'007' code helps stop Spectre exploits before they exist

Singaporeans boffins offer Spectre-protector as Fortinet ponders Android inoculation

By Richard Chirgwin 17 Jul 2018 at 05:34

Black hats haven't yet found a way to mass-exploit the Spectre vulnerability – but mitigations are already arriving.

Beyond chip vendor and operating system patches, there remain reasons to seek out additional defences: there are still circumstances in which protective coverage is incomplete – and over in the world of Android phones, updates dribble out slowly.

Be of good heart, sysadmins. At arXiv, Singaporean and US researchers have published work, appropriately dubbed "007", which checks code to see if it's trying to exploit Spectre; and at Virus Bulletin, Fortinet's Axelle Apvrille takes a look at the bug from an Android point of view.

Apvrille's work backs up what we've heard from other researchers: so far, Spectre exploitation is theoretical, with no exploits in the wild. She wrote that while there was a flurry of “Spectre exploit” stories based on AV-Test sample collection, it turned out that all of the reported samples were proofs-of-concept rather than genuine malware.

She adds: “there is a significant difference between a PoC of Spectre and a piece of
malware using Spectre. Turning a PoC into a malicious executable is far from a trivial process.”

That doesn't make this kind of work pointless, though, since it's a good thing to stay ahead of whatever nasties black hats might devise.

In developing a detection technique, Apvrille's second conclusion was also good news: an attack against Spectre, she found, seems relatively easy to detect.

She wrote that “we had expected several false positives with this signature, but that was not the case: this imperfect signature turns out to be quite good in practice.”

The signature Apvrille searched for (using the in-practice impractically-slow technique of searching whole binaries) was to identify “Flush+Reload cache attacks in ELF x86-64 executables”.

Although slow, that technique detected all of the viable samples in the proof-of-concept code gathered by AV-Test. Those that weren't successfully scanned, it turned out, wouldn't have worked anyway: “they were all damaged: the cache flush instruction was missing”.

And there's yet more good news for Android users: all of the proof-of-concept samples so far identified are for x86-64 architectures, and code doesn't easily port from there to ARMv7 architectures.

**Double-oh Seven**

The paper that landed at arXiv also seeks to detect code that attacks Spectre, at a generic level the authors describe as a “binary analysis framework to check and fix code snippets against potential vulnerability to Spectre attacks”.

Such things already exist, but the authors – Guanhua Wang, Tulika Mitra, and Abhik Roychoudhury from the National University of Singapore; Sudipta Chattopadhyay from the Singapore University of Technology and Design; and Ivan Gotovchits of Carnegie-Mellon University – explain that they impose heavy overheads, while their 007 framework imposed less than two per cent overhead, as measured by GNU Core Utilities.
They also claim to have detected “fourteen out of the fifteen Spectre vulnerable code patterns proposed by Paul Kocher, a feat that could not be achieved by the Spectre mitigation in C/C++ compiler proposed by Microsoft” (Kocher was one of the discoverers of Spectre, and he wrote this critique of Microsoft's C/C++ compiler fixes).

In the abstract, the 007 crew says their approach includes: “control flow extraction, taint analysis and address analysis to detect tainted conditional branches and their ability to impact memory accesses. Fixing is achieved by selectively inserting a small number of fences, instead of inserting fences after every conditional branch”.

The detection algorithm proposed in 007 is shown below (from the paper).

```
Input: P: Program binary
Output: \Phi: A set of triplets of the form (CB, IM1, IM2) capturing Spectre vulnerability
1: \Phi \gets \emptyset;
2: TS.policy \gets VtoV \triangleright Taint policy set value-to-value
3: step \gets None \triangleright Initialize Spectre detection stage
4: Let inst be the first instruction of P
5: while inst, ex it do
6:     GS \gets Interpreter.exe(inst) \triangleright GS: Global State
7:     TaintEngine.taint(inst, GS) \triangleright propagate taints
8:     if \tau\text{ (inst)} then \triangleright oo7 is invoked only for tainted instruction
9:         DS \gets oo7.check(inst) \triangleright DS: Detector State
10:    end if
11: inst \gets P.next() \triangleright fetch next instruction
12: end while
13: procedure oo7.check(inst)
14:     step \gets DS.step() \triangleright Checks the stage of detection
15:     if br(inst) then \triangleright check for CB
16:         DS \gets DS.setCB(inst) \triangleright recognize that inst might capture CB
17:     end if
18:     step \gets STEP_CB \triangleright progress the detection stage to CB
19:     TS.policy \gets PtoV \triangleright enable pointer-to-value taint
20: end if
21: if (load(inst) \land step = STEP_CB) then
22:     cb \gets DS.CB() \triangleright get CB from detection stage
23:     if (Dep(cb, inst) \land \Delta(cb, inst) \leq SEW) then \triangleright check
24:         DS \gets DS.setIM1(inst) \triangleright recognize that inst misses CB
25:     end if
26: end if
```
if (mem(inst) ∧ step = STEP.IM1) then
    DS ← DS.setCB(inst) ▷ get CB from detection state
    if (Dep(cb, inst) ∧ Δ(cb, inst) ≤ SEW) then ▷ check
        DS ← DS.setIM2(inst) ▷ recognize that inst
        Φ U = (DS.CB(), DS.IM1(), DS.IM2()) ▷ catch
        step ← None ▷ reset checker
        TS.policy + VtoV ▷ disable pointer-to-value
    end if
end if

if (step = STEP.CB ∧ Δ (DS.CB(), inst) > SEW) then ▷ Outside
    step ← None ▷ Reset detection beyond speculation window
    TS.policy + VtoV
end if

if (step = STEP.IM1 ∧ Δ (DS.CB(), inst) > SEW) then ▷ Outside
    step ← None ▷ Reset detection beyond speculation window
    TS.policy + VtoV
end if

return DS

The researchers note that their detection code is available on request from the National University of Singapore's Website.

Sponsored: Minds Mastering Machines - Call for papers now open