20. Authentication and Access Control Part 2

Blase Ur and David Cash
February 26th, 2021
CMSC 23200 / 33250
Some Ways to Understand Users

• Retrospective analysis of user-created passwords
• Large-scale online studies
• Examine real passwords
• Qualitative studies
Password Cracking

Statistical Metrics For Passwords

• Traditionally: Shannon entropy
• Recently: $\alpha$-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
1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$

Attack Model

$80d561388725fa74f2d03cd16e1d687c$
1. \( h(“123456”) = e10adc3949ba59abbe56e057f20f883e \)
2. \( h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99 \)
Attack Model

1. $h(“123456”) = e10adc3949ba59abbe56e057f20f883e$
2. $h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885$
Attack Model

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

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

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
jamesbond007!

Guess # 366,163,847,194
n(c$JZX!zKc^bIA^X^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
- Professionals
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

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
<th>Guesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super</td>
<td>1. Append “1”</td>
<td>Super1</td>
</tr>
<tr>
<td>Password</td>
<td>2. Replace “a” → “4”</td>
<td>Password1</td>
</tr>
<tr>
<td>Chicago</td>
<td>3. Lowercase all</td>
<td>Chicago1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Super</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P4ssword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chic4go</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGS (≈ 20,000,000)</td>
<td>Korelogic (≈ 5,000)</td>
</tr>
<tr>
<td>Linkedin (≈ 60,000,000)</td>
<td>Megatron (≈ 15,000)</td>
</tr>
<tr>
<td>HIBP (≈ 500,000,000)</td>
<td>Generated2 (≈ 65,000)</td>
</tr>
</tbody>
</table>
Example Wordlists and Rulelists

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGS (≈ 20,000,000)</td>
<td>Korelogic (≈ 5,000)</td>
</tr>
<tr>
<td>Linkedin (≈ 60,000,000)</td>
<td>Megatron (≈ 15,000)</td>
</tr>
<tr>
<td>HIBP (≈ 500,000,000)</td>
<td>Generated2 (≈ 65,000)</td>
</tr>
</tbody>
</table>

$10^9 - 10^{15}$ guesses
Example Wordlists and Rulelists

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGS ($\approx 20,000,000$)</td>
<td>Korelogic ($\approx 5,000$)</td>
</tr>
<tr>
<td>Linkedin ($\approx 60,000,000$)</td>
<td>Megatron ($\approx 15,000$)</td>
</tr>
<tr>
<td>HIBP ($\approx 500,000,000$)</td>
<td>Generated2 ($\approx 65,000$)</td>
</tr>
</tbody>
</table>

+ Hackers’ private word/rule lists

10$^9$ - 10$^{15}$ guesses
John the Ripper

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

\[
\begin{align*}
\text{wordlist} & \quad \rightarrow \quad \text{rules} \\
\text{rules} & \quad \rightarrow \quad \text{guesses}
\end{align*}
\]
John the Ripper

usenix
security

wordlist

rules

guesses
John the Ripper

usenix
security

[ ]
[add 1 at end]
[change e to 3]
John the Ripper

usenix
security

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

wordlist

rules

usenix
security

usenix1
security1
us3nix
s3curity

guesses
John the Ripper

usenix
security

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

wordlist

usenix
security

rules

usenix
security

usenix1
security1

us3nix
s3curity

guesses
John the Ripper

- wordlist: `usenix security`
  - [add 1 at end]
  - [change e to 3]

- rules: `usenix security1 us3nix s3curity`

- guesses: `usenix security usenix1 security1`
Hashcat

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

- wordlist
  - rules

  hashcat
  advanced password recovery

  guesses
Hashcat

usenix
security

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

wordlist

rules

guesses
Hashcat

usenix

security

[ ]

[add 1 at end]

[change e to 3]

rules

guesses

wordlist

usenix

usenix1

us3nix

security

security1

s3curity
Hashcat

wordlist

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

guesses

rules

usenix

security

usenix1

us3nix

security

security1

s3curity
# Hashcat: Rule Language

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
<th>Example Rule</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td></td>
<td>do nothing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lower-case</td>
<td>i</td>
<td>Lowercase all letters</td>
<td>l p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Uppercase</td>
<td>u</td>
<td>Uppercase all letters</td>
<td>u p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Capitalize</td>
<td>c</td>
<td>Capitalize the first letter and lower the rest</td>
<td>c p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Invert Capitalize</td>
<td>C</td>
<td>Lowercase first found character, uppercase the rest</td>
<td>C p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Toggile Case</td>
<td>t</td>
<td>Toggle the case of all characters in word</td>
<td>t p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Toggles Case</td>
<td>TN</td>
<td>Toggle the case of characters at position N</td>
<td>T3 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Reverse</td>
<td>r</td>
<td>Reverse the entire word</td>
<td>r p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Duplicate</td>
<td>d</td>
<td>Duplicate entire word</td>
<td>d p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Duplicate N</td>
<td>pN</td>
<td>Append duplicated word N times</td>
<td>p2 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Reflect</td>
<td>f</td>
<td>Duplicate word reversed</td>
<td>f p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Rotate Left</td>
<td>l</td>
<td>Rotates the word left</td>
<td>l p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Rotate Right</td>
<td>r</td>
<td>Rotates the word right</td>
<td>r dp@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Append Character</td>
<td>SX</td>
<td>Append character X to end</td>
<td>S1 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Prepend Character</td>
<td>AX</td>
<td>Prepend character X to front</td>
<td>AX p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Truncate left</td>
<td>l</td>
<td>Deletes first character</td>
<td>l p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Truncate right</td>
<td>r</td>
<td>Deletes last character</td>
<td>r p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Delete N</td>
<td>DN</td>
<td>Deletes character at position N</td>
<td>D3 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Extract range</td>
<td>xNM</td>
<td>Extracts M characters, starting at position N</td>
<td>x04 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Insert N</td>
<td>iN</td>
<td>Inserts character X at position N</td>
<td>i41 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Overwrite N</td>
<td>oNX</td>
<td>Overwrites character X at position N with X</td>
<td>o33 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Truncate @ N</td>
<td>N</td>
<td>Truncate word at position N</td>
<td>6 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Replace SX</td>
<td>sX</td>
<td>Replace all instances of X with Y</td>
<td>s@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Purge @X</td>
<td>p@X</td>
<td>Purge all instances of X</td>
<td>p@X p@ssWOrd</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
<th>Example Rule</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject less</td>
<td>&lt;N</td>
<td>Reject plains if their length is greater than N</td>
<td>&lt;G</td>
<td></td>
</tr>
<tr>
<td>Reject greater</td>
<td>&gt;N</td>
<td>Reject plains if their length is less or equal to N</td>
<td>&gt;8</td>
<td></td>
</tr>
<tr>
<td>Reject equal</td>
<td>_N</td>
<td>Reject plains of length not equal to N</td>
<td>_7</td>
<td></td>
</tr>
<tr>
<td>Reject contain</td>
<td>IX</td>
<td>Reject plains which contain char X</td>
<td>Iz</td>
<td></td>
</tr>
<tr>
<td>Reject not contain</td>
<td>/X</td>
<td>Reject plains which do not contain char X</td>
<td>/e</td>
<td></td>
</tr>
<tr>
<td>Reject equal first</td>
<td>(X</td>
<td>Reject plains which do not start with X</td>
<td>(h</td>
<td></td>
</tr>
<tr>
<td>Reject equal last</td>
<td>X</td>
<td>Reject plains which do not end with X</td>
<td>)l</td>
<td></td>
</tr>
<tr>
<td>Reject equal at</td>
<td>=NX</td>
<td>Reject plains which do not have char X at position N</td>
<td>=1a</td>
<td></td>
</tr>
<tr>
<td>Reject contains</td>
<td>%NX</td>
<td>Reject plains which contain char X less than N times</td>
<td>%2a</td>
<td></td>
</tr>
<tr>
<td>Reject contains Q</td>
<td>Q</td>
<td>Reject plains where the memory saved matches current word</td>
<td>rMrQ</td>
<td>e.g. for palindrome</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
<th>Example Rule</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap front</td>
<td>k</td>
<td>Swaps first two characters</td>
<td>k p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Swap back</td>
<td>K</td>
<td>Swaps last two characters</td>
<td>K p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Swap @ N</td>
<td>*NM</td>
<td>Swaps character at position N with character at position M</td>
<td>*34 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Bitwise shift left</td>
<td>LN</td>
<td>Bitwise shift left character @ N</td>
<td>L2 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Bitwise shift right</td>
<td>RN</td>
<td>Bitwise shift right character @ N</td>
<td>R2 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Ascii increment</td>
<td>+N</td>
<td>Increment character @ N by 1 ascii value</td>
<td>+2 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Ascii decrement</td>
<td>-N</td>
<td>Decrement character @ N by 1 ascii value</td>
<td>-1 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Replace N + 1</td>
<td>N</td>
<td>Replaces character @ N with value at @ N plus 1</td>
<td>.1 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Replace N – 1</td>
<td>N</td>
<td>Replaces character @ N with value at @ N minus 1</td>
<td>.1 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Duplicate block front</td>
<td>yN</td>
<td>Duplicates first N characters</td>
<td>y2 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Duplicate block back</td>
<td>YN</td>
<td>Duplicates last N characters</td>
<td>Y2 p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>E</td>
<td>Lower case the whole line, then upper case the first letter and every letter after a space</td>
<td>E p@ssWOrd</td>
<td></td>
</tr>
<tr>
<td>Title w/separater</td>
<td>eX</td>
<td>Lower case the whole line, then upper case the first letter and every letter after a custom separator character</td>
<td>eX p@ssWOrd</td>
<td></td>
</tr>
</tbody>
</table>
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

1-gram model:
[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
- Speed: Slow
- "PCFG"
structure model:

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

Digit Model:
\[ D_1 \rightarrow 3 \, (0.5) \, 5 \, (0.5) \]
\[ D_3 \rightarrow 123 \, (1.0) \]

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’’
Approach

4 password sets

password
iloveyou
teamo123
...

censored_password
censored_password
censored_password
...

Pa$$w0rd
iLov3you!
1QaZ2W@x
...

pa$$word1234
12345678asDF
!q1q!q1q!q1q
...

5 approaches

hashcat
applied advanced password recovery

KoreLogic
security
Configuration Is Crucial

LongComplex

Percent guessed

Guesses

0%
10%
20%
30%
40%

10^1  10^3  10^5  10^7  10^9  10^{11}  10^{13}  10^{15}

HC-MWR-big
HC-MWR
HC-Generated2-big
HC-Generated2
HC-SpiderLabs-big
HC-SpiderLabs
HC-Best64-big
HC-Best64
Comparison for Complex Passwords

![Graph showing comparison for complex passwords]
Per-Password Highly Impacted

Password!
Per-Password Highly Impacted

• JTR guess # 801

Password!
Per-Password Highly Impacted

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

Password!
Per-Password Highly Impacted

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

Password: P@ssw0rd!
Neural Networks For Passwords

Better Password Scoring

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

Recurrent Neural Networks (RNNs)

LSTM Architecture
Generating Passwords
Generating Passwords

password → o or maybe 0 or O or ...
Generating Passwords

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

<table>
<thead>
<tr>
<th>Character</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10%</td>
</tr>
<tr>
<td>B</td>
<td>1%</td>
</tr>
<tr>
<td>C</td>
<td>4%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>O</td>
<td>8%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Z</td>
<td>0.02%</td>
</tr>
<tr>
<td>0</td>
<td>3%</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>END</td>
<td>6%</td>
</tr>
</tbody>
</table>

“C”
Prob: 5%
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)
How Do We Help Users Make Better Passwords?
**Problem 1: Bad Advice**

### Password Requirements

**Must Contain**
- At least 8-characters.
- At least one uppercase alphabetic character (e.g., A-Z).
- At least one lowercase alphabetic character (e.g., a-z).
- At least one number (e.g., 0-9).
- At least one special character (e.g., []~!@#$%^&*()?><./_-+=).

**Cannot Contain**
- Known information (i.e., first name, last name, Andrew user ID, date of birth, 9-digit Carnegie Mellon ID number, SSN, job title).
- Four or more occurrences of the same character (e.g., aaaa, 2222, a123a345a678a).*
- A word that is found in a standard dictionary.*
  (after removing non-alpha characters).

*This requirement does not apply to Andrew account passwords that are more than 19 characters in length (e.g., passphrase).

### Additional Policies
- Last five passwords cannot be used.
- Cannot be changed more than four times in a day.
Problem 2: Inaccurate Feedback
Problem 3: Unhelpful Feedback

Twitter

Please enter a stronger password.

Please enter a stronger password.
Building a Data-Driven Meter

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…

Create Your Password

Username
blase

Password
**********
Show Password & Detailed Feedback

Confirm Password

Your password could be better.

- Don't use dictionary words or words used on Wikipedia
  (Why?)
- Consider inserting digits into the middle
  (Why?)
- Consider making your password longer
  (Why?)

See Your Password With Our Improvements

How to make strong passwords

Continue
...Displays Score Visually
…Provides Text Feedback

Create Your Password

Username
blase

Password

Show Password & Detailed Feedback

Confirm Password

Your password could be better.

- Don’t use dictionary words or words used on Wikipedia (Why?)
- Consider inserting digits into the middle (Why?)
- Consider making your password longer (Why?)

See Your Password With Our Improvements

How to make strong passwords
...Gives Detail (Password Shown)

Create Your Password

Username: blase
Password: CryptoUnicorn3|
Confirm Password:

Your password could be better:
- Don’t use dictionary words (Unicorn) or words used on Wikipedia (Crypto)
- Consider inserting digits into the middle, not just at the end
- Consider making your password longer than 14 characters

A better choice: C3ryptoUnicorn@

How to make strong passwords
…Offers Explanations
Explanations Shown in Modal

A better choice: CryptoUnicorn3

Your password could be better.

- Don’t use dictionary words (Unicorn) or words used on Wikipedia (Crypto). Attackers use software that automatically guesses millions of words commonly found in dictionaries, wordlists, or other people’s passwords.
- Consider inserting digits into the middle, not just at the end. 38% of people also put digits at the end of the password.
- Consider making your password longer than 14 characters. In recent years, attackers have gotten much better at guessing passwords under 16 characters.

How to make strong passwords

OK
Standard Feedback

Create Your Password

Username
blase

Password

Your password could be better.
- Don’t use dictionary words (Unicorn) or words used on Wikipedia (Crypto)

A better choice: C3ryptoUniCorn@

Confirm Password

A better choice: C3ryptoUniCorn@

Continue

How to make strong passwords