CSci 4271W
Development of Secure Software Systems
Day 8: More Threat Modeling, More Defenses

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

More perspectives on threat modeling

Threat modeling: printer manager

Announcements intermission

Return address protections

Software-oriented modeling

- This is what we've concentrated on until now
  - And it will still be the biggest focus
- Think about attacks based on where they show up in the software
- Benefit: easy to connect to software-level mitigations and fixes

Asset-oriented modeling

- Think about threats based on what assets are targeted / must be protected
- Useful from two perspectives:
  - Predict attacker behavior based on goals
  - Prioritize defense based on potential losses
- Can put other modeling in context, but doesn't directly give you threats

Kinds of assets

- Three overlapping categories:
  - Things attackers want for themselves
  - Things you want to protect
  - Stepping stones to the above

Attacker-oriented modeling

- Think about threats based on the attacker carrying them out
  - Predict attacker behavior based on characteristics
  - Prioritize defense based on likelihood of attack
- Limitation: it can be hard to understand attacker motivations and strategies
  - Be careful about negative claims

Kinds of attackers (Intel TARA)

- Competitor
- Data miner
- Radical activist
- Cyber vandal
- Sensationalist
- Civil activist
- Terrorist
- Anarchist
- Irrational individual
- Gov't cyber warrior
- Corrupt gov't official
- Legal adversary

Kinds of attackers (cont'd)

- Internal spy
- Government spy
- Thief
- Vendor
- Reckless employee
- Information partner
- Disgruntled employee

Outline

More perspectives on threat modeling

Threat modeling: printer manager

Announcements intermission

Return address protections

Setting: shared lab with printer

- Imagine a scenario similar to CSE Labs
  - Computer labs used by many people, with administrators
- Target for modeling: software system used to manage printing
  - Similar to real system, but use your imagination for unknown details

Example functionality

- Queue of jobs waiting to print
  - Can cancel own jobs, admins can cancel any
- Automatically converting documents to format needed by printer
- Quota of how much you can print

Assets and attackers

- What assets is the system protecting?
  - What negative consequences do we want to avoid?
- Who are the relevant attackers?
  - What goals motivate those attackers?
- Take 5 minutes to brainstorm with your neighbors

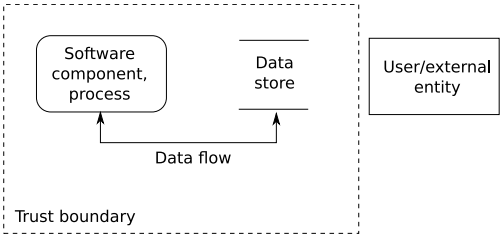
Assets and attackers

- Administrators:
  - Want to let students do printing needed for classes
  - While minimizing spending on paper, toner, and admins responding to problems
- Attackers:
  - Non-students might try to print
  - Students might try to print too much
  - Students might interfere with each other

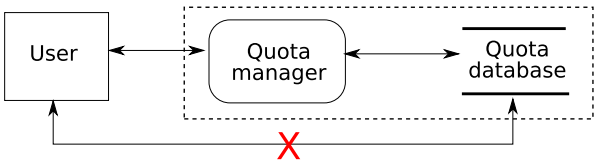
Data flow diagram

- Show structure of users, software/hardware components, data flows, and trust boundaries
- For this exercise, can mix software, OS, and network perspectives
- Include details relevant to security design decisions
- Take 15 minutes to draw with your neighbors

Data flow diagram: key

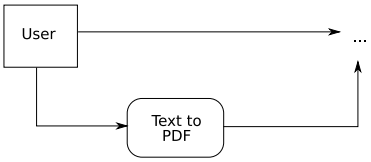


DFD #1: access control



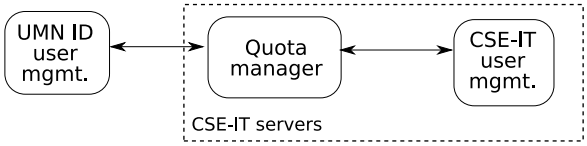
- The absence of data flow will need an implementation

DFD #2: optional processing



- Text-to-PDF can't add much risk here

DFD #3: a trust boundary



- Different risks from where authentication lies

STRIDE threat brainstorming

- Think about possible threats using the STRIDE classification
- Are all six types applicable in this example?
- Take 10 minutes to brainstorm with your neighbors

STRIDE examples

S: make your jobs look like a different student's
T: insert mistakes in another student's homework
R: claim you don't know why your quota is used up
I: read another student's homework
D: break printing before an assignment deadline
E: student performs administrator actions

Outline

More perspectives on threat modeling

Threat modeling: printer manager

Announcements intermission

Return address protections

Brief announcements

- Problem set 1 is available on the public web page now
  - Due a week from Friday, 2/17
- The first midterm exam will be a week from next Tuesday (2/21) in class
  - Open book, open notes
  - You will have the whole class period
  - Topics will be memory safety bugs and attacks, and threat modeling
  - Similar concepts, but less depth, than labs and p-set

Outline

More perspectives on threat modeling

Threat modeling: printer manager

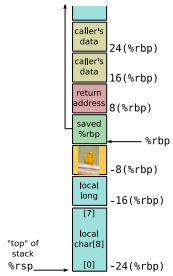
Announcements intermission

Return address protections

Canary in the coal mine

Photo credit: Fir0002 CC-BY-SA

Adjacent canary idea



Terminator canary

- Value hard to reproduce because it would tell the copy to stop
- StackGuard: 0x00 0D 0A FF
  - 0: String functions
  - newline: `fgets`(), etc.
  - -1: `getc`()
  - carriage return: similar to newline?
- Doesn't stop: `memcpy`, custom loops

Random canary

- Can't reproduce because attacker can't guess
- For efficiency, usually one per execution
- Ineffective if disclosed

XOR canary

- Want to protect against non-sequential overwrites
- XOR return address with value `c` at entry
- XOR again with `c` before return
- Standard choice for `c`: see random canary

Further refinements

- More flexible to do earlier in compiler
- Rearrange buffers after other variables
  - Reduce chance of non-control overwrite
- Skip canaries for functions with only small variables
  - Who has an overflow bug in an 8-byte array?

What's usually not protected?

- Backwards overflows
- Function pointers
- Adjacent structure fields
- Adjacent static data objects

Where to keep canary value

- Fast to access
- Buggy code/attacker can't read or write
- Linux/x86: `%gs:0x14`

Complex anti-canary attack

- Canary not updated on `fork` in server
- Attacker controls number of bytes overwritten

Complex anti-canary attack

- Canary not updated on `fork` in server
- Attacker controls number of bytes overwritten
- ANRY BNRY CNRY DNRY ENRY FNRY
- search 2^{32} $\rightarrow$ search $4 \cdot 2^8$

Shadow return stack

- Suppose you have a safe place to store the canary
- Why not just store the return address there?
- Needs to be a separate stack
- Ultimate return address protection