

Dependency Graphs

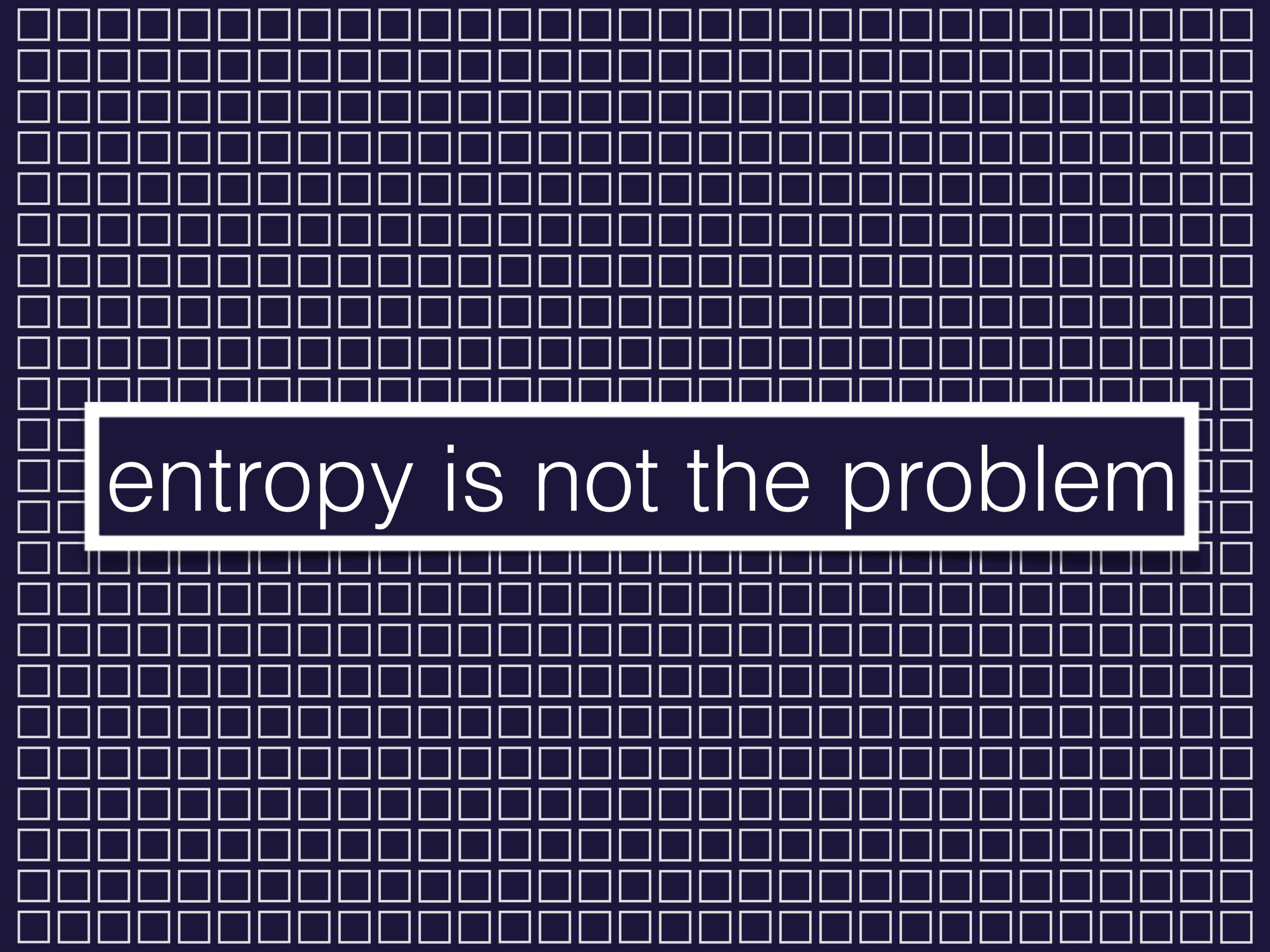
An approach for moving target defense selection

MOTIVATION

APPROACH

APPLICATION

why not more moving target defenses?



entropy is not the problem

ASLR

linux, windows, iOS, macos
gcc, clang,
etc.

Networks

?

yes!

arp, mitm, network sniffing, scanning

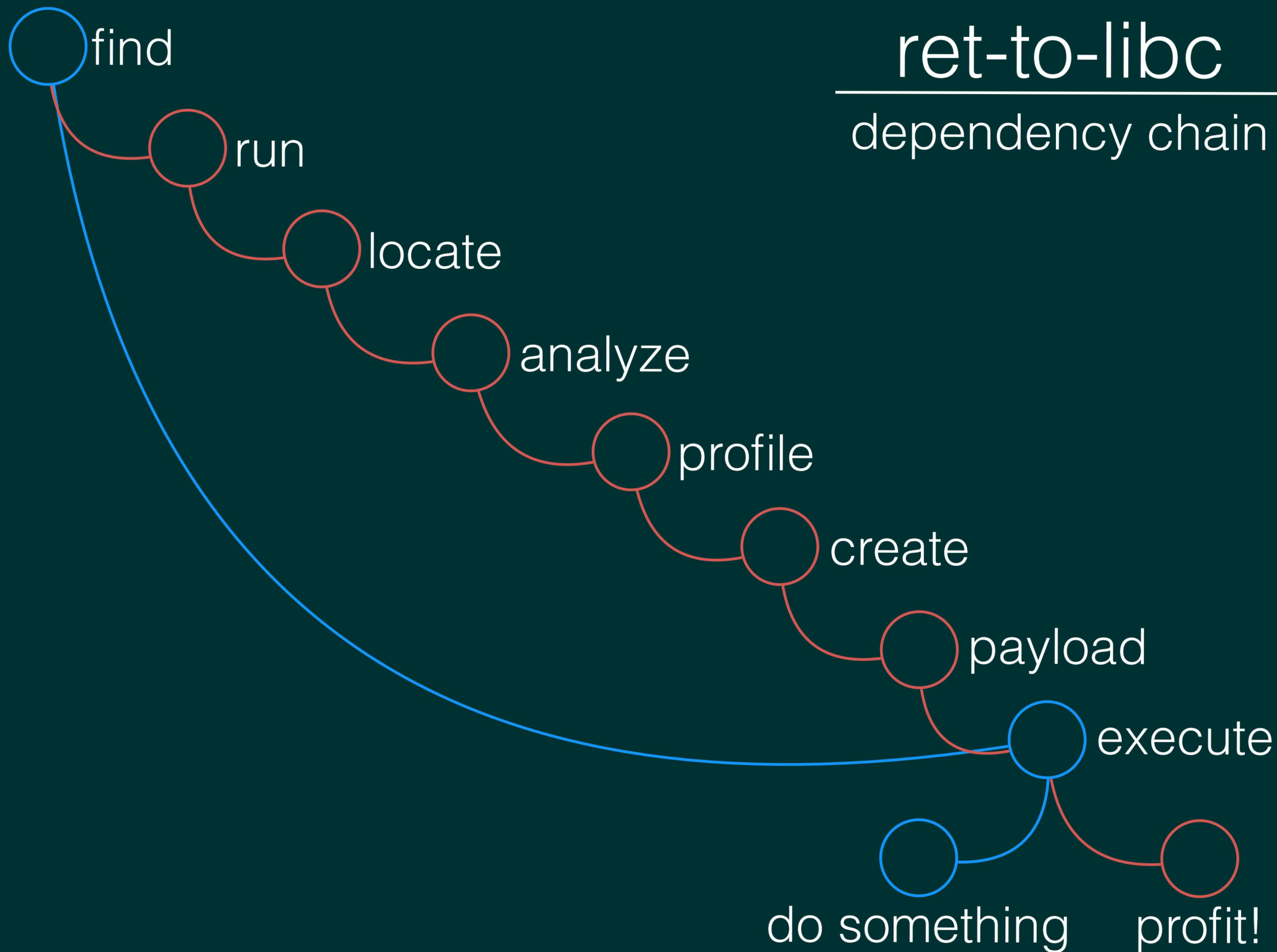
do attackers always look like admins?

no!

Shellcode, payloads, calling patterns

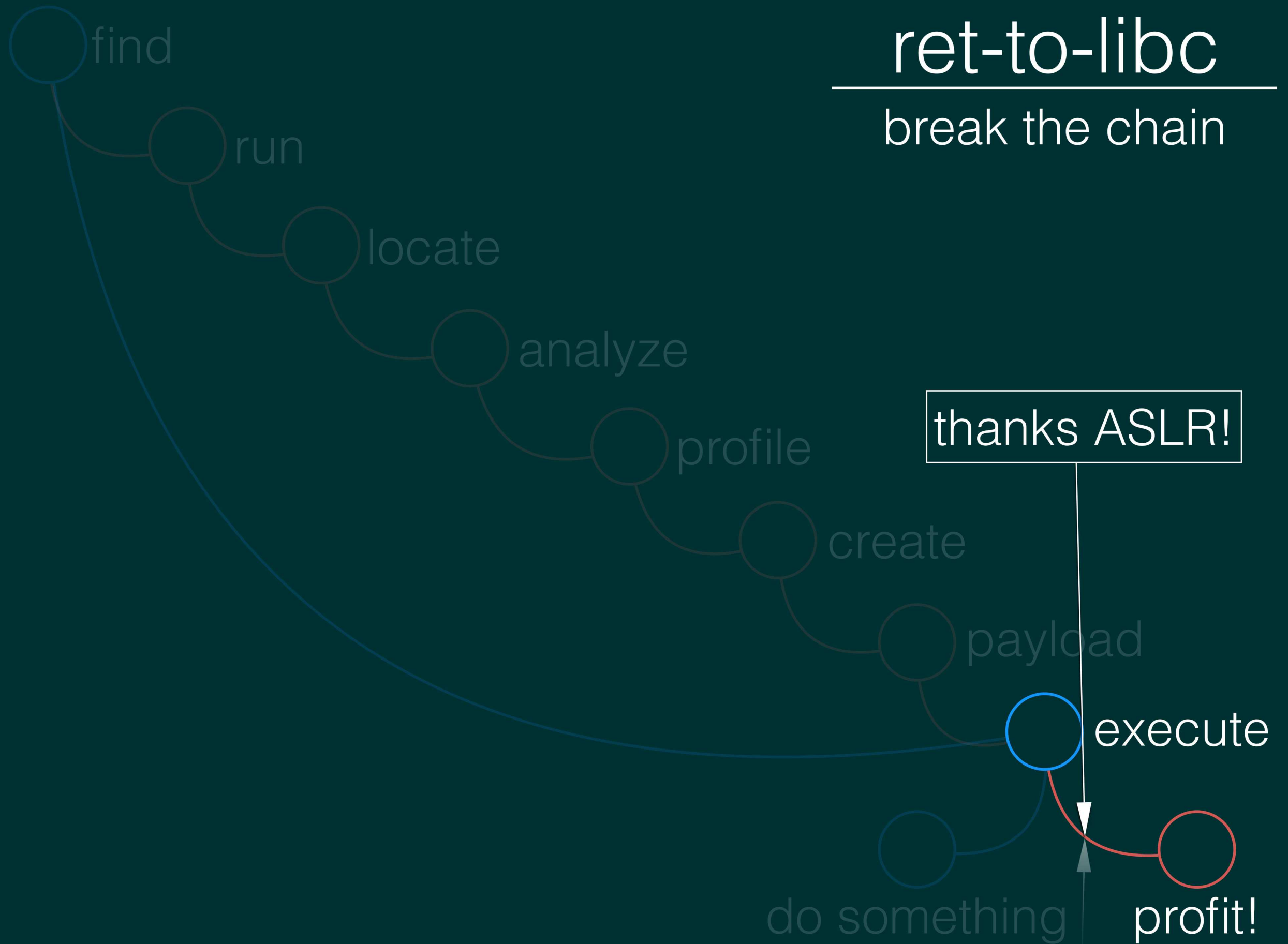
ret-to-libc

dependency chain



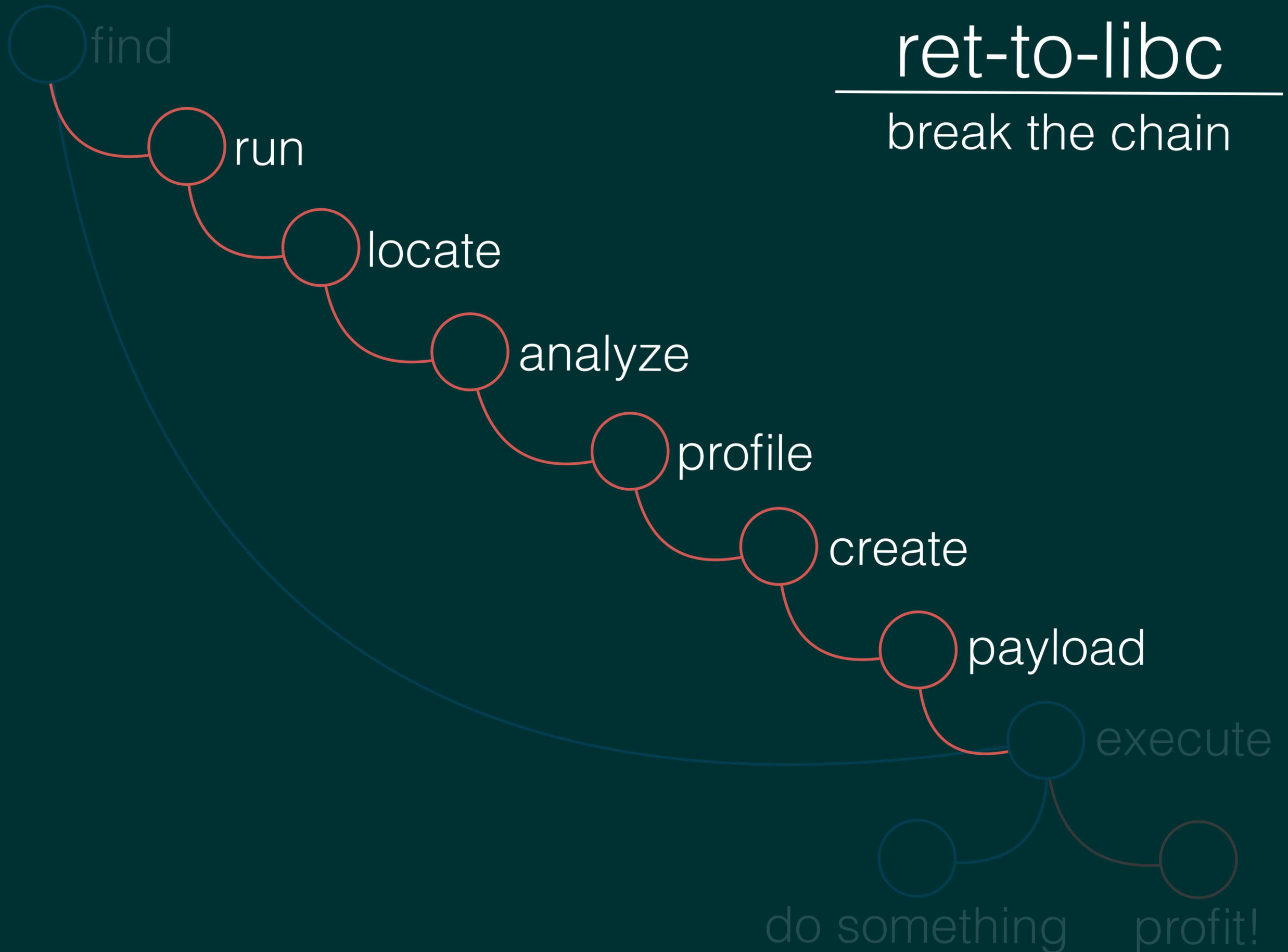
ret-to-libc

break the chain



ret-to-libc

break the chain



can we use this to dynamically select defenses?

use case: ASLR

weighted graphs reflecting relevant dependencies

adversaries

- time for access
- cost for access
- time for knowledge
- cost for knowledge
- unpredictability
- movement

users

- memory requirements
- CPU requirements
- system stability
- latency
- bandwidth
- stability

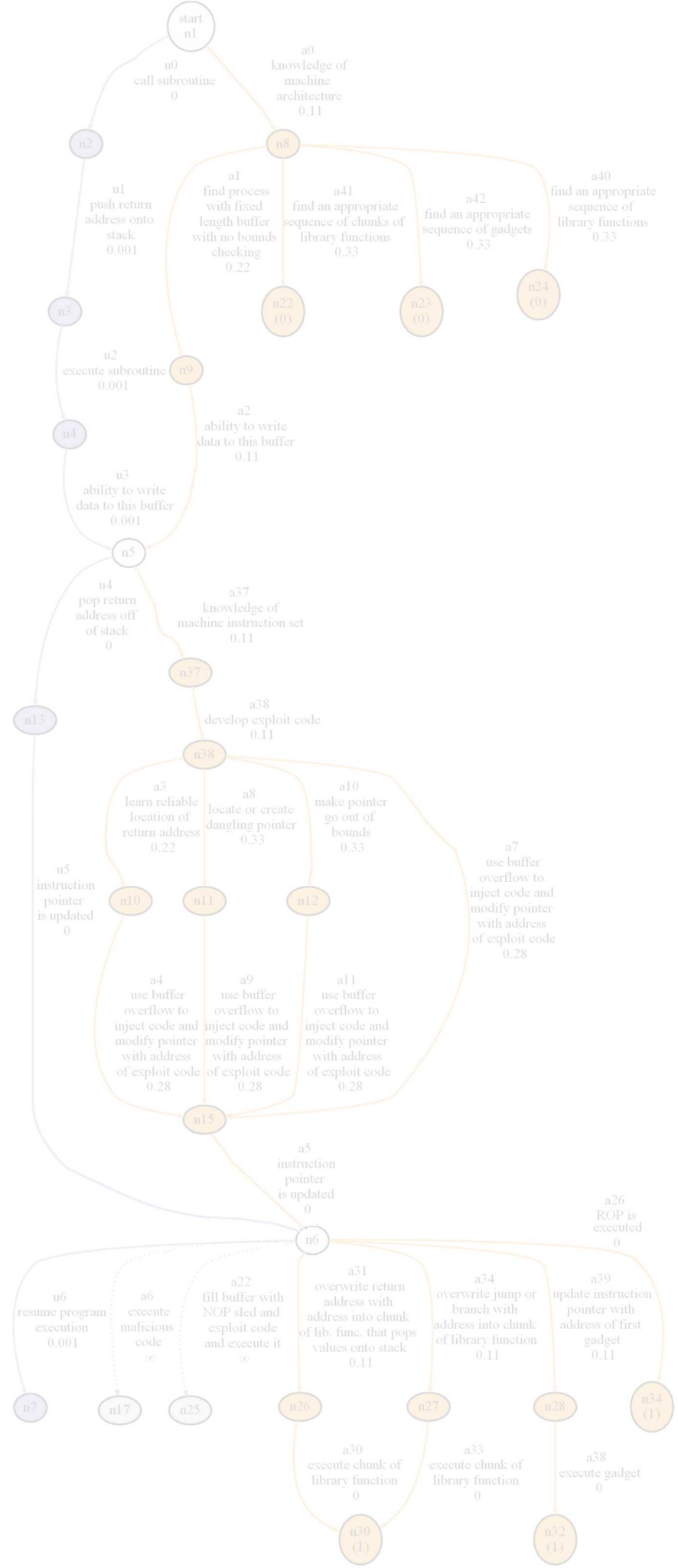
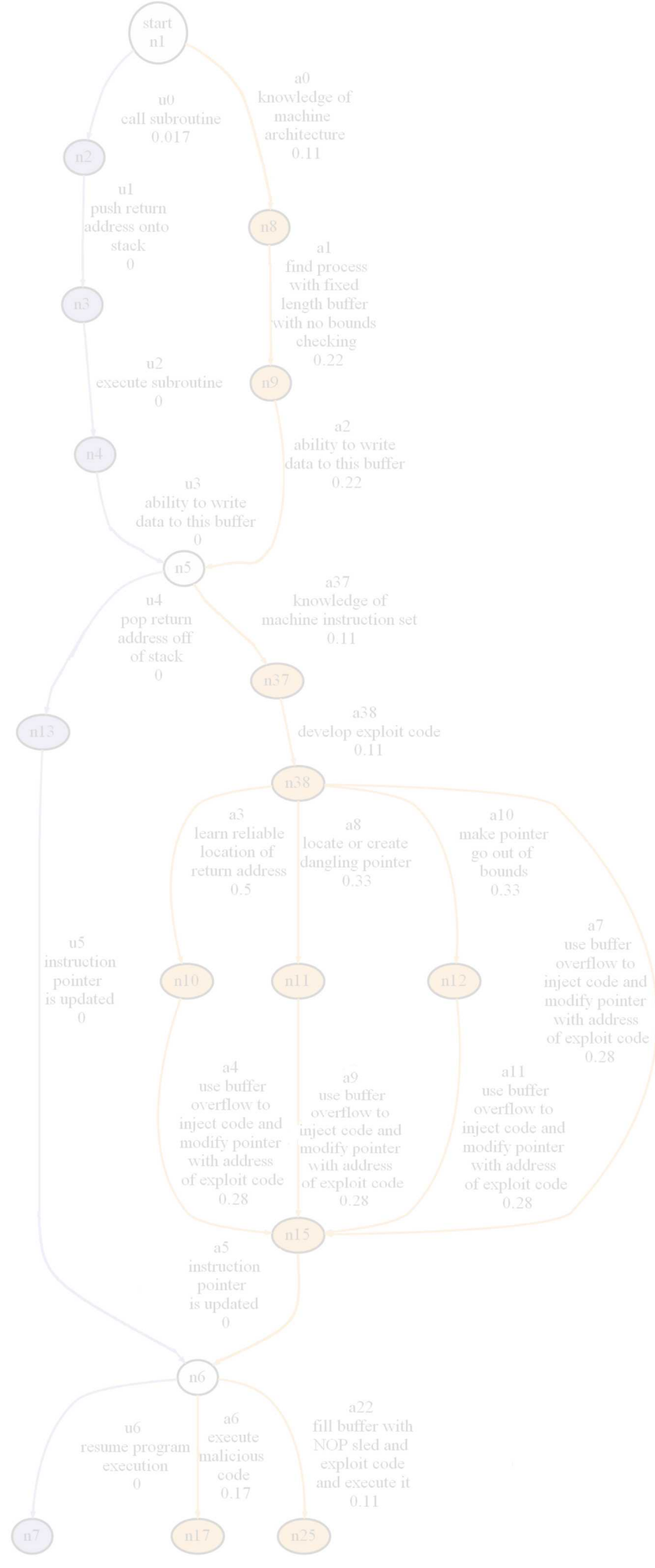
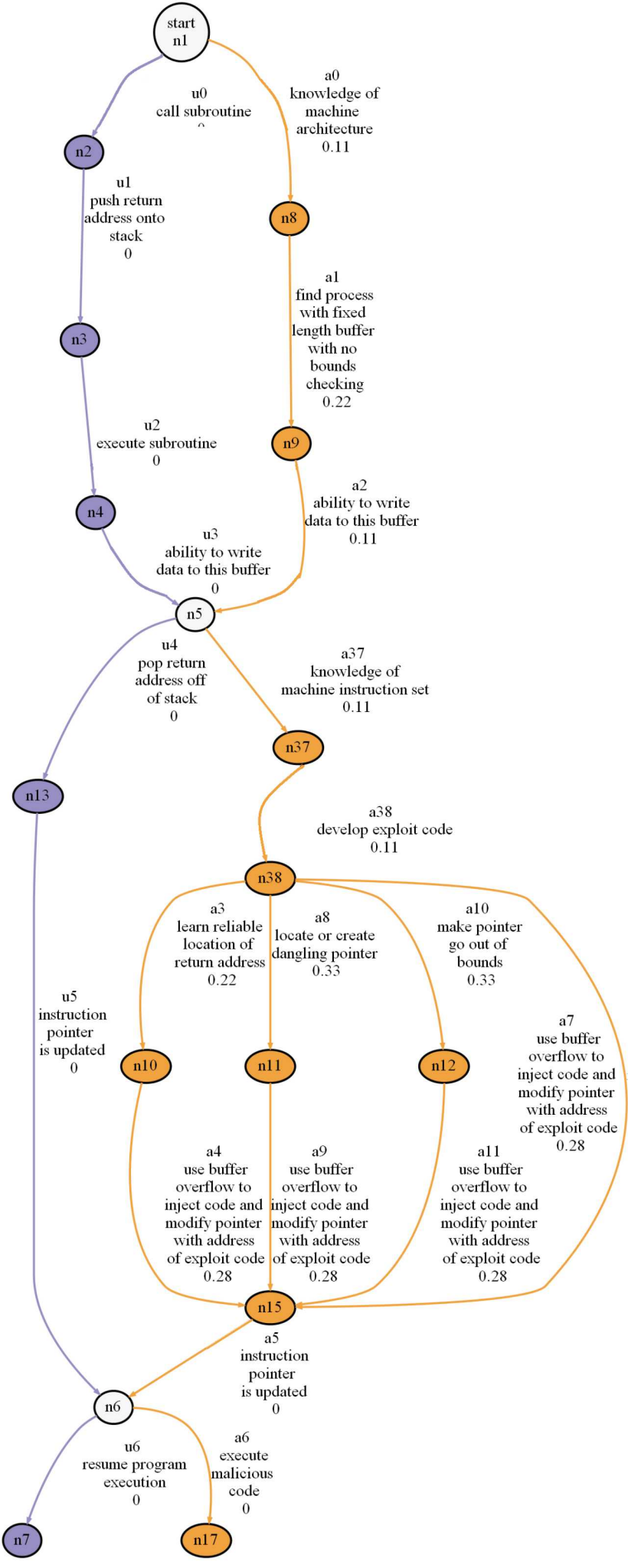
$$d = m - pm$$

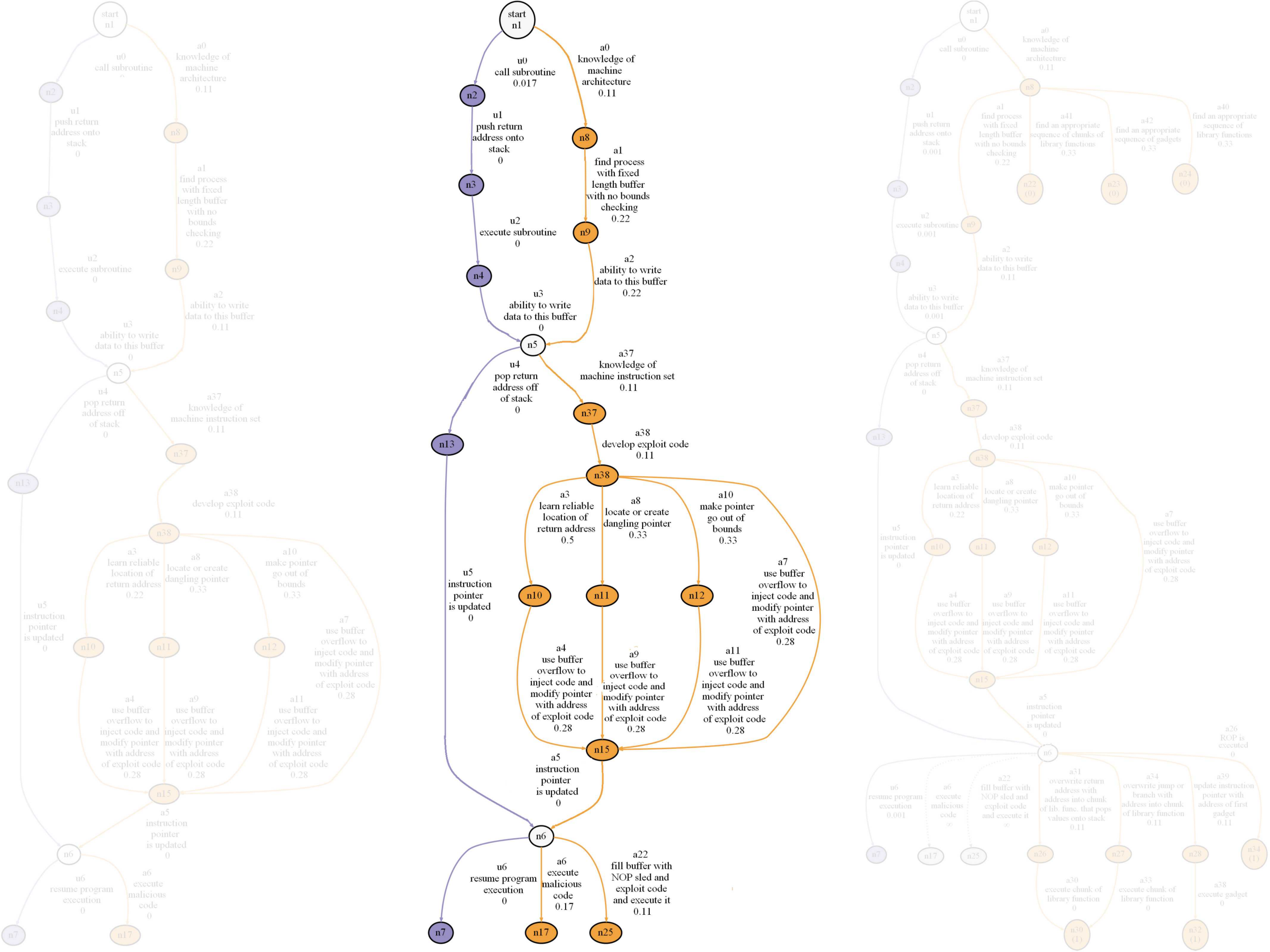
$$p = \frac{m - d}{m}$$

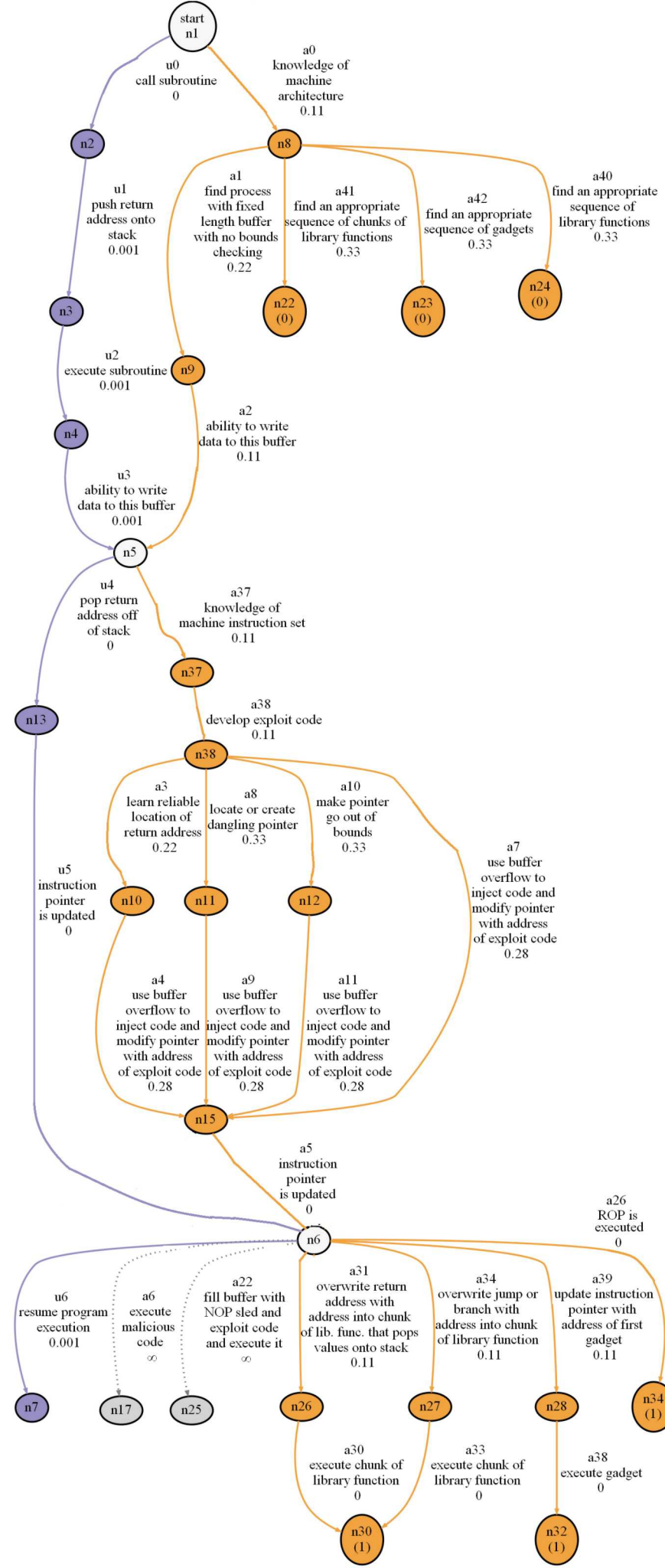
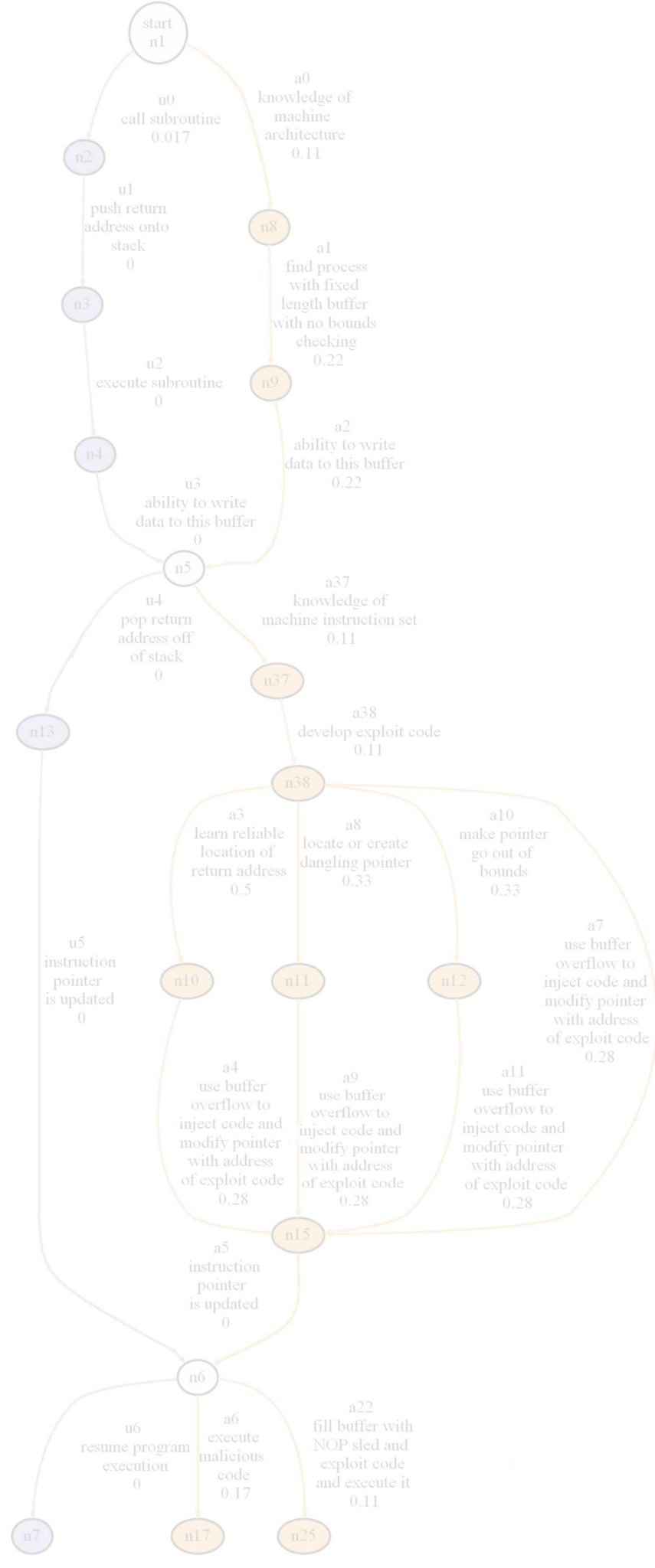
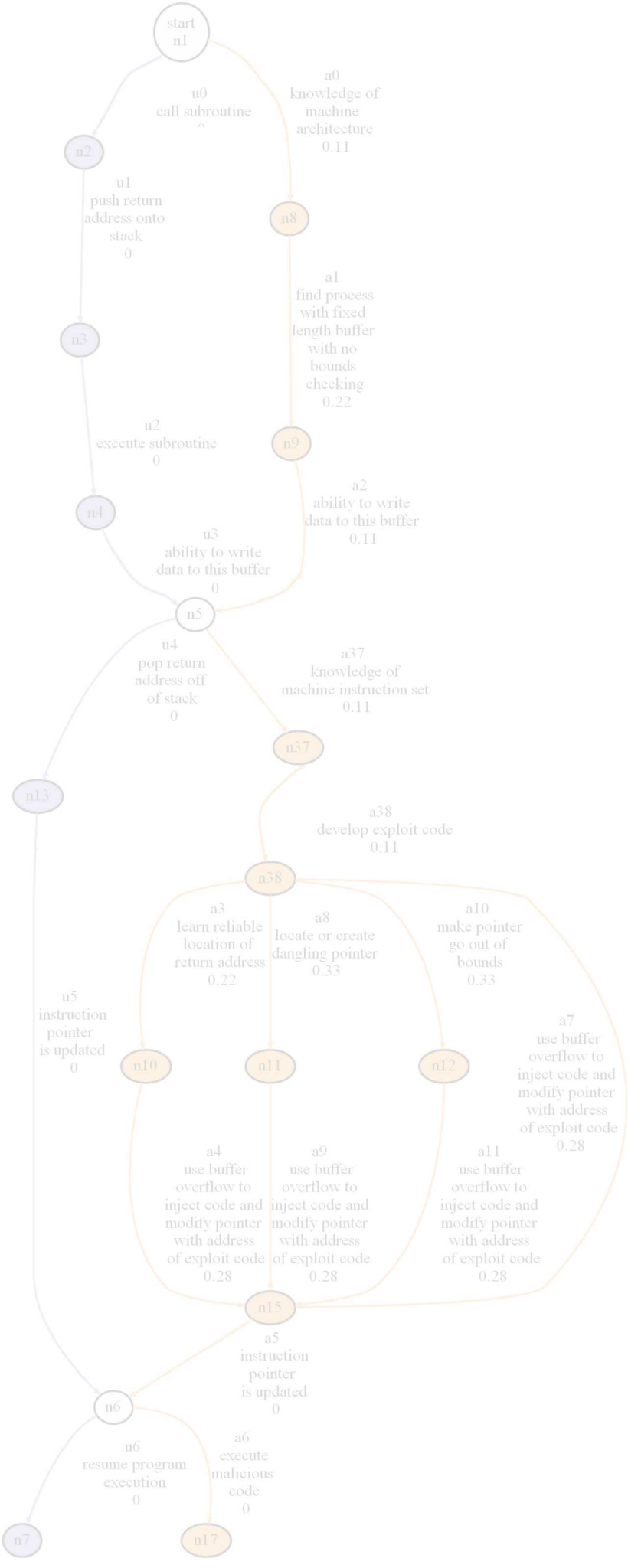
$$c = m - m \prod_i p_i$$

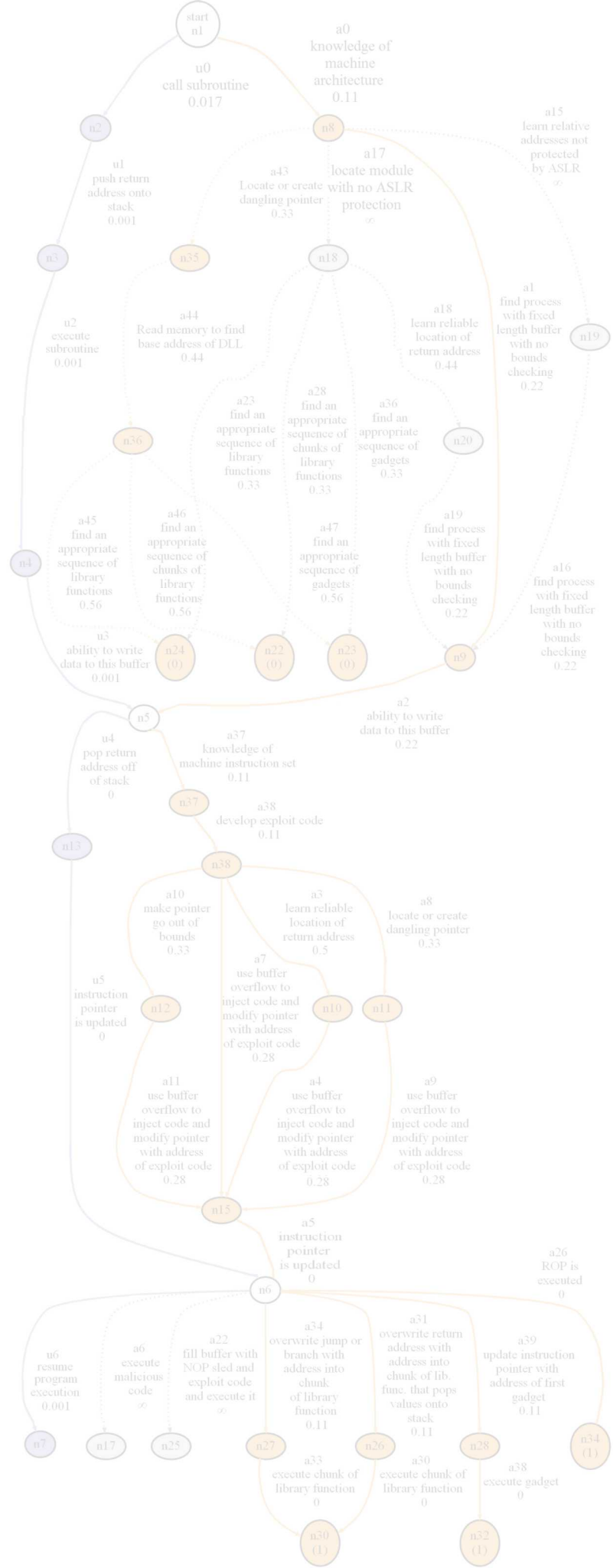
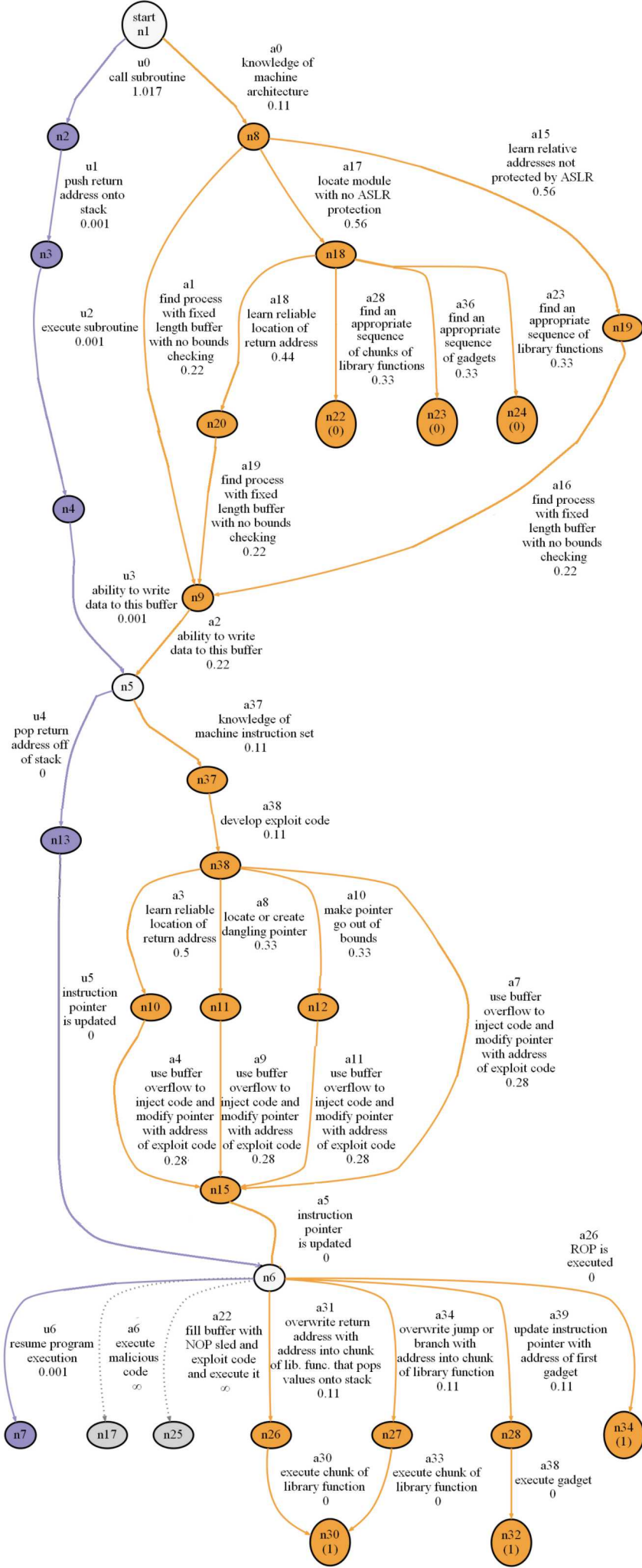
costs: not perfect, but seems reasonable

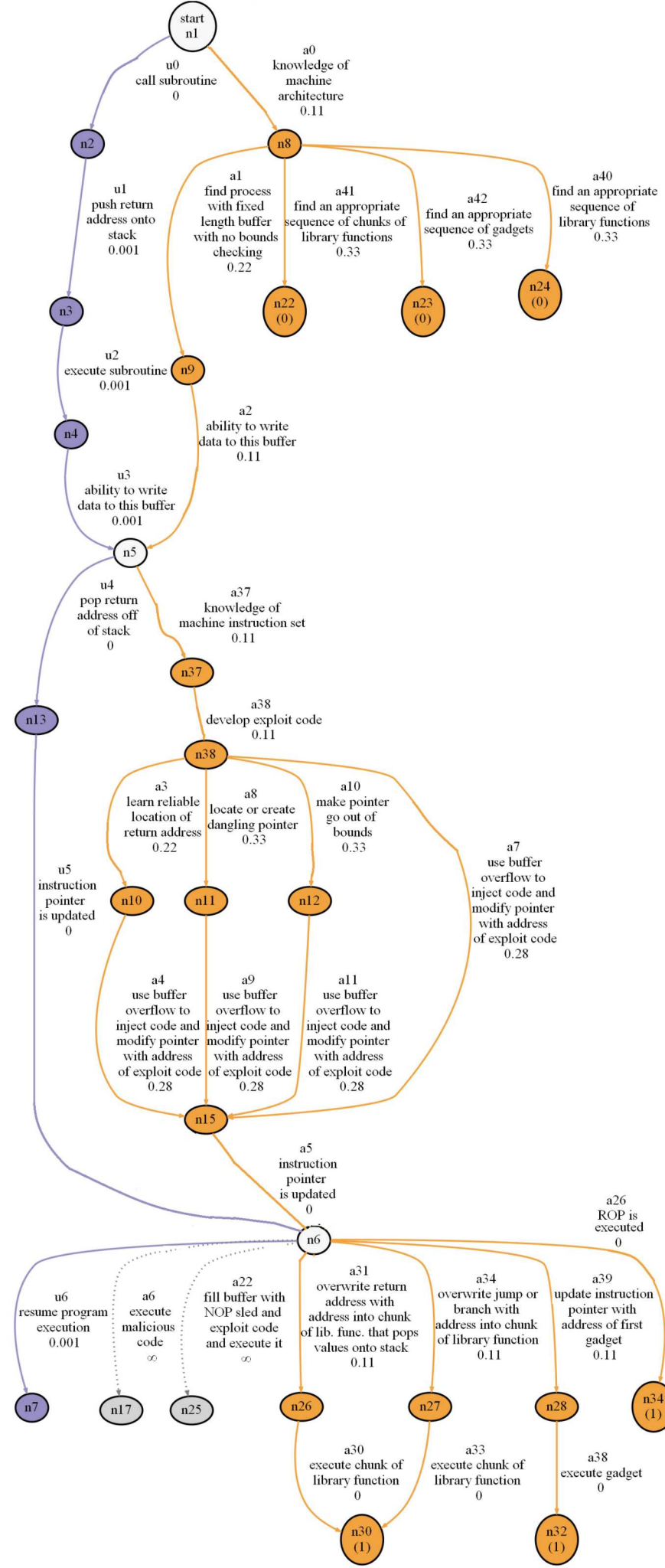
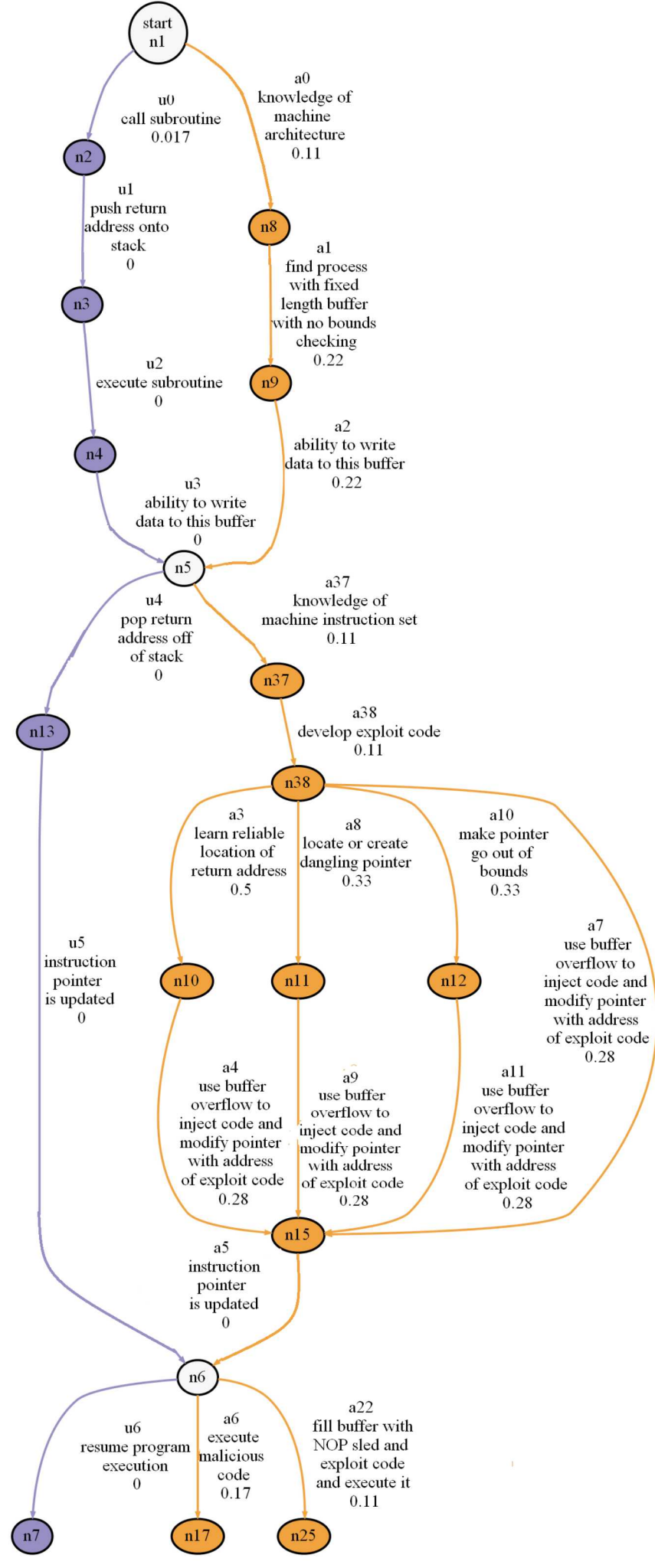
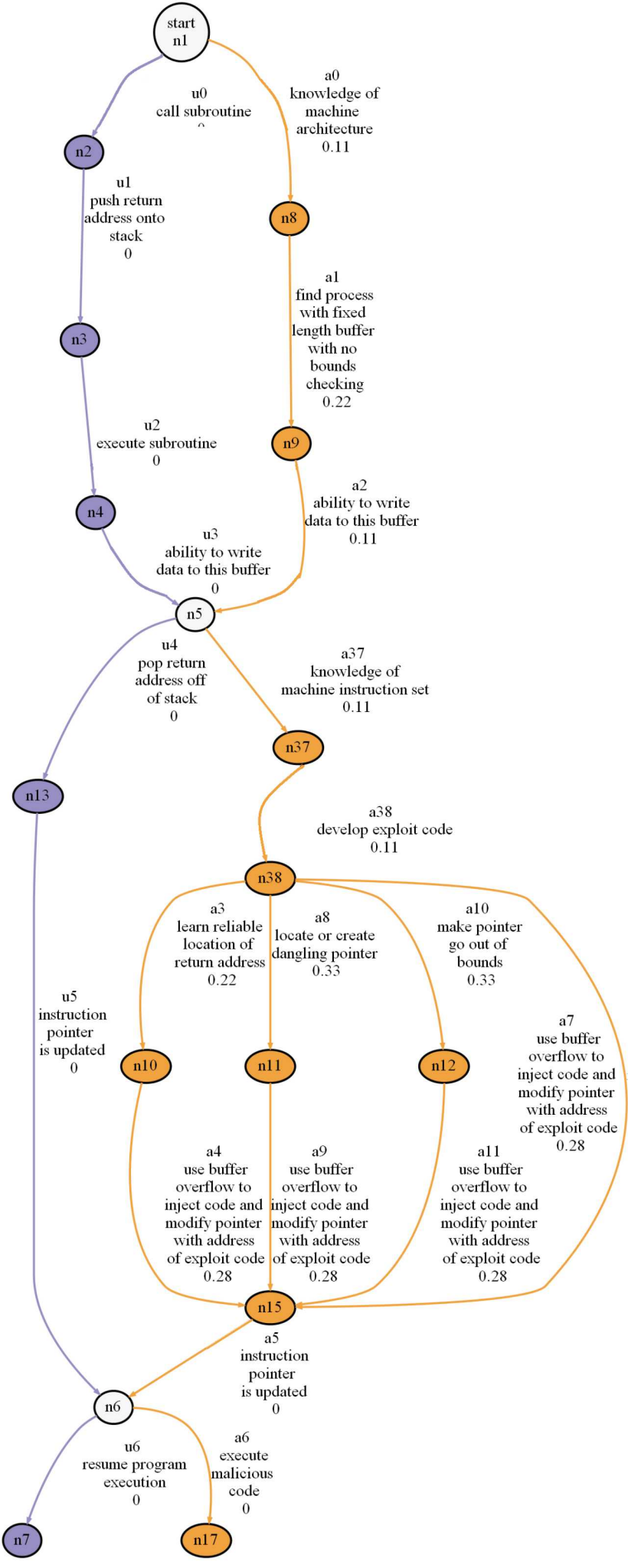
what do these graphs look like, and how does this work?

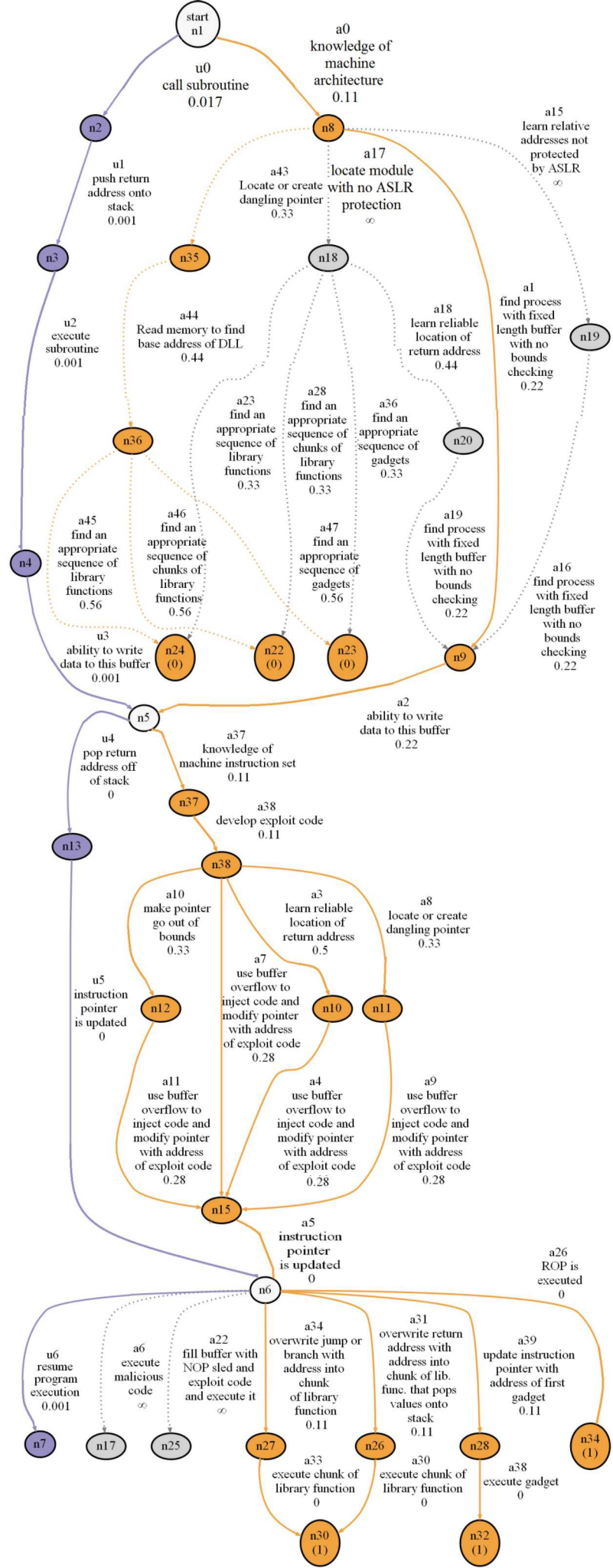
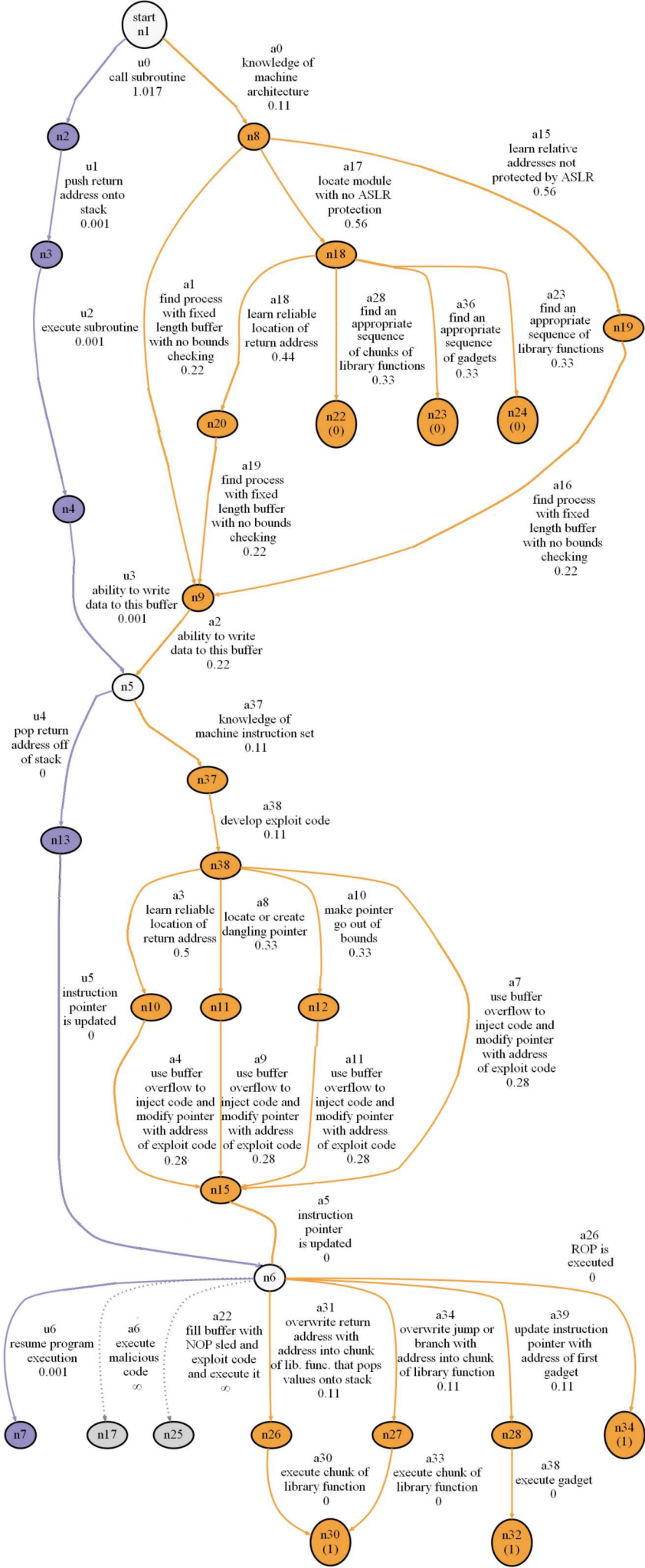












what can we do with these graphs?

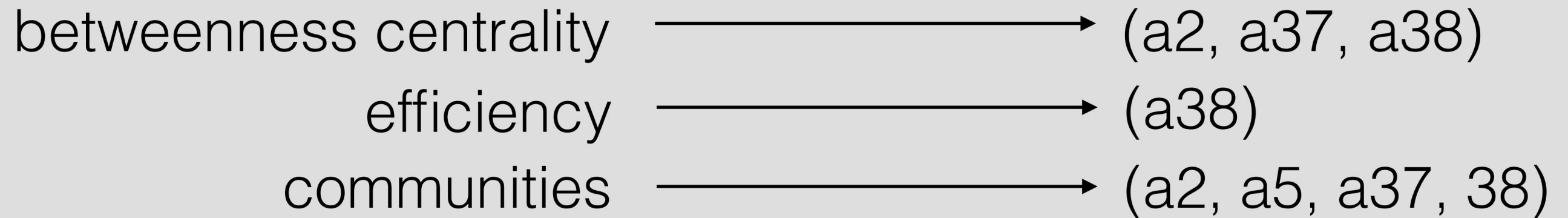
centrality

communities

cut finding

efficiency

automating defense discovery



ability to write to buffer, updating instruction pointer,
knowledge of instruction set, exploit code development

automating defense discovery

betweenness centrality	—————→	(a2, a37, a38)
efficiency	—————→	(a38)
communities	—————→	(a2, a5, a37, 38)

instruction set randomization?

ability to write to buffer, updating instruction pointer,
knowledge of instruction set, exploit code development



Thank you!