Lecture 26, June 1, 2026

SQL Injection Attacks

- Web app code:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

- Fill out form with “anthony”

```
SELECT * FROM Users WHERE UserId = anthony
```

- Fill out form with “admin OR 1=1”

```
SELECT * FROM Users WHERE UserId = admin OR 1=1
```

Frustration Personified: Trying to Change a Password

Please enter your new password:

cabbage

Sorry, the password must be more than 8 characters. Please enter your new password:

boiled cabbage

Sorry, the password must contain 1 numeric character. Please enter your new password:

1 boiled cabbage

Sorry, the password cannot have spaces. Please enter your new password:

50bleepingboiledcabbages

Sorry, the password must contain at least 1 upper case character. Please enter your new password:

50BLEEPINGboiledcabbages

Sorry, the password cannot have more than one upper case character consecutively. Please enter your new password:

50BleepingBoiledCabbagesShovedUpYourRearIfYouDon’tGiveMeAccessNow!

Sorry, the password cannot contain punctuation. Please enter your new password:

ReallyPissedOff50BleepingBoiledCabbagesShovedUpYourRearIfYouDontGiveMeAccessNow

Sorry, the password is already in use. Please enter your new password:

Password Cracking: Rainbow Tables

Hellman Time-Space Tradeoff

Originally used to find DES keys

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

$sp_1 \rightarrow xp_{2,1} = f(sp_2) \rightarrow xp_{2,2} = f(r(xp_{2,1})) \rightarrow \ldots \rightarrow ep_2 = f(r(xp_{2,t\text{-}1}))$ (sp_2, ep_2)

$\ldots$ $\ldots$

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

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

$sp_1 \rightarrow xp_{2,1} = f(sp_2) \rightarrow xp_{2,2} = f(r_1(xp_{2,1})) \rightarrow \ldots \rightarrow ep_2 = f(r_{t-1}(xp_{2,t-1}))$ (sp_2, ep_2)

$\ldots$ $\ldots$

$sp_k \rightarrow xp_{k,1} = f(sp_k) \rightarrow xp_{k,2} = f(r_1(xp_{k,1})) \rightarrow \ldots \rightarrow ep_k = f(r_{t-1}(xp_{k,t-1}))$ (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 ise salts of 128 bits
- Key stretching: iterate password hash value
  - Example: hash *password* & *salt* combination 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