Stop the bleeding from the heart

prism research group @ cse

Buffer overflow

- Overwriting memory contents beyond legal limits
  - Stack memory → hijack the running program by replacing function return address
  - Heap memory → copying information that should not be revealed
- One of the earliest (and newest) software security vulnerabilities
  - Morris worm, 1988
  - Heartbleed, 2014
- Essentially caused by a common programming mistake

The heartbleed bug

- The most “glamorous” security bug in recent history
- A serious security flaw in OpenSSL, the most widely used library for secure network communication such as “https”
- Hatched in Dec. 2011, discovered on April 1, 2014
- Around 500,000 websites affected including Google, Yahoo, Amazon, and our own CSE
- Over two dozen CISCO and Juniper routers also affected, tens of thousands of units to be patched

Enter the beast

tls1_process_heartbeat

```
@2514  memcpy(bp, pl, payload)
```

target memory

source memory

how many to copy

- Copying heartbeat echo message to outgoing buffer
- “pl” and “payload” controlled by attacker ,“bp” is a network buffer
- I can put 4 bytes in “pl” and claim “payload” to be 64kb

Finding the needle

- openssl : close to 1/2 million lines, 92 directories, over 3500 functions

- memcpy(target, source, length) : 647 places in over 100 files

- Inspection effort → heartbleed itself spans 7 files, over 13,000 lines in 44 functions @ 300 lines per function

- “Given enough eye balls, all bugs are shallow”

Software AI to replace eyeballs

- Objective: present a handful of reports through static program analysis
- For heartbleed, AI compiler should reason as follows
  - Culprit, memcpy, 647 usages, need to consider other constraints
  - Additional constraints
    - length comes from system read and unchecked
    - source data come from system read
    - source data size is much smaller than length
    - (Optional) target data goes to system write
- Theory limits: problem ultimately undecidable and exponential. false alarms inevitable.

memcpy(target, source, length)

To stop the bleeding ……

- Context-sensitive analysis : the capability to walk up and down the calling stack (exponential)
- Path-sensitive data flow analysis : to precisely understand the flow/propagation of data on the stack (exponential)
- Pointer analysis: to understand how values flow through the heap (undecidable)
- Abstract interpretation: to compute the value ranges program variables can take (exponential)
- Propositional satisfiability: to understand boolean predicates in programs (NP-complete)

pinpoint @ cse

- On-going research in tackling all these challenges
  - Cutting edge symbolic pointer analysis to understand heap
  - Massive use of theorem prover to understand path logic
  - Summary-based data indexing on cloud to address scalability