TaintDroid: An Information-Flow Tracking System for Real-time Privacy Monitoring on Smart-phones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Description  (Mayank Chaudhari) | Comment 1  (peer 1 name) | Comment 2  (peer 2 name) | Comment 3  (peer 3 name) |
| Malware Analysis Technique | dyanmic |  |  |  |
| Dataset description |  |  |  |  |
| Feature selection Technique | based on the standard API notations |  |  |  |
| Feature Extraction Technique |  |  |  |  |
| Classification Technique |  |  |  |  |
| Accuracy or confusion matrix scores |  |  |  |  |
| Scalability and Efficiency | Taint Droid is scalable in nature and adds an overhead of only 14% over the normal execution. |  |  |  |
| Overview | It leverages the Androids virtualized architecture to integrate four granularities of taint propagation : variable-level, method-level, message-level, and file-level. Taint droid identifies the flow of sensitive information and marks it as a tainted. Later the labeled data is tracked to analyze its impact over other data in order to study the leaks and finally the tainted data is identified before it leaves the system at a taint sink. This is implemented as a multilevel taint tracking system. The variable level tracking is performed at virtual machine level, message level tracking is performed between applications, method level tracking at the level of native libraries and finally file level tracking at network or storage level. The taint tags are stored alongside the variables in memory by doubling the memory size of actual variable.  Taint tracking is performed in following ways:  1) The taint propagation takes place by using the defined set of rules called DEX Taint propagation logic inside the Dalvik VM.  2) Native code is unmonitored in TaintDroid.  3) For internal VM methods, they manually inspected and patched them fro taint propagation as needed.  4) For JNI methods it uses as method profile table for tracking. This is a conservative heuristic and operate only on primitive and string arguments and return values. Thus having false positives for objects.  5) IPC taint propagation is performed at message level rather than method level as unpacking can be dealt differently at method level. This also leads to false positives.  6) Secondary level taint tracking is performed using a single file pointer. This also leads to false positives.  Limitations:  ============  1) It can track only data flows not control flows.  2) It can not track taint tags on Direct Buffer objects.  3) significant false positives can be observed when tracked information contains configuration identifiers |  |  |  |
|  |  |  |  |  |