Dynamic Analysis

Feb 27. 2014 (Chang Liu)

Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software

1. Summary

This paper discusses about how to binary emulator and instrumental tools to identify software vulnerabilities “very efficiently”. To mitigate such vulnerabilities, the authors proposed taint analysis, which consists of three part: TaintSeed to locate the taint data, TaintChecker to track the propagation of taint data, and TaintChecker to allow binary rewriting at runtime.

2. Questions

This paper talks about taint analysis. The discussion proceeded in the form of a serials of QA (on this paper, Mike Q most of the time), discussing each component: TaintSeed. TaintChecker. TaintAssert.

(I think I am confused about Nate and James. So some James in the following are actually Nate, but I cannot recall which ones…)

Q: How to identify Taint Data
A: Using TaintSeed Policy

Q: Why use all 4 bytes?
(Still don’t understand, but seems to classify taint instructions into categories)

Q: How to propagate Taint data?
James: Copy-based propagation

Q: Why not tracking implicit flow
Aseem: prevent false positive

Q: What are bad use of taint data?
Aseem: Jump, Function Pointers, Format String, etc.

Q: Why these?
Matt: control flow

Q: Why format string?
Andrew: implicit flow

Q: What is an alternative policy?
Aseem: system call

Q: What are the bad thing of system call?
A: Code injection

Q: why not consider it as an attack at the first place?
A: will check on the user's input data to make sure it is ok.

Q: What are the big sources of false positive and false negative?
(Andrew made many comments, but I sat behind him, and could hardly hear clearly what he said)

James: make the checker more aware of it?
Mike: that's another problem. This problem is about sometimes taint data can be used, and so it is a false negative.

Andrew commented on the slow performance

Q: why nullgrind in exp goes down and up and down again?
A: no explanation in the paper

Q: why it is so slow (40x) but still important?
Andrew: employed in real application like email server. Email throughput is not large, so the slow speed is tolerable..
James: Verify only sampled code can help

Q: what is the relation of this paper and other injection papers?
A: copy-based
James: control flow

Matt: is this the state-of-the-art
James: some other emulators are used in the following work.

3. Criticism

Andrew criticize that is too slow.

4. New ideas

Supporting control flow might be an interesting direction.

Dynamic vs. Static Flow-sensitive security analysis

1. Summary

While static analysis tools such as type systems can provide soundness guarantee of the information flow of a program, they are often too restrictive. This paper seeks whether dynamic analysis tools can help improve the status, e.g. if assigning a high security variable to a low security variable happens in an untaken path, then we treat it as not violating the security policy (permissive) The authors formalize four intuitive properties a desired dynamic monitor should satisfy, and then show the impossibility result: monitor satisfying all of the four property does not exist.

2. Questions

(I fall asleep from here for about 20 mins, since I've been stayed up for almost 24 hours…)
(When I wake up, Mike was discussing the flow sensitive type system, then proceeded to the impossibility result. When starting to discuss the monitor, the time was over. I think I only capture one question as follows.)

Q: can the output channel be high?
Mike: yes. the output rule allows outputting secret output or in high security context to high output channel.

(Andrew made many comments afterwards, but again, I can hardly hear what he was saying…)

3. Criticism

Kris: not interesting
Mike: but still have 600 citations.

(I'd like to make some my own comments here. Maybe it is better to move to scriptroute:

Let's re-consider why the counter example doesn't work in dynamic system: the key statement is

if(h) then l=1; else skip;

the intuitive explanation is that on h=false, skip is checked, without knowing that l=1 makes l a high variable. Therefore, permissive allows type checker to ignore those *untaken path*, and thus

allows those untaken path to contain *unsound code*. In this sense, permissiveness intuitively contradicts to soundness.

Let's take another look at the this program: what if we change that to SSA? The SSA program looks like the following one:

```
if(h) then l1=1; else l2=l;
l = phi(l1, l2);
```

Bingo, the type checker can infer that both l1 and l2 are high, so l is high as well. Let's write it without phi:

```
if(h) then l=1; else { l=l; skip; }
```

Right, the key point is that l=l (which is semantically a dummy statement) is missing in the false branch, which will allows type checker to label l as high. In fact, we can arbitrarily add such dummy statements to make arbitrary variables to be labeled high. However, the problem is that *without knowing the untaken path, permissiveness is agnostic to when such insertions are reasonable.*

Therefore, permissiveness seems unreasonable to motivate dynamic type checker. Then what is a better motivation? I think the answer should be efficiency. The rationale behind is that by ignoring the untaken path, the type checker do not need to reason about the effect of the code in those path so as to reduce the reasoning cost. But why does this make sense? A reason might be, the adversary wants to deterministically perform their attack rather than trying their luck. Therefore, a majority of executions should reveal information to adversaries, which can be trapped by such an unsound dynamic system.

We can see Dawn Song's work also follows such a philosophy: they execute the code, to check which data are taint. Those code not executed which may even pollute the data will not be treated as taint. However, that work has a low efficiency, but I guess the emulator's efficiency has a big impact on this.

Notice that efficiency also motivates of dynamic symbolic execution over its static counterpart, which do not reason about untaken path to alleviate the path explosion problem. So I am wondering that all such dynamic versions can improve the efficiency at the cost of accuracy (soundness).

)

4. New ideas