First level protections evaluation process

Core ideas

- white player: defender
  - the first turn is for the defender
- black player: attacker
  - all the turns but the first are for the attacker
- chessboard: the application
  - the ADSS see the application as a set of metrics
- given an asset a_j
  - it has a set of enforced security properties $\mathcal{P}^i(a_j) \geq 0$, the higher the securer
  - in a vanilla application all the properties are zero
  - it has a set of overheads $\mathcal{O}^i(a_j) \geq 0$
  - a weight $\mathcal{W}(a_j) \geq 0$
- defender moves: solutions (combination of protections)
  - solutions increase the property values
- attacker moves: attack paths
  - attack paths decrease the property values
- victory
  - attacker: when at least one security property goes below some threshold λ
  - defender: never, so he tries to delay the attacker victory, pushing it down in the search tree

Leaves evaluation

$$
\begin{aligned}
\textsc{Evaluate}(n) &= \textsc{Security}(n) - \textsc{Overhead}(n) - \textsc{Penalty}(n) \\
\textsc{Security}(n) &= \sum_i \sum_j \mathcal{P}^i(a_j) \\
\textsc{Overhead}(n) &= \sum_i \sum_j \mathcal{O}^i(a_j) \\
\textsc{Penalty}(n) &= P \cdot \sum_j \mathcal{W}(a_j) \cdot \textsc{Broken}(a_j)
\end{aligned}
$$

- a leaf n is informally a protected application attacked with some attack paths
- $0 \ll P < \infty$ is a very big user constant
- $\textsc{Broken}(a_j)$ is the number of broken properties whose value is less than the threshold λ

Properties evaluation

$$
\mathcal{P}^i(a_j) = \mathcal{W}(a_j) \cdot \sum_k \alpha_k^i \cdot \textsc{Metric}_k(a_j)
$$

- $\mathcal{P}^i(a_j)$ is the value of the i-th property on the asset a_j
- $\textsc{Metric}_k(a_j)$ is the delta of the k-th metric w.r.t. the vanilla application for the asset a_j
- $\alpha_k^i \geq 0$ is a coefficient relating the i-th property to the k-th metric

Playing a solution

- when a solution is played
  - it increases the metrics
  - it increases the overheads
- metrics can be computed via the ACTC or estimated via formulas
- overheads are computed via the PIs' formulas

Playing an attack path

$$\textsc{Metric}_k(a_j) = \pi^i \cdot \beta_k^i \cdot \textsc{OldMetric}_k(a_j)$$
$$\pi^i = \epsilon / \textsc{Effort}(i)$$

- when an attack path is played it decreases the metrics
  - if the attack is active it changes the code
  - if the attack is passive it bypasses the code complexity, emulating a sort of code simplification
- $0 \leq \beta_k^i \leq 1$ is a coefficient stating how the i-th attack path influences the k-th metric
  - can be inferred by looking at the attack steps and their types
  - the user can give a constant for each attack step type
  - these values become β_k^i if the attack path contains the related attack step types
- $0 \leq \pi^i \leq 1$ is the probability to execute the i-th attack path
- $0 < \epsilon \leq 1$ is a constant for the attacker expertise
- $\textsc{Effort}(i)$ is the effort of the i-th attack path (computed with the UEL formulas)

Approximating the metrics

Formulas

$$M_i = \bar{M}_i \cdot A_i + B_i$$

- we suppose PI independence
- we have: c code correlation sets with a assets each one, p PIs per asset and m metrics per asset
- M_i is a matrix where the (j, k) element if the k-th metric for the j-th asset