Systems Thinking + Web Security

January 10, 2013 *Michael Stone* *mistone@akamai.com*

Desiderata

- A convincing theory of accident causality, reporting, and prevention; e.g.,
  - *able to adequately explain known historical accidents*,
  - supporting counterfactual reasoning, feedback loops, delay, and
  - sensitive to known cognitive biases and theories of human behavior
- Reproducible analyses.
- A teachable analysis method.

Definitions

safety:	absence of accidents
accident:	an unplanned and unacceptable loss event
hazard:	system conditions in which accidents are possible
constraint:	a condition (control objective) required to prevent an accident

Rubric

System Diagram:	what's the context? who are we relying on to behave in accordance with our protocol?
Goals:	availability, authentication, secrecy, integrity, access control, resource usage, control-flow integrity, anonymity, concealment, privacy, ...
Adversary Powers:	reading, writing, spamming, spoofing, fuzzing, phishing, encrypting, decrypting, concatenating, parsing, delaying, dropping, reordering
Control System:	what work have we done to make ourselves safe?

Qualities

Awareness	Are people deeply aware of their decisions' consequences?
Justification	Are the results of each decision defensible?
Economy	Are things as simple as possible but no simpler?
Craft	Is the work good?
Coherence	Does each successive decision harmonize with previous ones?

Modes, Edges, and Gaps

mode:	a pattern of system behavior with distinctive requirements for safe operation; i.e., "flying", "gliding", "taxi-ing", and "parked". Mode confusion can be extremely dangerous.
edge:	boundaries between modes; i.e., "take-off", "stall", "low-fuel".
gaps:	an area of ignorance where contradictions or unsafe interactions may lurk.

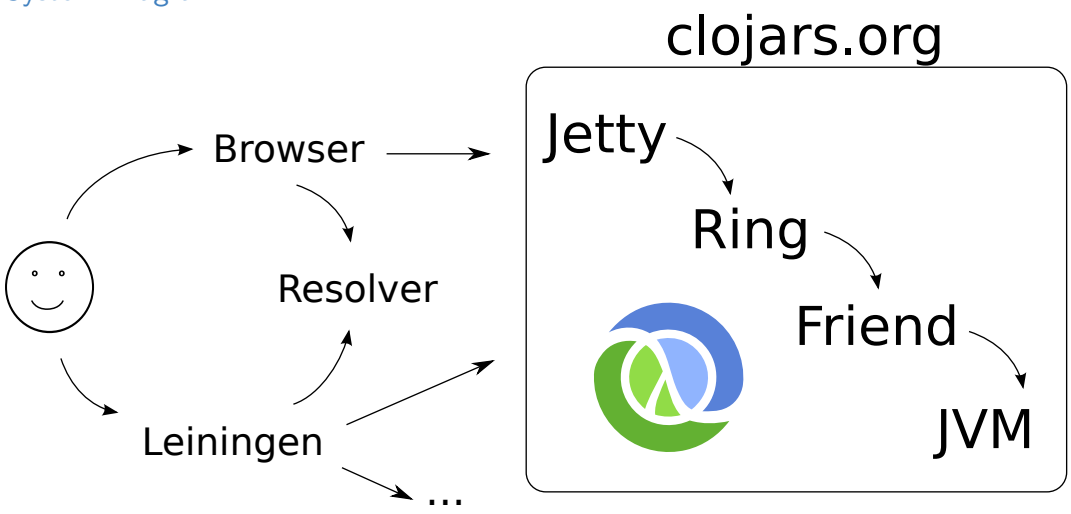
Training Aids

1. *Engineering a Safer World*, Nancy Leveson, MIT Press, 2012.
   (Available at no cost from mitpress.com.)
2. Read the security considerations discussions of well-studied problems like writing MTAs (Bernstein), censorship-resistant communication systems (Dingledine & Mathewson), and voting systems (Rubin, Felten, ...).
3. Read about famous attacks and broken software (Stuxnet, Aurora, Melissa, Flash, Sendmail, ...). How did they work, what did they achieve, and what conditions made them possible?
4. Practice breaking things. There are some great "swiss-cheese apps" and puzzle sites (Google Gruyere, 0x41414141.com) designed for this purpose.
5. Read other people's code. Learn to distinguish good code from bad code (kernel, libc, apache, X.org, postfix, sqlite, lua, ...).
6. Learn more programming languages (C, C++, ML, Lisp, Prolog) so that you learn about what problems and solutions exist.
7. Learn to write proofs, essays, and scientific reports, so you know what "evidence" means.

clojars.org example

January 10, 2013 *Michael Stone* *mistone@akamai.com*

System Diagram



Safety Constraint

Goals

Adversary Powers

Control System

Note