



**POLITECNICO
DI MILANO**



Security Workshop - Masibty

Stefano Zanero, Claudio Criscione



- Stefano Zanero
 - Assistant Professor @ Politecnico di Milano
- Claudio Criscione
 - Principal Consultant @ Secure Network
 - Hopefully soon-to-be PhD student @ Politecnico di Milano



What is our speech all about?



It's about letting people in charge of web applications security sleep at night*



* terms and conditions apply. We do not take care of your partner snoring



- Difficult, IRW to
 - **Detect attacks**
 - **Apply patches (without support from developers)**
 - **Have the time to follow all those 2458 unitasker web applications**
- In the meantime, you're likely going to get hacked by a pack of Monkeys (which can successfully hack web application, as scientifically demonstrated)





- Web Application Firewalls – a must?
 - Patching is not always possible due to “obscure reasons”
 - *Application* and *infrastructure/security* are different departments
 - You just have to do “something” for web application security, and you have to do that **yesterday**
- Most WAF solutions suffer from the “Grep Dilemma”
 - Should I really use something which is little more than a complex **Grep**?



- Inherent issues with signature based systems!
 - Application of blacklisting, and we all know blacklisting is intrinsically flawed
 - “Things that you do not hope for happen more frequently than things that you do hope for” (Plauto, “Mostellaria”)
 - You cannot enumerate all the possible attacks, and “generic signatures” yadda yadda simply do not work nearly well enough
- Applying whitelisting (i.e. only allowing through what is supposed to go through) would work, but it is a configuration nightmare
 - List **every** parameter of **every** form on **every** page of **every** application on **every** server
 - And then we can discuss “change management”, folks...
- This is why WAFs require careful configuration and constant updating
 - And **time** and **skills** are scarce resources, as usual



What are we trying to do?

- Recreate the “**Old Lady at the Window**” effect
 - You know, the old lady spotting “strange things happening” and dialing 9-1-1
- Which means...
 - Learning what's normal: Whitelisting : **Anomaly detection**
 - Block what's not: **Intrusion prevention**
 - Without administrator intervention : **Unsupervised learning**
 - With no (well, just a few) false positives
 - With attacks in the learning set – because that's what happens in the real world!



So, what is Masibty?



- A web application IPS
 - **Anomaly based**, and capable of doing unsupervised learning
 - Able to work in the “real-world”
 - Partly language-indipendant (Java **reverse proxy**) and partly language dependant (PHP PoC)
 - A flexible **architecture** where modules can be plugged into

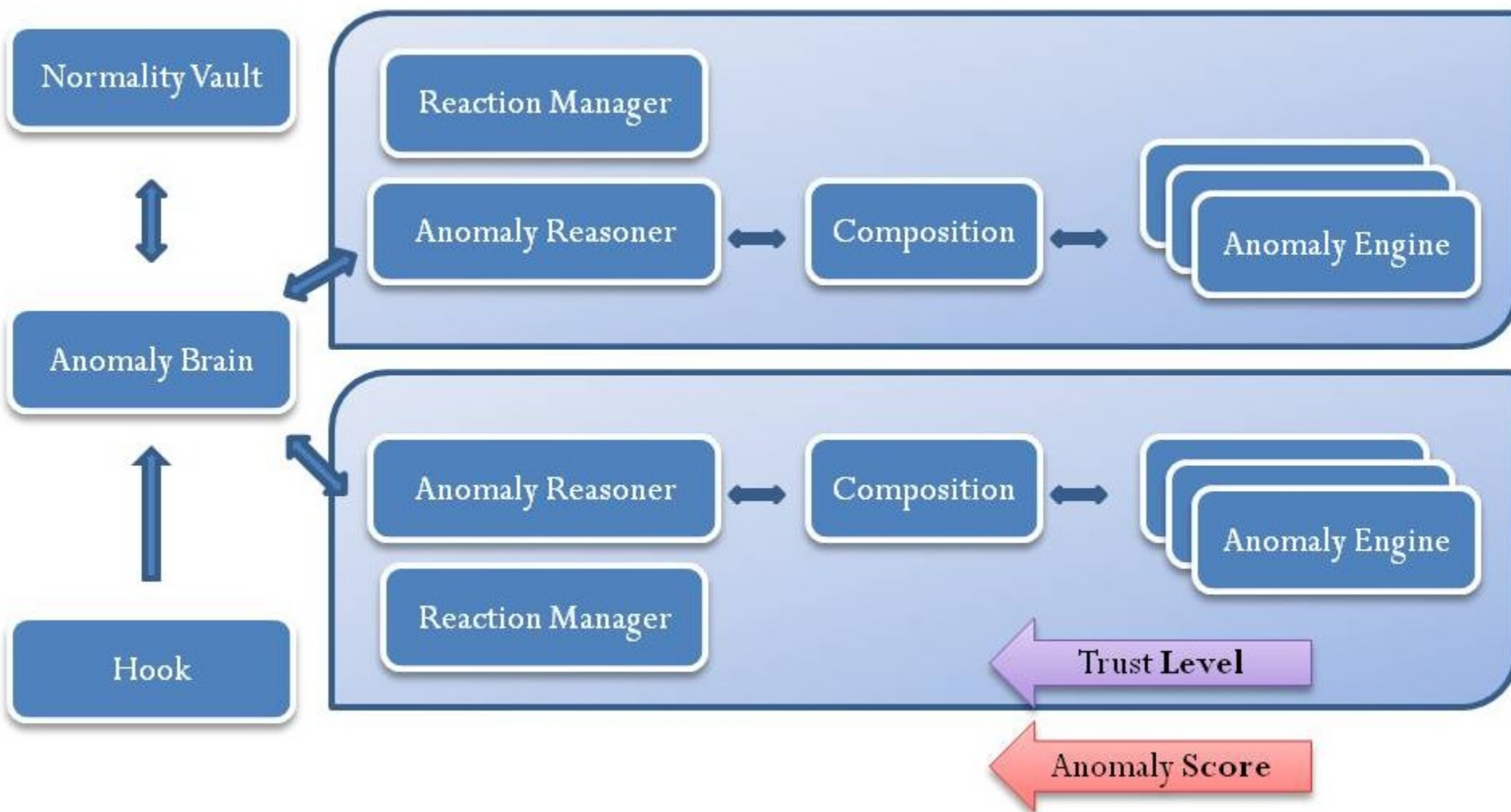


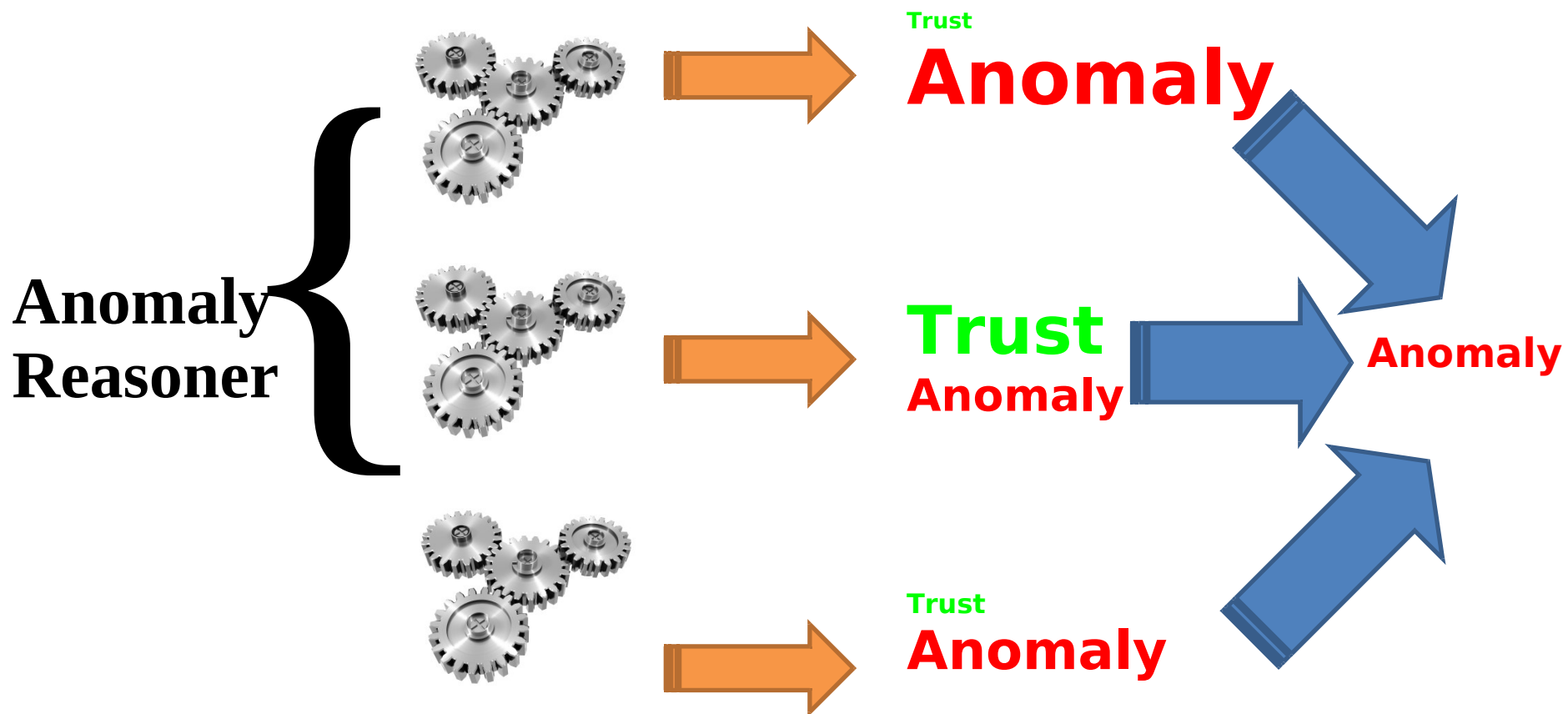


- **What are we going to learn?**
- **How are we going to learn it?**
- **How are we going to use it?**



How are we going to process the data?







- For each parameter, we build a profile using various **engines**
 - Order Engine
 - Presence Engine
 - Numbers Engine
 - Aliens Engine
 - Token Engine
 - Distribution Engine
 - Length Engine
- You can notice similarities with other models (like the ones proposed by Vigna and others)
 - We have improved some of their models or rebuilt them according to our new requirements



- Some of the engines take care of the “values” of the Parameters
 - **Number engine:** if we put a non-numerical value in an “almost always” numerical attribute, we get an anomaly
 - **Token Engine:** some parameters can only assume predefined values. They're *Tokens*.
 - **Length Engine:** parameters usually have a “similar” size
 - **Distribution Engine:** we should be able to identify notable peaks in the usage of a single character
 - **Alien Engine:** most parameters won't accept EVERY printable character



- Web applications often are “regular”, parameters are usually in the same order
 - **Order Engine**
 - **Presence Engine**
- Most structural engines can be bypassed, but are very accurate against many automated attacks!



- We now have a broad range of tools to identify attacks aimed at the server
- But yet, during the coding of Masibty, we wondered

“Since we already see all of these server responses, why don't we analyze those as well?”

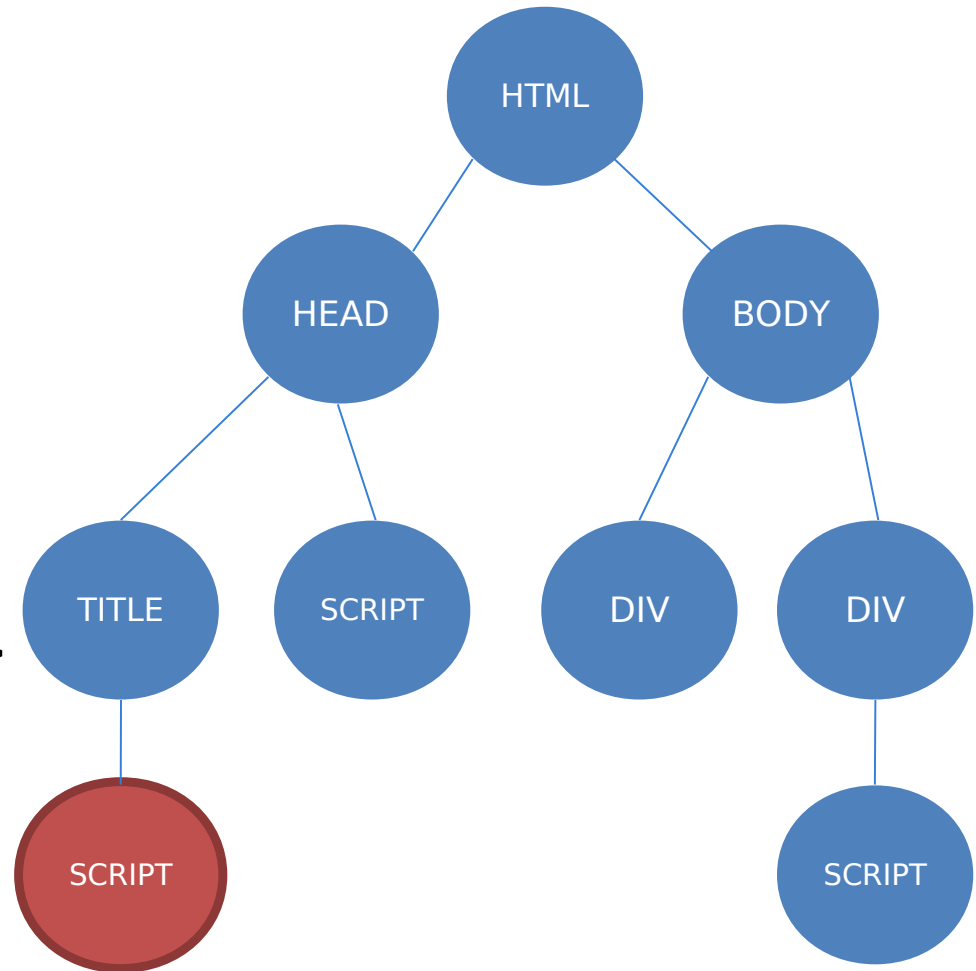


- Build a representation of server responses
 - Plant a (DOM) tree, save the environment!
- Once we have generated the tree, we can “learn” it
- If we see at some point in the future an unexpected branch on the tree...





<script>attack</script>

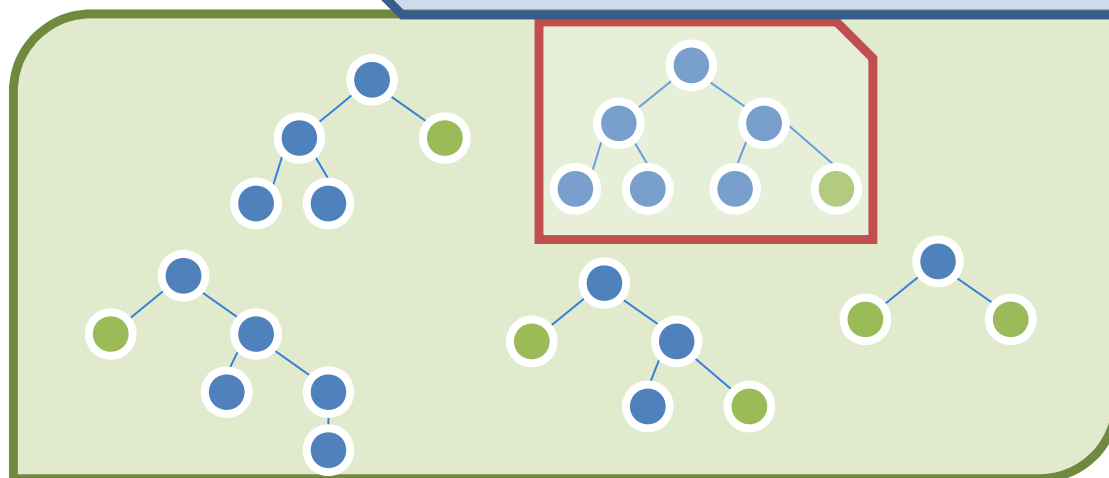
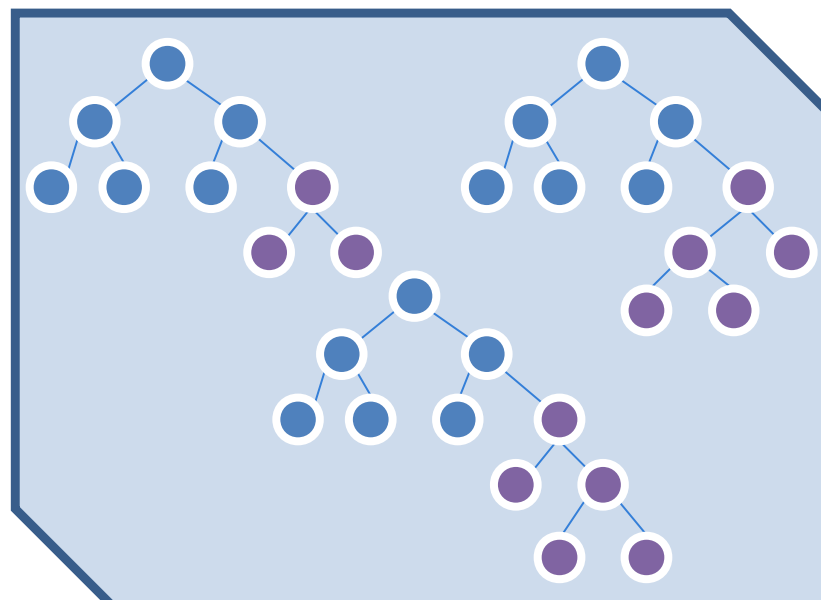
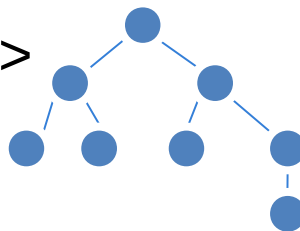




- A trivial “difference” between trees would be very false-positive prone
 - And would cause a lot of issues on each update
- **Templates** : identify areas of the tree where new branches are more likely to happen.



```
<HTML>
<HEAD>
  <TITLE></TITLE>
  <SCRIPT>JS</SCRIPT>
</HEAD>
<BODY>
  <DIV> TEST 123 </DIV>
  <DIV>
    <SCRIPT>JS</SCRIPT>
  </DIV>
</BODY>
</HTML>
```





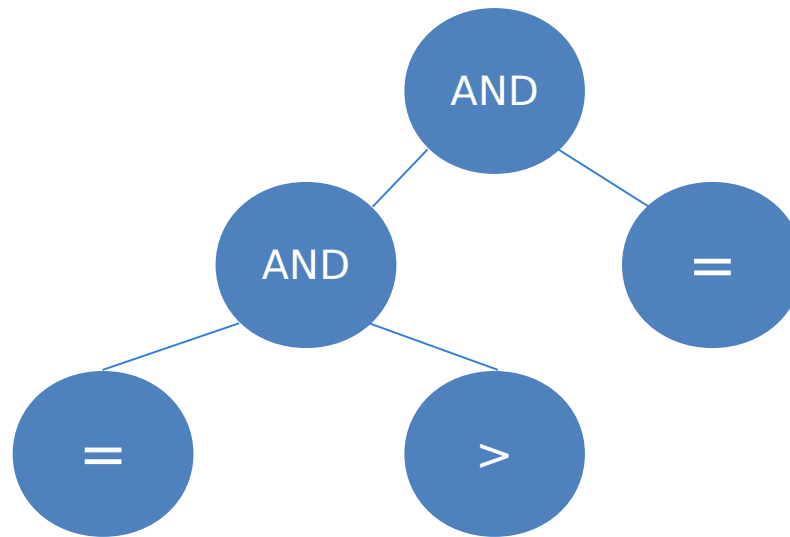
- 2 issues
 - Are we looking at the **SAME** tree the user would see?
 - We only care about **JavaScript**
- **Gecko!**
- We build the DOM tree as the browser would do it
- We can ask Gecko where the javascripts lie
 - So we only have *meaningful* branches in the trees



Oh no, more trees! SQL Anomaly

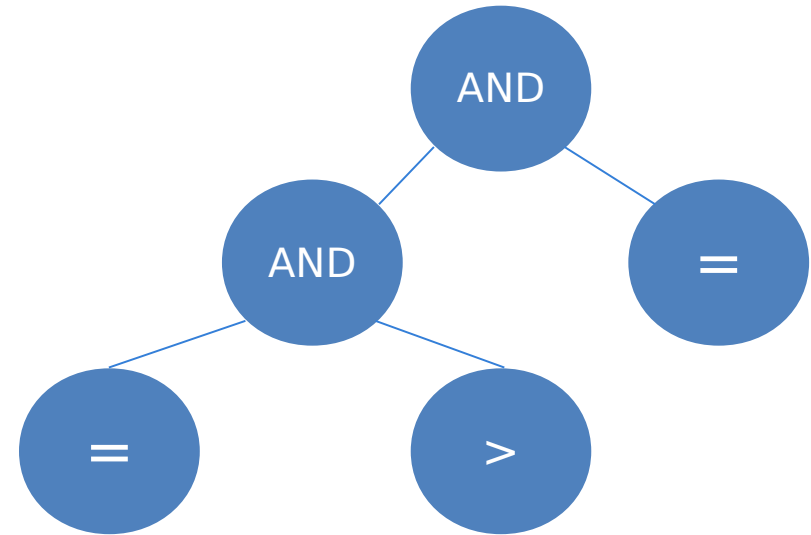
- Once we had Anomaly Tree algorithms working reliably on DOM documents, it was “easy” to port them on SQL
- Each SQL query can be represented as a tree
 - We can spot changes in the tree as we've done with the XSS Reasoner

SELECT * FROM USERS WHERE NAME = 'USER' AND
(PASSWORD = 'PASS' AND ROLE > 0)

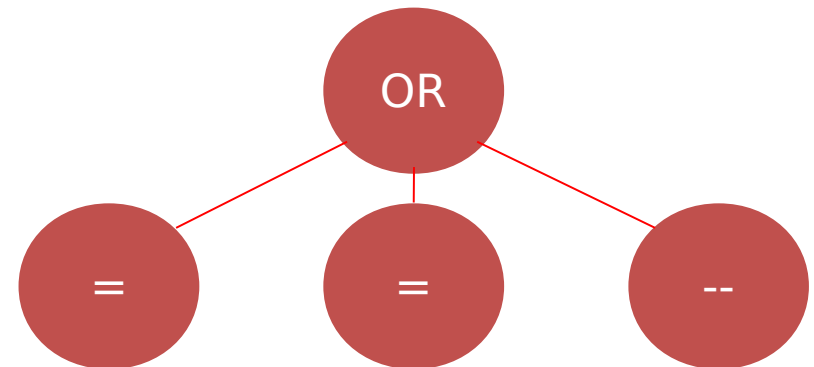




SELECT * FROM USERS
WHERE NAME = 'USER'
AND (PASSWORD = 'PASS'
AND ROLE > 0)



SELECT * FROM USERS
WHERE NAME = 'USER'
OR '1'='1' -- AND
(PASSWORD = 'PASS' AND
ROLE > 0')





- Evaluating the performance of an IDS isn't an easy task
- We tested 7 “real” applications
- A simple methodology
 - Install the application
 - Use the application “through Masibty” as normal users would do
 - Add some attacks during “learning”, either background noise like worms or real, successful attacks to the application
 - Switch to detection and repeat the tests
- Excellent (if not conclusive) results
 - 84% detection rate with a modest 0.14% false positive rate
 - Which gets to 93% DR if we take Badstore (yes, we've tested that one too) out of the pool
 - And gets to 100% DR, 0% FP if we remove the attacks from the training set...

which is what everybody else does!



How slow is it?

- Codebase is not optimized
 - No really, it's just a PoC for now, blame Claudio :-)
- In our testing environment we got an average 4-50ms delta in response times during the training phase and 1-20 ms during the detection phase
- RAM and CPU usage were usually quite low – and it was running in Eclipse!
- More testing is on its way



How can I get it? and future works



- It is going to be released for testing
 - And hopefully we'll have a paper on that sooner or later
- We're building a *working* GUI
- Next steps include
 - Supervised learning addon
 - New dedicated reasoners (JSON, Flash, Headers...)
 - Some advanced agent based stuff



Thank you!



POLITECNICO
DI MILANO

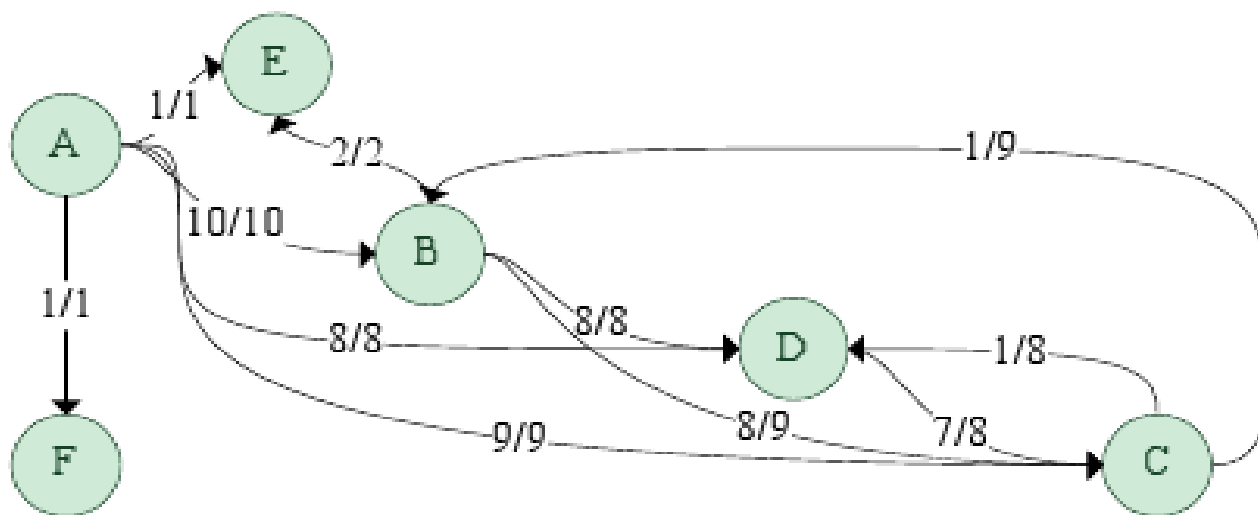


Questions!?!?

stefano.zanero@polimi.it
c.criscione@securenetwork.it



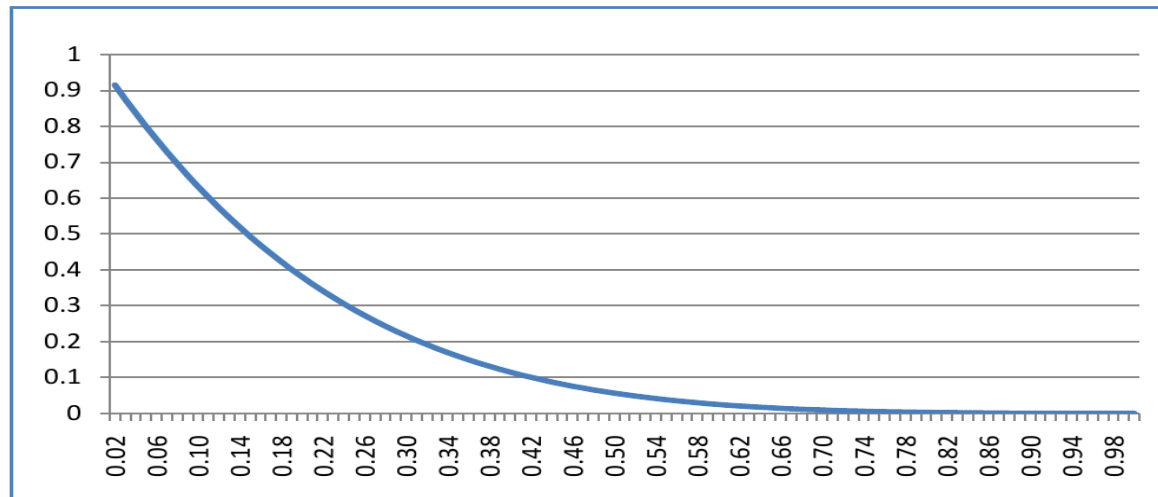
- During the learning phase, we build a profile of the order the parameters are usually presented in
 - We use the relative frequencies as our lead
- For instance, for an EP with parameters A B C D E...



- A mimicry attack is easy to conceive
- However, this is practically useful against real world attacks!



- The Numbers engine tries to pinpoint numerical only parameters
 - We find this happening quite often in real-world applications
- We use a Yule Simon distribution to filter in only the most “relevant” numbers-containing attributes
- If we put a non-numerical value in an “almost always” numerical attribute, we get an anomaly
 - The higher the number of times we observed numerical values, the higher the anomaly and the trust level





- Some parameters can only assume predefined values: they're *Tokens*
- During the learning phase we try to identify and enumerate their allowed values
 - There's a strong risk of overfitting here!
 - We use a variance based approach to identify token parameters
- This is a common approach, but it is very sensitive to the presence of attacks in the learning set
 - You end up whitelisting an attack!
- Masibty should be resilient to this kind of things... but how?



- Let's assume we have identified a token, which can have values as follows
 - V1,V2,V3,Attack1
- Attack1 will most likely have a low frequency
 - We can't just use the frequency itself!
 - What if we had V1...V100?
- We assume that all the parameters are equally likely to show up
 - We obtain the expected frequency
 - We get a ratio between expected and observed, then we use a YS distribution to highlight the most “different” distributions
- We (should) get the anomaly in bright red letters
- For this model the trust level is an evaluation of how certain we are that the parameter is actually a token



- The Length Engine measures the normal length of each parameter
- We profile each parameter during learning, measuring its minimum, maximum and average lengths and various statistical values
- Once we have enough data, it's easy to spot the anomalies
 - Think about injections in the ID field of a query



- This engine measures the relative distribution of characters in the string
 - We use “wildcards” for characters: it doesn't mind if we have a lot of As or Bs
- It can identify notable peaks in the usage of a single character
 - Think about * or / in an encoded query, NOPS in overflows and so on



- We measure whether all the parameters we're expecting are there or not
- We can identify unexpected variables
 - Think about register_globals powered attacks
- The least seen parameter is the one we use to determine the anomaly level
- Trust level is obtained leveraging a “Miss Rate” during the learning phase
- Like the previous one, this heuristic is also easy to fool
 - But still very useful against worms or automated attacks