Automatic Hot Patch Generation for Android Kernels

Zhengzi Xu, Yulong Zhang, Longri Zheng,
Liangzhao Xia, Chenfu Bao, Zhi Wang, Yang Liu

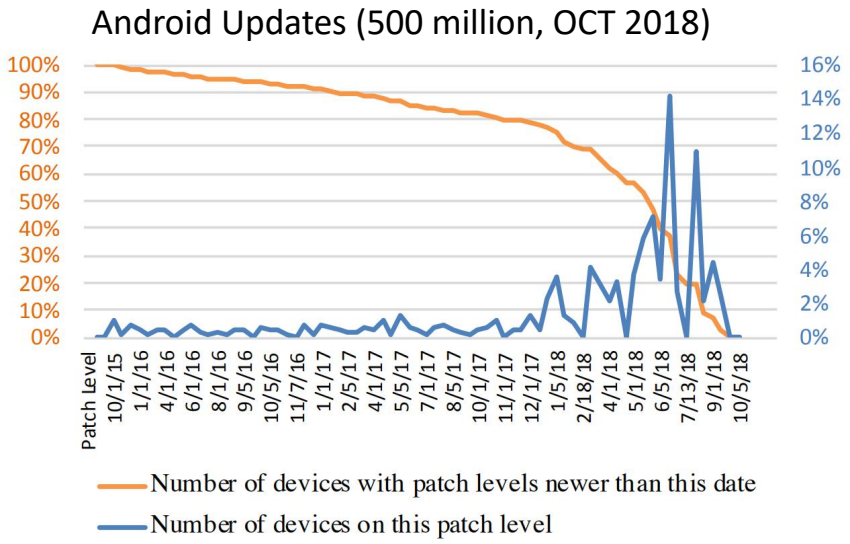
Motivation:

Android system has low upgrade rate (legacy systems).
Known vulnerabilities remain unfixed.
It is time consuming to develop patches manually.

Hot patch is an ideal solution for legacy systems.

Android Version Distribution (OCT 2018)

Android Major Version	Release Date	Percentage
Android 4.x	Oct-11	6.65%
Android 5.x	Nov-14	18.11%
Android 6.x	Oct-15	19.96%
Android 7.x	Aug-16	25.47%
Android 8.x	Aug-17	29.60%
Android 9.x	Aug-18	0.04%
Others	-	0.17%



Goal:
Input Fiter Generation

Given a vulnerable function F and its official patch P at location L, we would like to find a suitable location L_0 of F in binary form to insert an automatically generated hot patch P_0, which has the same semantics as P.

Why use filter rather than direct patch?

- In binary, it is easier to locate function beginning than other places.
- In different legacy systems, function beginning remains the same.

To ensure that the filter is safe, we limit the filter to only read the memory content without write operation.

Vulnerability Patch Survey:

Patch type classification on 375 CVEs of Android Kernel between year 2012 -2016.

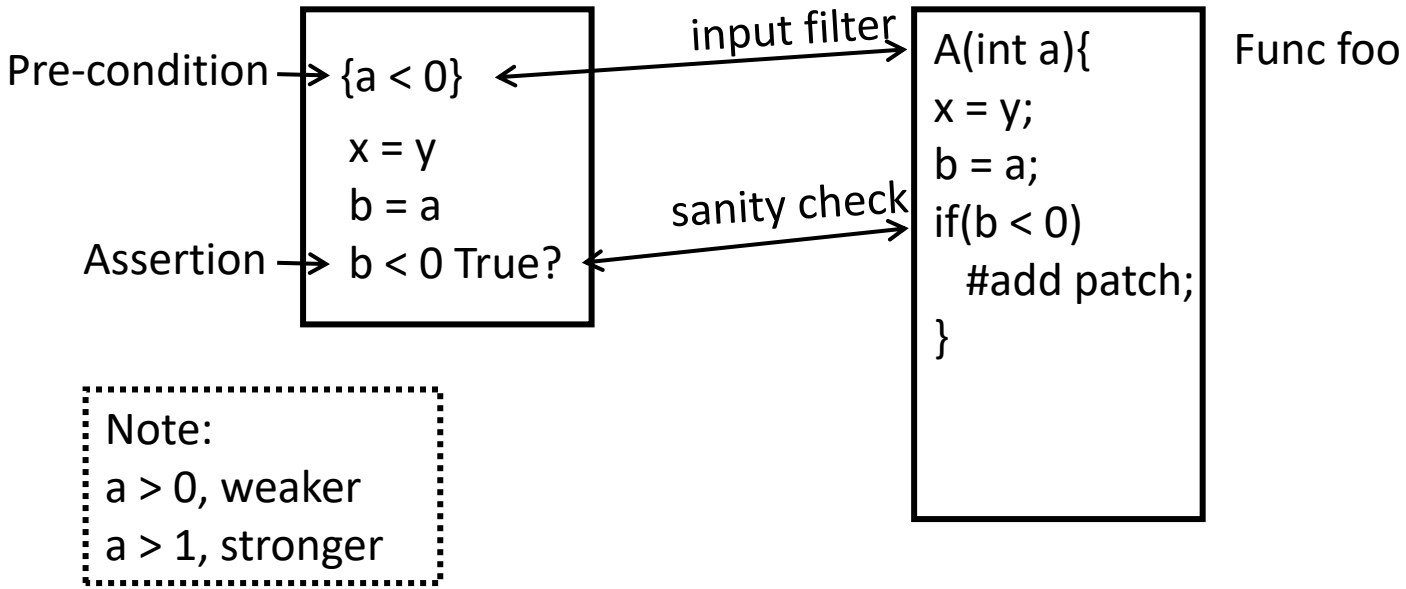
Patch Type	NO.	Percent
Sanity Testing	157	42.10%
Function Calling	65	17.40%
Change of Variable Values	37	9.90%
Change of Data Types	9	2.40%
Redesign	65	17.40%
Others	40	10.70%

Insight: The vulnerability type and the patch type are different.

Approach:
Weakest Precondition Reasoning

- To find what must be TRUE in order for a post assertion to be TRUE.

Predicate Transformers:
Transform a post-condition into a weakest pre-condition



Note:
a > 0, weaker
a > 1, stronger

Approach:
Function Calls Handling

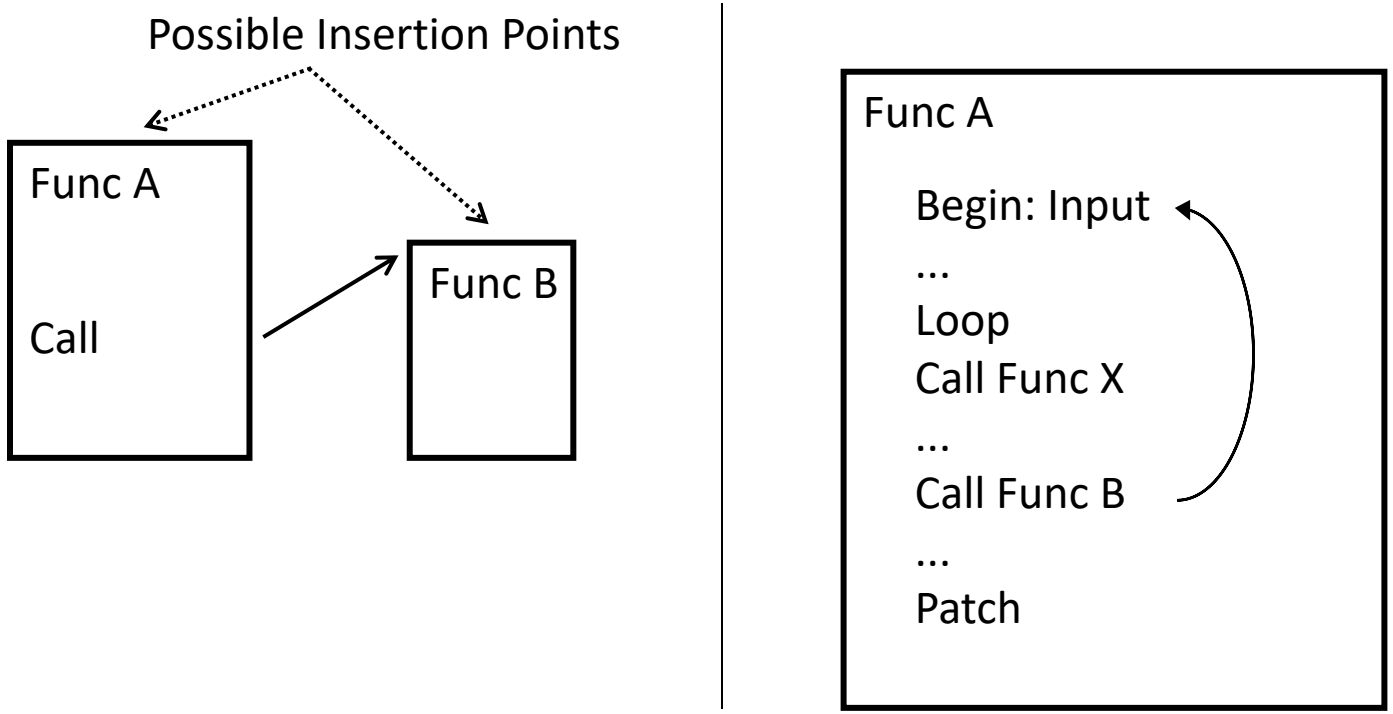
- Importing the inlined function
- Function summarization
  - Variable value propagation
- Skip nested function calls
  - Kernel has function calls, which are too deep to be analyzed
- Function hookings

Approach:
Loops Handling

- Loop can be classified to different types:
  - Loop which includes the patch.
    - re-build the loop logic in the filter
  - Loop which updates the target varibales.
    - loop summarizaion
  - Irrelevant loop. (safe, skip)
  - Complexed loop. (unsafe, skip)
    - nested loops
    - loops with function calls

In program analysis, loops and function calls are the two problems that need to be addressed.

Approach:
Insertion Point Optimization



Evaluation:

- Accuracy:
  - Manually verified: 55 (out of 59) correct generated patches.
- Performance:
  - Less than 0.1% in overhead.
- Robustness:
  - 21 cases tested. All of them pass the testing.
- Other:
  - 54 (out of 55) generated patches have similar logic with human written patches.

Future Improvements:

- Source Code Level Adaptive Patch Generation.
  - More flexiable and easiler to be implemented
- Enable the Write Operation in Binary.
  - Support more type of patch generation
- Patch Semantic Summarization.
  - There lacks researches on patch semantics
- Large and Complexed Patches.

Q&A

Thank you for listening.