Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries

Dario Pasquini, Marco Cianfriglia, Giuseppe Ateniese and Massimo Bernaschi

SAPIENZA
UNIVERSITÀ DI ROMA

Institute for applied Computing
"Mauro Picone" (IAC), Rome

STEVENS
INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

30TH USENIX SECURITY SYMPOSIUM

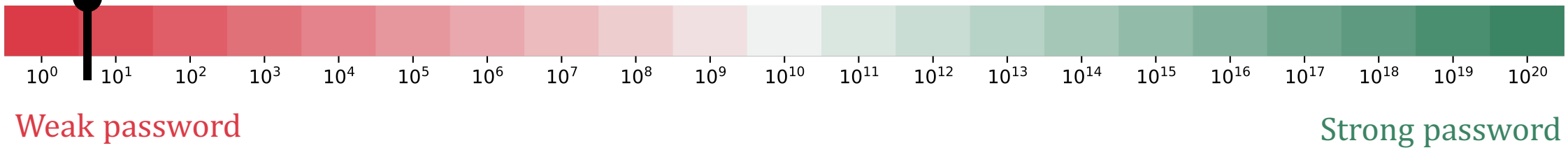
Password Security in a slide:

Adversary model

Guess number

(how many guesses A has to perform to guess it)

$$A(\text{"}pa\$\$w0rd!\text{"}) = 1123$$



Weak password

Strong password

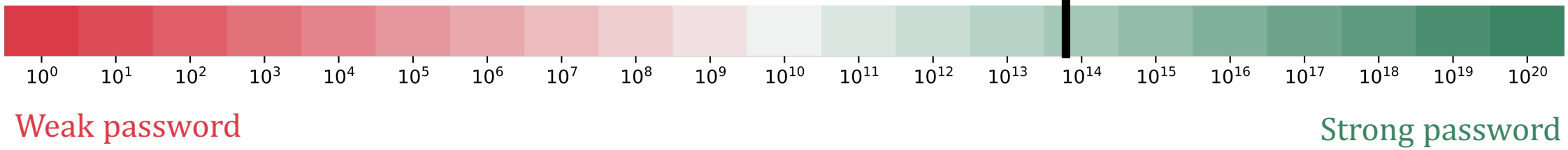
Password Security in a slide:

Adversary model

Guess number

(how many guesses ***A*** has to perform to guess it)

$$\boldsymbol{A}(\text{"}7H1d6{\sim}3.qA\text{"}) = \mathbf{10^{14}}$$

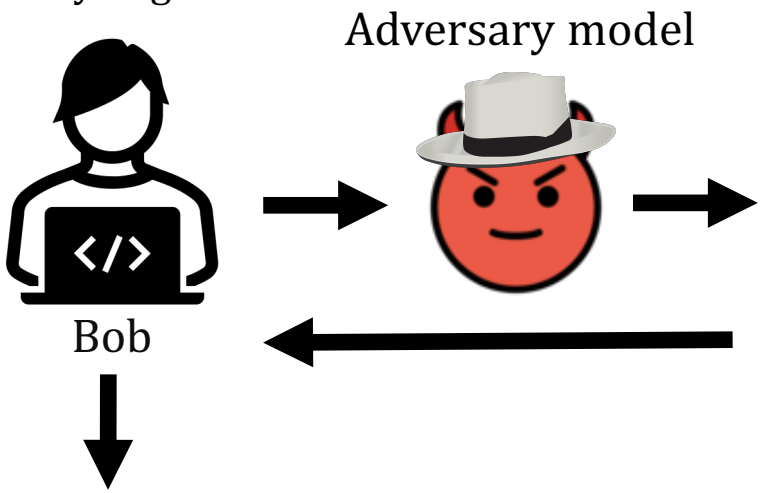


Weak password

Strong password

Why measuring password strength?

e.g., system administrator/
security engineer:



Questions Bob must answer:

- Should I make the hash function more expensive? (e.g., increasing the work–factor)
- Should I impose a different password composition policy?
- Is this new password composition policy making users choose stronger passwords?
- etc..

Real password cracker:

Reality check:

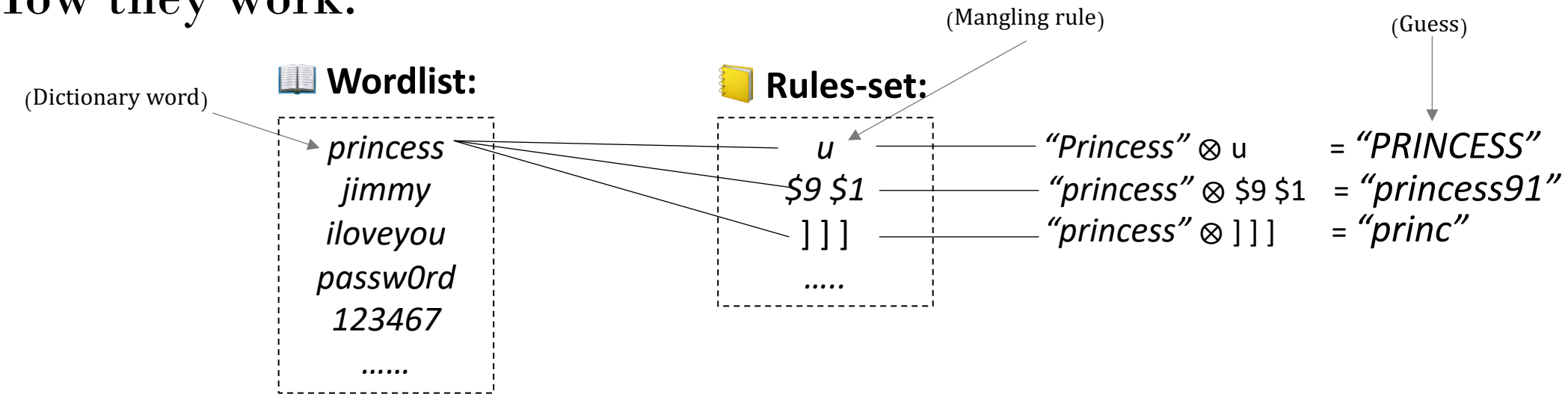
Our model of the **real-world password cracker**

The **real-world password cracker**

[1] *Measuring Real-World Accuracies and Biases in Modeling Password Guessability*, Ur et al. USENIX Sec 15

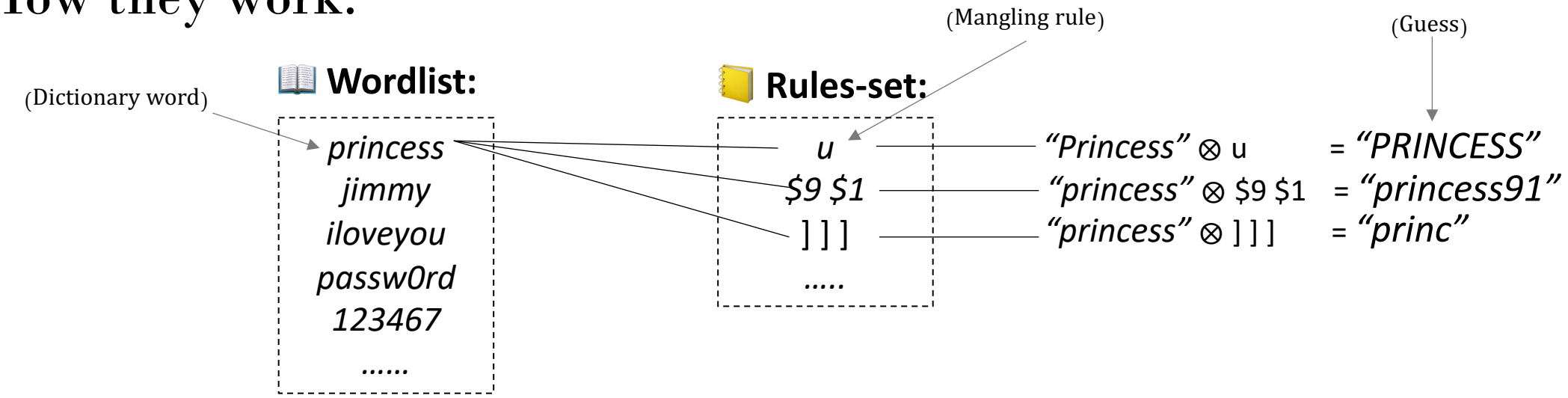
Dictionary attacks with Mangling rules:

- How they work:



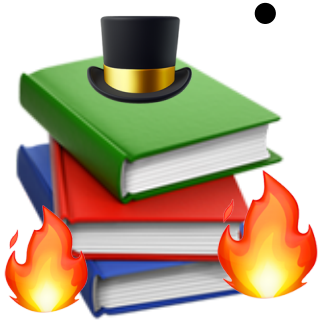
Dictionary attacks with Mangling rules:

- How they work:



- Why Dictionary attacks are so important:
  - **Password crackers use Dictionary Attacks:**
    - **Dictionary Attacks** are fast (offline attacks are also about time)
    - **Dictionary Attacks** are flexible (every target is different)

A real Dictionary Attack:

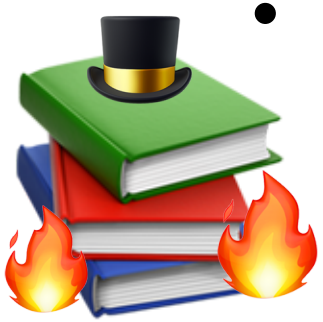


- Real-world password crackers:
  1. Use **optimized setups:**
     - Carefully chosen Word-lists
     - Carefully crafted Rules-sets

Manual processes that require a profound domain knowledge.

  2. **Manually tailor** the setup for the very **target:**
     - Collect information on the target before the attack
     - Tune in consideration of the initial passwords guessed during the attack

A real Dictionary Attack:



- Real-world password crackers:
  1. Use **optimized setups:**
     - Carefully chosen Word-lists
     - Carefully crafted Rules-sets

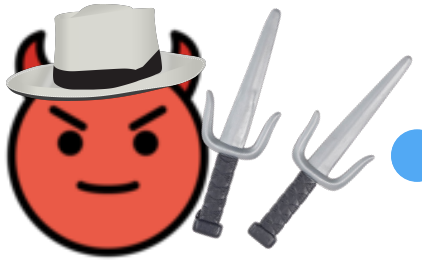
Manual processes that require a profound domain knowledge.

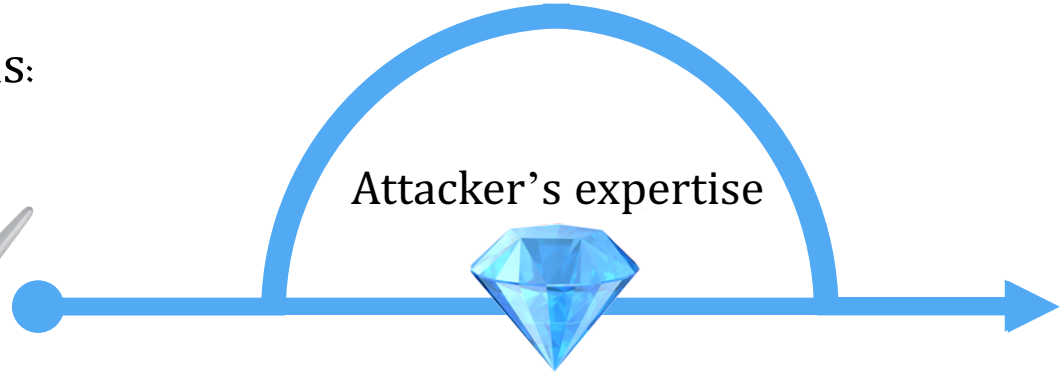
  2. **Manually tailor** the setup for the very **target:**
     - Collect information on the target before the attack
     - Tune in consideration of the initial passwords guessed during the attack

UNDERESTIMATION

Let's fill the gap

Trying to get from this:





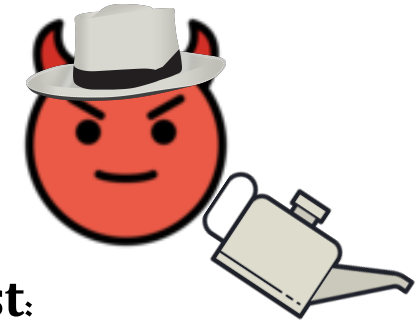
To this:

How we do it:

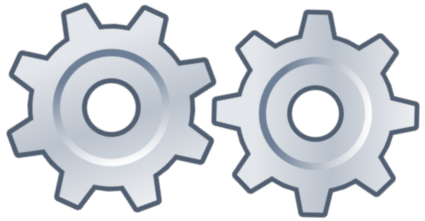
1. **Adaptive Mangling Rules**
(simulate high-end configurations)
2. **Dynamic Dictionary** (hits-recycling)
(simulates customized setups for the target)

On the functional interaction between rules-set and dictionary:

Adaptive Mangling Rules:



Wordlist:

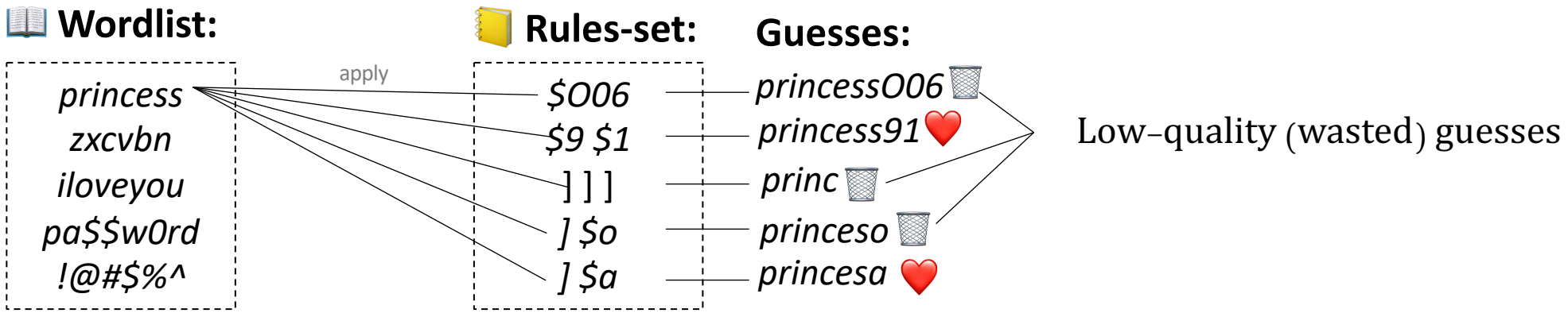


Rule-set:

The idea: It is a good attack if the **wordlist and the rule-set work well together**

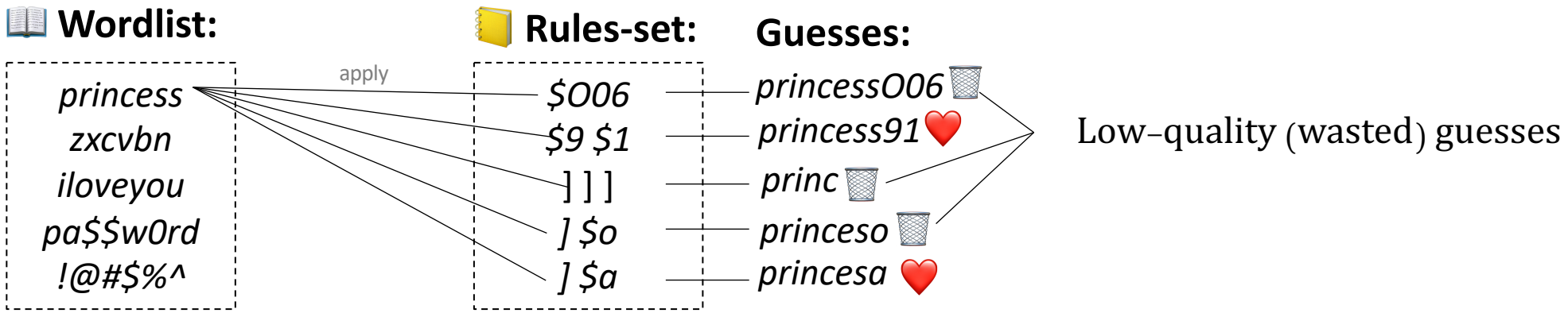
Optimizing compatibility between Wordlist and Rule-set

- Not all the rules equally interact with all the dictionary words:



Optimizing compatibility between Wordlist and Rule-set

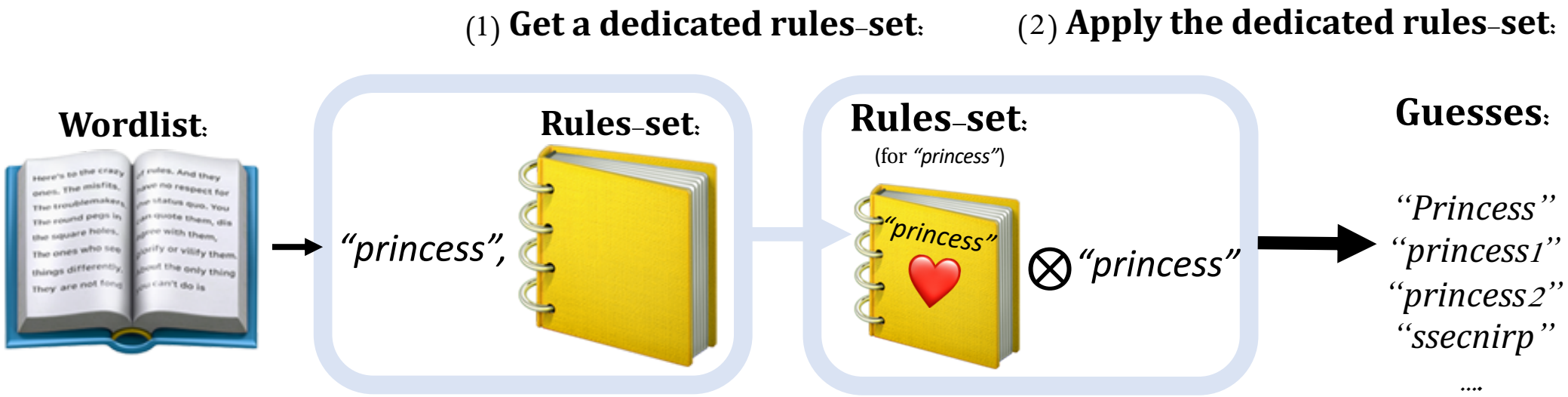
- Not all the rules equally interact with all the dictionary words:



- In unoptimized setups, most of the guesses are generated by low-compatibility pairs (rule/word).
- **Most of the generated guesses have very low-quality!**

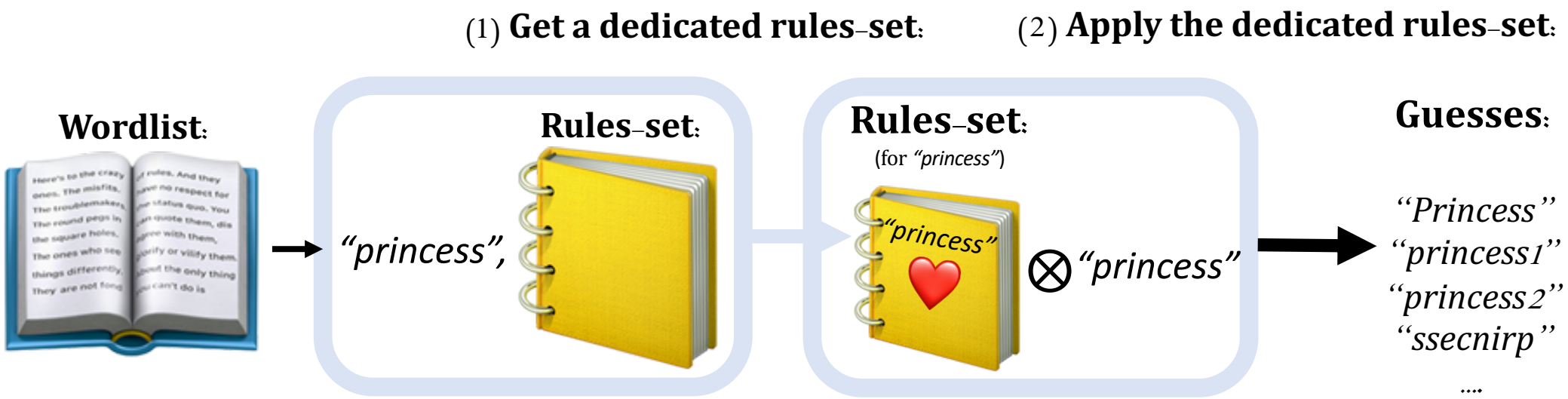
Adaptive Mangling Rules: The idea

- Every word from the dictionary gets its own rules-set:



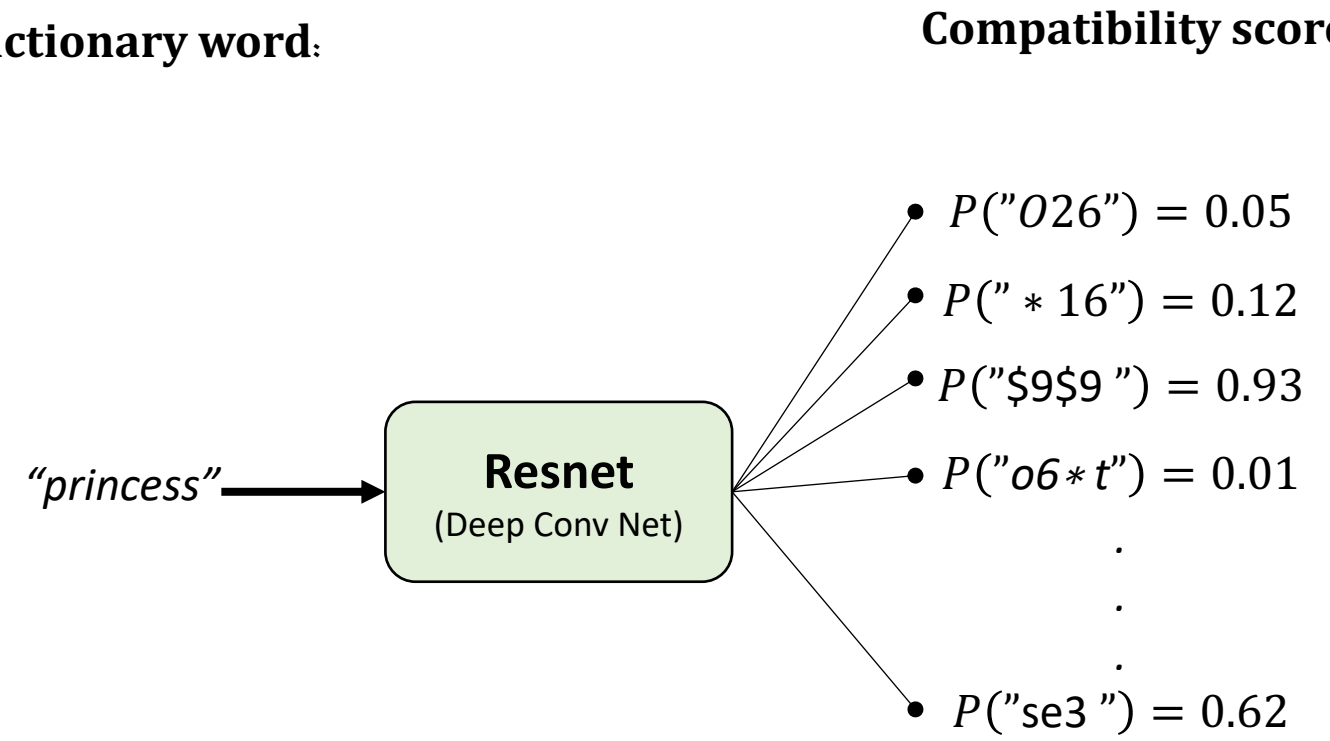
Adaptive Mangling Rules: The idea

- Every word from the dictionary gets its own rules-set:

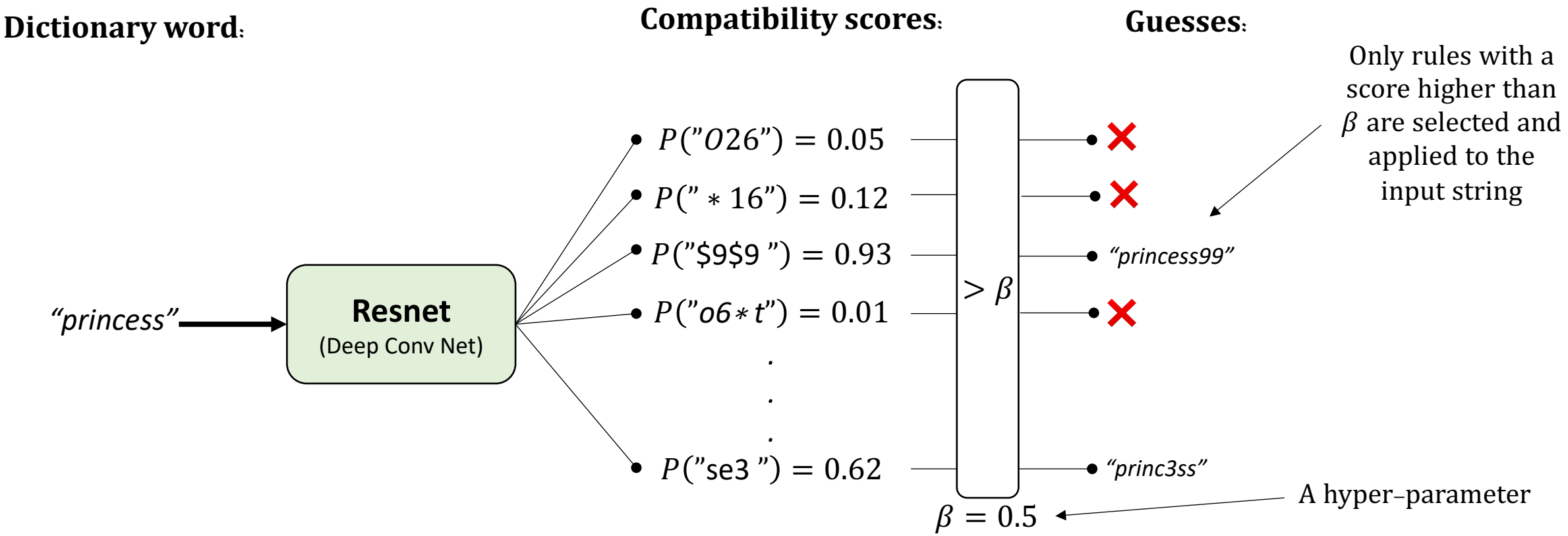


- 🏃 **It works at run-time!** (we'll see why later)

- Let's model rule/word compatibility with a **Deep Neural Network**
  - It's a multi-label classification task (each mangling rule is a class)
  - A rule-set is fixed (e.g., *generated2.rule*) and the network outputs a "*compatibility score*" for each rule in the set with a single inference.



- Let's model rule/word compatibility with a **Deep Neural Network**
  - It's a multi-label classification task (each mangling rule is a class)
  - A rule-set is fixed (e.g., *generated2.rule*) and the network outputs a *"compatibility score"* for each rule in the set with a single inference.

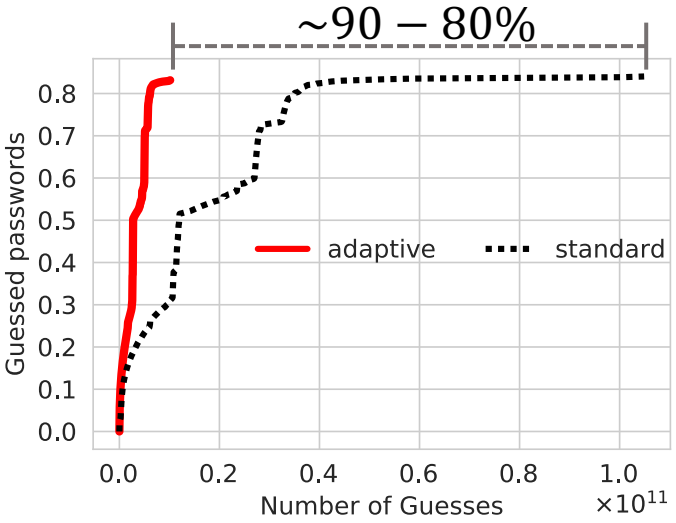


Is it Effective? An example:

Setup:

- **Rule engine**: *HashCat*
- **Wordlist**: *MyHeritage.com*
- **Rules-set**: *InsidePro-PasswordsPro* (3120 rules)
- **Target**: *animoto.com*

Performance gain:



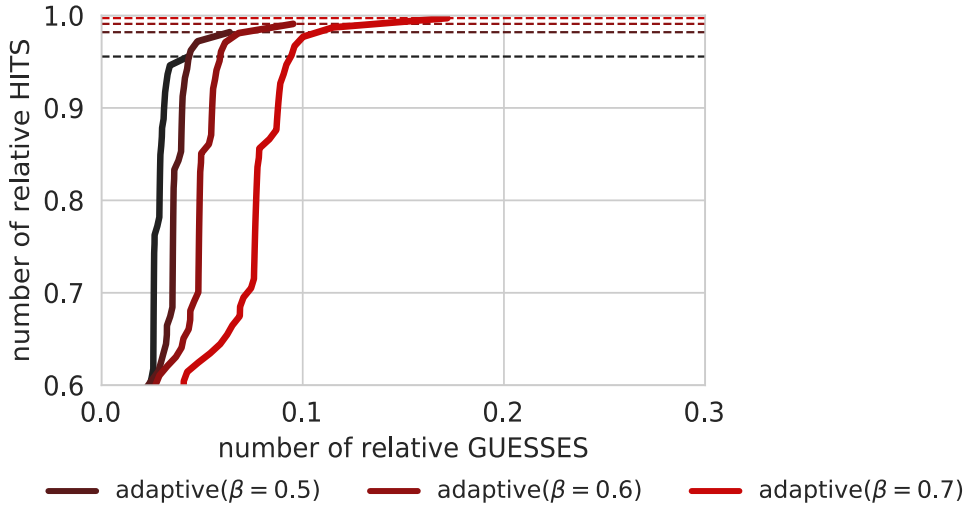
- It's also fast:

Numbers of compatibility scores per second (*On a NVIDIA V100*):

generated2.rule	*generated.rule*	*PasswordPro.rule*
#rules: 64528	#rules: 14424	#rules: 3120
1.528.346.738 c/s	**611.793.885 c/s**	**164.854.041 c/s**

Intrinsic parallel + It can run on tensor cores
(totally idle during the hash computation on GPU).

The effect of β:

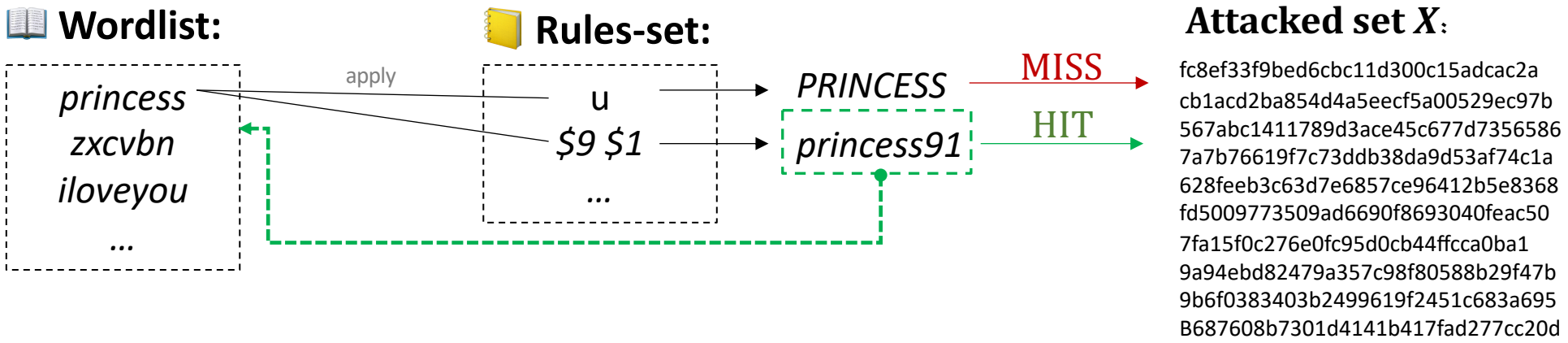


♻ Hits-Recycling: Simple, yet effective

- Passwords created under the same environment (e.g., webapp) are not independent from each other.
- Therefore, the passwords you guessed during the attack can easily bring you to guess new related passwords that occur in the target.

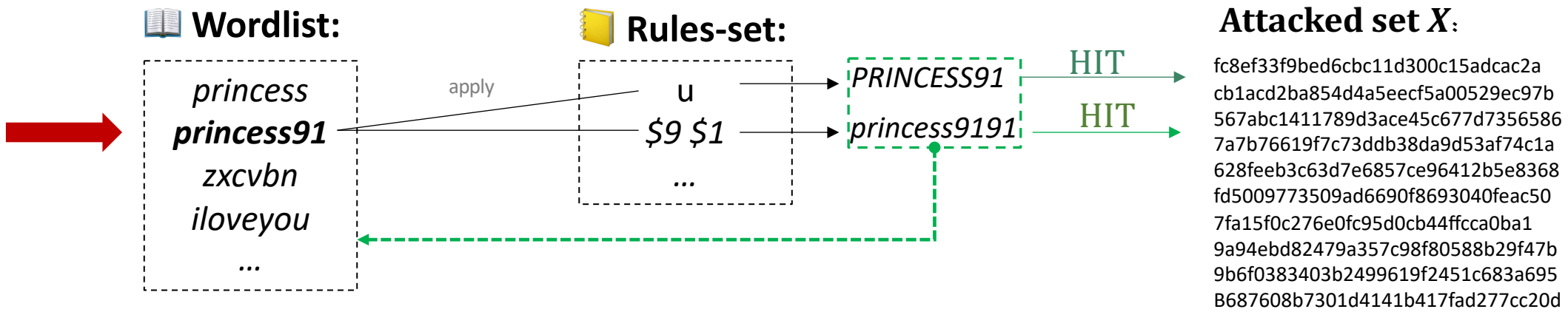
Hits-Recycling: Simple, yet effective

- Passwords created under the same environment (e.g., webapp) are not independent from each other.
- Therefore, the passwords you guessed during the attack can easily bring you to guess new related passwords that occur in the target.
- **Hits-Recycling: The wordlist is dynamically augmented with the passwords guessed during the running attack.**

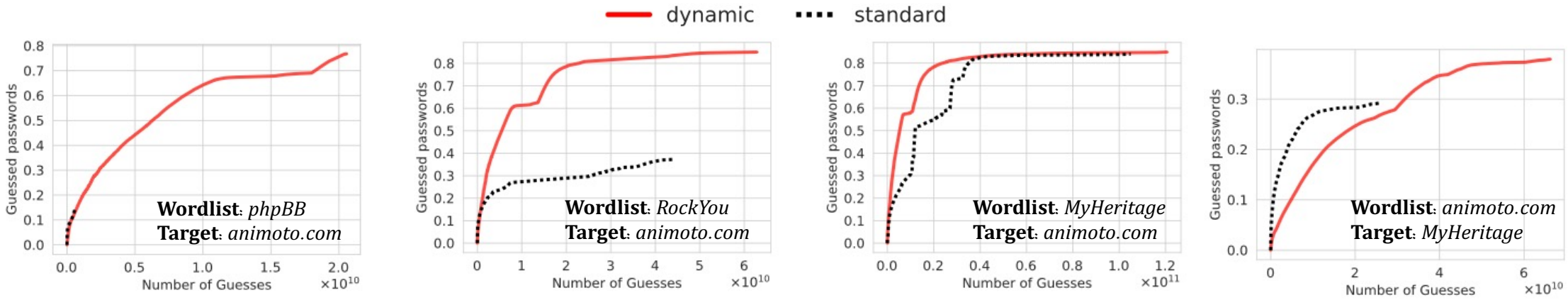


Hits-Recycling: Simple, yet effective

- Passwords created under the same environment (e.g., webapp) are not independent from each other.
- Therefore, the passwords you guessed during the attack can easily bring you to guess new related passwords that occur in the target.
- **Hits-Recycling: The wordlist is dynamically augmented with the passwords guessed during the running attack.**



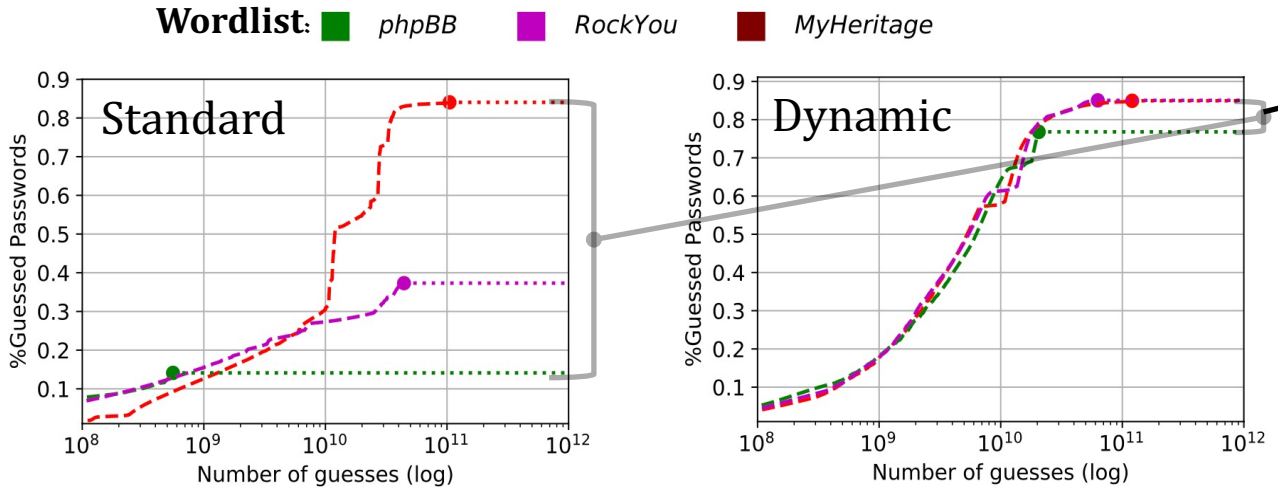
Hits-Recycling: the Effect



Setup:

- **Rule engine:** *HashCat*
- **Rules-set:** *InsidePro-PasswordsPro* (3120 rules)

Hits-Recycling: the Effect

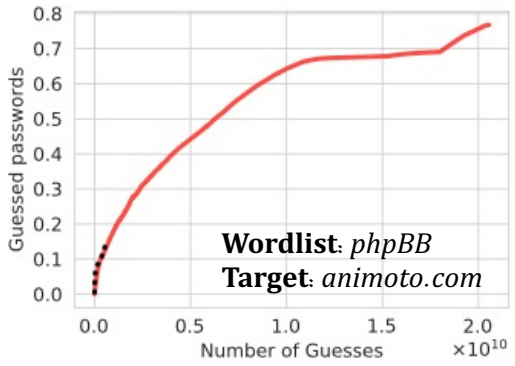


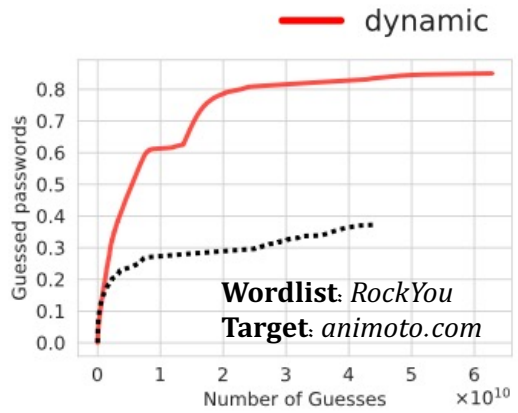
Reduces performance variance.

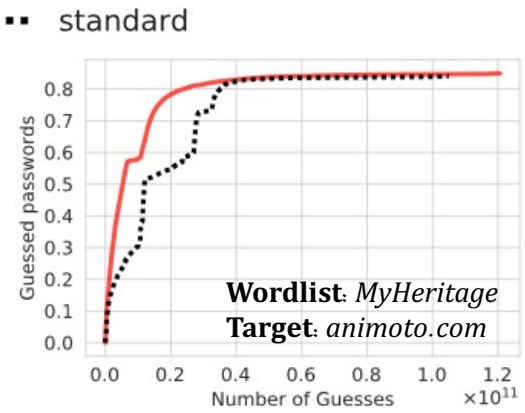
Setup:

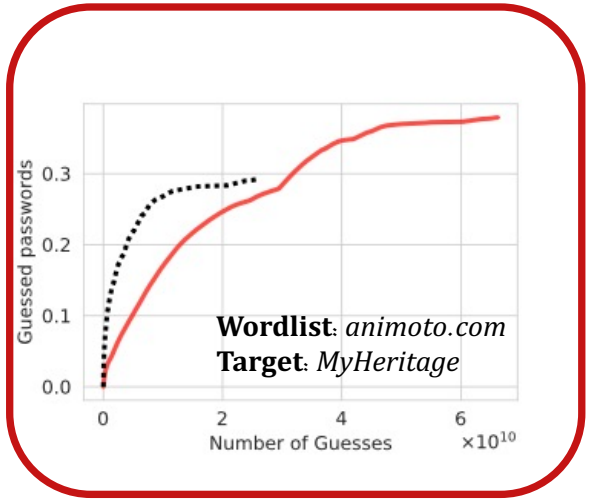
- **Rule engine:** *HashCat*
- **Rules-set:** *InsidePro-PasswordsPro* (3120 rules)

Hits-Recycling: the Effect









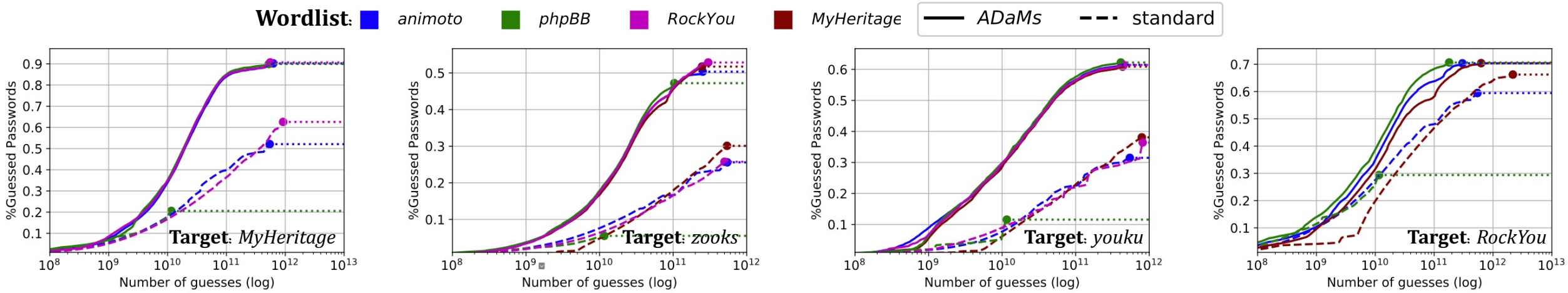
Setup:

- **Rule engine**: *HashCat*
- **Rules-set**: *InsidePro-PasswordsPro* (3120 rules)

Fixing Hits-Recycling with the **Adaptive rules**

- Hits–recycling + Adaptive Rules = **Adaptive, Dynamic Mangling rules** (***ADaMs***).
- Adaptive mangling rules offers a run–time optimization for the **dynamic created dictionary**.

***ADaMs* vs Standard** attack with same configuration (rules–set/dictionary):

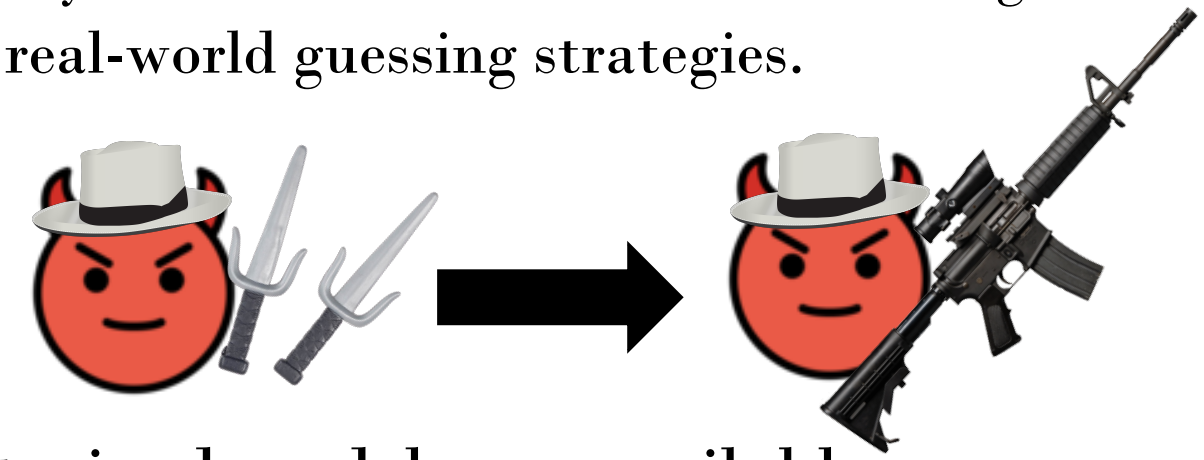


Rules–set: *generated2* (64528 rules)

Conclusion:

The proposed techniques:

- Make dictionary attacks more resilient to misconfigurations.
- Get closer to real-world guessing strategies.



- The code and trained models are available.

Use it instead of plain dictionary attacks in our security analysis!

https://github.com/TheAdamProject/adams

Thanks for your Attention!

For any questions: pasquini@di.uniroma1.it

We are working on a full–fledged (GPU–based) implementation with additional modules:

Adam, The First Cracker