CSE 484/M584: Computer Security (and Privacy)

Spring 2025

David Kohlbrenner
dkohlbre@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner, Nirvan Tyagi. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials

Admin

- Office hours are posted
  - They will start FRIDAY (none today or Thursday)
  - My office hours are Friday afternoons 230-330.
  - We have a LOT of office hours. Use them!
- Lab 1 will be posted tomorrow/tonight.
  - 1a will be due next Wednesday (Section should get you 90% of the way done)

Threat Modeling

Threat Modeling

- Assets: What are we trying to protect?
- Adversaries: Who might try to attack, and why?
- Vulnerabilities: How might the system be weak?
- Threats: What actions might an adversary take to exploit vulnerabilities?
- Risk: How important are assets? How likely is exploit?
- Possible Defenses: What mitigations do we have available?
- Not "traditional" threat modeling, but important (both in general, and to help better understand the system prior to threat modeling):
  - Benefits: Who might the system benefit, and how?
  - Harms: Who might the system harm, and how?

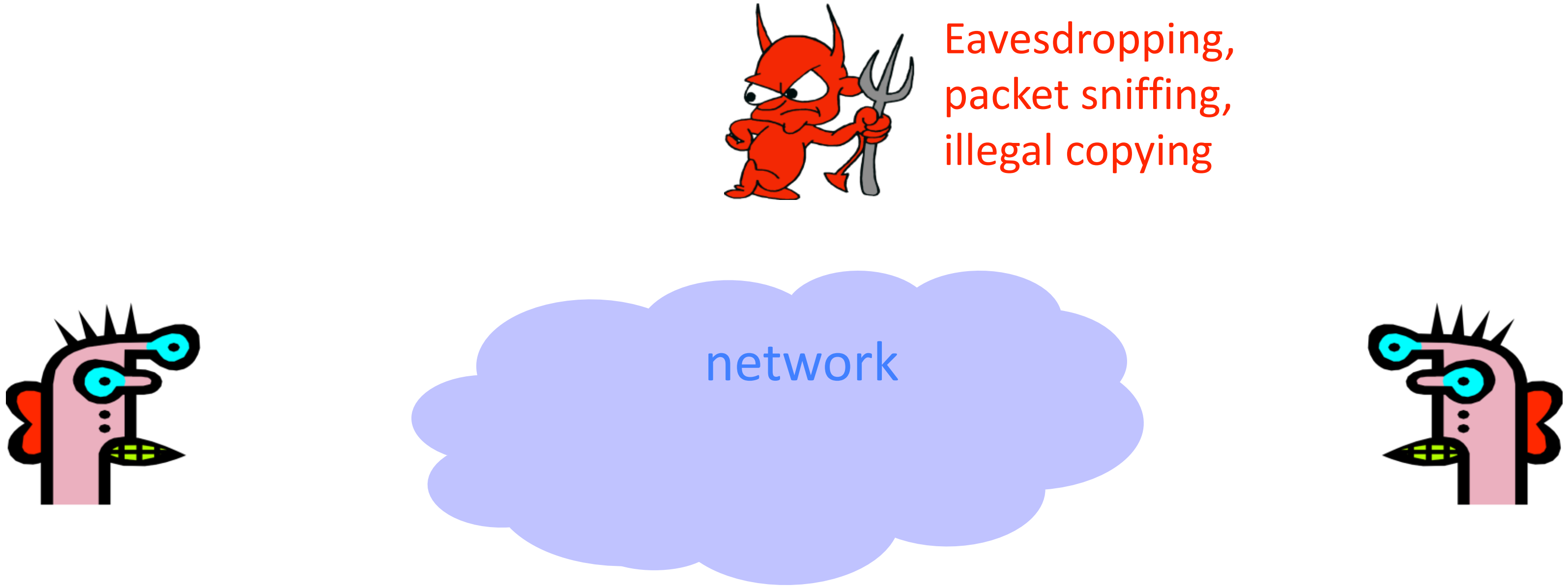
What's *Security*, Anyway?

- Common general security goals: "CIA"
  - Confidentiality
  - Integrity
  - Availability

- Or the extension: CPIAAU (Parkerian Hexad)
  - Control
  - Authenticity
  - Utility

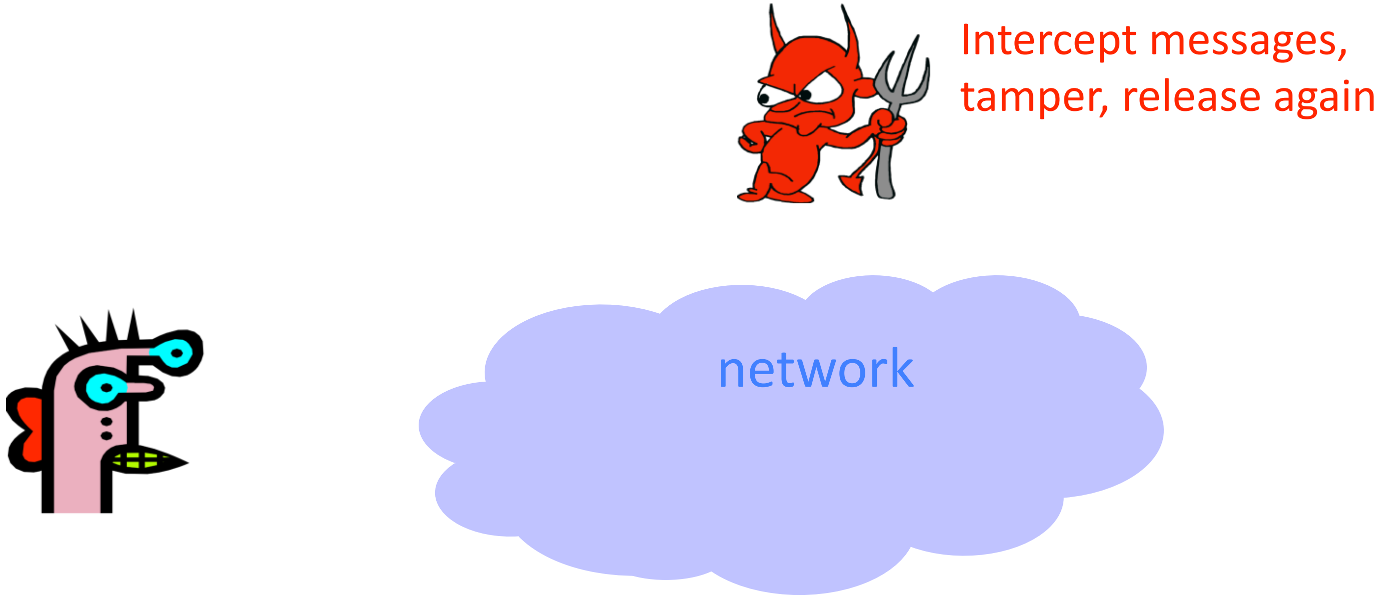
Confidentiality (Privacy)

- Confidentiality is concealment of information.



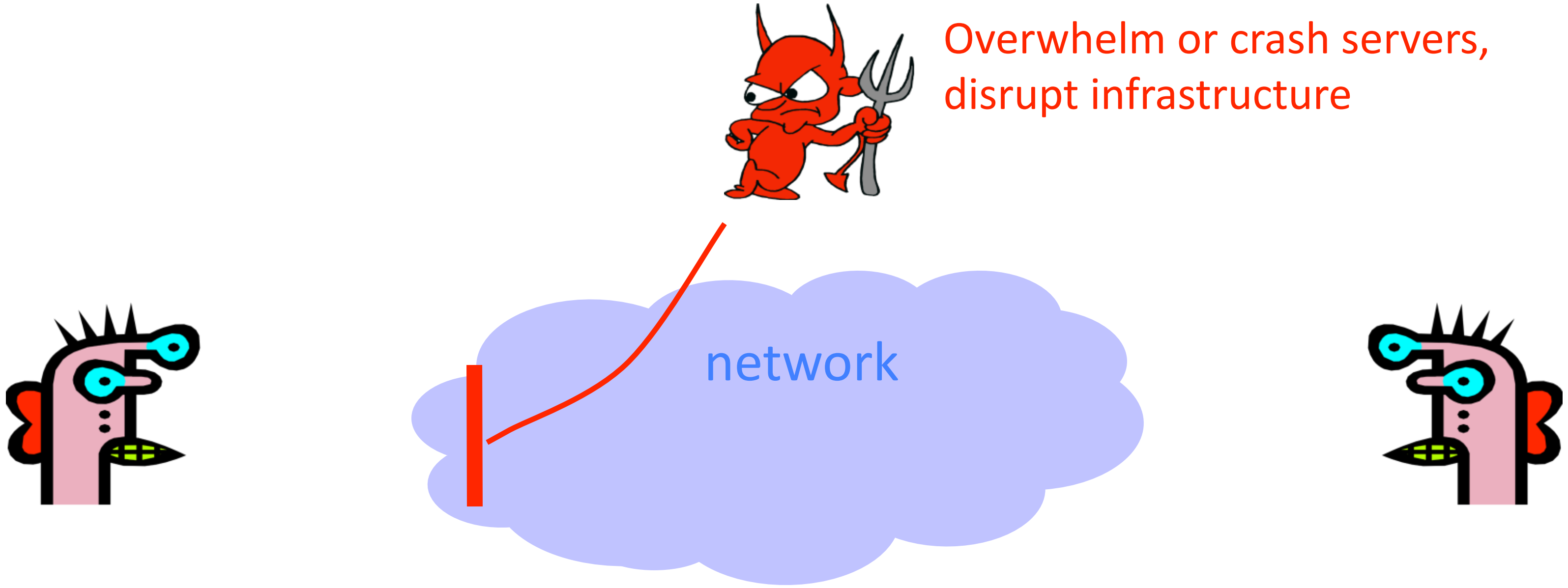
Integrity

- Integrity is prevention of unauthorized changes.



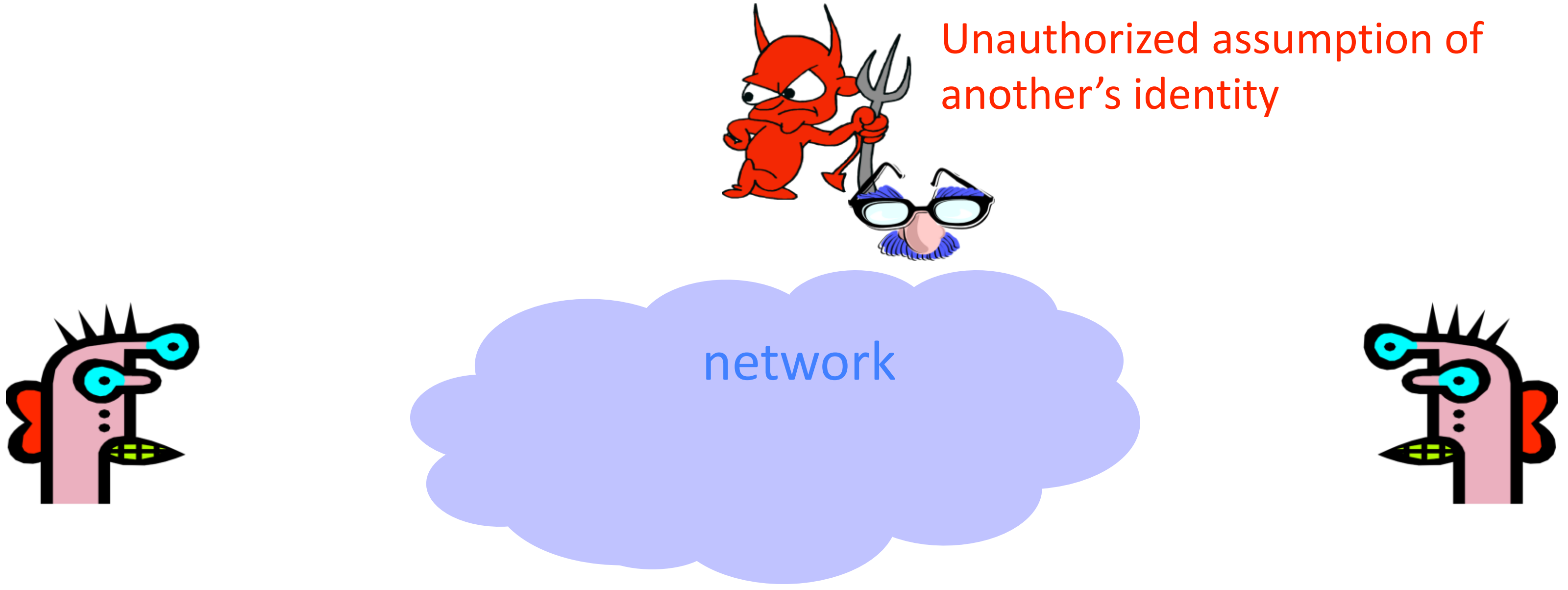
Availability

- Availability is ability to use information or resources.



Authenticity

- Authenticity is knowing who you're talking to.



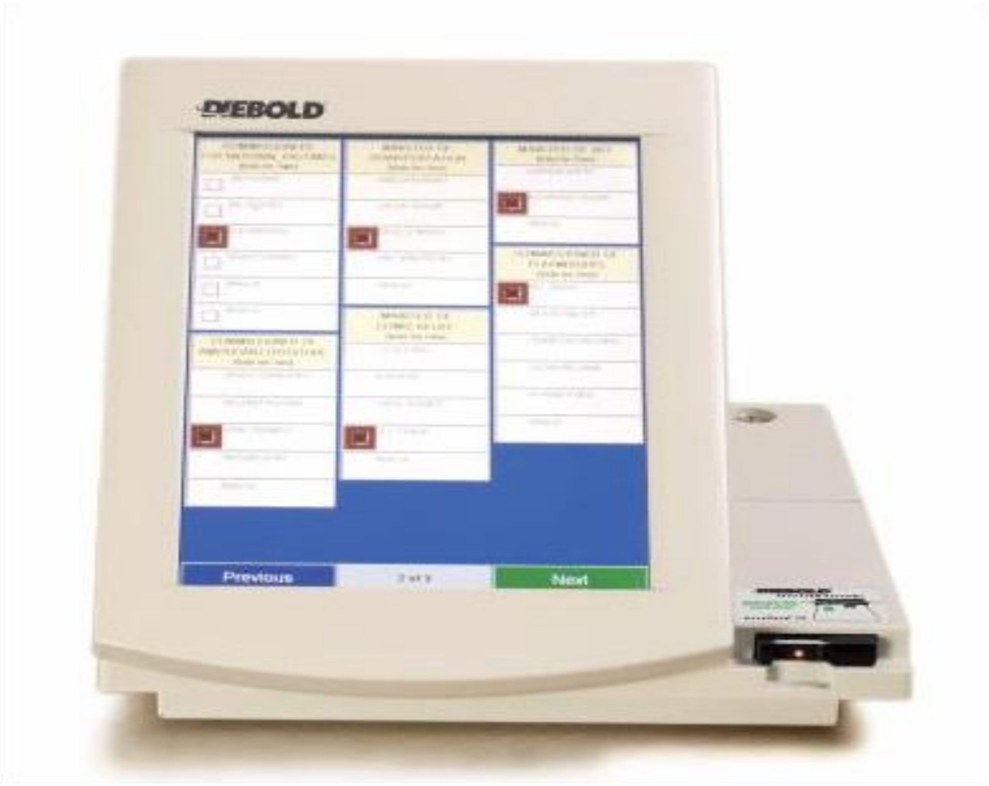
Threat Modeling

- There’s no such thing as perfect security
  - But, attackers have limited resources
  - **Make them pay unacceptable costs / take on unacceptable risks to succeed!**

- Defining security per context: identify assets, adversaries, motivations, threats, vulnerabilities, risk, possible defenses

Threat Modeling Example: Electronic Voting

- Popular replacement to traditional paper ballots

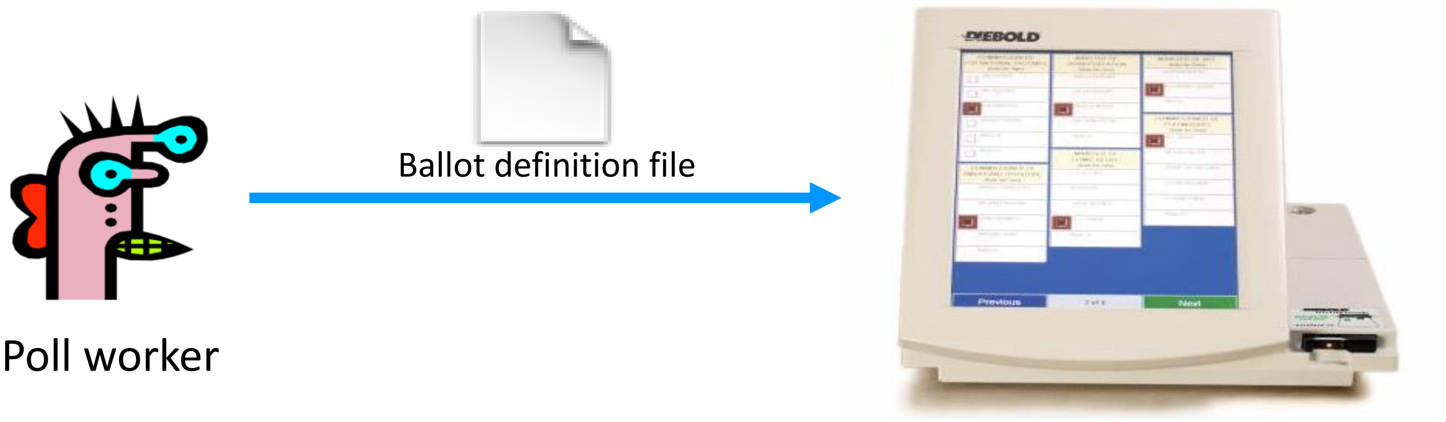


AP

Before we get into the system itself

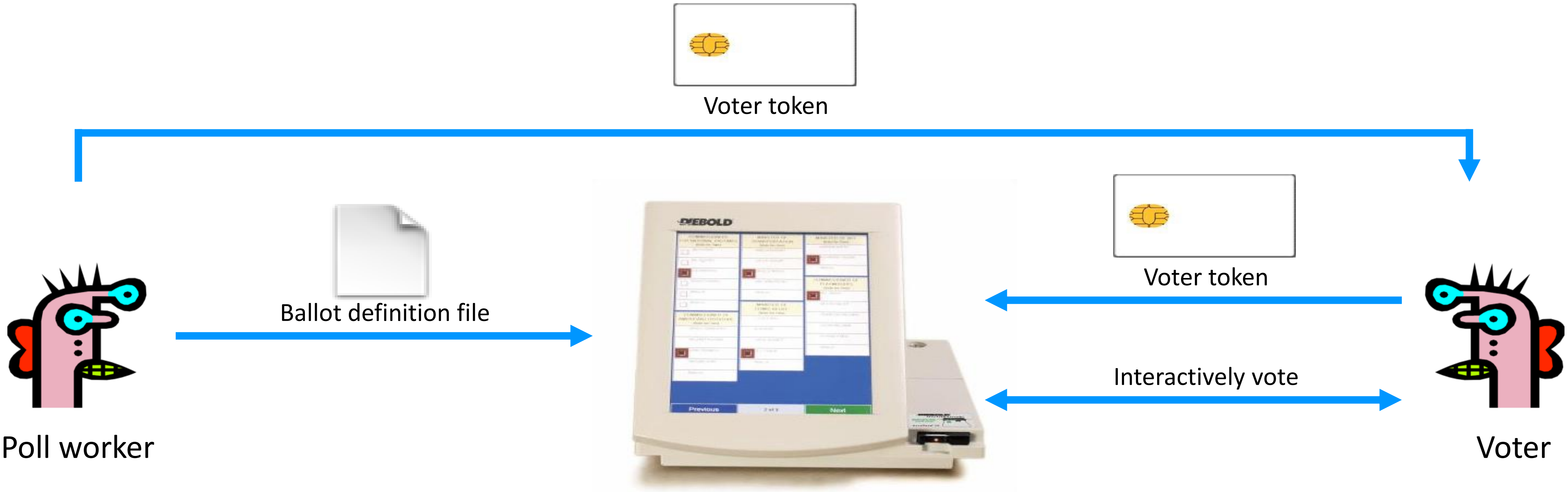
- Think about how in-person electronic voting must work
  - Machines
  - Votes
  - Voters
  - Etc.
- Lightning round threat modeling!
  - Assets?
  - Adversaries?
  - Risks (What might happen if assets are compromised?)
  - Defenses?

Pre-Election



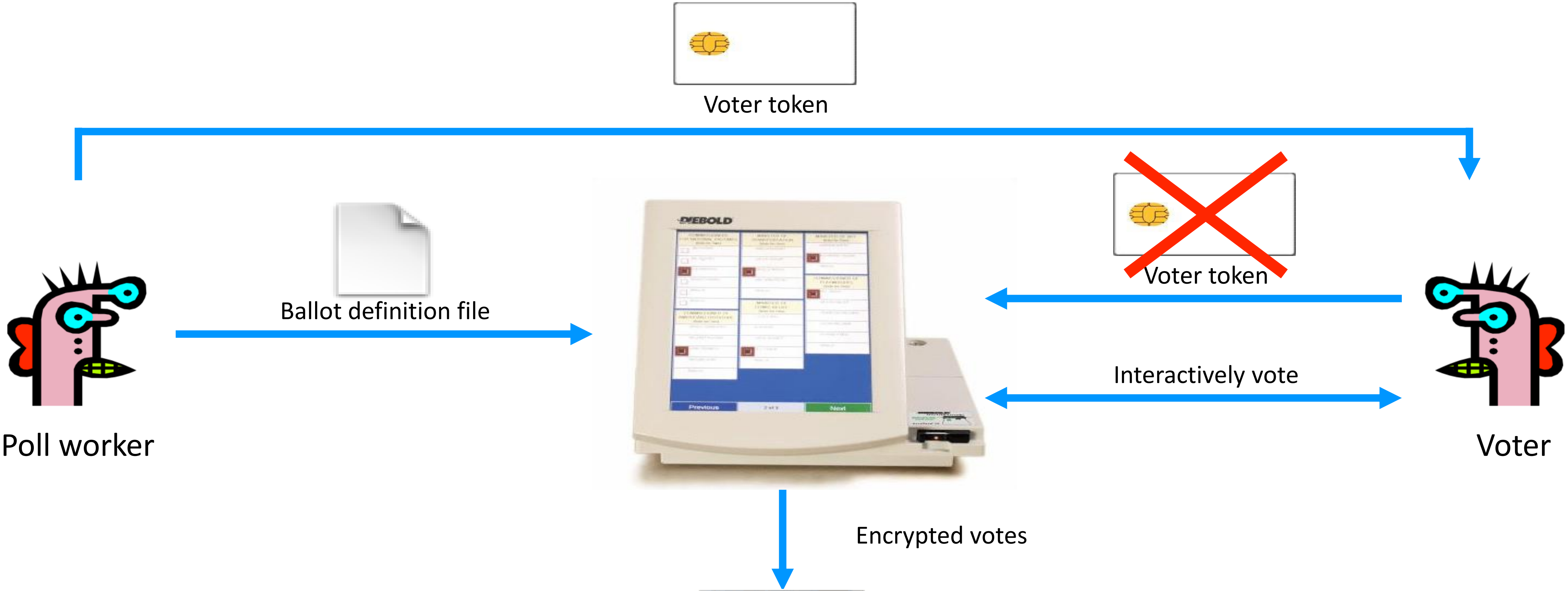
Pre-election: Poll workers load “ballot definition files” on voting machine.

Active Voting



Active voting: Voters obtain single-use tokens from poll workers. Voters use tokens to activate machines and vote.

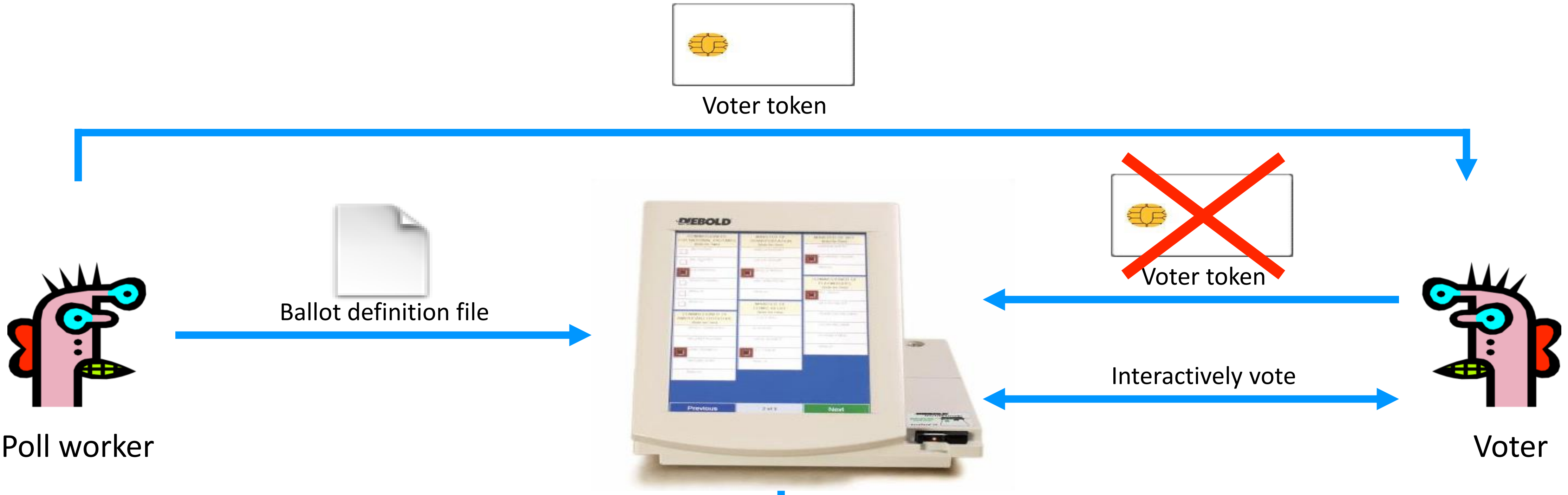
Active Voting

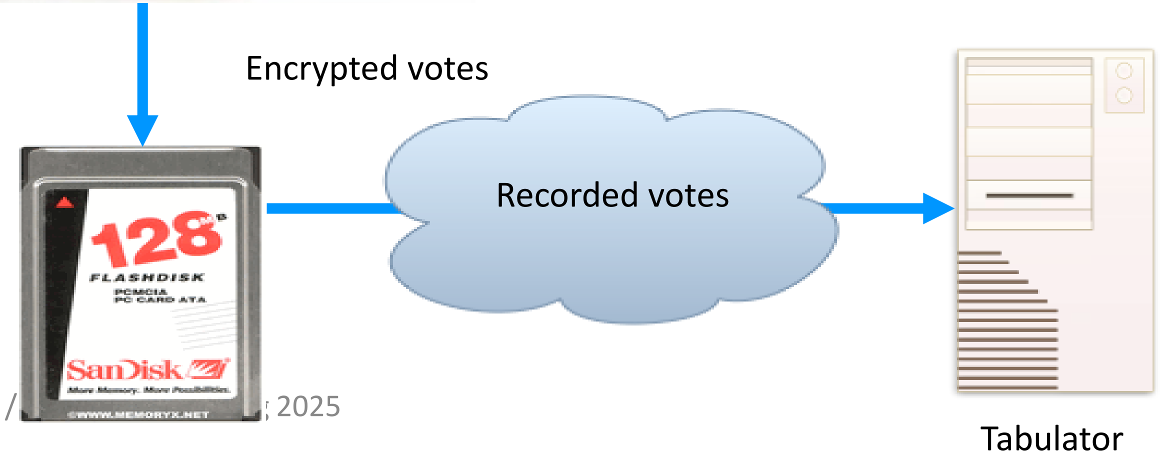


Active voting: Votes encrypted and stored.

Voter token canceled.

Post-Election



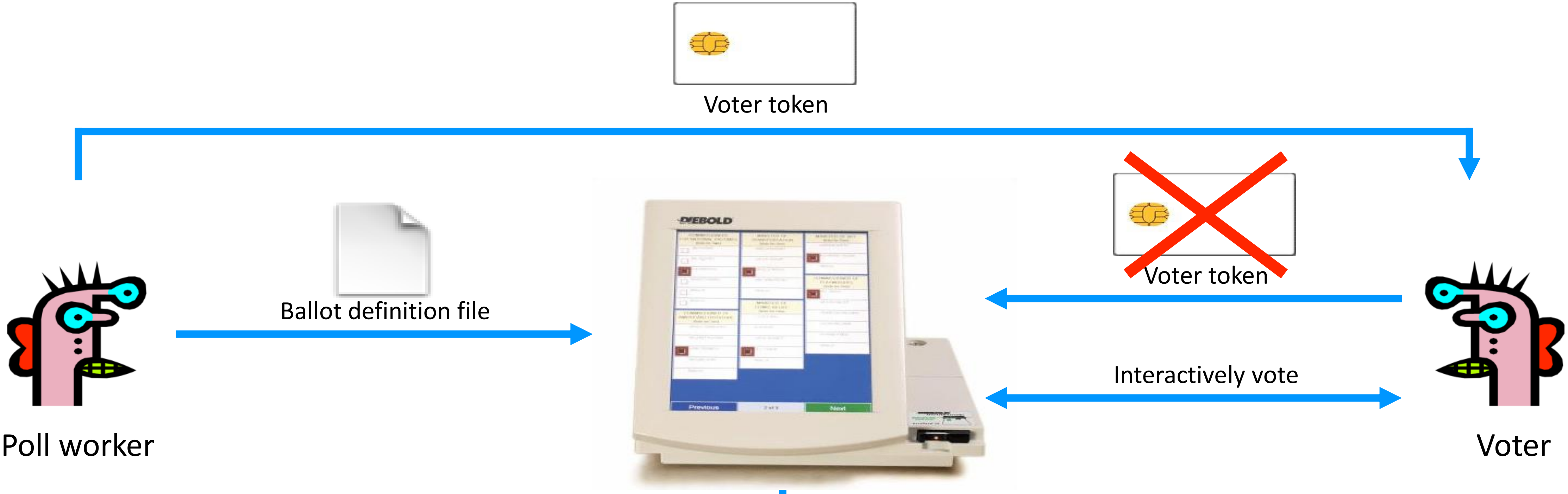


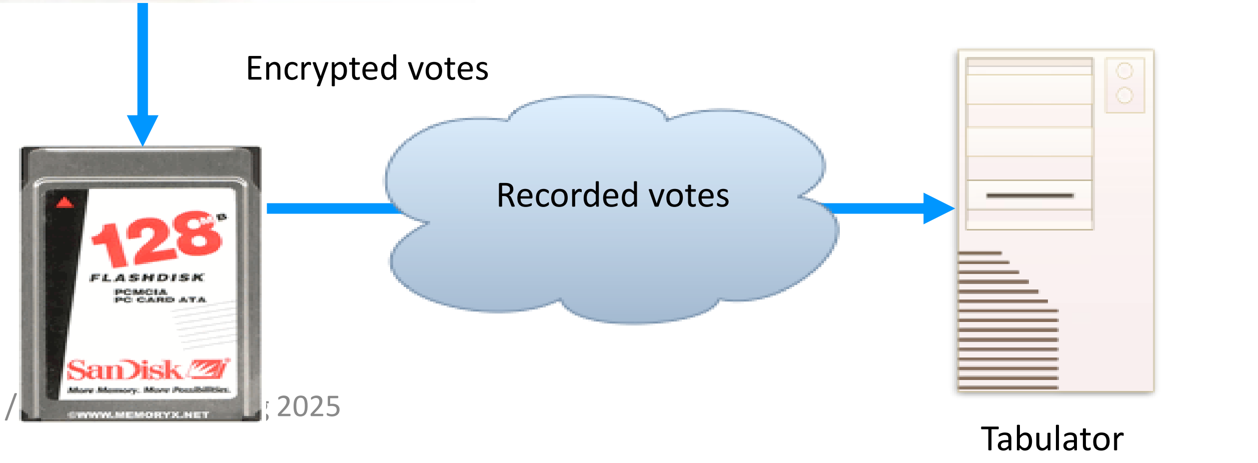
Post-election: Stored votes transported to tabulation center.

Threat modeling – In detail

- Gradescope
- Fill out the questions while discussing with your neighbors
  - Everyone should submit their own
  - **Polish not required, get down some good ideas!**

Post-Election





Post-election: Stored votes transported to tabulation center.


KEYS TO THE KINGDOM
Photo taken from Diebold's online store. The keys that open every Diebold touch-screen voting machine. Working copies have been made from the photo.

The x86(_64)

First technical component of the course

- Understanding classic binary vulnerabilities and exploitation techniques
- We'll be doing everything on x86 (32 or 64bit)
- Code will be in C
  - Lab 1: very little C
  - FP: More C

Reminders

- Manual memory management
- Strings are ‘just’ arrays of bytes, *no length field*
  - Strings only end when there is a NUL(NULL) byte.
- Pointers/integers/etc are all just bytes

C strings

```
void main(int argc, char* argv[]){
    char string1[32];

    memset(string1, 'a', 32);
    string1[31] = 0x00; // or, '\0'

    printf("String1: %s\n", string1);

    memset(string1, 'a', 32);

    printf("String1, again: %s\n", string1);
}
```

Attacks on Memory Buffers

- Buffer is a pre-defined data storage area inside computer memory (stack or heap)
- Typical situation:
  - A function takes some input that it writes into a pre-allocated buffer.
  - The developer forgets to check that the size of the input isn't larger than the size of the buffer.
  - Uh oh.
    - "Normal" bad input: crash
    - "Adversarial" bad input : take control of execution

Stack Buffers

- Suppose Web server contains this function

```
void func(char *str) {
    char buf[126];
    ...
    strcpy(buf,str);
    ...
}
```

- No bounds checking on strcpy()
- If str is longer than 126 bytes
  - Program may crash
  - Attacker may change program behavior

Example: Changing Flags

- Suppose Web server contains this function

```
void func(char *str) {
        byte auth = 0;
    char buf[126];
    ...
    strcpy(buf,str);
    ...
}
```

- Authenticated variable non-zero when user has extra privileges
- Morris worm also overflowed a buffer to overwrite an authenticated flag in fingerd

Memory Layout

- Text region: Executable code of the program
- Heap: Dynamically allocated data
- Stack: Local variables, function return addresses; grows and shrinks as functions are called and return

Stack Buffers

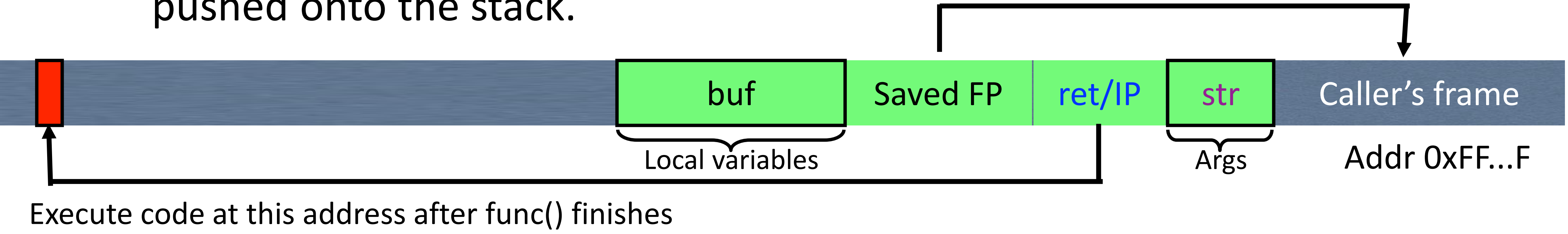
- Suppose Web server contains this function:

```
void func(char *str) {
        char buf[126];
        strcpy(buf,str);
}
```

Allocate local buffer (126 bytes reserved on stack)

Copy argument into local buffer

- When this function is invoked, a new frame (activation record) is pushed onto the stack.



What if Buffer is Overstuffed?

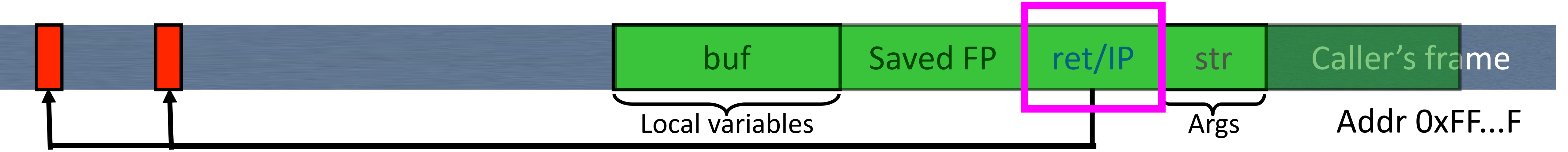
- Memory pointed to by str is copied onto stack…

```
void func(char *str) {
        char buf[126];
        strcpy(buf,str);
}
```

strcpy does NOT check whether the string at *str contains fewer than 126 characters

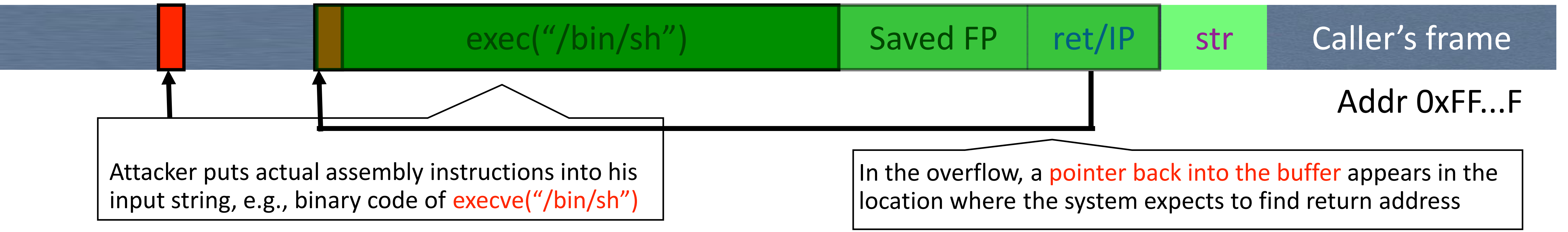
- If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations.

This will be interpreted as return address!



Executing Attack Code

- Suppose buffer contains attacker-created string
  - For example, str points to a string received from the network as the URL



- When function exits, code in the buffer will be executed, giving attacker a shell **("shellcode")**
  - Root shell if the victim program is setuid root