Dynamic taint analysis

Summary

All you ever wanted to know about dynamic taint analysis and forward symbolic execution is a systemization of knowledge paper about symbolic execution and taint analysis. It provides an overview of what it means to do dynamic taint analysis and symbolic execution on binary code and provides semantics for both.

Unleashing MAYHEM on binary code details a system that automatically identifies vulnerabilities and creates exploits in binary programs. They describe a system that alternates between concrete and symbolic execution, describing it as hybrid execution. They provide a new model of symbolic memory. When describing their automatic exploit generation they refer to prior work done on AEG from C programs.

Questions

Symbolic Execution

Mike: Is this much of a contribution? Why not?

James: No?

Mike: No? The paper defines this standard language with operational semantics. The formalisms you choose matter. They make particular choices in their semantics. Why use "goto" in these semantics?

James: It models assembly, what precise machines will do

Mike: Yeah, and we can model what the program does with stacks and function calls

James: Not compositional, have to encode all of this by hand

Mike: This is true with analyses encoded in the lambda calculus

Aseem: They call this an intermediate language, it's not supposed to be a source language

Mike: Does getting this down to assembly reveal anything? Does it tell you anything new? But what if you wanted to do SE on binaries? Take the binaries and conver them into something like SSA?

Matt: Is it important that you go through memory to get values?

Mike: Maybe? If we put statements in e1 and e2 in the FCond rule, what changes?

Matt: Can sigma be mutated?

James: Don't think so

Mike: You can have computed gotos though

Matt: If labels are first class values, then you could jump around in the program

Mike: Expressions being tainted captures the idea of a program counter hijacking exploit. How would that look at a higher level of abstraction?

Matt: How would this look in TAL?

Mike: Exactly the same. Anyway, that would be one reason why gotos are good. They have a heap with store and load, and variables. Why have variables? Why not stick everything in the heap? Variables are registers? A fixed number of variables according to delta, with a heap. You can only index into the delta with fixed names (like registers) but heap indexes are via computed values. And they have assertions. Why?

Aseem: Symbolic execution to verify some property, encode them in asserts

Matt: Mark infeasible paths?

Mike: No builtin notation for arrays, assertions can communicate expectations about security parameters

Andrew: Or any time the PC contains something that is tainted

Mike: Then the semantics for tainting. Could be more elegant. Extended previous rules to include tau-mu and tau-delta that are the taintedness of those regions, and they produce values with taints. All values are passed around with their taint. How do you introduce taint? How do you propagate taint? There is a policy that introduces taint at the conclusions of the rules with Pbinop, and as checks in the premises. Policy tells you whether or not it is okay to proceed with the evaluation. Taint is introduced as part of assignment. So, did anyone look at these long enough to judge the design? Is this the best way to express tainting as a policy? The dynamic monitor semantics from the Sabelfeld work is cool with normal semantics, but they have big-step semantics so you can't use that trick. Another choice they made was that they made policies for every different step. There is also no PC taint. Only values or addresses are tainted.

Matt: These are like transfer functions in a dynamic analysis

Mike: That the rules are in the same box is confusing, it implies they have something to do with each other but they don't beyond being unconditionally true

James: Done to save space?

Mike: They never describe the way to think about the policies.

Matt: Some are mapping taint to taint, some are mapping to bools.

Mike: The taint is also a logic of bools. They also don't give a high level intuition of what the "typical tainted jump target policy" means. The "mem" case ignores tainted addresses, if the value you're storing is tainted then the taintedness of the result is the same taint.

Mike: Then they talk about a bunch of practical concerns with the tainting analysis. The system also won't track taint through implicit flows, just explicit assignment. They also never query the SMT solver in any of the rules, directly. Where do they use it?

Mike: You also have a bunch of different strategies which paths to explore in the symbolic execution. It would be cool if those policies were reflected in the rules. They talk about symbolic jumps also. The rule expects that the expression steps to a value. There is no provided rule for what a rule would be where the expression does not step to a value. What to do?

MAYHEM

Mike: What is the MAYHEM paper about?

Kris: It's about doing symbolic execution on binaries to find exploits that generate a shell?

Mike: How does it work? What machinery do they use?

Kris: You take in some stuff and iterate the PC, and they call into something to do something.

Mike: What is this symbolic view of memory? They have these different memory objects, memory as this giant collection of formula.

Criticisms

Symbolic Execution

Aseem: They forget the t2, they drop it in their conclusion in their semantics

Matt: I don't like the use of commas to delimit the relations in their semantics

Mike: That's because the v1 label is just the PC label. Rules on the bottom just have instructions, rules on the top have variables/values.

Matt: Appel paper that says that SSA is a functional program. After reading that paper, I never want to design a semantics like this again. I don't know what these kinds of semantics buys you when you could have something higher level.

Mayhem

This paper is hard to understand.