ContextIDS

Presentation of program

Slides

Application Scenario

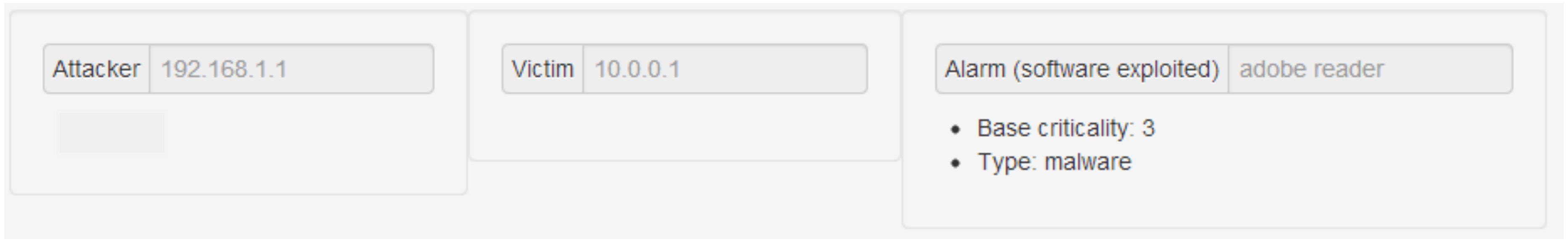
Intrusion Detection Systems often have no context

- Know nothing about victim
- Know nothing about attacker

What can we do with context once we have it?

- Give analyst basic insight on systems
- Enrich context with rules

No Context



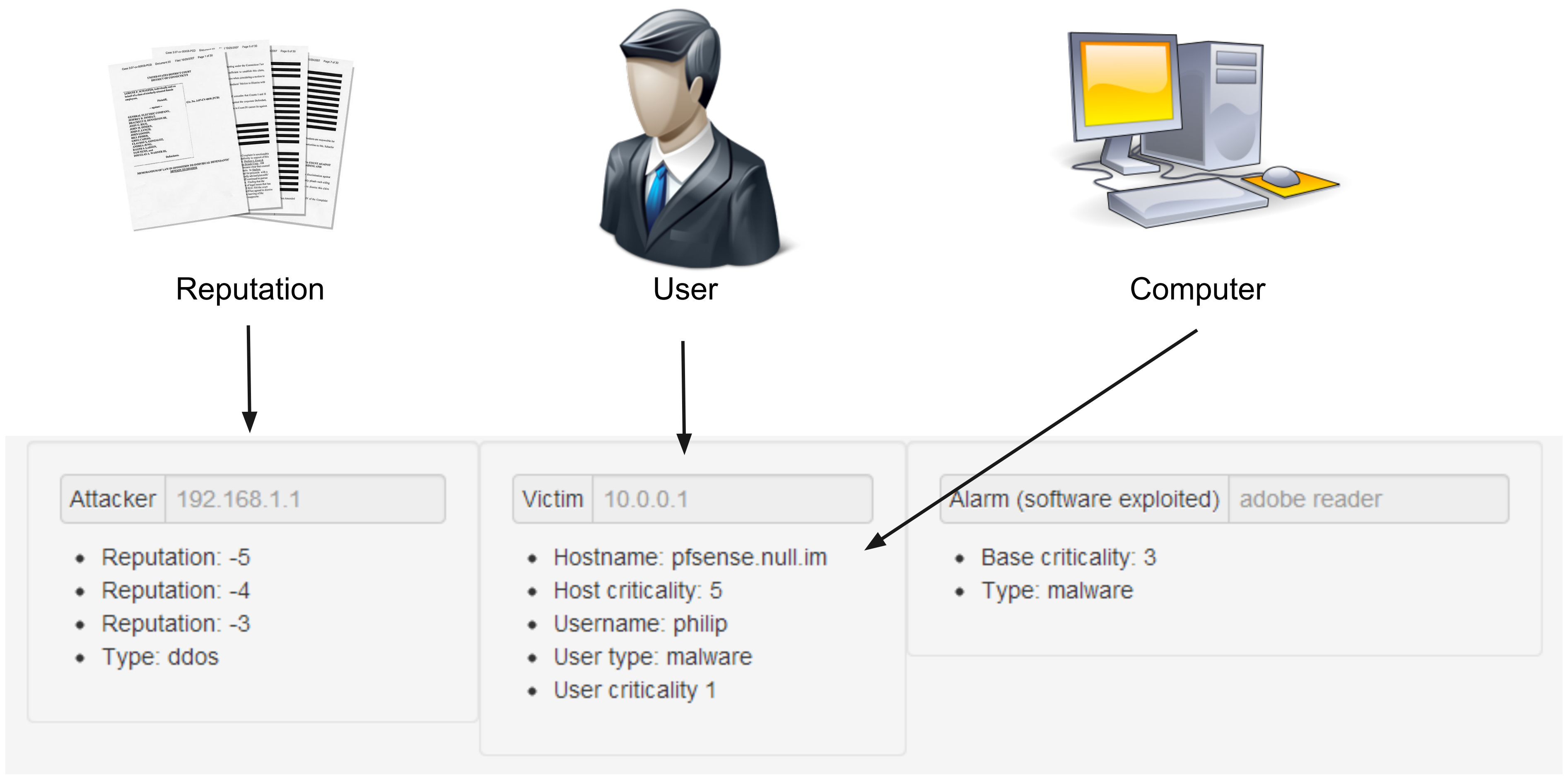
Attacker 192.168.1.1

Victim 10.0.0.1

Alarm (software exploited) adobe reader

- Base criticality: 3
- Type: malware

Integration



Attacker 192.168.1.1

- Reputation: -5
- Reputation: -4
- Reputation: -3
- Type: ddos

Victim 10.0.0.1

- Hostname: pfsense.null.im
- Host criticality: 5
- Username: philip
- User type: malware
- User criticality 1

Alarm (software exploited) adobe reader

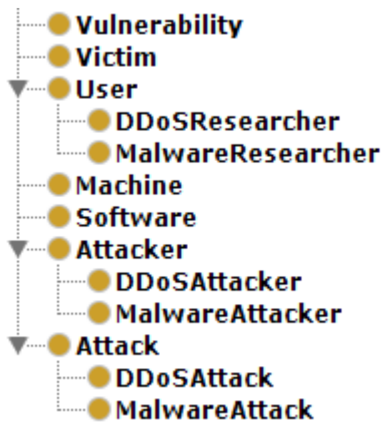
- Base criticality: 3
- Type: malware

Potential

What can we do with the context?
- Dynamic criticality
  - Based on users
  - Based on machines
  - Based on attacker
- Context aware rules
  - Attack matching victim

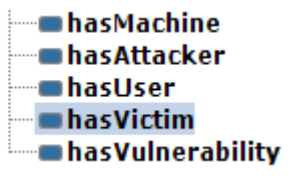
Protege - implementation

Classes



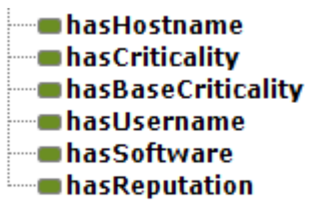
- Vulnerability
- Victim
- User
  - DDoSResearcher
  - MalwareResearcher
- Machine
- Software
- Attacker
  - DDoSAttacker
  - MalwareAttacker
- Attack
  - DDoSAttack
  - MalwareAttack

Object properties



- hasMachine
- hasAttacker
- hasUser
- hasVictim
- hasVulnerability

Data properties



- hasHostname
- hasCriticality
- hasBaseCriticality
- hasUsername
- hasSoftware
- hasReputation

Protege - implementation (2)

	Vulnerability	Victim	User	Machine	Software	Attacker	Attack
hasMachine		x					
hasUser		x					
hasVictim							x
hasVulnerability							x
hasHostname				x			
hasCriticality	x		x	x			x
hasUsername			x				
hasSoftware	x			x			
hasReputation						x	

Protege - Problems

Initially, I used entities instead of properties

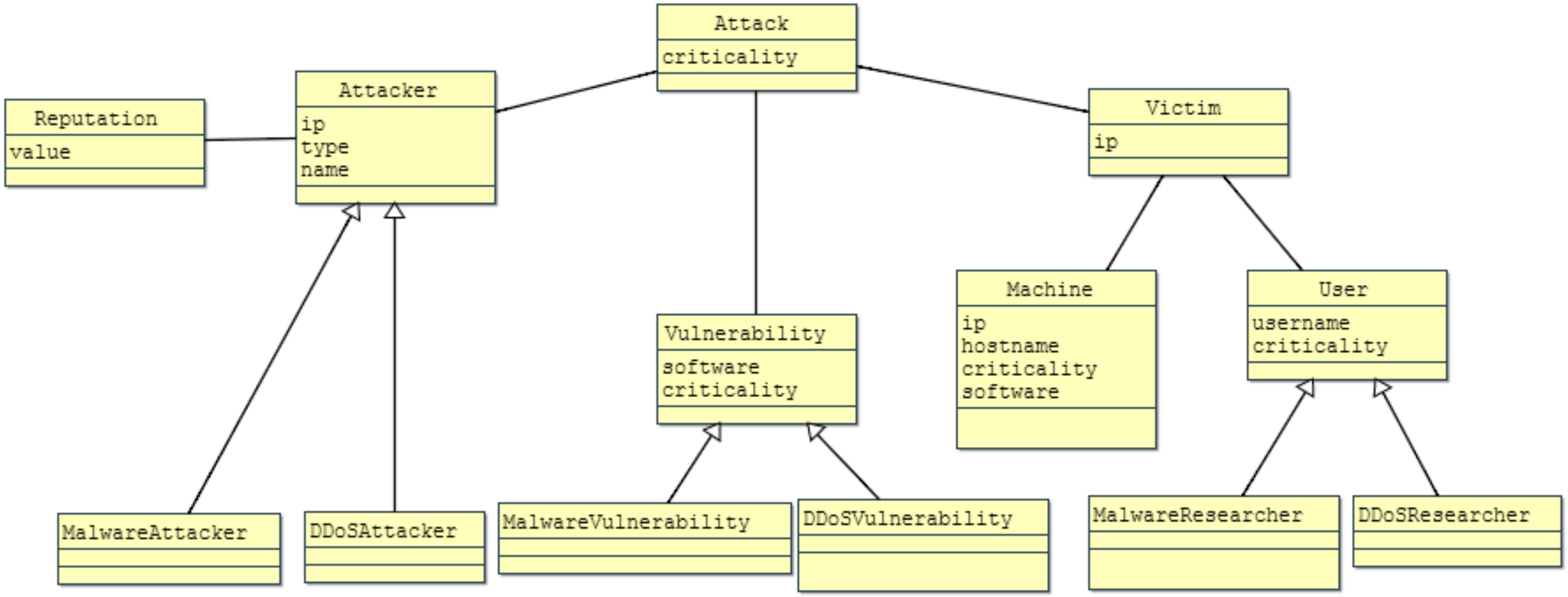
- Simplified implementation
- But no swrl over properties where calculations are needed

When implementing I had to use properties (and swrl)

- Could not make protege 4.2 set properties with rules
- Therefore, all rules are what I believe is correct, but not checked in protege.

Python - implementation

Python, with SQLAlchemy as data backend, and Flask as frontend (html)



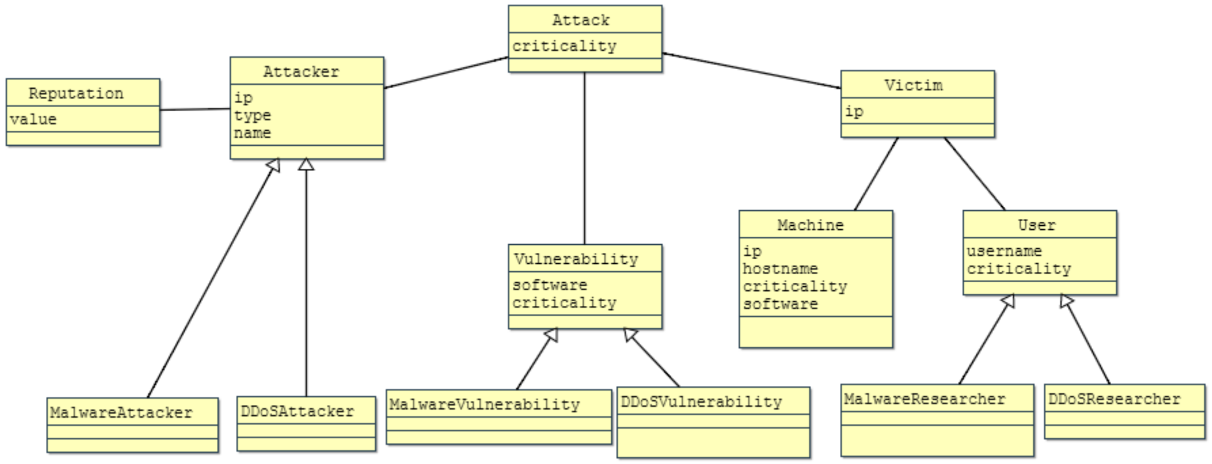
Python - problems

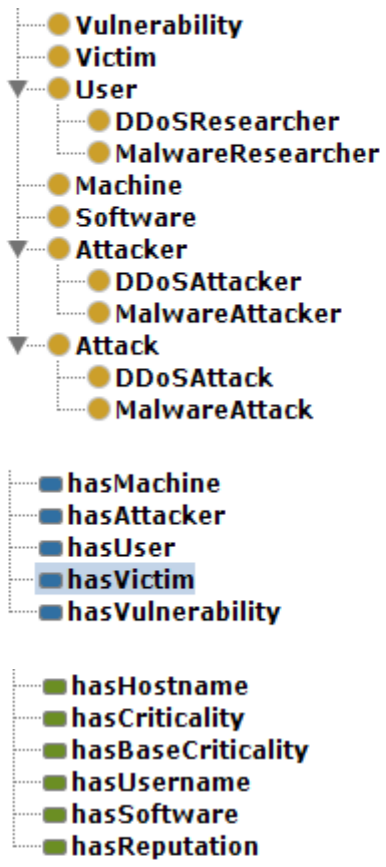
Mostly straightforward.

Retrieving objects and working with them require much more code than in Protege

Harder to fix code when changing structure

Comparison - Classes





- Vulnerability
- Victim
- User
  - DDoSResearcher
  - MalwareResearcher
- Machine
- Software
- Attacker
  - DDoSAttacker
  - MalwareAttacker
- Attack
  - DDoSAttack
  - MalwareAttack

- hasMachine
- hasAttacker
- hasUser
- hasVictim
- hasVulnerability

- hasHostname
- hasCriticality
- hasBaseCriticality
- hasUsername
- hasSoftware
- hasReputation

Comparison - Rules (1)

Description

Initially, the attack criticality is based on vulnerability criticality

Protege

```
Attack(?a), hasVulnerability(?a, ?v), hasCriticality(?v, ?result) -> hasCriticality(?a, ?result)
```

Python

```
vulnerability = session.query(Vulnerability).filter(Vulnerability.software == attack.software).first()
if vulnerability:
    attack.criticality = vulnerability.criticality
```

Comparison - Rules (2)

Description

Use the criticality of the victim user to modify the overall criticality of the attack

Protege

Attack(?attack), hasCriticality(?attack, ?crit), hasVictim(?attack, ?victim), hasUser(?victim, ?user), hasCriticality(?user, ?ucrit), swrlb:add(?result, ?crit, ?ucrit) -> hasCriticality(?a, ?result)

Python

```
if attack.victim:
    if attack.victim.user:
        attack.criticality += attack.victim.user.criticality
```

[attack.criticality = attack.criticality + attack.victim.user.criticality]

Comparison - Rules (3)

Description

If attacker is a known malware distributor, and the attack is a malware attack, chances are it is a true attack, so we escalate.

Protege

MalwareAttack(?a), hasAttacker(?a, ?attacker), MalwareAttacker(?attacker), hasCriticality(?a, ?c), swrlb:add(?result, ?c, 5) -> hasCriticality(?a, ?result)

Python

```
if attack.attacker:
    if attack.vulnerability:
        if isinstance(attack.attacker, MalwareAttacker):
            if isinstance(attack.vulnerability, MalwareVulnerability):
                attack.criticality += 5
```

Comparison - Rules (4)

Description

If an attack uses a vulnerability which the victim is known to be vulnerable to, we want to escalate.

Protege

Attack(?a), hasVictim(?a, ?victim), hasMachine(?victim, ?machine), hasSoftware(?machine, ?software), hasVulnerability(?a, ?v), hasSoftware(?v, ?software) hasCriticality(?a, ?crit), swrlb:add(?result, ?crit, 5) -> hasCriticality(?a, ?result)

Python

```
if attack.victim:
    if attack.victim.machine:
        for x in attack.victim.machine.software:
            if attack.vulnerability:
                if x.name == attack.vulnerability.software:
                    attack.criticality += 5
```

Live demo

(Say a prayer to the demo-gods, and will away the demon Murphy)

Potential improvements

More types (attacks) - Classes

Multiple types on each attacker

Dynamic criticality through dependencies

Rules over time, context in the time axis

Multiple types on each victim

- User
- Machine

Thoughts

Protege as a prototyping/modeling tool

Not using OWL in code

My development road

Protege has an overhead

But: can be used as an interactive modeling tool

- Rules in protege (pure logic) is relatively easily translated to code

Easy to prototype since

- Classes/data can quickly be set up
- Changes do not require massive change in code
- Rules allow simple reasoning over data
    - without sqwrl, it is harder to work over sets

Questions?