Securing the Personal Automated Scheduling System

CSCE 548 | Dr. Farkas

Members

- Will Reade
- Breland Miley
- Matthew Zimmermann
- Ryan Bowen

Overview

- Current system
  - Collate desired classes and create combinations for every possible schedule
  - Register students for classes
  - Eventually handle entire advisement process to simplify registration time

- Goal of 548 Research (In no order)
  - Discover potential vulnerabilities
  - Increase knowledge of security procedures
  - Get an A.
  - Graduate.
  - Make millions.

Vulnerabilities

API

- Malicious Abuse (brute force attack)
- DDoS

SQL Server

- SQL Injection
- Data redundancy
- Data availability (hardware failure)

Application (Front End)

- Cross-Site Scripting

API Research

Key-Based API Access

- Each API call has unique key attached
- Hashed key is validated before any "work" is done

Geographic Distribution

- Distributed servers can help to prevent DDoS
- Relies on consistent, quick key checking and multiple servers

Load Balancing

- Load balancing allows us to ensure even if a DDoS attack is attempted, attack requests will be forwarded to least busy server, ensuring at worst, a "higher than average" load across all servers.

API Research

Key Distribution

- Our generated keys are uniformly distributed for our entire keyspace
  - 50 "A"s have an equally likely chance of being 50 "Z"s or 50 "0"s.
- Random key distribution helps to ensure true key entropy.

Break it

Testing

- Self DDoS
  - Hosted instances on EC2, allowed to auto deploy new instance, distributed geographically. Front and back on separate groups of servers.
  - Dev server environment for attacker environment (courtesy of SCANA [dual OC-3c @ 149.76 Mbps/line])
  - Objective: Test Front End/Back End load distribution and test API key brute force attack

API/DDoS Attack Results

- **BF Key Testing + DoS Attack**
  - At worst, 15 Amazon EC2 servers spawned (6 DB servers, 7 Web Servers)
  - 25 servers generating requests at ~50 reqs/second (1,250 HTTP reqs/second, ~75,000 HTTP API reqs/min)
  - **Results**
    - Never had key collision
    - ~60 minutes in, Amazon decided "malicious activity taking place on your account"

API/DDoS Attack Results

- **BF Key Testing + DoS Attack**
  - In reality, keyspace is 50^62 (50 character key, 26+26+10), had ~5 billion requests
    - Didn't make a dent in keyspace testing (testing for key collisions)
  - Load statistics: Web --> 56% CPU Avg, DB -->**87%** CPU Avg; Nearly 100% usage at peak attack times for DB server
    - Amazon auto scaled and distributed requests, wanted more instances of DB servers, but setup constraints wouldn't allow for it
  - **Conclusion : Don't tick off "Anonymous"**

SQL Research

"Treat all input as evil."

Parameterized Queries

- Keeps user input separate from query string
- Try to rely on integer inputs for majority of API calls.

Externally Stored SALT

- Explanation of "salting" a password
- Keep SALT separately stored from database

SQL Research

- LINQ provides automatic parameterized queries
- SQL account set to only allow updates and reads on specific tables necessary for each operation
- GreenSQL (database firewall) running between API and SQL
    - Looks for things like tautologies and non-known queries against the database

How we tested

- SQL Ninja - SQL Server Injection and takeover tool
    - Ran against firewalled and non-firewalled databases
    - With GreenSQL, SQL ninja queries never even touched the database = A+
    - Without GreenSQL, app rejected non-acceptable user input = A+
- Absinthe - Blind SQL injection tool
    - No results found = A+

XSS Research

Cross Site Scripting

- Allows Javascript and HTML to be injected into code and deployed to users
- Causes tremendous problem with cookies and local browser storage
- Careful coding and scripting can only work to reduce the threat level of such attacks

XSS Results

- No perfect solution
- PASS does store input from the user and allows it to be reproduced on certain pages
    - Is vulnerable to XSS attacks.
- Risk minimized in sense that no input from one user is ever displayed to another user
    - Worst case: User can initiate XSS attack on themselves
- Working on encoding all user input/output

At the end of the day...

- Research provided great insight into how to secure PASS
  - Application redesigned with security as top priority
- Knew some of the larger security principles, but needed to implement specific risk mitigation tools
- Based on testing, we managed to build a fairly robust and secure application

Questions?

Here's my dog. Hugging a toy.