David Brumley et al.
Presentation by Paul Marinescu

Automatically Identifying Trigger-based Behavior in Malware

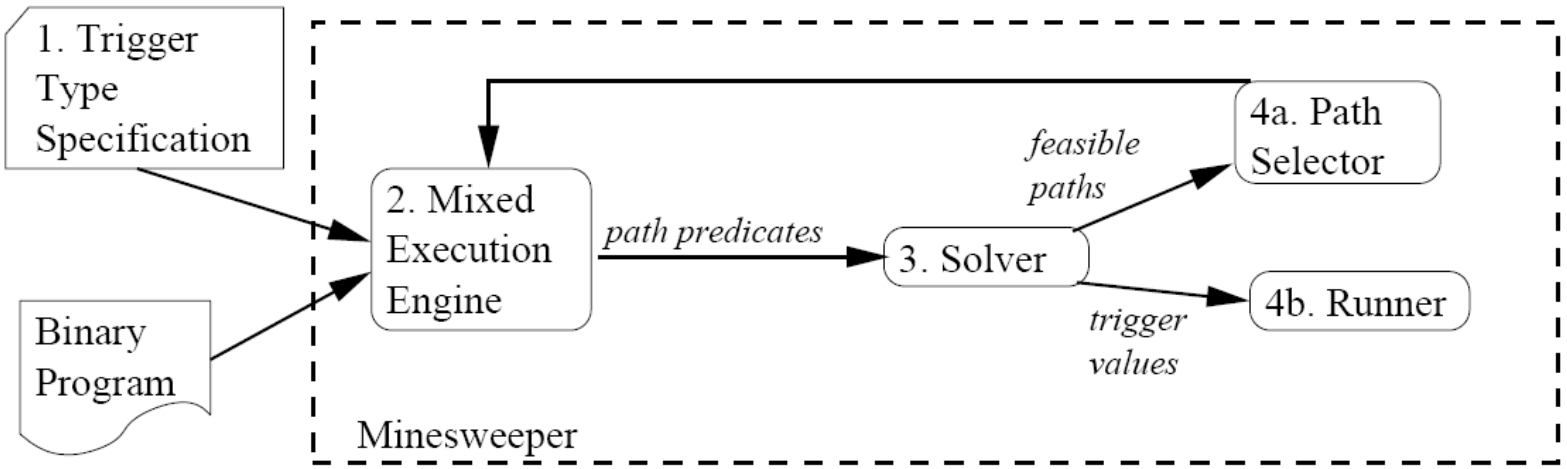
Motivation

- Malware is bad – economic loss made by Conficker:$9.1 billion dollars (US)
- Manual analysis is tedious
- Certain code paths are only be executed when certain *trigger conditions* are met

MineSweep

- MineSweep: an automatic way to identify and reason about trigger-based behaviors
- Triggers are basically inputs that excersise (chains of) conditional jumps – MineSweep tries to construct values for the triggers such that the conditions evaluate in the right direction
- …using mixed concrete and symbolic execution

System Overview



Triggers

- System time, system events, network and keyboard inputs and return values from library or system calls
- Technically, they are API calls e.g., GetLocalTime, recv
- The symbolic execution engine treats their output as symbolic

Mixed Execution Engine

- Instruction is to be executed symbolically?
  - No
    - Execute it on the real CPU
  - Yes
    - Retrieve concrete operands from the real machine
    - Translate the instruction to IR
    - Execute it symbolically

IR Translation

- *add eax, ox2* becomes…

```
tmp1 = EAX; EAX = EAX + 2;
//eflags calculation
CF:reg1_t = (EAX<tmp1);
tmp2 = cast(low, EAX, reg8_t);
PF =(!cast(low,
((((tmp2>>7)^(tmp2>>6))^((tmp2>>5)^(tmp2>>4)))^
(((tmp2>>3)^(tmp2>>2))^((tmp2>>1)^tmp2)))), reg1_t);
AF = (1==(16&(EAX^(tmp1^2))));
ZF = (EAX==0);
SF = (1==(1&(EAX>>31)));
OF = (1==(1&(((tmp1^(2^oxFFFFFFFF))&(tmp1^EAX))>>31)));
```

Symbolic execution – bounding size

- Pure symbolic execution can produce formulas exponential in the size of the program
- e.g., x1 = x0+x0; x2 = x1+x1; x3 = x2+x2;
  - x3 = x0+x0+x0+….+x0 where there are 8 x0's.
- Solution: common sub-expressions can be named using a let expression:
  - let x1 = x0+x0 in let x2 = x1+x1 in x2+x2

Symbolic execution – function summaries

- In order to speed up the analysis, well-known (pure) functions are replaced with a *summary* when executed symbolically
- strstr, strlen and other string manipulation function
- The summary has the same `symbolic` effect as the original

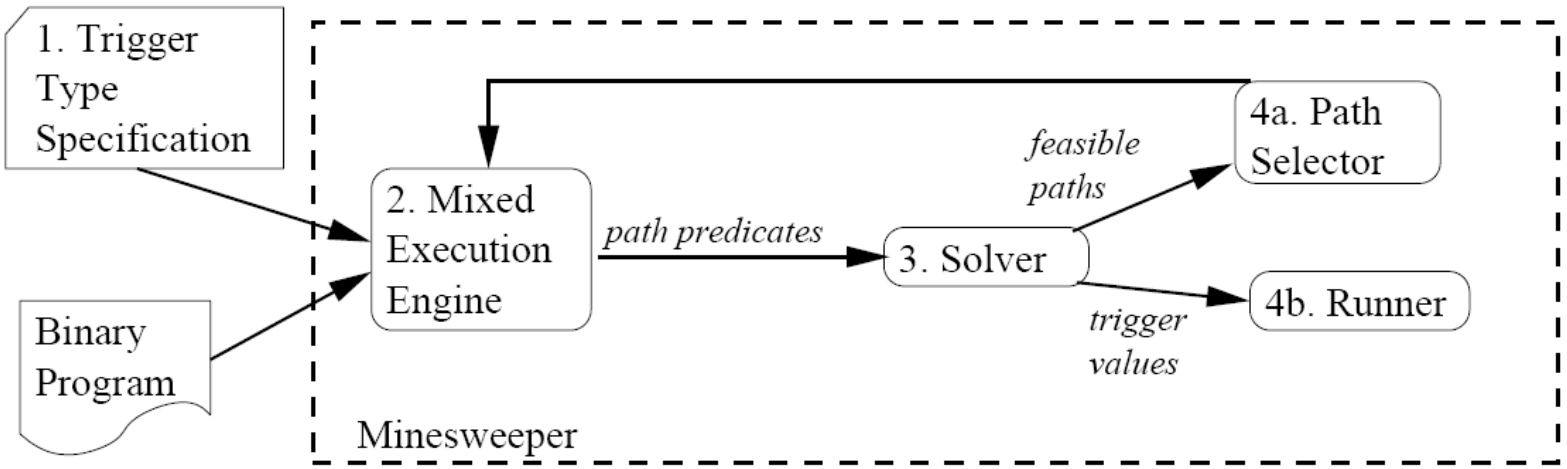
Symbolic execution - jumps

- **cjmp(e, true branch, false branch)**

- If current path predicate is ϕ
- Then new path predicates are:
  - $\phi \wedge e$ for the true branch
  - $\phi \wedge (\neg e)$ for the false branch

- Conceptually, the mixed symbolic execution engine forks to explore each (feasible) path

System Overview - Solver

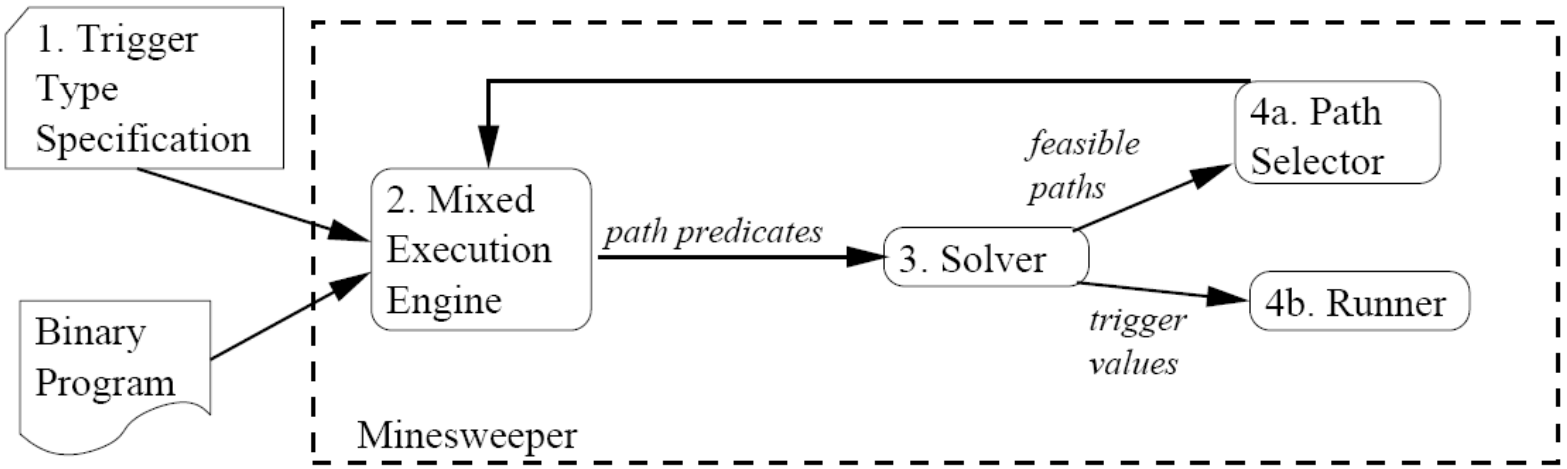


Solver (STP)

- For each generated path predicate, the Solver checks whether it is *satisfiable* i.e. whether there is an input that can make execution follow that path

- Might not always find the solution:
  - if (md5(x) == y) …

System Overview – Path Selector



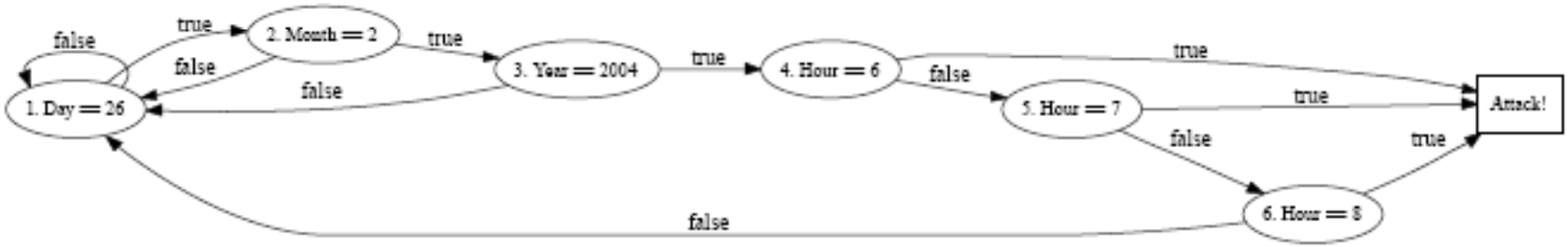
Path Selector

- Responsible for giving the *best* currently discovered path to the symbolic execution engine
- Best = most likely to contain (or lead to) malicious code
- Uses a BFS approach - loop bodies are executed once

Evaluation

Program	Total Time	Total STP Time	#Trigger Jumps	% Sym. Instr.
MyDoom	28 min	2.2 min	11	0.00136%
NetSky	9 min	0.3 min	6	0.00040%
Perfect Keylogger	2 min	< 0.1 min	2	0.00508%
TFN	21 min	6.5 min	14	0.00052%

Evaluation – sample output



Limitations

- Unsupported
  - System calls with symbolic arguments
  - Indirect jumps to symbolic locations

Sidenotes - BitBlaze

- The authors also created *BitBlaze* – a binary analysis framework (similar in principle to LLVM and Phoenix) but that also directly supporting mixed symbolic execution
- …and similar projects like BitScope, Panorama (taint analasys), Renovo (extract original code from packed executables), HookFinder…