Taint Analysis

Incremental Approach

Example

```
1: public class CurrencyRate {
2: DataStore dataStore = Bean.getService("CurrencyRateService");
3: protected void doGet(HttpServletRequest req,HttpServletResponse resp) throws IOException {
4: String source = req.getParameter("source");
5: String target = req.getParameter("target");
6: try {
7:      DataStoreRequest request = new DataStoreRequest();
8:      request.set("sourceCurrency", source); // BAD
9:      request.set("targetCurrency", URLEncoder.encode(target)); // GOOD
10:     request.set("date", new String(Date.getDate())); // GOOD
11:     DataStoreResponse response = dataStore.invoke(request);
12:     resp.set(response.get("rate")); // BAD
13: }
14: catch(Exception e) {
15:     e.printStackTrace();
16: }
17: }
18: }
```

Problem Specification

Security rule (Source,Sanitizer,Sink)

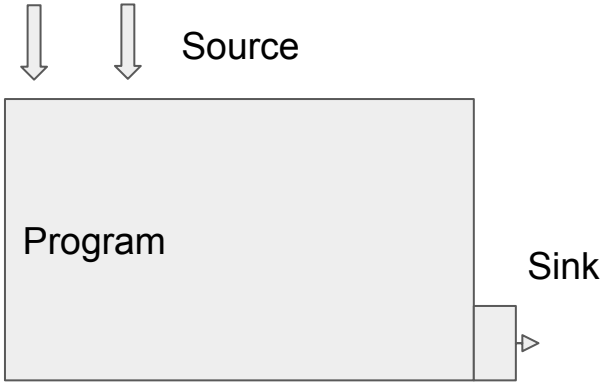
Source => [SourceCurrency,targetCurrency]

Sanitizer => [URLEncoder.encode()]

Sink => [rate]

Source -> Sink // Problem

Source -> Sanitizer -> Sink // 🆗



Slicing

```
1.  int i;
2.  int sum = 0;
3.  int product = 1;
4.  for(i = 1; i < N; ++i) {
5.    sum = sum + i;
6.    product = product * i;
7.  }
8.  write(sum);
9.  write(product);
```

Slice[8,sum]

```
1.int i;

2. int sum = 0;

4. for(i = 1; i < N; ++i) {

5. sum = sum + i;

7. }

8. write(sum);
```

Slicing as a Data Flow Problem

Relevant(j) = (Relevant(i) ∩ Referenced(j) not empty)? Relevant(i) + Defined(j): Relevant(i)

Program P

Slicing Criterion - (n,V)

Defined(n)

Referenced(n)

Relevant(n)

Slice computed by condition (1,i) => {1,5,6}

n	Statement	Defined	Referenced	Relevant
1	Int i = n	{i}	{n}	{i}
2	Int sum = 0	{sum}	{}	{i}
3	Int product = 0	{product}	{}	{i}
5	Sum = sum + i	{sum}	{sum,i}	{i,sum}
6	Product = product * i	{product}	{product,i}	{i,sum,product}

Implementation Details

1. WALA library to compute call graph and a SSA representation.

   sum = sum + i; => 6 = binaryop(add) 9 , 11

2. Maintain a work list of instructions & compute fix point by interpreting instruction by instruction forward from worklist & stop only when no new facts are learned.

3. For Incremental changes, use the computed table to build the next version. Slight modification to existing instruction, new instruction gets added or existing instruction gets deleted, add only their successors to the worklist. Compute the new relevance set for them & continue till there is no new facts available from the changes.

Thanks!

References:

1. Omer Tripp, Marco Pistoia, Stephen J. Fink, Manu Sridharan, and Omri Weisman. 2009. **TAJ: effective taint analysis of web applications.** DOI=http://dx.doi.org/10.1145/1543135.1542486
2. Omer Tripp, Marco Pistoia, Patrick Cousot, Radhia Cousot, and Salvatore Guarnieri. 2013. **ANDROMEDA: accurate and scalable security analysis of web applications.** DOI=http://dx.doi.org/10.1007/978-3-642-37057-1_15
3. Mark Weiser. 1981. **Program slicing.** (ICSE '81). DOI=http://dl.acm.org/citation.cfm?id=802557
4. Frank Tip.1994. **A Survey of Program Slicing Techniques.** DOI=http://www.franktip.org/pubs/jpl1995.pdf
5. Vida Ghodssi. 1983. **Incremental Analysis of Programs.** Ph.D. Dissertation.
6. Ryder, B. G., T. J. Marlowe, and M. C. Paull. **Conditions for incremental iteration: Examples and counterexamples** DOI=http://dx.doi.org/10.1016/0167-6423(88)90061-5