CSci 4271W
Development of Secure Software Systems
Day 27: Authentication and Security Testing

Stephen McCamant

University of Minnesota, Computer Science & Engineering

Outline

Web authentication

Announcements intermission

Names and identities

ROC curve exercise

Testing and fuzzing

Per-website authentication

- Many web sites implement their own login systems
  - + If users pick unique passwords, little systemic risk
  - − Inconvenient, many will reuse passwords
  - − Lots of functionality each site must implement correctly
  - − Without enough framework support, many possible pitfalls

Building a session

- HTTP was originally stateless, but many sites want stateful login sessions
- Built by tying requests together with a shared session ID
- Must protect confidentiality and integrity

Session ID: what

- Must not be predictable
  - Not a sequential counter
- Should ensure freshness
  - E.g., limited validity window
- If encoding data in ID, must be unforgeable
  - E.g., data with properly used MAC
  - Negative example: crypt(username || server secret)

Session ID: where

- Session IDs in URLs are prone to leaking
  - Including via user cut-and-paste
- Usual choice: non-persistent cookie
  - Against network attacker, must send only under HTTPS
- Because of CSRF, should also have a non-cookie unique ID

Session management

- Create new session ID on each login
- Invalidate session on logout
- Invalidate after timeout
  - Usability / security tradeoff
  - Needed to protect users who fail to log out from public browsers

Account management

- Limitations on account creation
  - CAPTCHA? Outside email address?
- See previous discussion on hashed password storage
- Automated password recovery
  - Usually a weak spot
  - But, practically required for large system

Client and server checks

- For usability, interface should show what's possible
- But must not rely on client to perform checks
- Attackers can read/modify anything on the client side
- Easy example: item price in hidden field

Direct object references

- Seems convenient: query parameter names resource directly
  - E.g., database key, filename (path traversal)
- Easy to forget to validate on each use
- Alternative: indirect reference like per-session table
  - Not fundamentally more secure, but harder to forget check

Function-level access control

- E.g. pages accessed by URLs or interface buttons
- Must check each time that user is authorized
  - Attack: find URL when authorized, reuse when logged off
- Helped by consistent structure in code

Outline

Web authentication

Announcements intermission

Names and identities

ROC curve exercise

Testing and fuzzing

Midterm 2 statistics

```
<=4 | *
  5 | 056666
  6 | 2345667888
  7 | 00012368899
  8 | 00344444466667799
  9 | 02344666789
 10 | 00
```

Mean 75.7, median 79, difficulty adjustment +5

SRT logistics

- All online this semester
  - Open through the last regular class day (next Monday)
- Requested but not required; won't affect your grade one way or the other
- Primary evaluation combines Prof. McCamant and the course
- Please also evaluate Aditya separately if you have comments or suggestions about his performance

SRT URL

- `https://srt.umn.edu/blue`
- We'll take a 15-minute break in class material that we request you use for filling out the evaluation

Outline

Web authentication

Announcements intermission

Names and identities

ROC curve exercise

Testing and fuzzing

Accounts versus identities

- "Identity" is a broad term that can refer to a personal conception or an automated sytem
- "Name" is also ambiguous in this way
- "Account" and "authentication" refer unambiguously to institutional/computer abstractions
- Any account system is only an approximation of the real world

Real human names are messy

- Most assumptions your code might make will fail for someone
  - ASCII, length limit, uniqueness, unchanging, etc.
- So, don't design in assumptions about real names
- Use something more computer-friendly as the core identifier
  - Make "real" names or nicknames a presentation aspect

Zooko's triangle

- Claims (2001) it is hard/impossible for a naming scheme to be simultaneously:
  - Human-meaningful
  - Secure
  - Decentralized
- Too imprecise to be definitively proven/refuted
  - Blockchain-based name systems are highest-profile claimed counterexamples
- A useful heuristic for seeing design tensions

Identity documents: mostly unhelpful

- "Send us a scan of your driver's license"
  - Sometimes called for by specific regulations
  - Unnecessary storage is a disclosure risk
  - Fake IDs are very common

Identity numbers: mostly unhelpful

- Common US example: social security number
- Variously used as an identifier or an authenticator
  - Dual use is itself a cause for concern
- Known by many third parties (e.g., banks)
- No checksum, guessing risks
- Published soon after a person dies

"Identity theft"

- The first-order crime is impersonation fraud between two other parties
  - E.g., criminal trying to get money from a bank under false pretenses
- The impersonated "victim" is effectively victimized by follow-on false statements
  - E.g., by credit reporting agencies
  - These costs are arguably the result of poor regulatory choices
- Be careful w/ negative info from 3rd parties

Outline

Web authentication

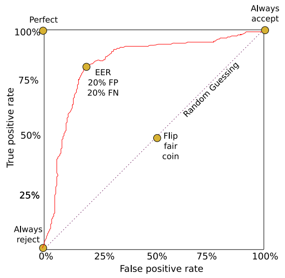
Announcements intermission

Names and identities

ROC curve exercise

Testing and fuzzing

Error rates: ROC curve



Extreme biometrics examples

- `exact_iris_code_match`: very low false positive (false authentication)
- `similar_voice_pitch`: very low false negative (false reject)

Where are these in ROC space?

```
A if (iris()) return REJECT; else return ACCEPT;
B return REJECT;
C if (iris()) return ACCEPT; else return REJECT;
D if (iris() && pitch()) return ACCEPT; else return REJECT;
E return ACCEPT;
F if (rand() & 1) return ACCEPT; else return REJECT;
G if (pitch()) return ACCEPT; else return REJECT;
H if (iris() || pitch()) return ACCEPT; else return REJECT;
```

Outline

Web authentication

Announcements intermission

Names and identities

ROC curve exercise

Testing and fuzzing

Testing and security

- “Testing shows the presence, not the absence of bugs” – Dijkstra
- Easy versions of some bugs can be found by targeted tests:
  - Buffer overflows: long strings
  - Integer overflows: large numbers
  - Format string vulnerabilities: `%x`

Random or fuzz testing

- Random testing can also sometimes reveal bugs
- Original ‘fuzz’ (Miller): `program </dev/urandom`
- Even this was surprisingly effective

Mutational fuzzing

- Instead of totally random inputs, make small random changes to normal inputs
- Changes are called *mutations*
- Benign starting inputs are called *seeds*
- Good seeds help in exercising interesting/deep behavior

Grammar-based fuzzing

- Observation: it helps to know what correct inputs look like
- Grammar specifies legal patterns, run backwards with random choices to generate
- Generated inputs can again be basis for mutation
- Most commonly used for standard input formats
  - Network protocols, JavaScript, etc.

What if you don’t have a grammar?

- Input format may be unknown, or buggy and limited
- Writing a grammar may be too much manual work
- Can the structure or interesting inputs be figured out automatically?

Coverage-driven fuzzing

- Instrument code to record what code is executed
- An input is interesting if it executes code that was not executed before
- Only interesting inputs are used as basis for future mutation

AFL

- Best known open-source tool, pioneered coverage-driven fuzzing
- American Fuzzy Lop, a breed of rabbits
- Stores coverage information in a compact hash table
- Compiler-based or binary-level instrumentation
- Has a number of other optimizations