10. Authentication Part 2

Blase Ur and David Cash
February 2nd, 2022
CMSC 23200 / 33250

THE UNIVERSITY OF CHICAGO

Credential Breaches

Some Breached Companies

LinkedIn

Adobe

SONY®

Domino's Pizza

GAWKER

000webhost.com
better than paid hosting

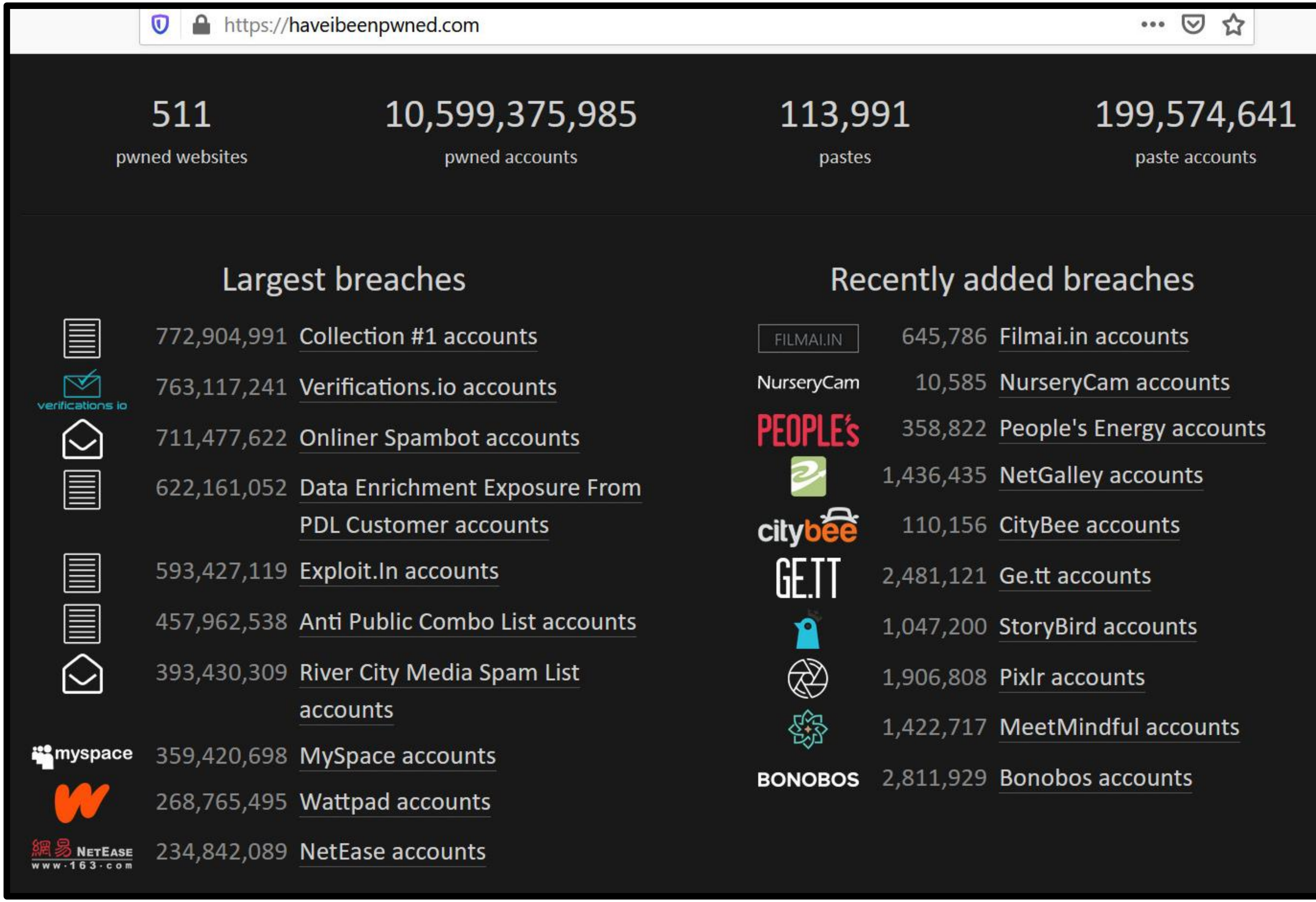
YAHOO!®

STRATFOR
GLOBAL INTELLIGENCE

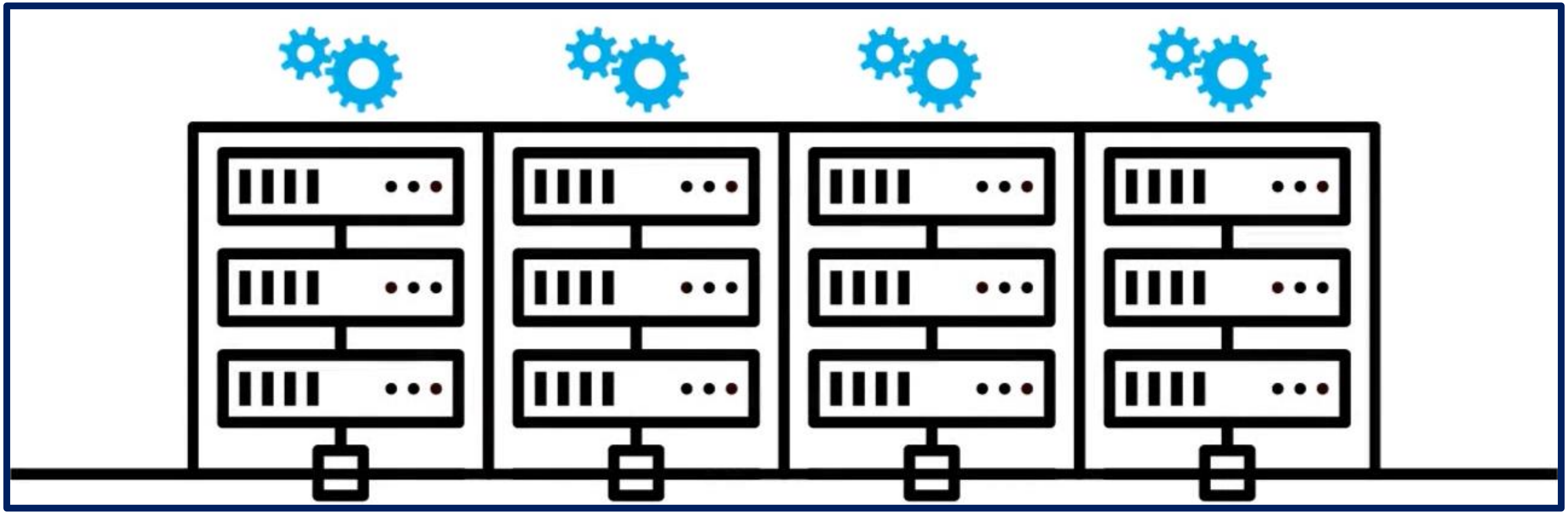
Data-Driven Statistical Attacks

- (2009) 32 million passwords:
- (2016) 117 million passwords:
- (2017) 3 <u>billion</u> passwords:
- Total: >10 billion passwords stolen from >500 services

Have I Been Pwned (HIBP)



Password Cracking



Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, Richard Shay. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In *Proc. USENIX Security Symposium*, 2015.

Statistical Metrics For Passwords

- Traditionally: Shannon entropy
- Recently: α-guesswork
- Disadvantages of statistical approaches
  - Entropy does not consider human tendencies
  - Usually no per-password estimates
  - Huge sample required
  - Not real-world attacks

Parameterized Guessability

- How many guesses a particular cracking algorithm with particular training data would take to guess a password

Attack Model

80d561388725fa74f2d03cd16e1d687c

Attack Model

80d561388725fa74f2d03cd16e1d687c

1. h(“123456”) = e10adc3949ba59abbe56e057f20f883e

Attack Model

80d561388725fa74f2d03cd16e1d687c

1. h("123456") = e10adc3949ba59abbe56e057f20f883e
2. h("password") = 5f4dcc3b5aa765d61d8327deb882cf99

Attack Model

80d561388725fa74f2d03cd16e1d687c

1. h("123456") = e10adc3949ba59abbe56e057f20f883e
2. h("password") = 5f4dcc3b5aa765d61d8327deb882cf99
3. h("monkey") = d0763edaa9d9bd2a9516280e9044d885

Attack Model

80d561388725fa74f2d03cd16e1d687c

1. h(“123456”) = e10adc3949ba59abbe56e057f20f883e
2. h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99
3. h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885
4. h(“letmein”) = 0d107d09f5bbe40cade3de5c71e9e9b7

Attack Model

80d561388725fa74f2d03cd16e1d687c

1. h(“123456”) = e10adc3949ba59abbe56e057f20f883e
2. h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99
3. h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885
4. h(“letmein”) = 0d107d09f5bbe40cade3de5c71e9e9b7
5. h(“p@ssw0rd”) = 0f359740bd1cda994f8b55330c86d845

Attack Model

80d561388725fa74f2d03cd16e1d687c

1. h(“123456”) = e10adc3949ba59abbe56e057f20f883e
2. h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99
3. h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885
4. h(“letmein”) = 0d107d09f5bbe40cade3de5c71e9e9b7
5. h(“p@ssw0rd”) = 0f359740bd1cda994f8b55330c86d845
6. h(“Chic4go”) = **80d561388725fa74f2d03cd16e1d687c**

`Chic4go`

Guess # 6

```
j@mesb0nd007!
```

Guess # 366,163,847,194

```
n(c$JZX!zKc^bIAX^N
```

Guess # past cutoff

Broad Cracking Approaches

- Brute force (or selective brute force)
- Wordlist
- Mangled wordlist
  - Hashcat and John the Ripper
- Markov models
- Probabilistic Context-Free Grammar
- Neural networks
- In practice: manual, iterative updates

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1
Password1

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1
Password1
Chicago1

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Guesses

Super1
Password1
Chicago1
Super
P4ssword
Chic4go

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1
Password1
Chicago1
Super
P4ssword
Chic4go
super
password
chicago

Example Wordlists and Rulelists

Wordlist

PGS (≈ 20,000,000)

Linkedin (≈ 60,000,000)

HIBP (≈ 500,000,000)

Example Wordlists and Rulelists

Wordlist

PGS (≈ 20,000,000)

Linkedin (≈ 60,000,000)

HIBP (≈ 500,000,000)

Rulelist

Korelogic (≈ 5,000)

Megatron (≈ 15,000)

Generated2 (≈ 65,000)

Example Wordlists and Rulelists

Wordlist	Rulelist
PGS (≈ 20,000,000)	Korelogic (≈ 5,000)
Linkedin (≈ 60,000,000)	Megatron (≈ 15,000)
HIBP (≈ 500,000,000)	Generated2 (≈ 65,000)

→ 10^9 - 10^{15} guesses

Example Wordlists and Rulelists

Wordlist	Rulelist
PGS (≈ 20,000,000)	Korelogic (≈ 5,000)
Linkedin (≈ 60,000,000)	Megatron (≈ 15,000)
HIBP (≈ 500,000,000)	Generated2 (≈ 65,000)

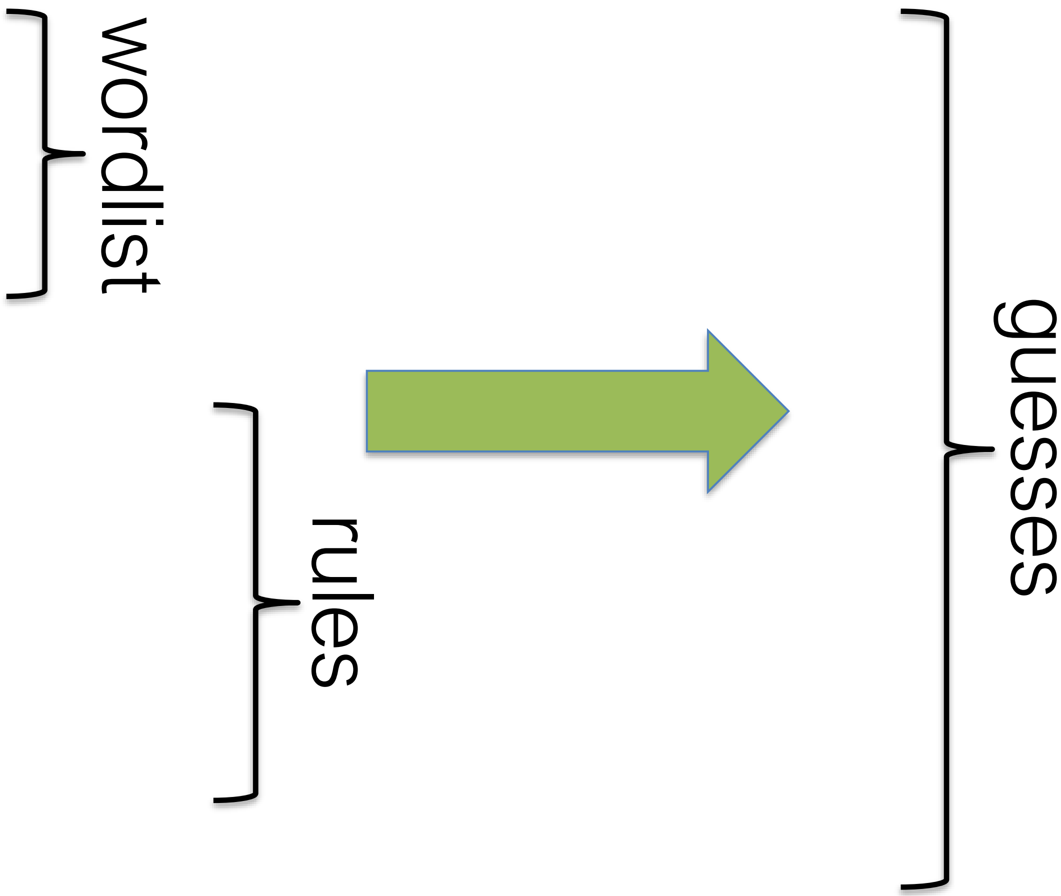
→ 10^9 - 10^{15} guesses

+ Hackers' private word/rule lists

John the Ripper

- Wordlist mode requires:
  - Wordlist (passwords and dictionary entries)
  - Mangling rules
- Guesses variants of input wordlist
- Speed: Fast
- “JTR”

John the Ripper



John the Ripper

usenix

security

wordlist

rules

guesses

John the Ripper

usenix

security

wordlist

[ ]

[add 1 at end]

[change e to 3]

rules

guesses

John the Ripper

usenix
security

wordlist

[ ]
[add 1 at end]
[change e to 3]

rules

usenix
security
usenix1
security1
us3nix
s3curity

guesses

John the Ripper

usenix

security

wordlist

[ ]

[add 1 at end]

[change e to 3]

rules

usenix

security

usenix1

security1

us3nix

s3curity

guesses

John the Ripper

usenix
security

wordlist

[ ]
[add 1 at end]
[change e to 3]

rules

usenix
security
usenix1
security1
us3nix
s3curity

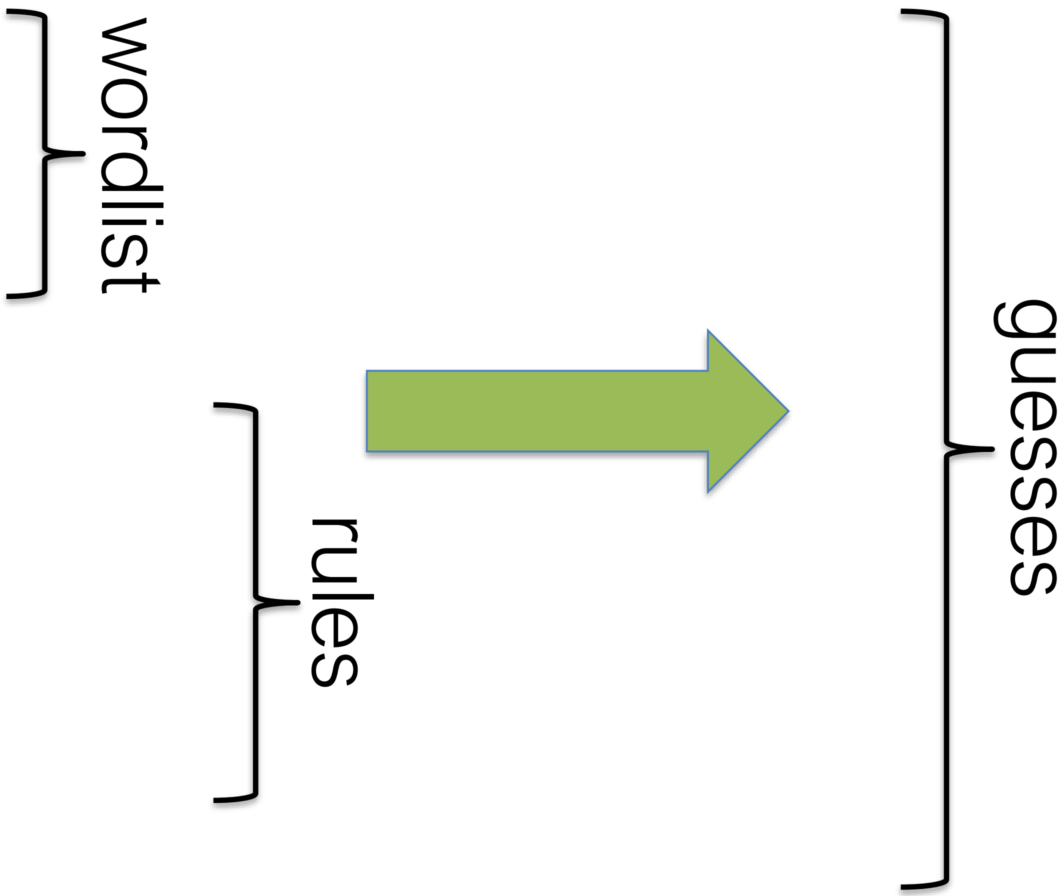
guesses

Hashcat

- Wordlist mode requires:
  - Wordlist (passwords and dictionary entries)
  - Mangling rules
- Guesses variants of input wordlist
- Speed: Fast

Hashcat

hashcat
advanced
password
recovery



Hashcat

hashcat
advanced
password
recovery

usenix

security

wordlist

[ ]

[add 1 at end]

[change e to 3]

rules

guesses

Hashcat

hashcat
advanced
password
recovery

usenix
security

wordlist

[ ]
[add 1 at end]
[change e to 3]

rules

usenix
usenix1
us3nix
security
security1
s3curity

guesses

Hashcat

hashcat
advanced
password
recovery

usenix
security

wordlist

[ ]
[add 1 at end]
[change e to 3]

rules

usenix
usenix1
us3nix
security
security1
s3curity

guesses

Hashcat: Rule Language

Name	Function	Description	Example Rule	Input Word	Output Word	Note
Nothing	:	do nothing	:	p@ssW0rd	p@ssW0rd	
Lowercase	l	Lowercase all letters	l	p@ssW0rd	p@ssw0rd	
Uppercase	u	Uppercase all letters	u	p@ssW0rd	P@SSW0RD	
Capitalize	c	Capitalize the first letter and lower the rest	c	p@ssW0rd	P@ssw0rd	
Invert Capitalize	C	Lowercase first found character, uppercase the rest	C	p@ssW0rd	p@SSW0RD	
Toggle Case	t	Toggle the case of all characters in word.	t	p@ssW0rd	P@SSw0RD	
Toggle @	TN	Toggle the case of characters at position N	T3	p@ssW0rd	p@sSW0rd	*
Reverse	r	Reverse the entire word	r	p@ssW0rd	dr0Wss@p	
Duplicate	d	Duplicate entire word	d	p@ssW0rd	p@ssW0rdp@ssW0rd	
Duplicate N	pN	Append duplicated word N times	p2	p@ssW0rd	p@ssW0rdp@ssW0rdp@ssW0rd	
Reflect	f	Duplicate word reversed	f	p@ssW0rd	p@ssW0rddr0Wss@p	
Rotate Left	{	Rotates the word left.	{	p@ssW0rd	@ssW0rdp	
Rotate Right	}	Rotates the word right	}	p@ssW0rd	dp@ssW0r	
Append Character	$X	Append character X to end	$1	p@ssW0rd	p@ssW0rd1	
Prepend Character	^X	Prepend character X to front	^1	p@ssW0rd	1p@ssW0rd	
Truncate left	[	Deletes first character	[	p@ssW0rd	@ssW0rd	
Trucate right	]	Deletes last character	]	p@ssW0rd	p@assW0r	
Delete @ N	DN	Deletes character at position N	D3	p@ssW0rd	p@sW0rd	*
Extract range	xNM	Extracts M characters, starting at position N	x04	p@ssW0rd	p@ss	* #
Omit range	ONM	Deletes M characters, starting at position N	O12	p@ssW0rd	psW0rd	*
Insert @ N	iNX	Inserts character X at position N	i4!	p@ssW0rd	p@ss!W0rd	*
Overwrite @ N	oNX	Overwrites character at position N with X	o3$	p@ssW0rd	p@s$W0rd	*
Truncate @ N	'N	Truncate word at position N	'6	p@ssW0rd	p@ssW0	*
Replace	sXY	Replace all instances of X with Y	ss$	p@ssW0rd	p@$$W0rd	
Purge	@X	Purge all instances of X	@s	p@ssW0rd	p@W0rd	+

Name	Function	Description	Example Rule	Note
Reject less	<N	Reject plains if their length is greater than N	<G	*
Reject greater	>N	Reject plains if their length is less or equal to N	>8	*
Reject equal	_N	Reject plains of length not equal to N	_7	*
Reject contain	!X	Reject plains which contain char X	!z	
Reject not contain	/X	Reject plains which do not contain char X	/e	
Reject equal first	(X	Reject plains which do not start with X	(h	
Reject equal last	)X	Reject plains which do not end with X	)t	
Reject equal at	=NX	Reject plains which do not have char X at position N	=1a	*
Reject contains	%NX	Reject plains which contain char X less than N times	%2a	*
Reject contains	Q	Reject plains where the memory saved matches current word	rMrQ	e.g. for palindrome

Name	Function	Description	Example Rule	Input Word	Output Word	Note
Swap front	k	Swaps first two characters	k	p@ssW0rd	@pssW0rd	
Swap back	K	Swaps last two characters	K	p@ssW0rd	p@ssW0dr	
Swap @ N	*NM	Swaps character at position N with character at position M	*34	p@ssW0rd	p@sWs0rd	*
Bitwise shift left	LN	Bitwise shift left character @ N	L2	p@ssW0rd	p@æsW0rd	*
Bitwise shift right	RN	Bitwise shift right character @ N	R2	p@ssW0rd	p@9sW0rd	*
Ascii increment	+N	Increment character @ N by 1 ascii value	+2	p@ssW0rd	p@tsW0rd	*
Ascii decrement	-N	Decrement character @ N by 1 ascii value	-1	p@ssW0rd	p?ssW0rd	*
Replace N + 1	.N	Replaces character @ N with value at @ N plus 1	.1	p@ssW0rd	psssW0rd	*
Replace N - 1	,N	Replaces character @ N with value at @ N minus 1	,1	p@ssW0rd	ppssW0rd	*
Duplicate block front	yN	Duplicates first N characters	y2	p@ssW0rd	p@p@ssW0rd	*
Duplicate block back	YN	Duplicates last N characters	Y2	p@ssW0rd	p@ssW0rdrd	*
Title	E	Lower case the whole line, then upper case the first letter and every letter after a space	E	p@ssW0rd w0rld	P@ssw0rd W0rld	+
Title w/separator	eX	Lower case the whole line, then upper case the first letter and every letter after a custom separator character	e-	p@ssW0rd-w0rld	P@ssw0rd-W0rld	+

Hashcat: Rule Language

*05 O03 d '7

Switch the first and the sixth char;

Delete the first three chars;

Duplicate the whole word;

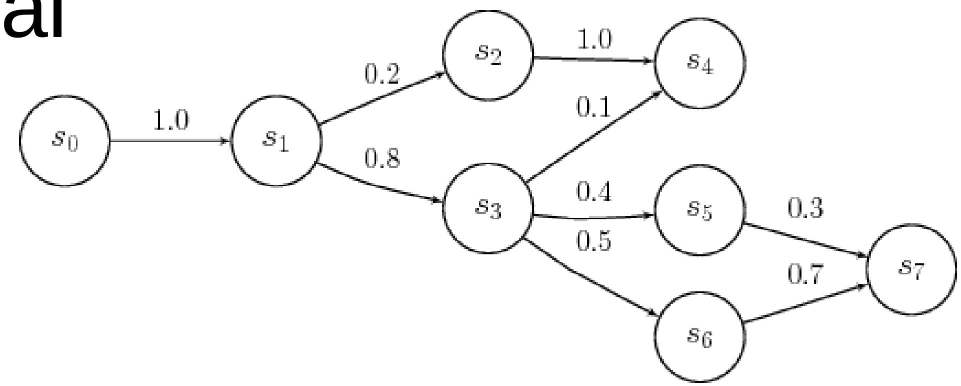
Truncate the word to length 7;

Hashcat (Other Modes)

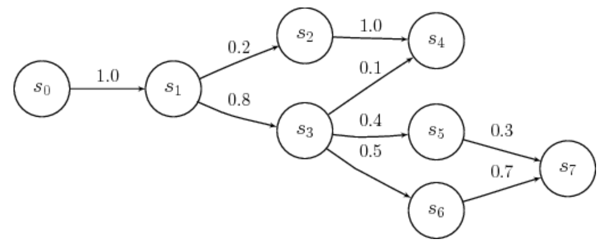
- Mask attack (brute force within a specified character-class structure)
- Combinator attacks
- Hybrid attacks
- Many more!

Markov Models

- Predicts future characters from previous
- Approach requires weighted data:
  - Passwords
  - Dictionaries
- Speed: Slow
- Smoothing is critical



Markov Models




```
chic4gooo
```

2-gram model (1 character of context):

[start] ➔ **c** (1.0)

4 ➔ **g** (1.0)

c ➔ **h** (0.5), **4** (0.5)

g ➔ **o** (1.0)

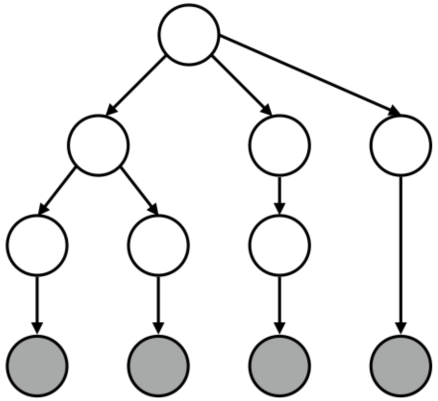
h ➔ **i** (1.0)

i ➔ **c** (1.0)

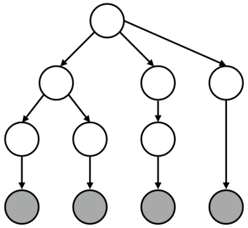
o ➔ **o** (0.67) **[end]** (0.33)

Probabilistic Context-Free Grammar

- Generate password grammar
  - Structures
  - Terminals
- OG: Weir et al. IEEE S&P 2009
- Speed: Slow
- “PCFG”



PCFG



passwordpassword

password123

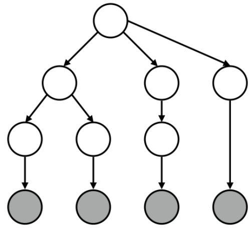
usenix3

5ecurity

iloveyou

nirvanaa123

PCFG



passwordpassword

password123

usenix3

5ecurity

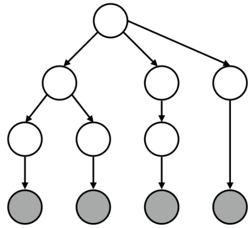
iloveyou

nirvanaa123

Structure Model:

$\mathbf{L_{16}}$	(1/6)
$\mathbf{L_8D_3}$	(2/6)
$\mathbf{L_6D_1}$	(1/6)
$\mathbf{D_1L_7}$	(1/6)
$\mathbf{L_8}$	(1/6)

PCFG



passwordpassword

password123

usenix3

5ecurity

iloveyou

nirvanaa123

Digit Model:

D_1 ➔ **3** (0.5) **5** (0.5)

D_3 ➔ **123** (1.0)

Repeat for letters, etc.

Professionals (“Pros”)

- Proprietary wordlists and configurations
  - 10^{14} guesses
  - Manually tuned, updated
- For example: KoreLogic
  - Password audits for Fortune 500 companies
  - Run DEF CON “Crack Me If You Can”

KoreLogic SECURITY

Approach

4 password sets ✖ 5 approaches

```
password
iloveyou
teamo123
…
```

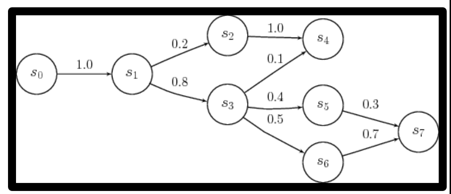
```
passwordpassword
1234567812345678
!1@2#3$4%5^6&7*8
…
```

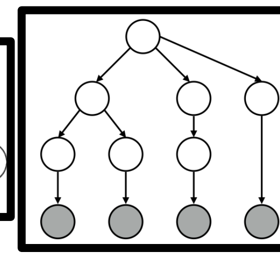
```
Pa$$w0rd
iLov3you!
1QaZ2W@x
…
```

```
pa$$word1234
12345678asDF
!q1q!q1q!q1q
…
```

John the Ripper

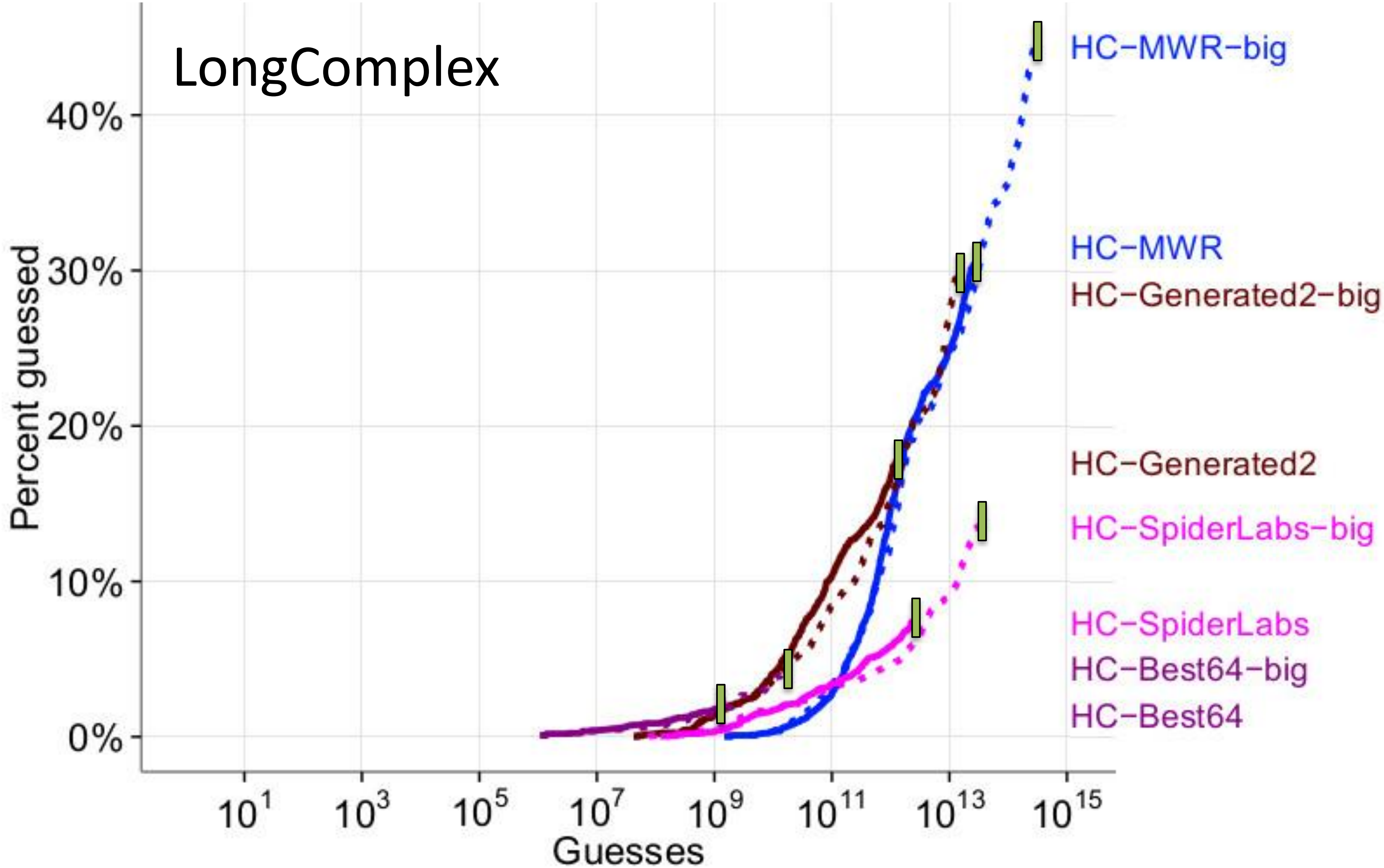
hashcat
advanced
password
recovery



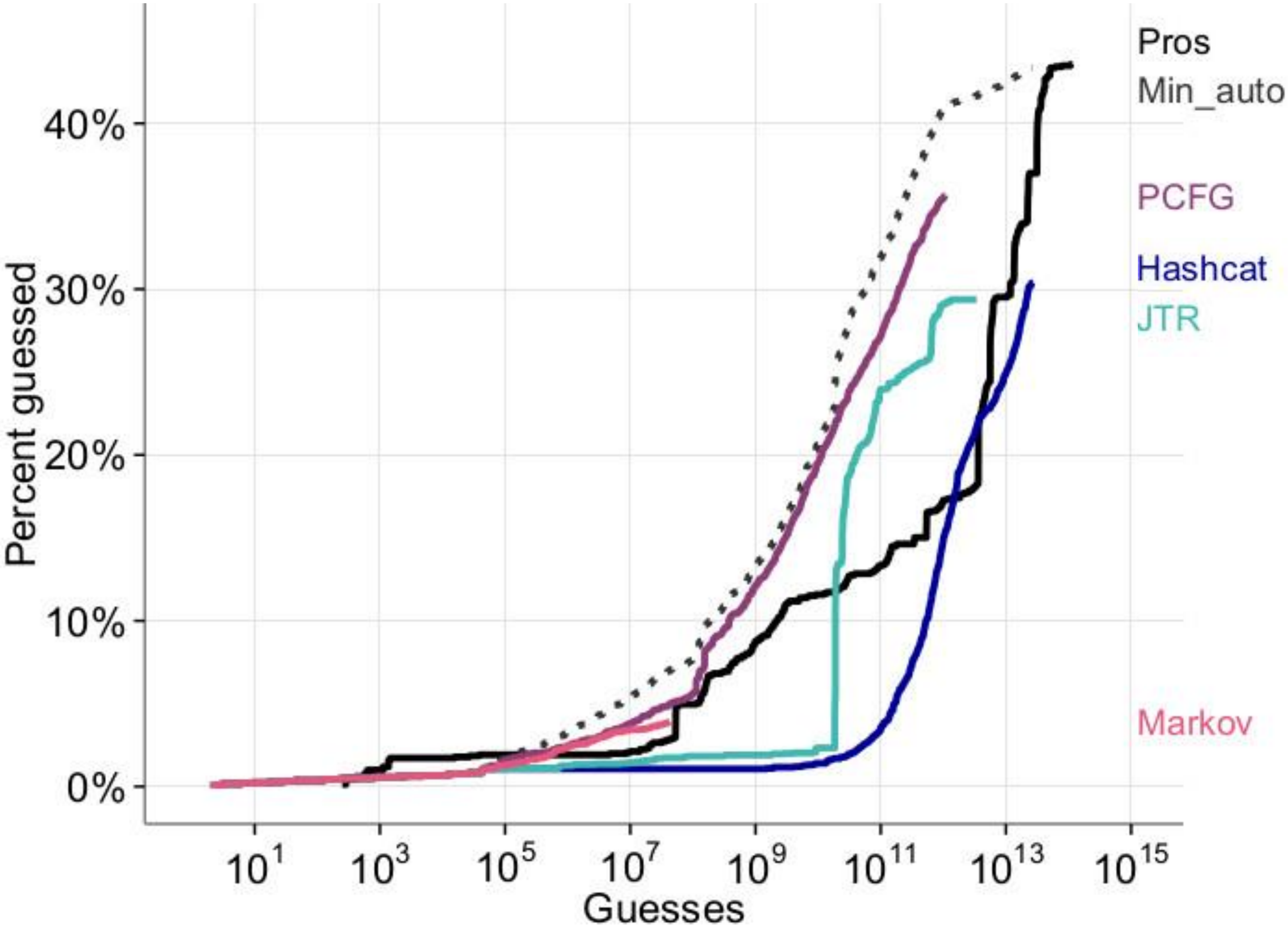


KoreLogic
SECURITY

Configuration Is Crucial



Comparison for Complex Passwords



Per-Password Highly Impacted

P@ssw0rd!

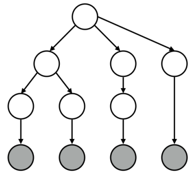
Per-Password Highly Impacted

- JTR guess # 801

P@ssw0rd!

Per-Password Highly Impacted

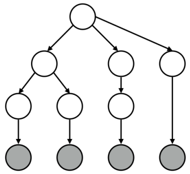
- JTR guess # 801
- Not guessed in 10^{14} PCFG guesses



`P@ssw0rd!`

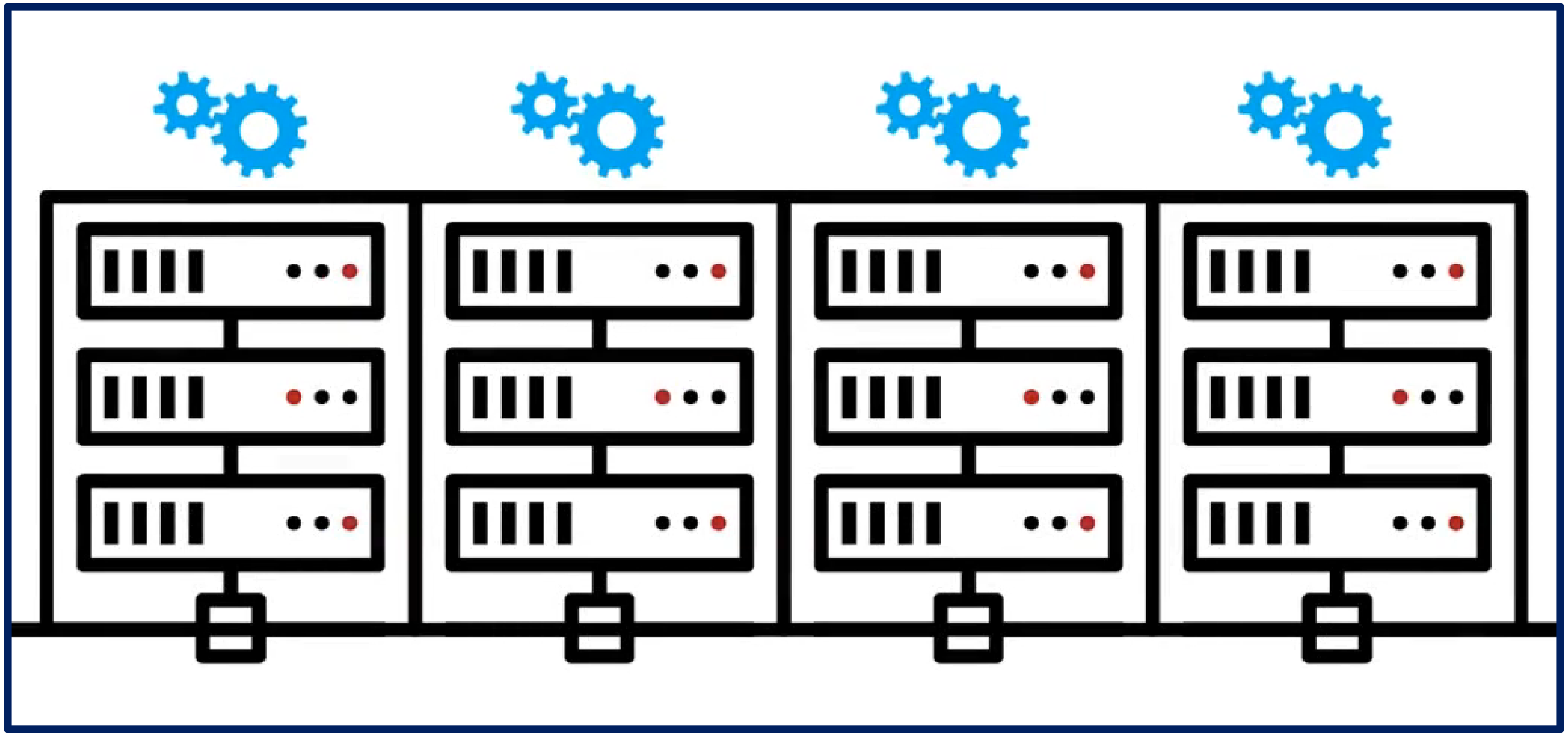
Per-Password Highly Impacted

- JTR guess # 801
- Not guessed in 10^{14} PCFG guesses



P@ssw0rd!

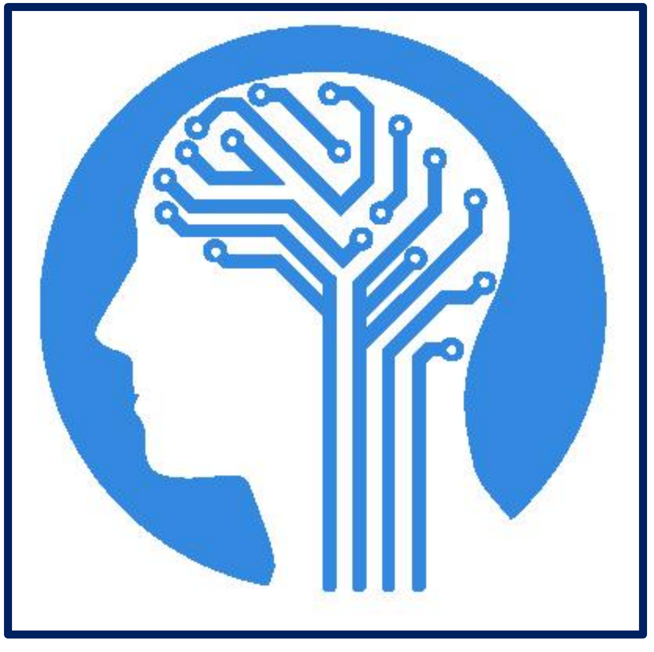
Neural Networks For Passwords



William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In *Proc. USENIX Security Symposium*, 2016.

Better Password Scoring

- Real-time feedback
- Runs entirely client-side
- Accurately models password guessability



Recurrent Neural Networks (RNNs)

LSTM Architecture

Generating Passwords

Generating Passwords

`passw` → o or maybe 0 or O or …

Generating Passwords

`passw` →

Next char is:

A: 3%

B: 1%

C: 0.6%

...

O: 55%

...

Z: 0.01%

0: 20%

1: ...

Generating Passwords

“”

Prob: 100%

→

Next char is:

A:	3%
B:	2%
C:	5%
…	
O:	2%
…	
Z:	0.2%
0:	1%
1:	…
END:	2%

Generating Passwords

""

Prob: 100%

Next char is:

A:	3%
B:	2%
C:	5%
…	
O:	2%
…	
Z:	0.2%
0:	1%
1:	…
END:	2%

Generating Passwords

"C"

Prob: 5%

Generating Passwords

"C"

Prob: 5%

Next char is:

A:	10%
B:	1%
C:	4%
…	
O:	8%
…	
Z:	0.02%
0:	3%
1:	…
END:	6%

Generating Passwords

"C"

Prob: 5%

→

Next char is:

A:	10%
B:	1%
C:	4%
…	
O:	8%
…	
Z:	0.02%
0:	3%
1:	…
END:	6%

Generating Passwords

“CA”

Prob: 0.5%

→

Next char is:

A:	3%
B:	10%
C:	7%
…	
O:	1%
…	
Z:	0.03%
0:	2%
1:	…
END:	12%

Generating Passwords

"CAB"

Prob: 0.05%

Next char is:

A:	3%
B:	10%
C:	7%
…	
O:	1%
…	
Z:	0.03%
0:	2%
1:	…
END:	3%

Generating Passwords

"CAB"

Prob: 0.05%

Next char is:

A:	4%
B:	3%
C:	1%
…	
O:	2%
…	
Z:	0.01%
0:	4%
1:	…
END:	12%

Generating Passwords

"CAB"
Prob: 0.05%

Next char is:

A:	4%
B:	3%
C:	1%
…	
O:	2%
…	
Z:	0.01%
0:	4%
1:	…
END:	12%

Generating Passwords

"CAB"

Prob: 0.006%

Descending Probability Order

`CAB` - 0.006%

`CAC` - 0.0042%

`ADD1` - 0.002%

`CODE` - 0.0013%

...

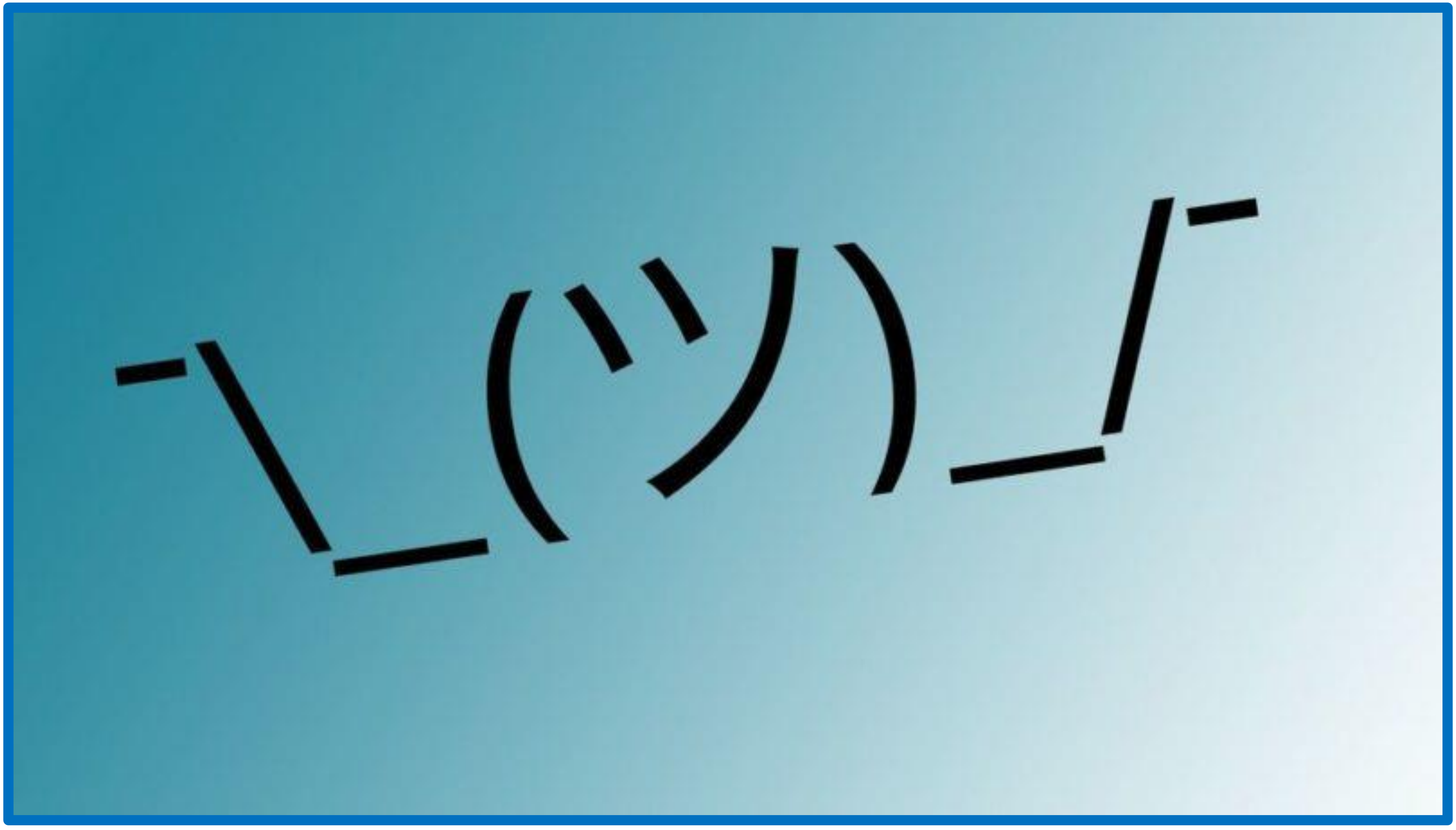
Design Space

- Model size: 3mb (browser) vs. 60mb (GPU)
- Transference learning
  - Novel password-composition policies
- Training data
  - Natural language
- (Many others)

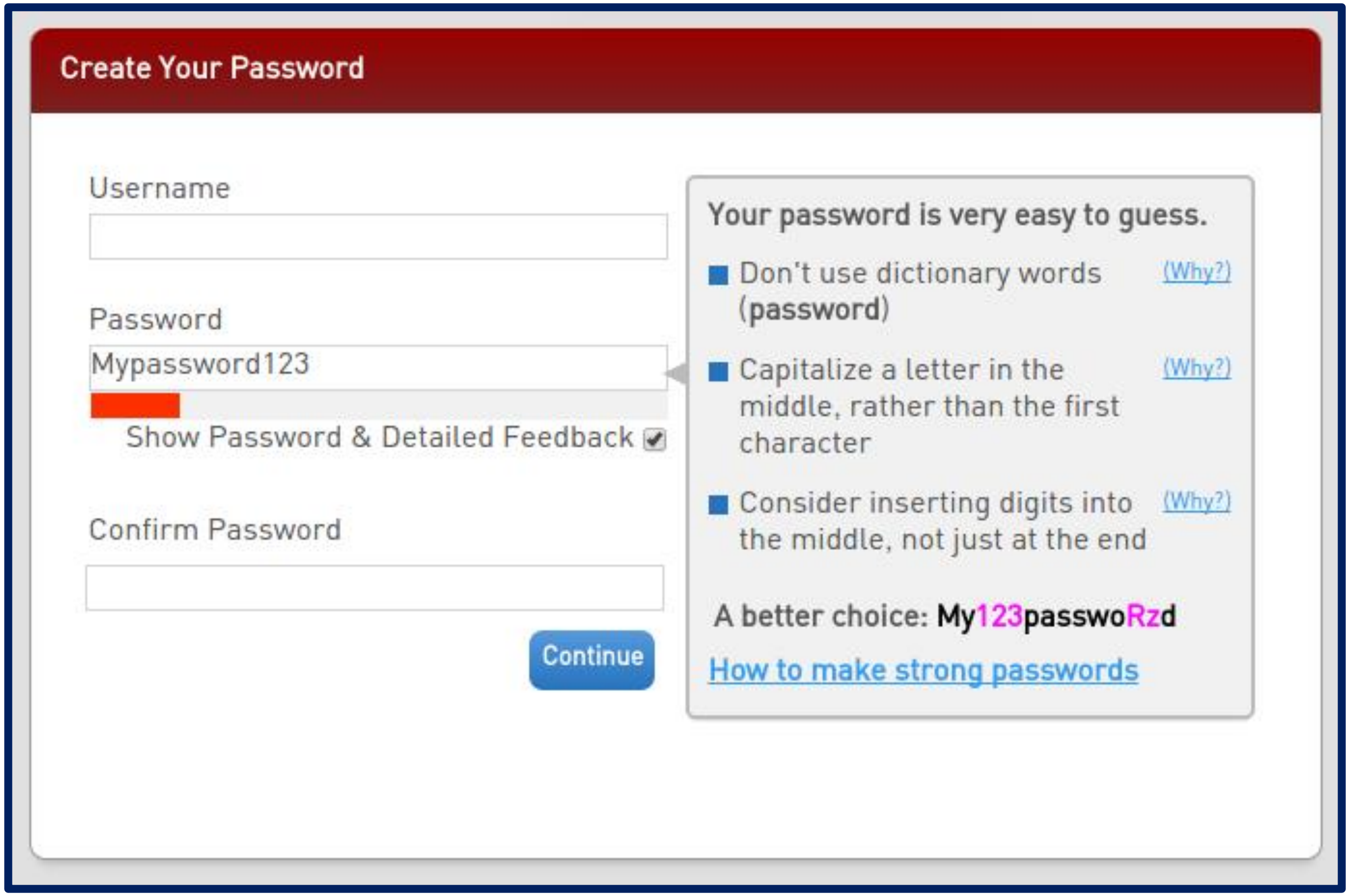
Key Results

- Neural networks produce better guesses than previous methods
- Larger model not a major advantage
- Browser implementation in Javascript

Intelligibility (Explanations)



Building a Data-Driven Meter



Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, William Melicher. Development and Evaluation of a Data-Driven Password Meter. In *Proc. CHI*, 2017.

We designed & tested a meter with:

1) Principled strength estimates
2) Data-driven feedback to users

We designed & tested a meter with:

1) Principled strength estimates (RNN)

2) Data-driven feedback to users

We designed & tested a meter with:

1) Principled strength estimates

2) Data-driven feedback to users

Provide Intelligible Explanations

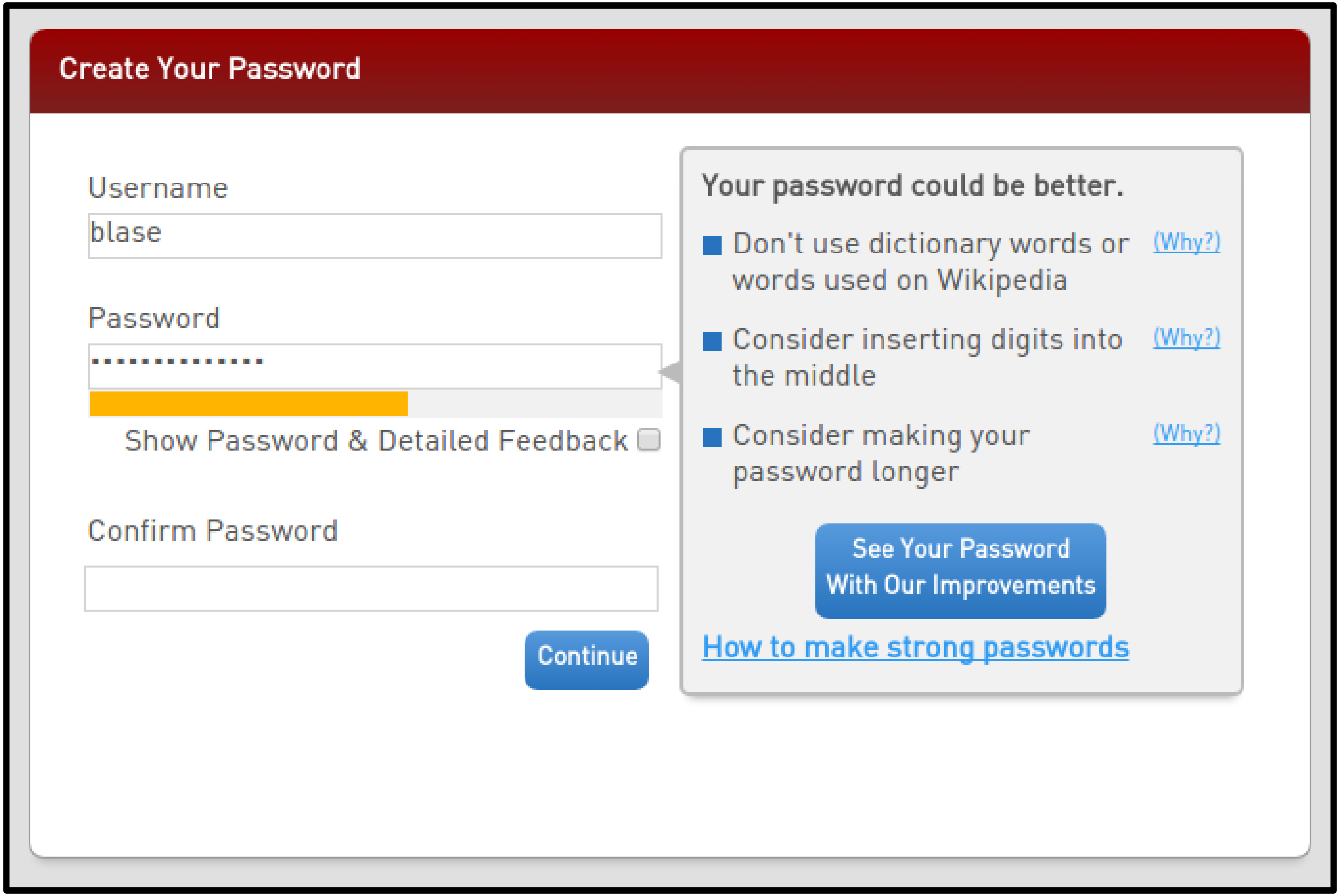
```
Unic0rns
```

Don't use simple transformations of words or phrases (**unicorns → Unic0rns**)

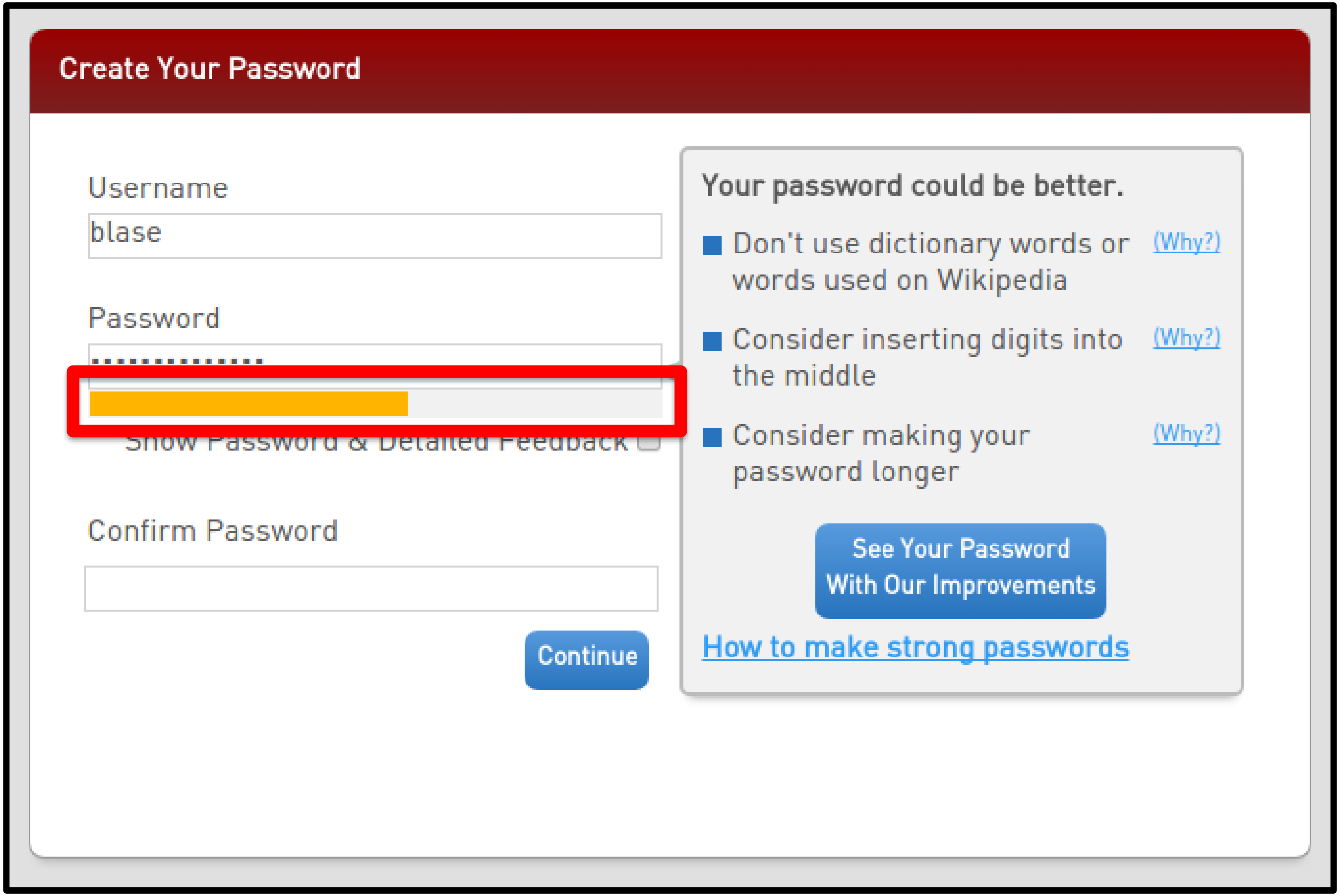
Capitalize a letter in the middle, rather than the first character

- 21 characteristics
- Weightings determined with regression

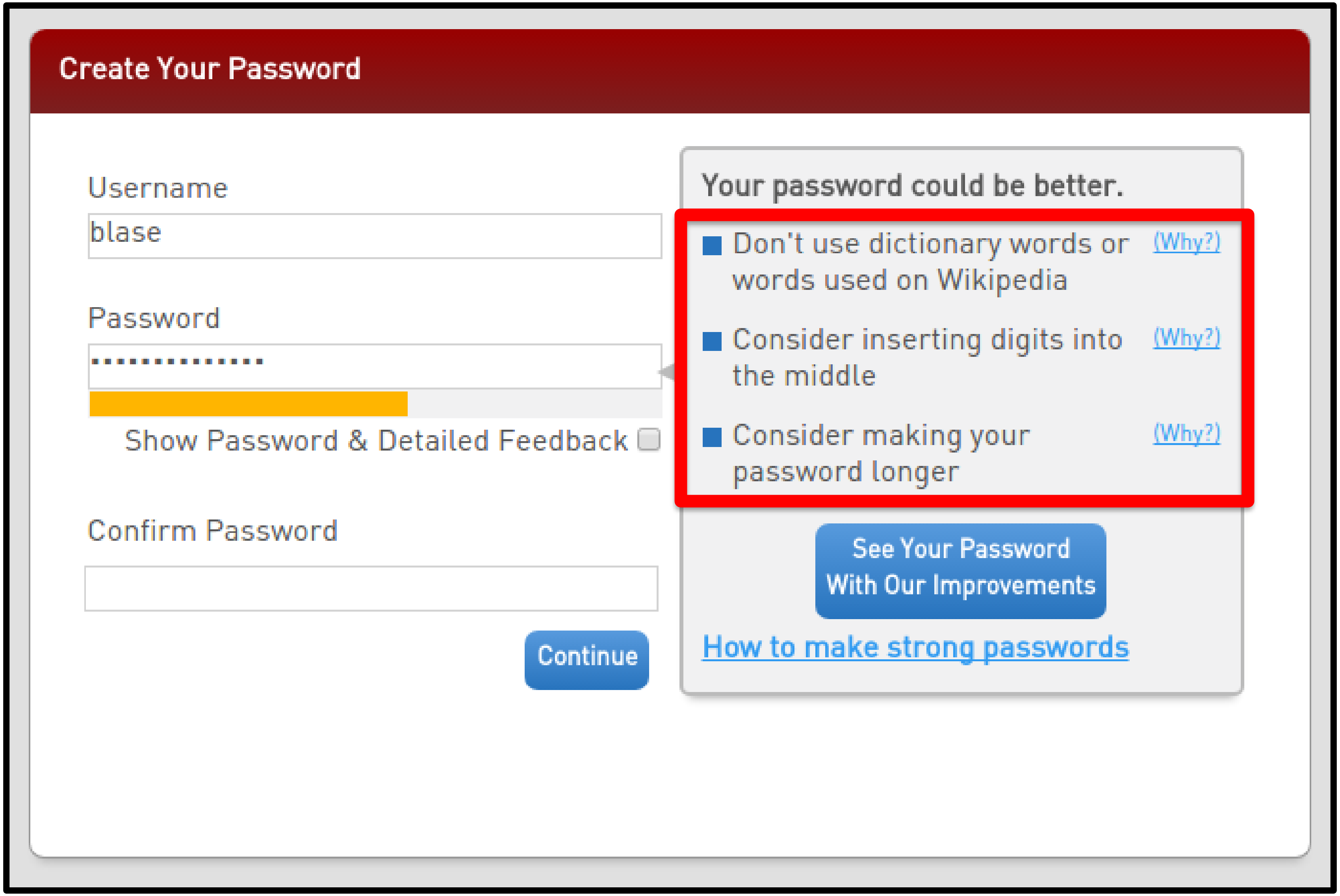
After Requirements Are Met…



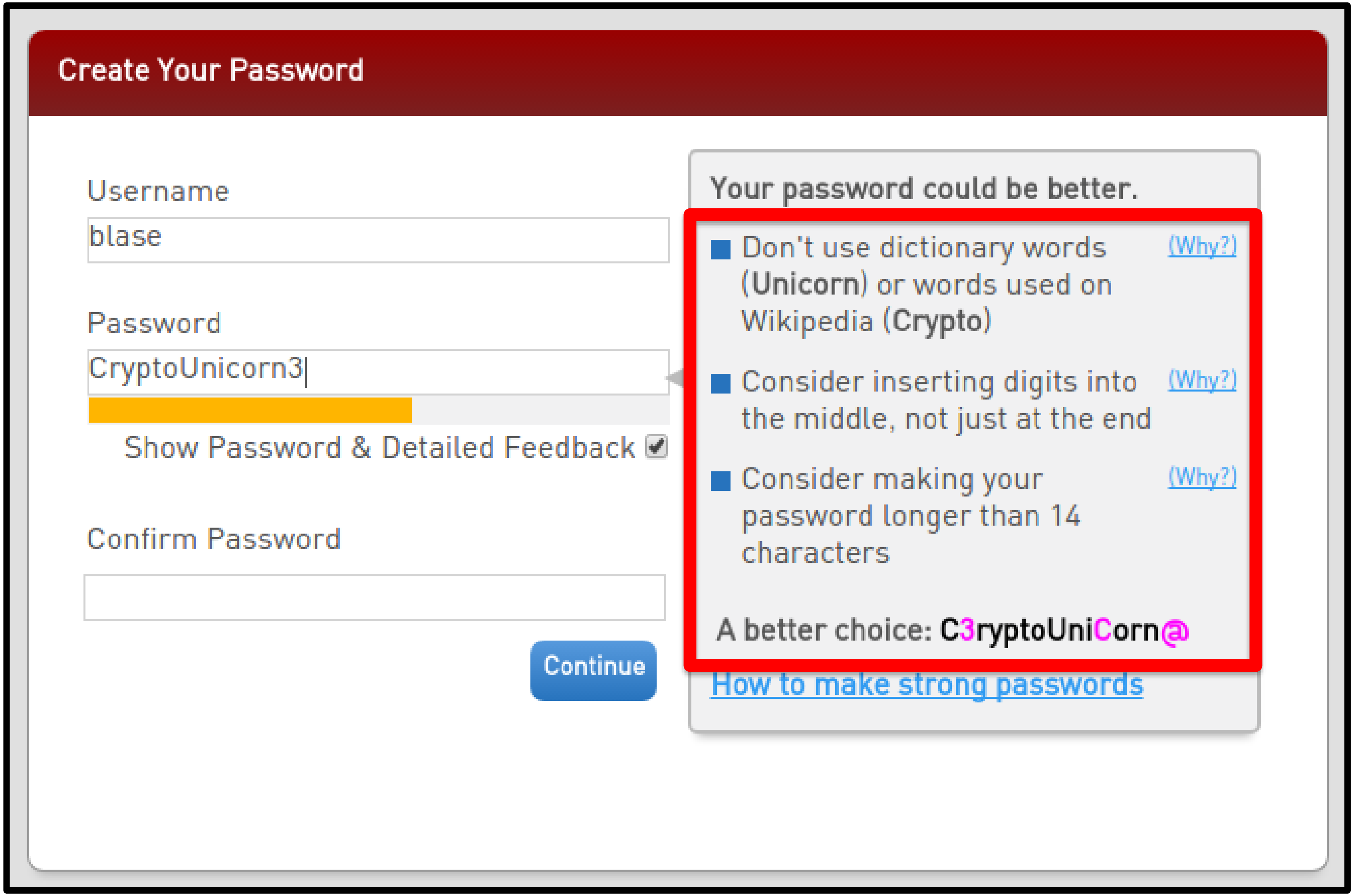
...Displays Score Visually



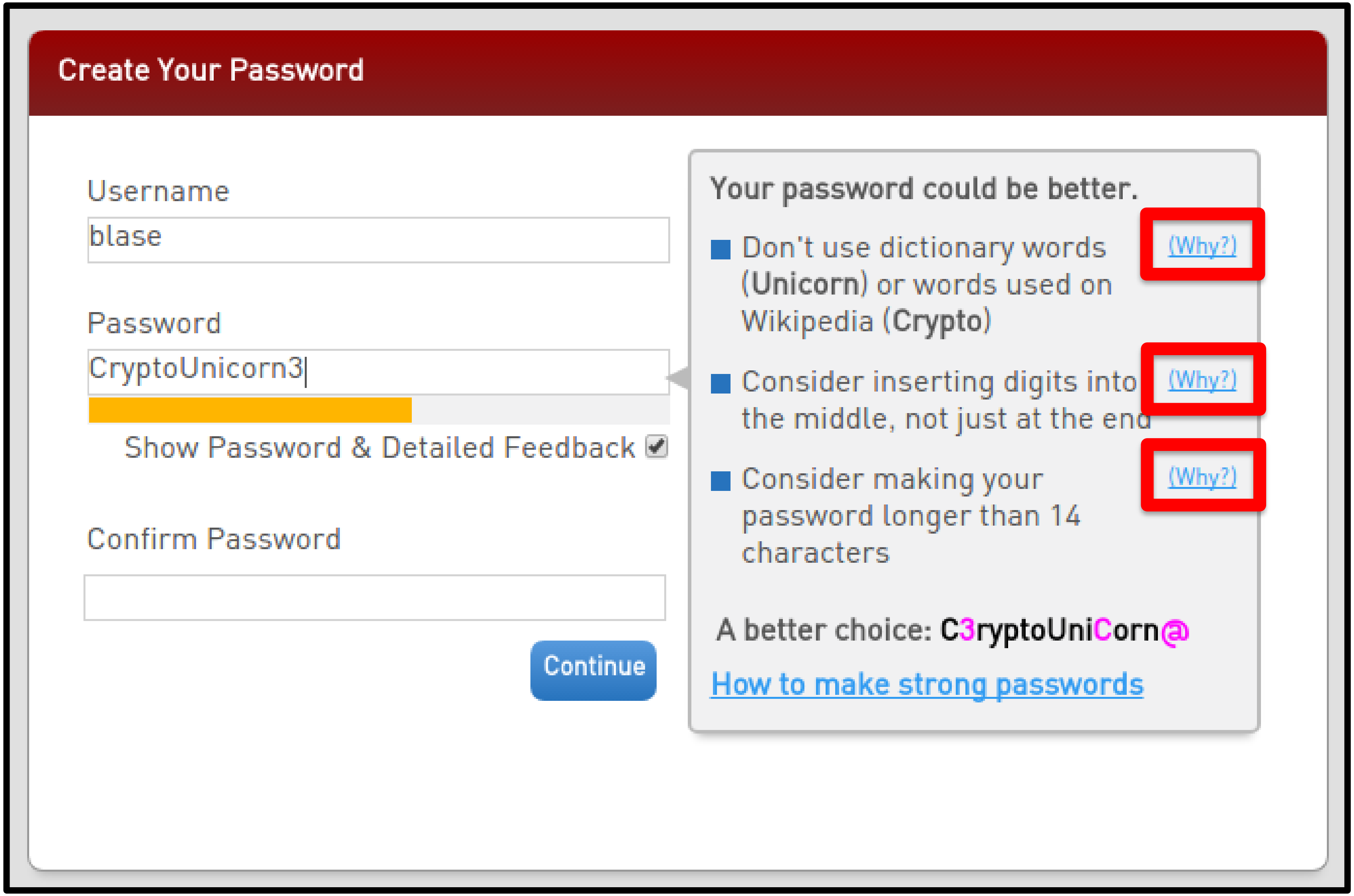
…Provides Text Feedback



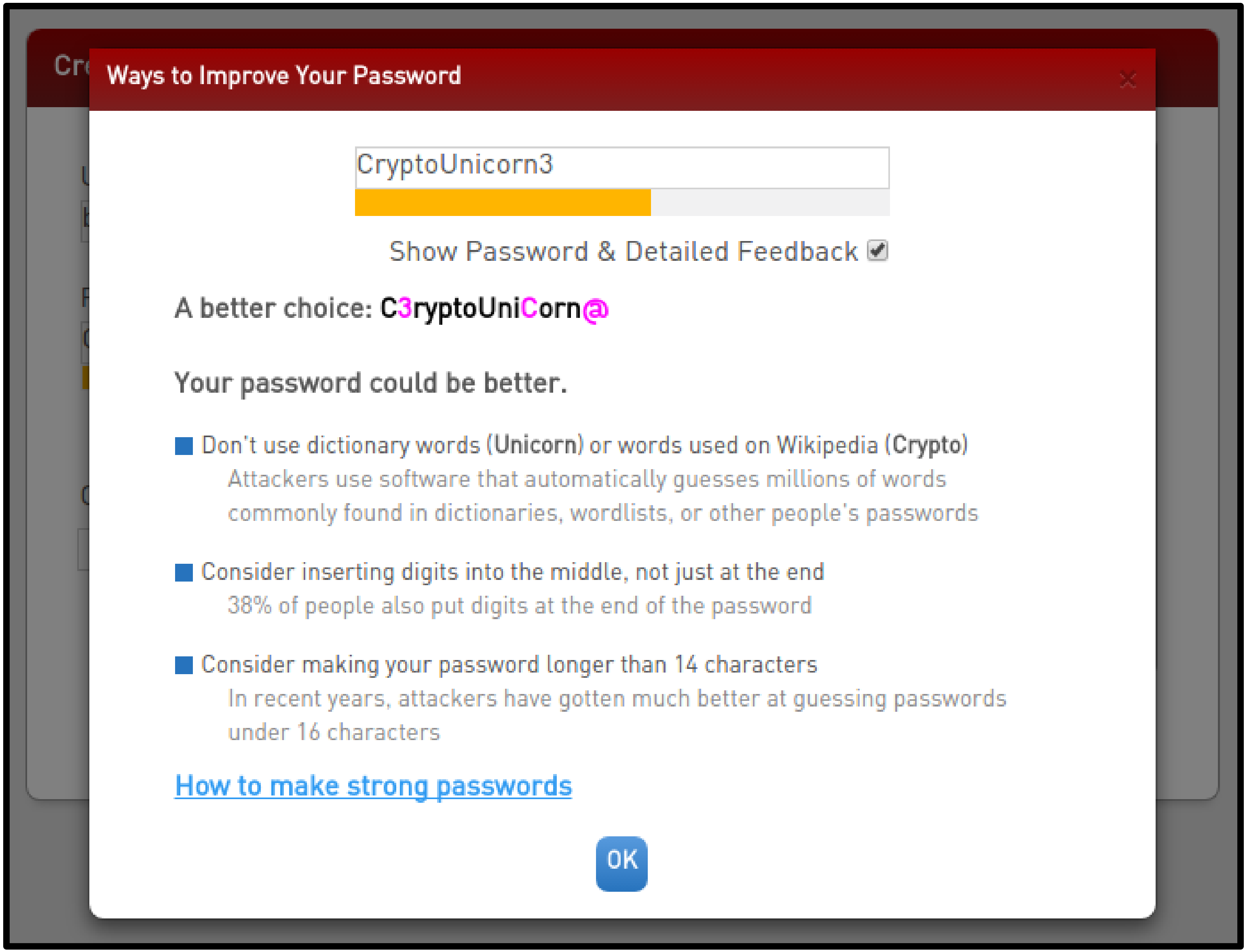
…Gives Detail (Password Shown)



…Offers Explanations



Explanations Shown in Modal



Standard Feedback