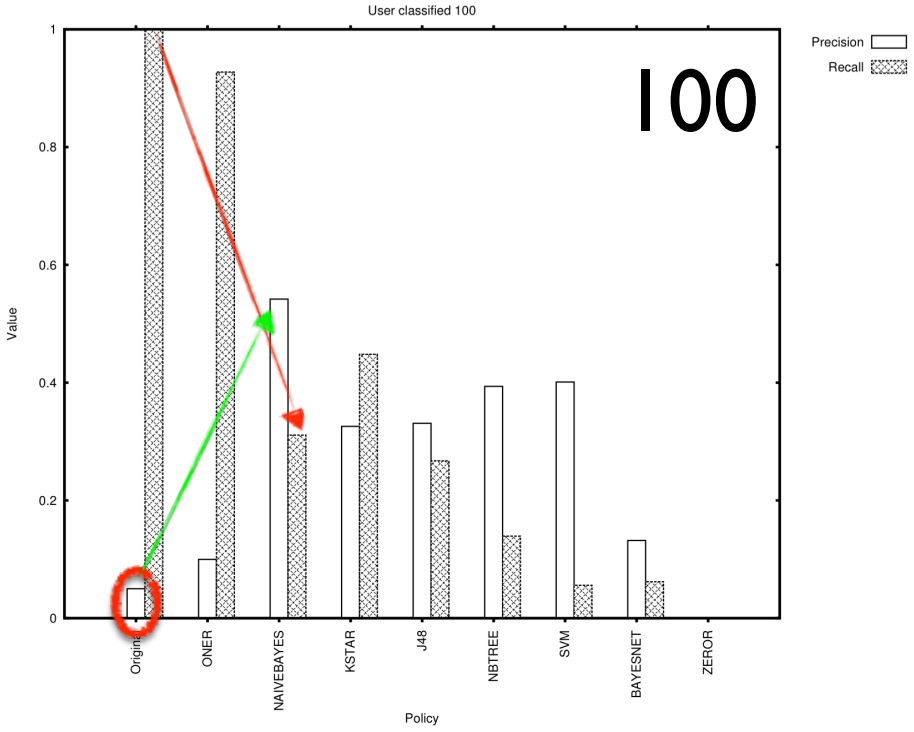
precision/recall by classifier

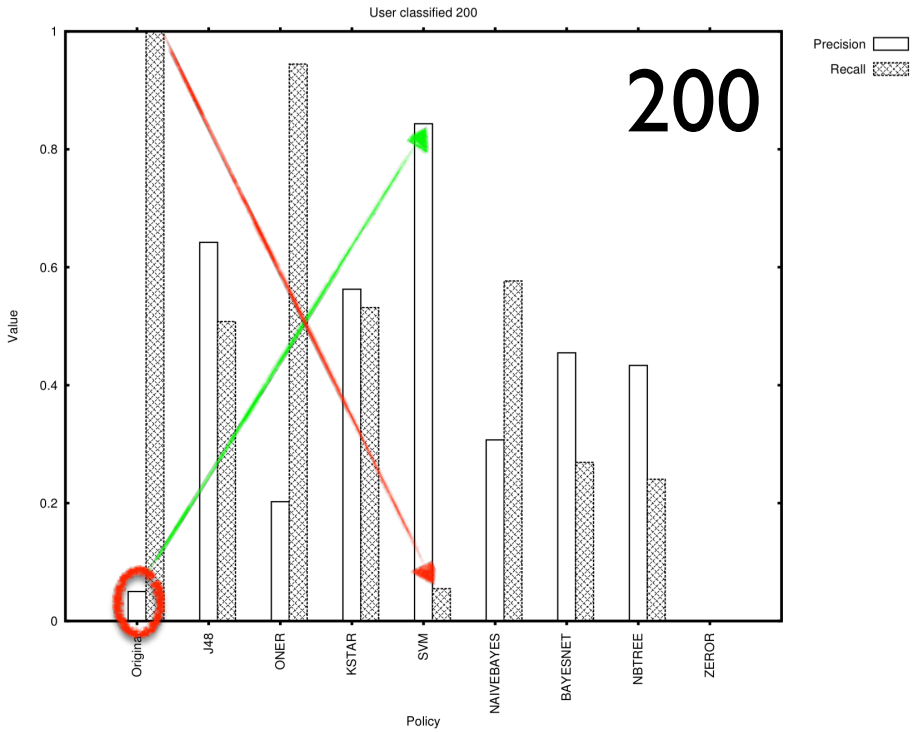




conservative: bias toward high recall

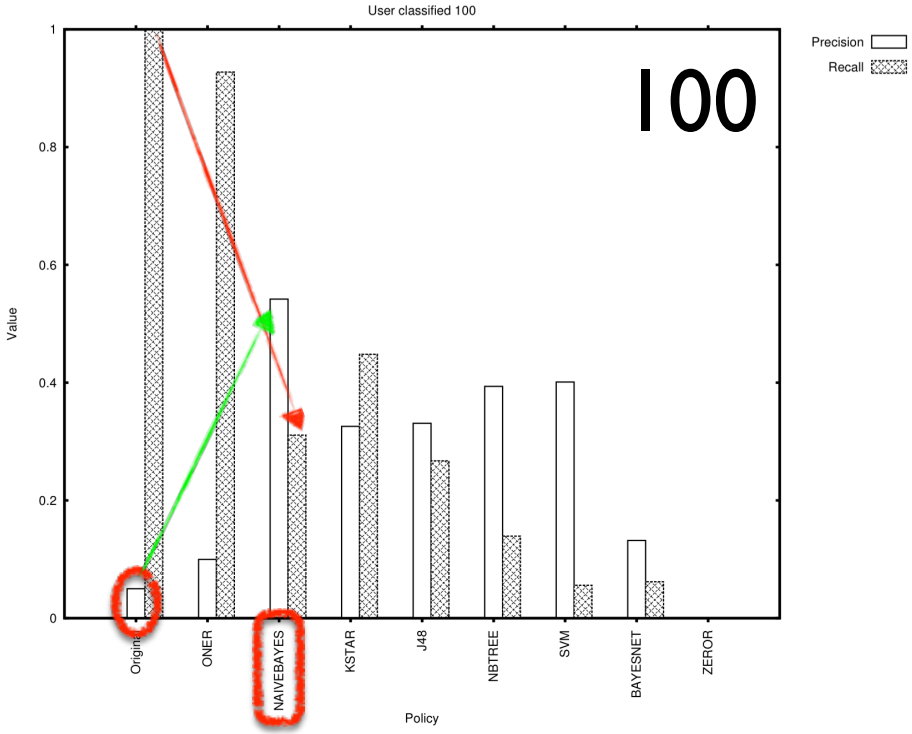
precision/recall by classifier

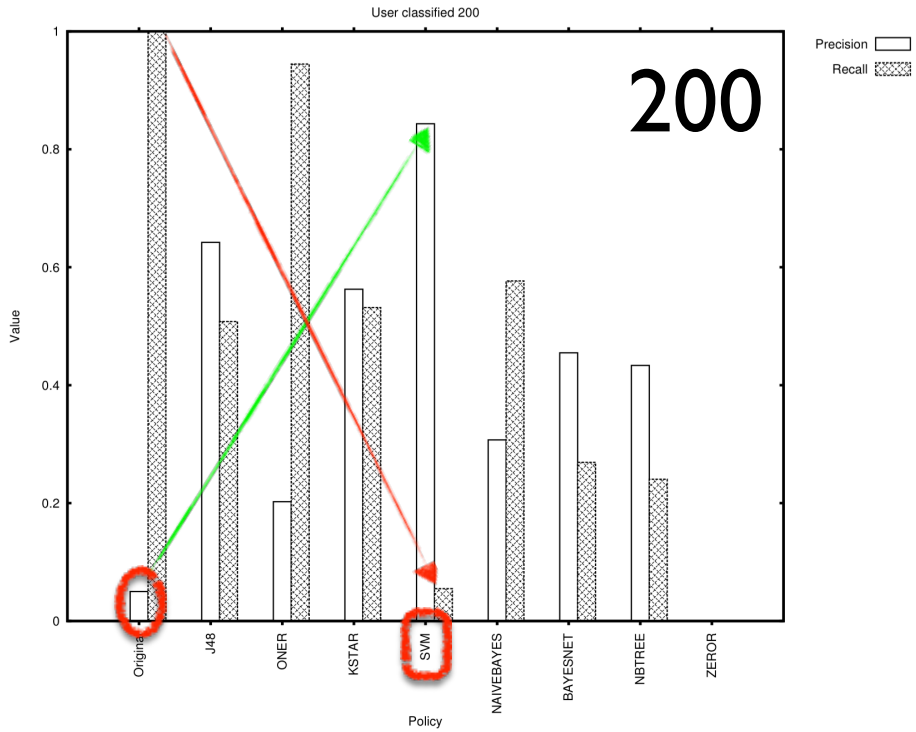




aggressive: I want the good stuff (high precision)!

precision/recall by classifier

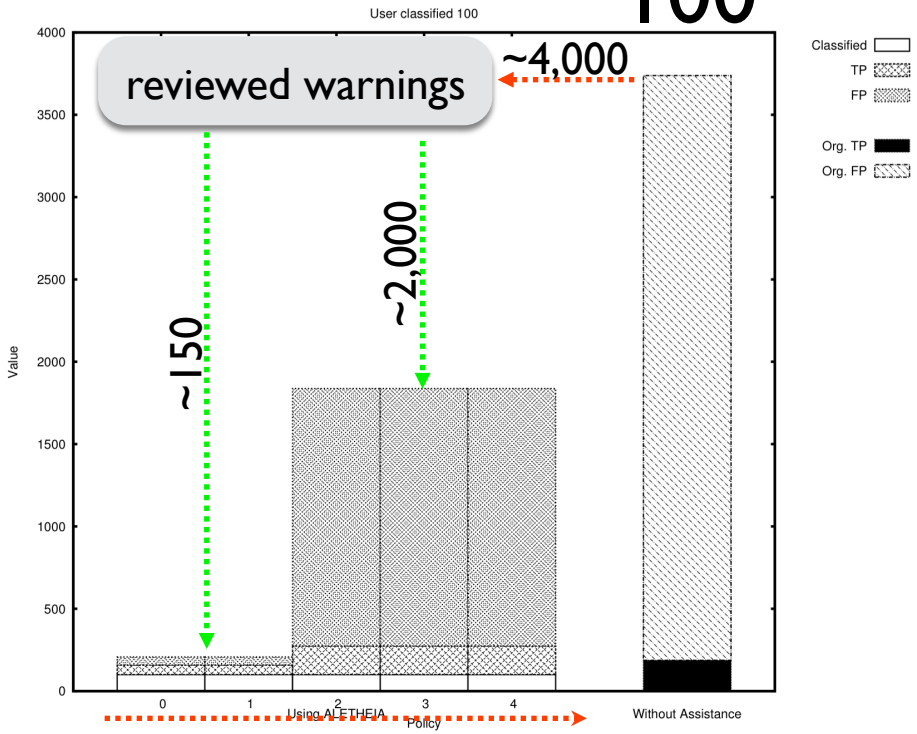




aggressive: I want the good stuff (high precision)!

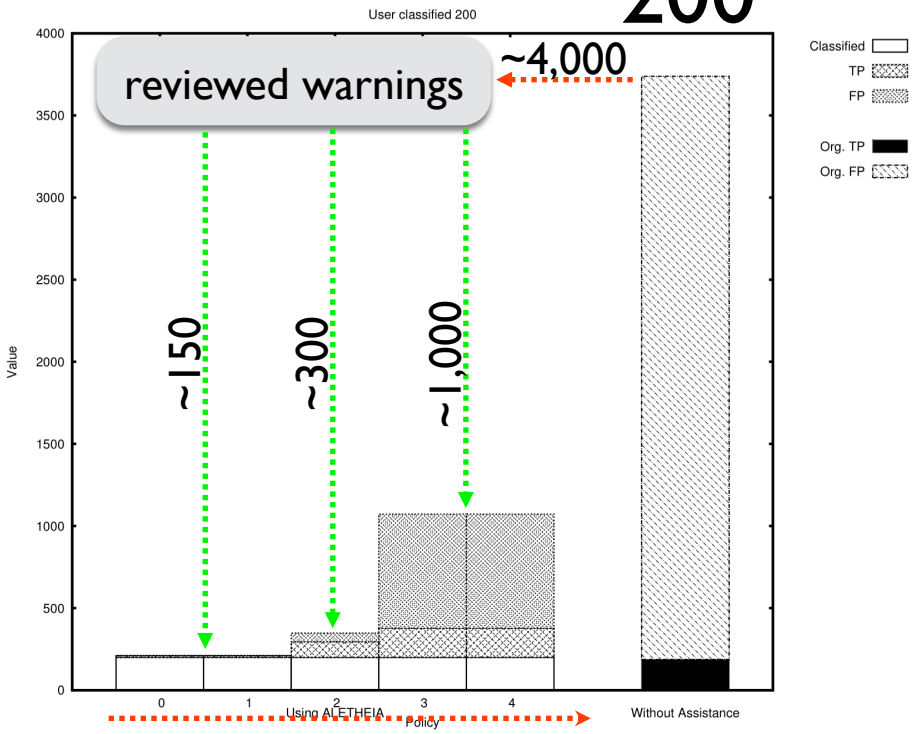
user effort by policy

100



precision recall

200



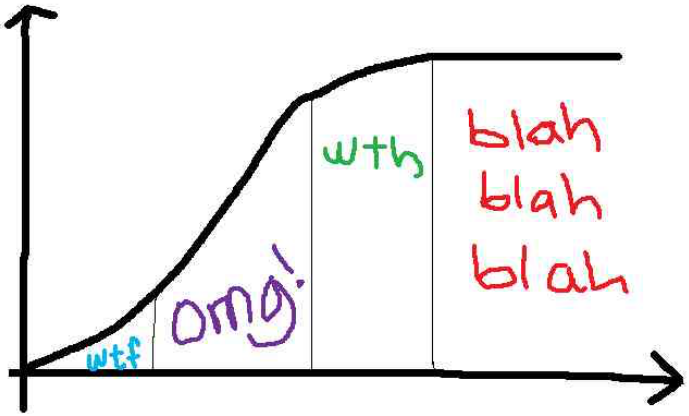
precision recall

Conclusion 2: Learning Framework

None of the classification algorithms in the Aletheia suite is either optimal or near optimal across all policies

Conclusion 3: Diminishing Returns

Improvement in filter quality, measured as policy score, diminishes with user effort



>= Present Work

Integrated in an IBM product for static security analysis: IBM Security AppScan Source

In the future:

- Experiment with more learning algorithms
- Integration of machine learning with static analysis algorithm

Thank You!

pistoia@us.ibm.com