

Rainbow Tables

ECS 153 Spring Quarter 2021

Hellman Time-Space Tradeoff

Originally used to find DES keys

$$sp_1 \rightarrow xp_{1,1} = f(sp_1) \rightarrow xp_{1,2} = f(r(xp_{1,1})) \rightarrow \dots \rightarrow ep_1 = f(r(xp_{1,t-1})) \quad (sp_1, ep_1)$$

$$sp_2 \rightarrow xp_{2,1} = f(sp_2) \rightarrow xp_{2,2} = f(r(xp_{2,1})) \rightarrow \dots \rightarrow ep_2 = f(r(xp_{2,t-1})) \quad (sp_2, ep_2)$$

...

...

$$sp_k \rightarrow xp_{k,1} = f(sp_k) \rightarrow xp_{k,2} = f(r(xp_{k,1})) \rightarrow \dots \rightarrow ep_k = f(r(xp_{k,t-1})) \quad (sp_k, ep_k)$$

$f: A \rightarrow C$ is complementation function

$r: C \rightarrow A$ is reduction function, which produces printed output

How It Works

- Attacker wants to find password p given hash $h = f(p)$
- Attacker sees if it is in list of ep_i
 - If there, attacker reconstructs chain; value in next-to-last column is p
 - If not there, attacker computes $f(h)$ and compare it to list of ep_i
 - If there, attacker reconstructs chain; value in second-to-last column is p
 - If not . . . iterate until p found or determined not to be in table
- Problem: collisions
 - If same intermediate values in two chains, the chains match from then on
- Solution: rainbow tables!

Rainbow Tables

- Same as Hellman's table, except that the reduction function r is replaced by a family of reduction functions $\{r_1, \dots, r_{t-1}\}$

$$sp_1 \rightarrow xp_{1,1} = f(sp_1) \rightarrow xp_{1,2} = f(r_1(xp_{1,1})) \rightarrow \dots \rightarrow ep_1 = f(r_{t-1}(xp_{1,t-1})) \quad (sp_1, ep_1)$$

$$sp_2 \rightarrow xp_{2,1} = f(sp_2) \rightarrow xp_{2,2} = f(r_1(xp_{2,1})) \rightarrow \dots \rightarrow ep_2 = f(r_{t-1}(xp_{2,t-1})) \quad (sp_2, ep_2)$$

...

...

$$sp_k \rightarrow xp_{k,1} = f(sp_k) \rightarrow xp_{k,2} = f(r_1(xp_{k,1})) \rightarrow \dots \rightarrow ep_k = f(r_{t-1}(xp_{k,t-1})) \quad (sp_k, ep_k)$$

- Fewer collisions as intermediate values must also be in the same column

Thwarting Rainbow Tables

- Use large salts unique to each user
 - Attacker has to compute one table per salt
 - Linux, FreeBSD, other systems typically use salts of 128 bits
- Key stretching: iterate password hash value
 - Example: hash *password* | *salt* multiple times
 - Slows construction of table
- Other issues
 - Will not get passwords with symbols outside the range of those used to construct the rainbow table
 - Will not get passwords longer than those used to construct the rainbow table