CSE P 590 / CSE M 590 (Spring 2010)

Computer Security and Privacy

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

High-level information

- Instructor: Tadayoshi Kohno (Yoshi)
  - Office: CSE 558
  - Office hours: Thursdays, 5:30 to 6:20pm (right before class)
  - Open door policy – don't hesitate to stop by!
- TAs: Tamara Denning and Karl Koscher
  - Office/hours: Thursdays, 5:30 to 6:20pm (also right before class)
- Course website
  - Assignments, reading materials, ...
- Course email list
  - Announcements
- Course forum
  - Discussion

Prerequisites

- Working knowledge of C and assembly
  - One of the projects involves writing buffer overflow attacks in C
  - You must have detailed understanding of x86 architecture, stack layout, calling conventions, etc.
- Working knowledge of software engineering tools for Unix environments (gdb, etc)
- Working knowledge of Java and JavaScript

Prerequisites

- Strongly recommended: Computer Networks; Operating Systems
  - Will help provide deeper understanding of security mechanisms and where they fit in the big picture
- Recommended: Complexity Theory; Discrete Math; Algorithms
  - Will help with the more theoretical aspects of this course.

Course Logistics (CSE P 590)

- Lectures: Thursdays: 6:30 to 9:20pm
- Security is a contact sport!
- Labs (30% of the grade)
- Homeworks (30% of grade)
- Research (20% of grade)
- Final (20% of the grade)

Exceptional work may be rewarded with extra credit

No make-up or substitute exam!
If you are not sure you will be able to take the exam on the assigned date and time, **do not take this course!**

Course Logistics (CSE M 590)

- Same as before, but...
- Labs (30% of the grade)
- Homeworks (25% of grade)
- Research (25% of grade)
- Final (20% of the grade)

Late Submission Policy

- Late assignments will (generally) be dropped 20% per day.
  - Late days will be rounded up
  - So an assignment turned in 26 hours late will be downgraded 40%.
  - See website / assignment announcements for exceptions
- Everything is generally due on Friday (Research summarizes generally due on Thursdays)

Course Materials

- Textbooks:
  - Daswani, Kern, Kesavan, "Foundations of Security"
  - Ferguson, Schneier, Kohno, "Cryptography Engineering"
  - Additional materials linked to from course website
- Attend lectures.
  - Lectures will not follow the textbooks
  - Lectures will focus on "big-picture" principles and ideas
  - Lectures will cover some material that is not in the textbook

Other Helpful Books (all online)

- Ross Anderson, "Security Engineering" (1st edition)
  - Focuses on design principles for secure systems
  - Wide range of entertaining examples: banking, nuclear command and control, burglar alarms
  - You should all at least look at the Table of Contents for this book.
- Kaashoek and Saltzer, "Principles of Computer System Design"
- Menezes, van Oorschot, and Vanstone, "Handbook of Applied Cryptography"

Others books, movies, …

- Pleasure books include:
  - Little Brother by Cory Doctorow
    - Available online here http://craphound.com/littlebrother/download/
    - Highly recommended reading
  - Cryptonomicon by Neal Stephenson
- Movies include:
  - Hackers
  - Sneakers
  - Diehard 4
  - Wargames
- Historical texts include:
  - The Codebreakers by David Kahn
  - The Code Book by Simon Singh

Ethics

- In this class you will learn about how to attack the security and privacy of (computer) systems.
- Knowing how to attack systems is a <u>critical</u> step toward knowing how to protect systems.
- But one must use this knowledge in an ethical manner.

Mailing List

- Make sure to sign up for the mailing list
- URL for mailing list on course website:
  - http://www.cs.washington.edu/education/courses/csep590c/10sp/administrivia/email.html
- Used for announcements

Forum

- We’ve set up a forum for this course
  - https://catalysttools.washington.edu/gopost/board/kohno/16358/
- Please us it to discuss the homeworks and labs and other general class materials

Homeworks

- Tentative schedule below (future dates subject to change based on progress, etc)
- General plan (tentative):
  - 4 homeworks, approximately once every two weeks
    - April 16, April 30, May 14, May 28
    - First one posted online over the weekend
  - Generally due Fridays at 6:30pm.
  - Submit to Catalyst system (URL on course page)
- http://www.cs.washington.edu/education/courses/csep590c/10sp/homework/index.html

Labs

- Tentative schedule below (future dates subject to change based on progress, etc)
- General plan (tentative):
  - 2 labs
    - May 7, May 28
    - First one posted online by next Monday
  - Generally due Fridays at 6:30pm.
  - Submit to Catalyst system (URL on course page)
- http://www.cs.washington.edu/education/courses/csep590c/10sp/projects/index.html

Labs (tentative plan)

- First lab: Software security
  - Buffer overflow attacks, ...
- Second lab: Web security
  - XSS attacks, ...

Research

- Read approximately 1 research paper every week (at most 2 papers per week)
- Submit review of paper online
- Come prepared to discuss research in class

- http://www.cs.washington.edu/education/courses/csep590c/10sp/homework/index.html

Research

- Contents of review:
  - What problem does the paper address?
  - Important new ideas in the paper, and why
  - Approach used to solve the problem
  - How the paper supports or justifies its arguments
  - Ways paper could be improved
  - Open research questions
- See course web page for more info

What does “security” mean to you?

Two key themes of this course

- How to **think** about security
  - The Security Mindset - “new” way to think about systems
  - Threat models, security goals, assets, risks, adversaries
  - Connection between security, technology, politics, ethics, ...
  - Woven throughout the course. See also:
    - http://cubist.cs.washington.edu/Security/ (last year)
    - https://catalysttools.washington.edu/gopost/board/kohno/14597/ (last quarter)
    - http://slashdot.org/
- **Technical aspects** of security
  - Attack techniques
  - Defenses

How to think about security

- Several approaches for developing “The Security Mindset” and for exploring the broader contextual issues surrounding computer security
  - Security reviews / current events (see Section 1.12 of Ferguson et al)
  - In class discussions
  - Participation in forums

Technical Themes

- Vulnerabilities of computer systems
  - Software problems (buffer overflows); crypto problems; network problems (DoS, worms); people problems (usability, phishing)
- Defensive technologies
  - Protection of information in transit: cryptography, security protocols
  - Protection of networked applications: firewalls and intrusion detection
  - “Defense in depth”

What This Course is <u>Not</u> About

- <u>Not</u> a comprehensive course on computer security
  - Computer security is a <u>broad</u> discipline!
  - Impossible to cover everything in one quarter
  - So be careful -- this course is not a "silver bullet"
- <u>Not</u> about all of the latest and greatest attacks
  - Read bugtraq or other online sources instead
- <u>Not</u> a course on ethical, legal or economic issues
  - We will touch on ethical issues, but the topic is huge
- <u>Not</u> a course on how to "hack" or "crack" systems
  - Yes, we will learn about attacks ... but the ultimate goal is to develop an understanding of attacks so that you can build more secure systems

Two key themes of this course

- How to **think** about security
  - The Security Mindset - “new” way to think about systems
  - Threat models, security goals, assets, risks, adversaries
  - Connection between security, technology, politics, ethics, ...
  - Woven throughout the course. See also:
    - http://cubist.cs.washington.edu/Security/ (last year)
    - https://catalysttools.washington.edu/gopost/board/kohno/14597/ (last quarter)
    - http://slashdot.org/
- **Technical aspects** of security
  - Attack techniques
  - Defenses

What is Computer Security?

- Systems may fail for many reasons
- Reliability deals with accidental failures
- Usability deals with problems arising from operating mistakes made by users
- Security deals with intentional failures created by intelligent parties
  - Security is about computing in the presence of an adversary
  - But security, reliability, and usability are all related

What Drives the Attackers?

- Adversarial motivations:
  - Money, fame, malice, curiosity, politics, terror....
- Fake websites, identity theft, steal money and more
- Control victim’s machine, send spam, capture passwords
- Industrial espionage and international politics
- Access copy-protected movies and videos
- Attack on website, extort money
- Wreak havoc, achieve fame and glory

Challenges: What is "Security?"

- What does security mean?
  - Often the hardest part of building a secure system is figuring out what security means
  - What are the assets to protect?
  - What are the threats to those assets?
  - Who are the adversaries, and what are their resources?
  - What is the security policy?
- Perfect security does not exist!
  - Security is not a binary property
  - Security is about risk management

Current events andsecurity reviews designed to exercise our thinking about these issues

From Policy to Implementation

- After you've figured out what security means to your application, there are still challenges
  - How is the security policy enforced?
  - Design bugs
    - Poor use of cryptography
    - Poor sources of randomness
    - ...
  - Implementation bugs
    - Buffer overflow attacks
    - ...
  - Is the system usable?

Don't forget the users! They are a critical component!

Many Participants

- Many parties involved
  - System developers
  - Companies deploying the system
  - The end users
  - The adversaries (possibly one of the above)
- Different parties have different goals
  - System developers and companies may wish to optimize cost (generalization)
  - End users may desire security, privacy, and usability
  - But the relationship between these goals is quite complex (will customers choose not to buy the product if it is not secure?)

Other (Mutually-Related) Issues

- Do consumers actually care about security?
- Security is expensive to implement
- Plenty of legacy software
- Easier to write “insecure” code
- Some languages (like C) are unsafe

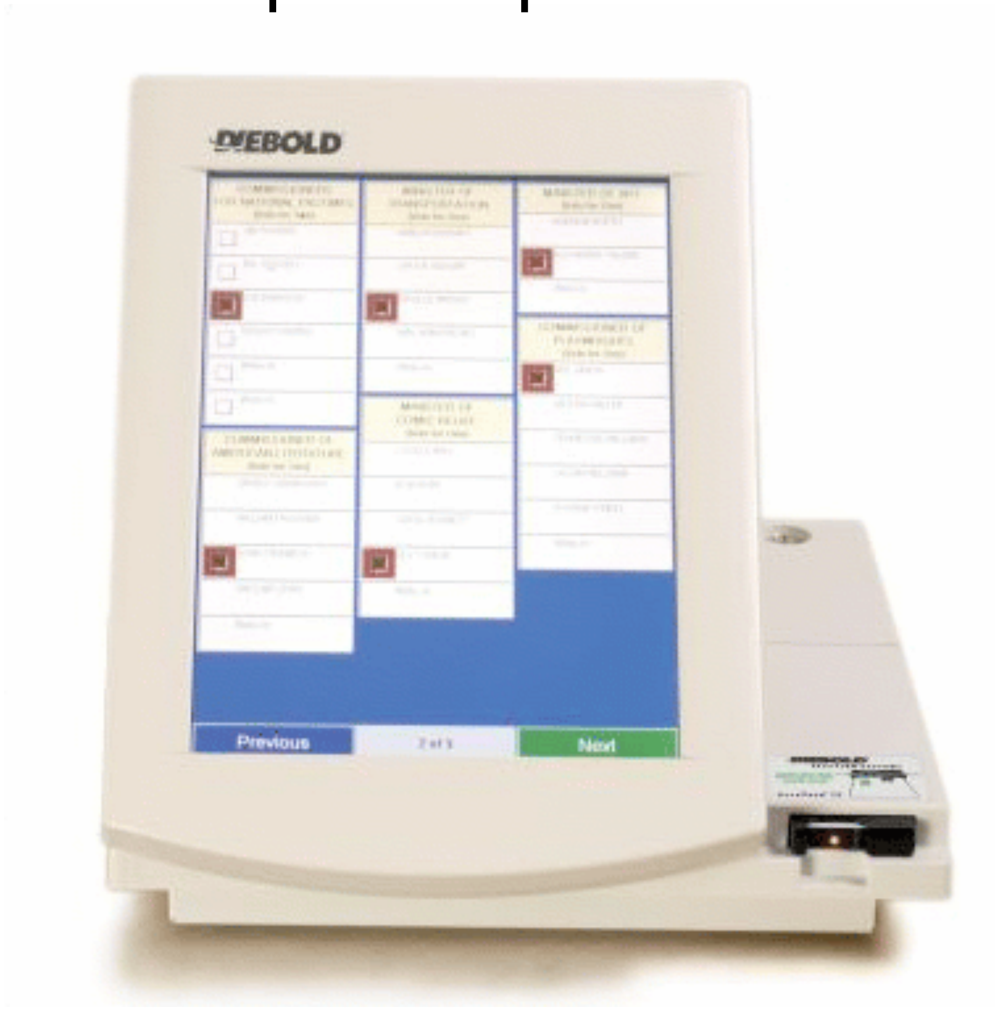
Approaches to Security

- Prevention
  - Stop an attack
- Detection
  - Detect an ongoing or past attack
- Response
  - Respond to attacks

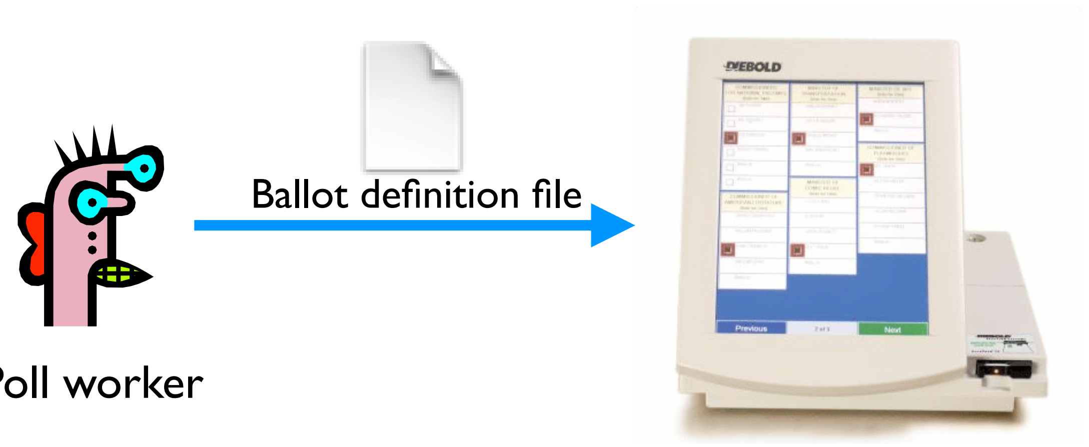
- The threat of a response may be enough to deter some attackers

Example: Electronic Voting

- Popular replacement to traditional paper ballots

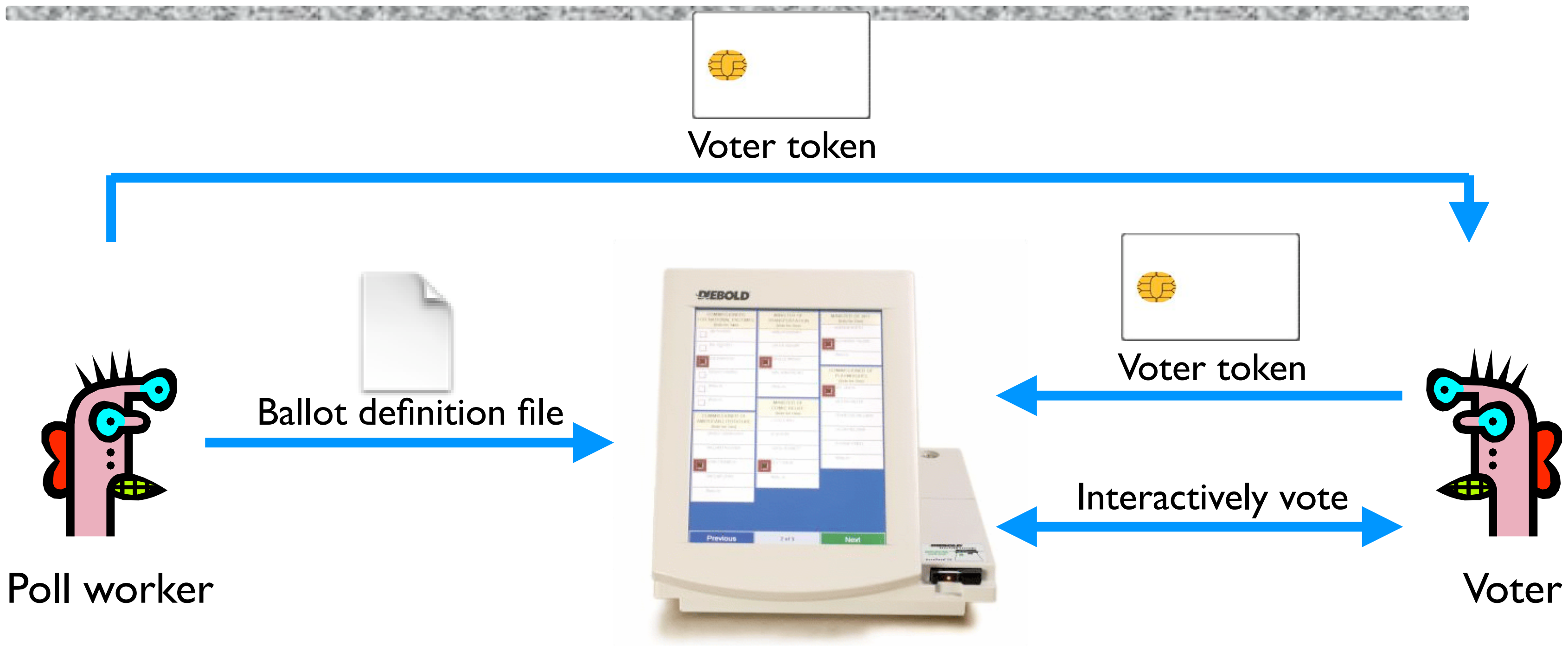


Pre-Election



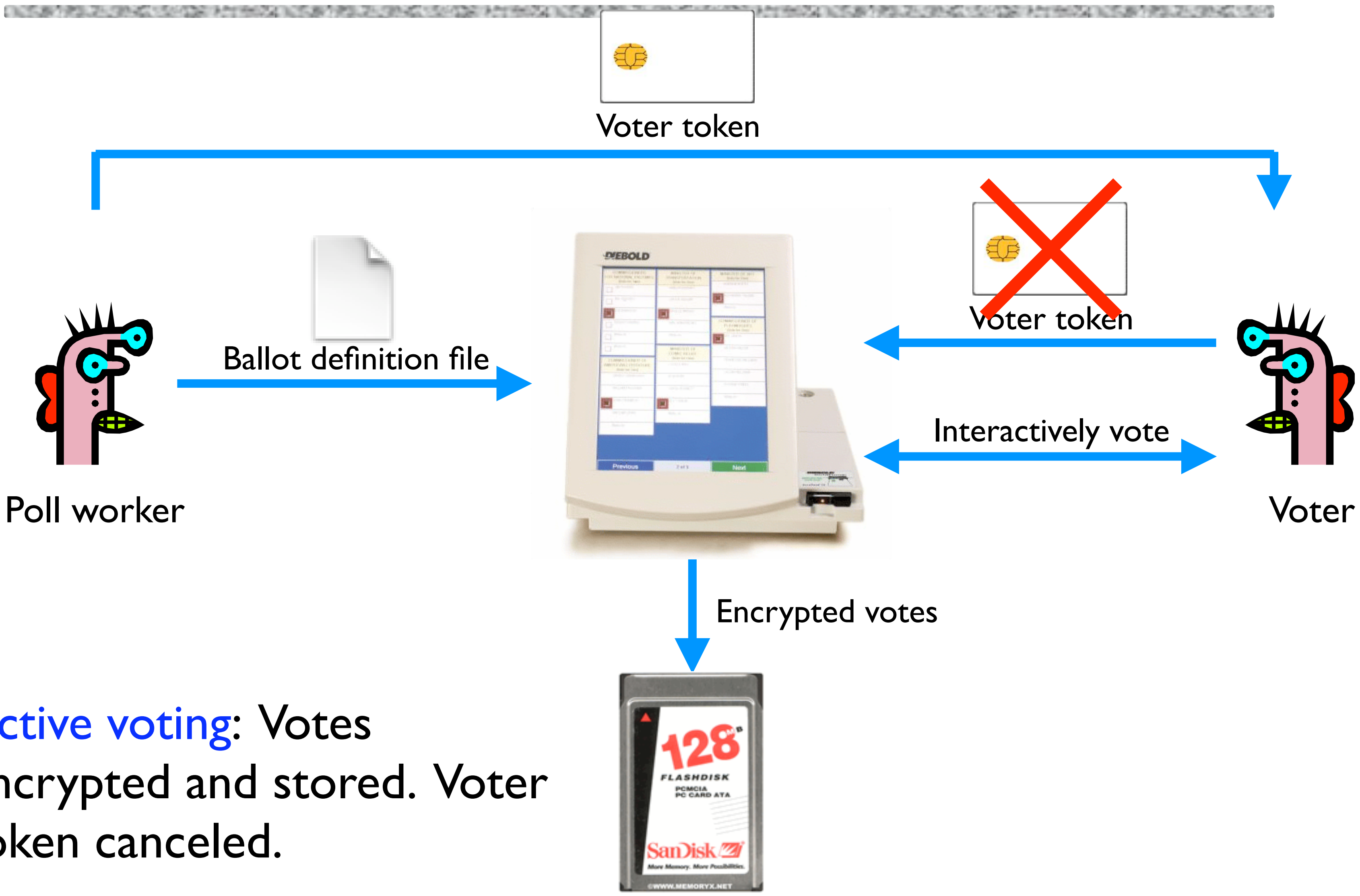
Pre-election: Poll workers load "ballot definition files" on voting machine.

Active Voting



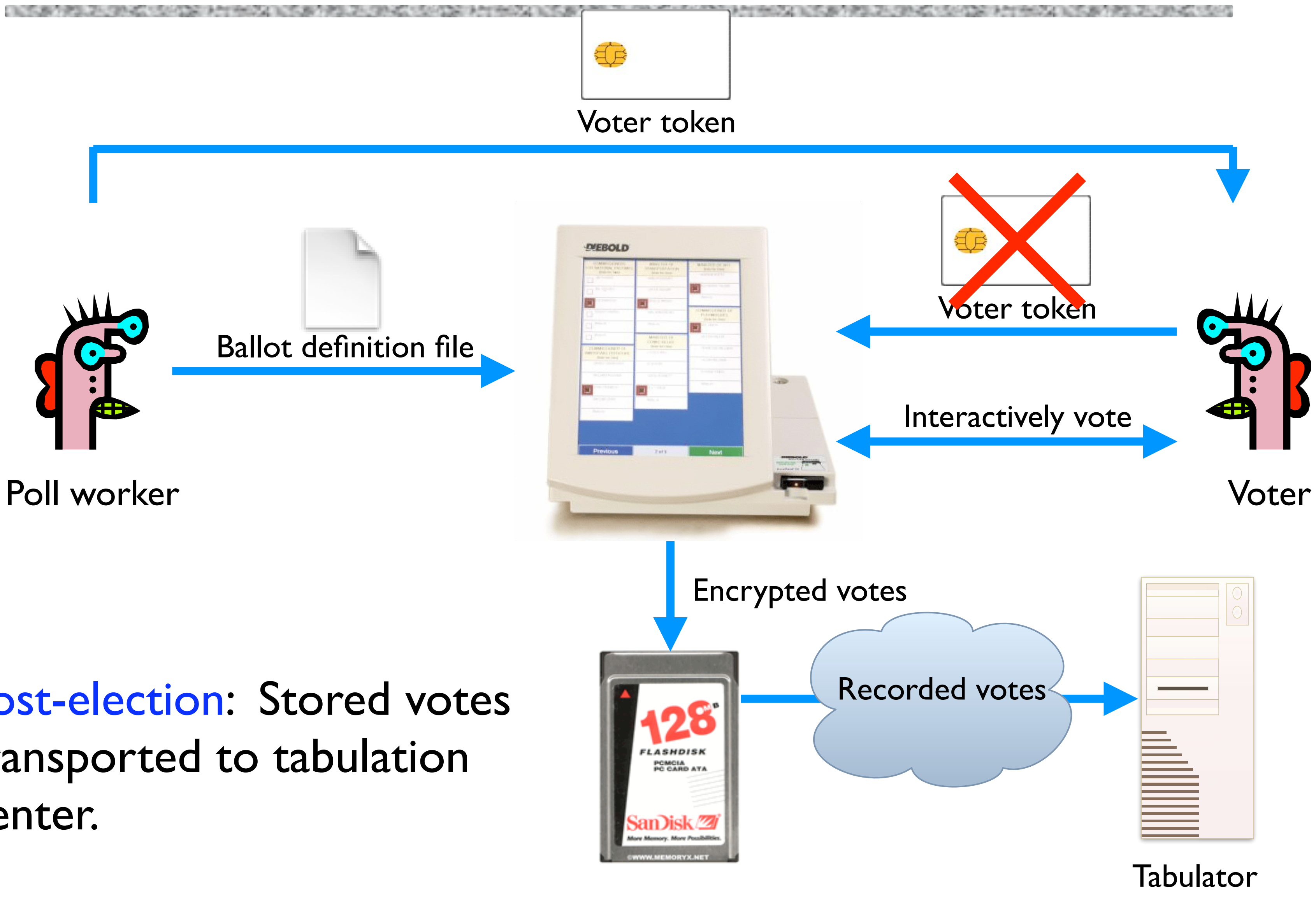
Active voting: Voters obtain single-use tokens from poll workers. Voters use tokens to active machines and vote.

Active Voting



Active voting: Votes encrypted and stored. Voter token canceled.

Post-Election

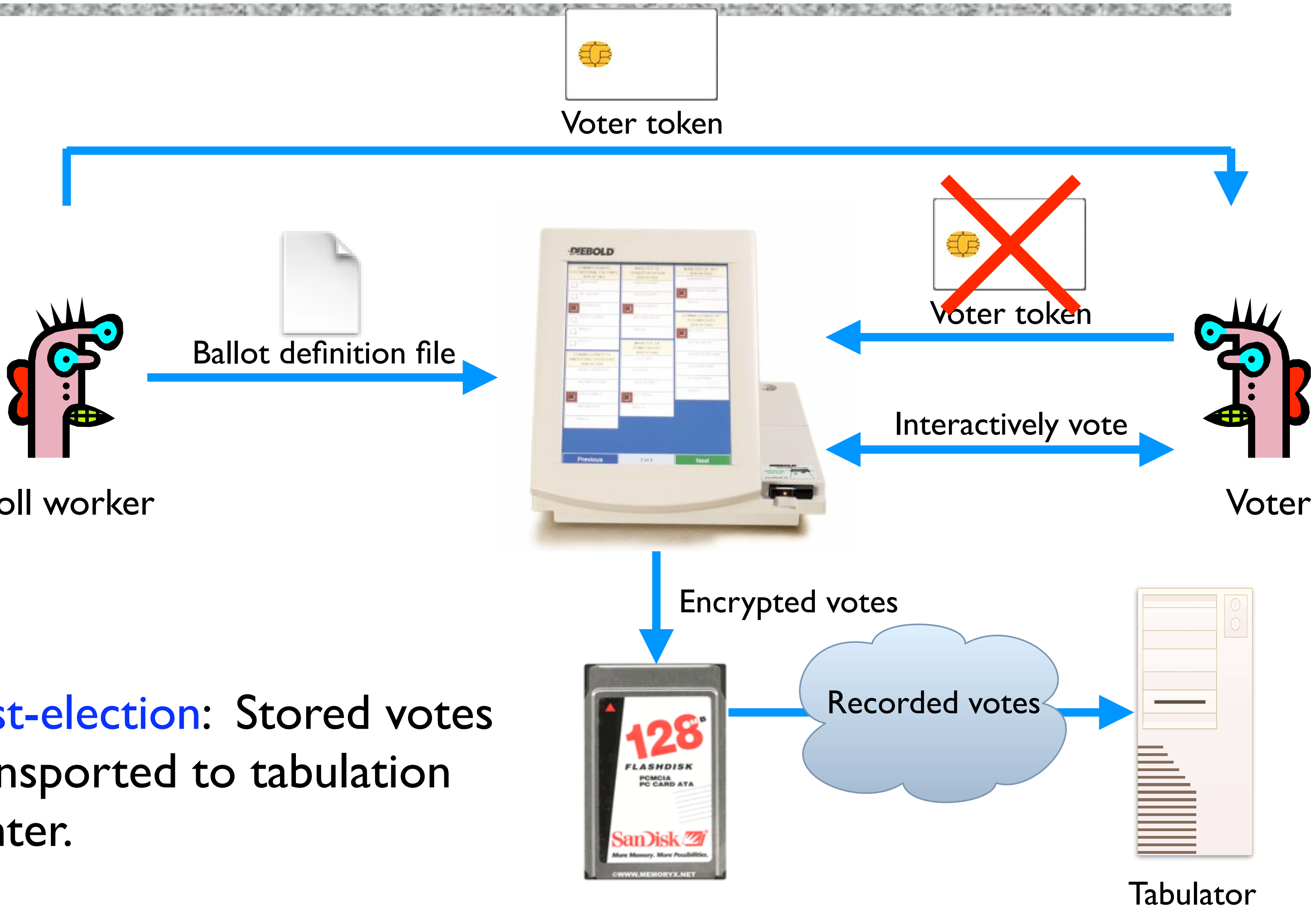


Post-election: Stored votes transported to tabulation center.

Security and E-Voting (Simplified)

- Functionality goals:
  - Easy to use
  - People should be able to cast votes easily, in their own language or with headphones for accessibility
- Security goals:
  - Adversary should not be able to tamper with the election outcome
    - By changing votes
    - By denying voters the right to vote
  - Is it OK if an adversary can do the above, assuming you can catch him or her or them?
  - Adversary should not be able to figure out how voters vote

Can You Spot Any Potential Issues?



Post-election: Stored votes transported to tabulation center.

Potential Adversaries

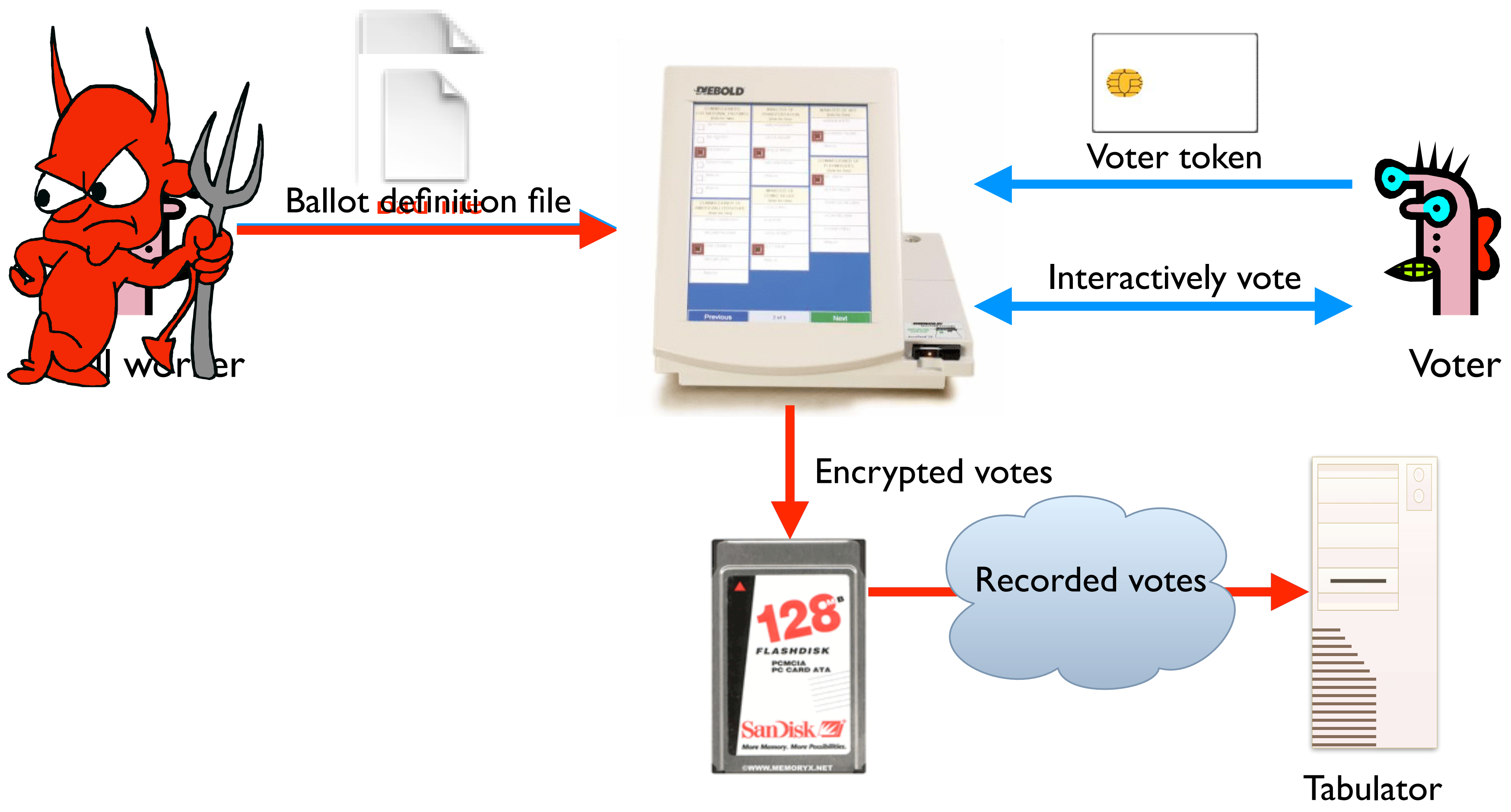
- Voters
- Election officials
- Employees of voting machine manufacturer
  - Software/hardware engineers
  - Maintenance people
- Other engineers
  - Makers of hardware
  - Makers of underlying software or add-on components
  - Makers of compiler
- …
- Or any combination of the above

What Software is Running?

Problem: An adversary (e.g., a poll worker, software developer, or company representative) able to control the software or the underlying hardware could do whatever he or she wanted.

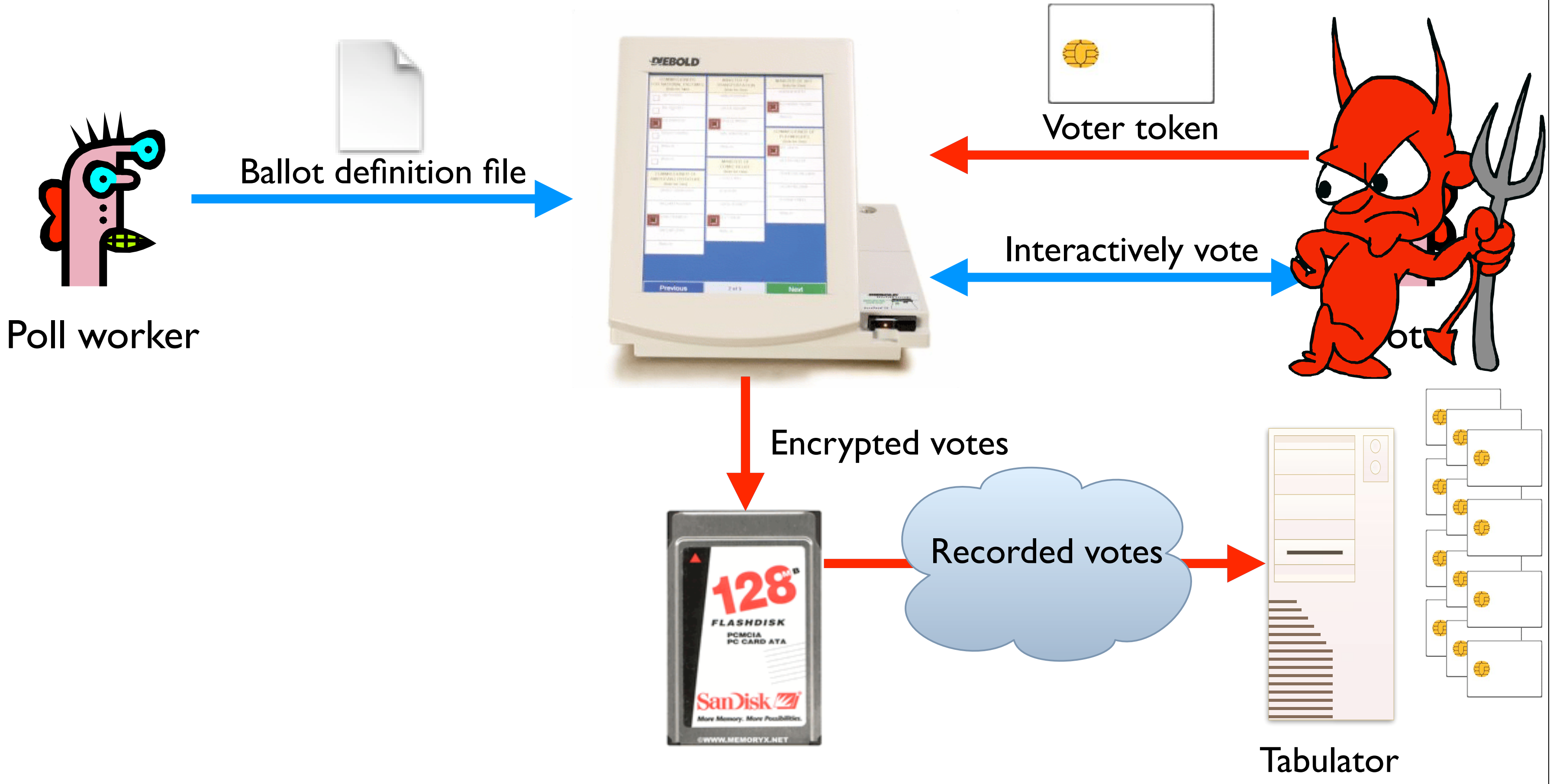
Problem: Ballot definition files are not authenticated.

Example attack: A malicious poll worker could modify ballot definition files so that votes cast for "Mickey Mouse" are recorded for "Donald Duck."



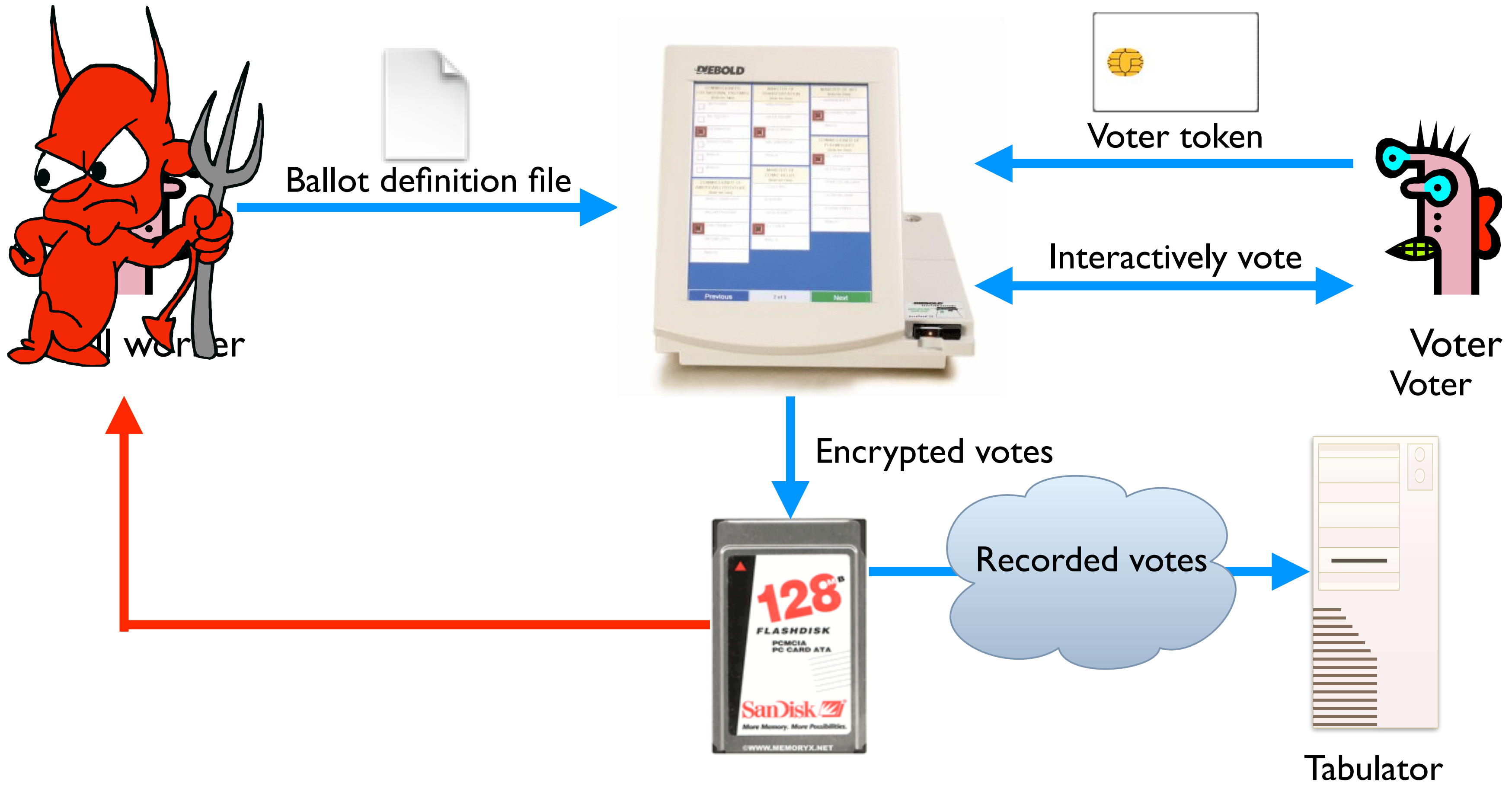
Problem: Smartcards can perform cryptographic operations. But there is no authentication from voter token to terminal.

Example attack: A regular voter could make his or her own voter token and vote multiple times.



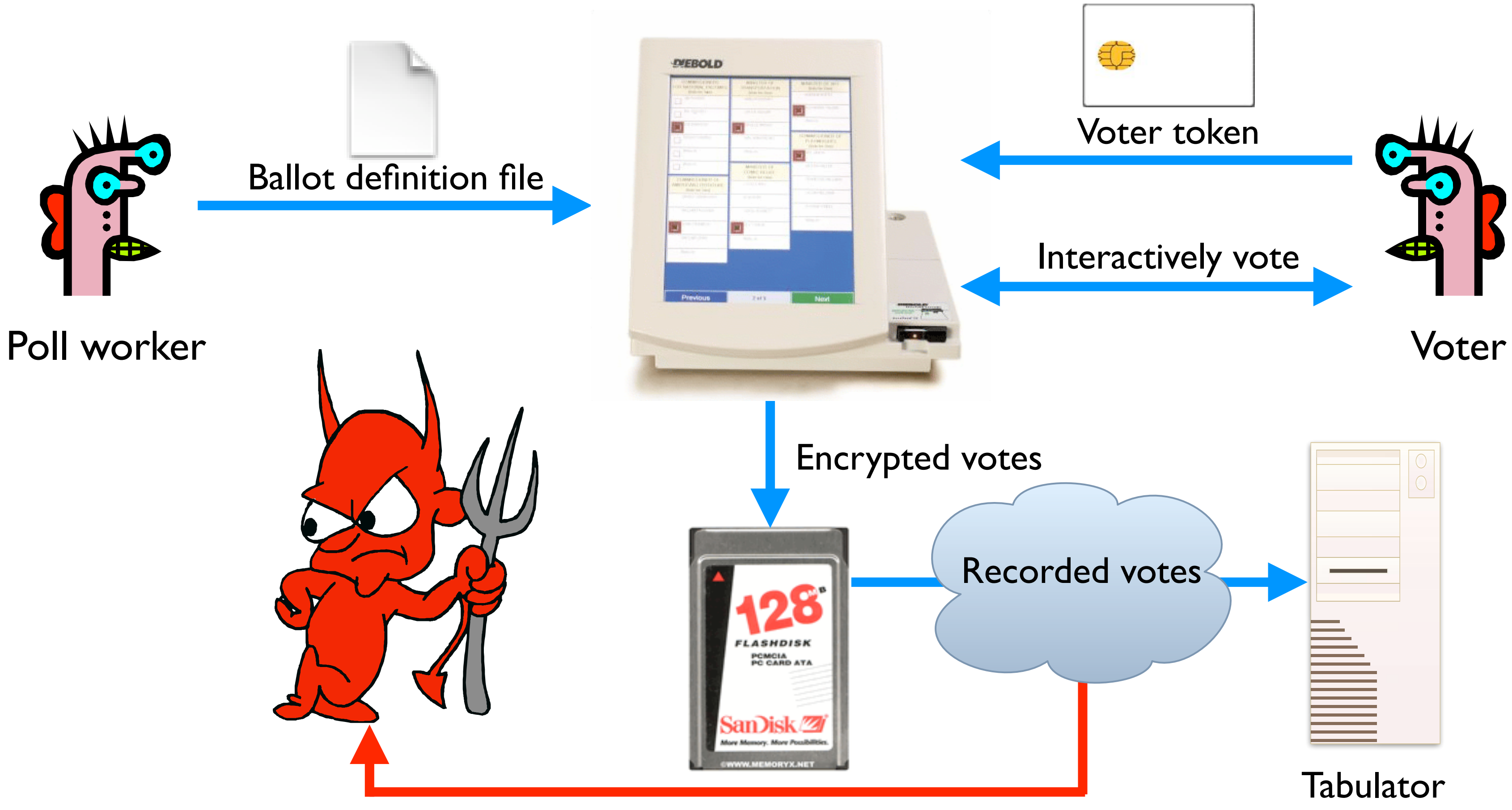
Problem: Encryption key ("F2654hD4") hard-coded into the software since (at least) 1998. Votes stored in the order cast.

Example attack: A poll worker could determine how voters vote.



Problem: When votes transmitted to tabulator over the Internet or a dialup connection, they are decrypted first; the cleartext results are sent the the tabulator.

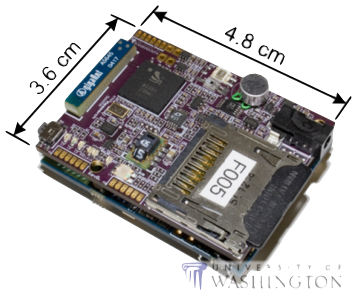
Example attack: A sophisticated outsider could determine how votes vote.



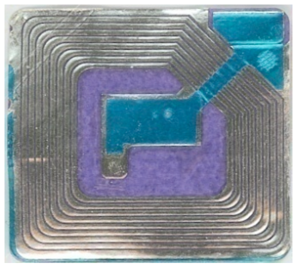
Why do these security issues exist?

What could we do to ward off such issues with future technologies?

Security not just for PCs



mobile sensing platforms



RFID

EEG Gaming

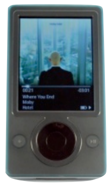
large displays

ambient displays

smart phones

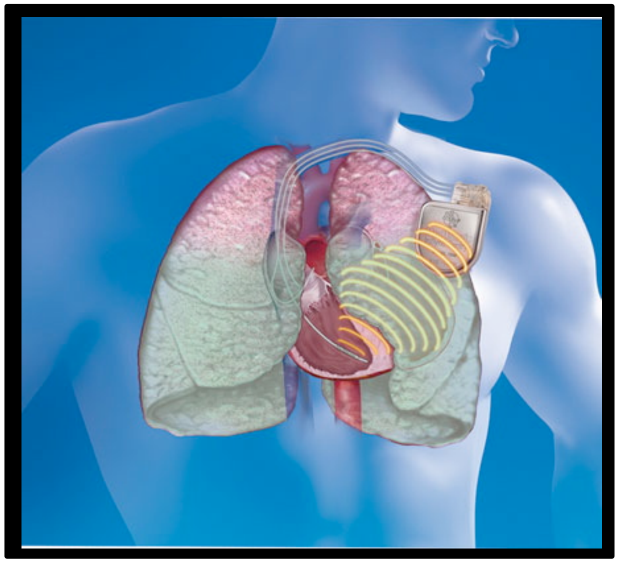
wearables

health displays

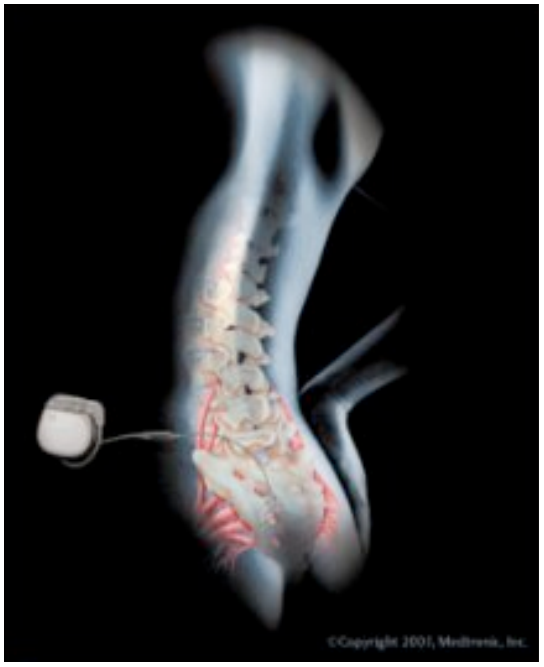


Implantable Medical Devices

Glucose monitors



Pacemakers and defibrillators



Neurostimulators

pumps

Examples of Past Security Reviews

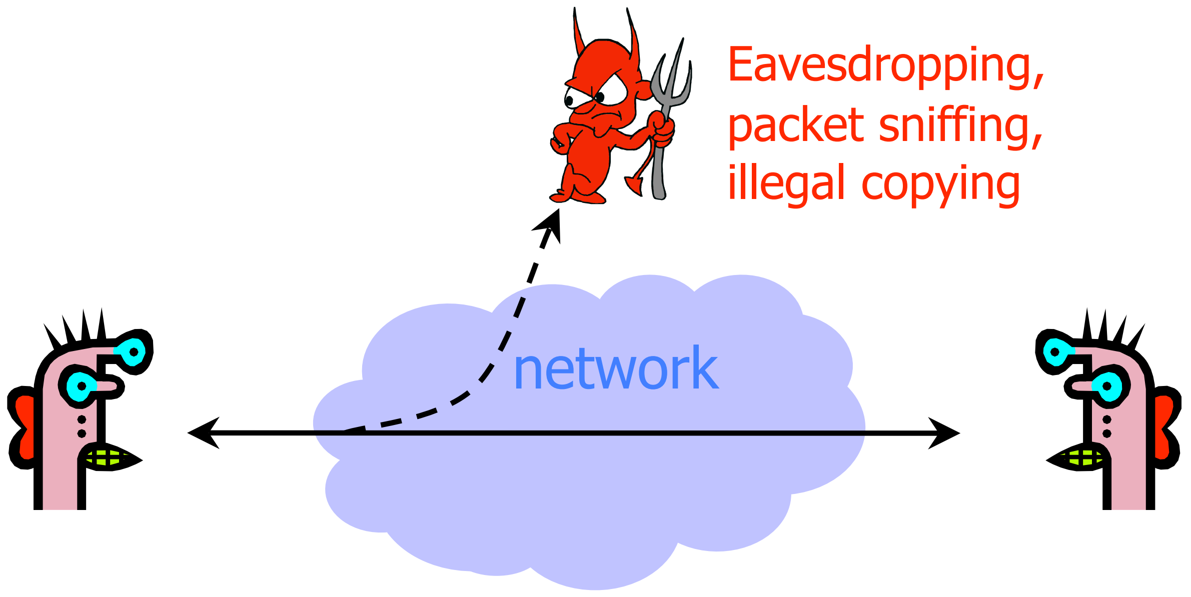
- http://cubist.cs.washington.edu/Security/category/security-reviews/ (Blog through 2009)
- https://catalysttools.washington.edu/gopost/board/kohno/14597/ (Forum for Winter 2010)
  - MyFord: https://catalysttools.washington.edu/gopost/conversation/kohno/328838
  - Caregiver’s Assistant: https://catalysttools.washington.edu/gopost/conversation/kohno/331994
  - Home Automation: https://catalysttools.washington.edu/gopost/conversation/kohno/332004

Next

- Basic security goals
- Thinking more about security

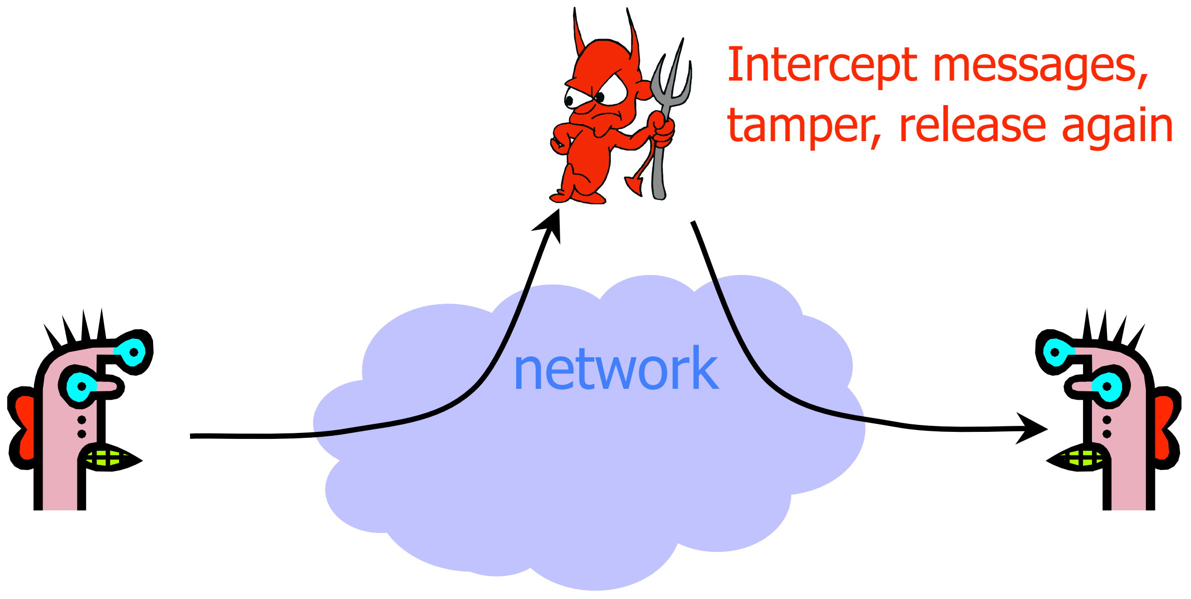
Confidentiality (Privacy)

- Confidentiality is concealment of information



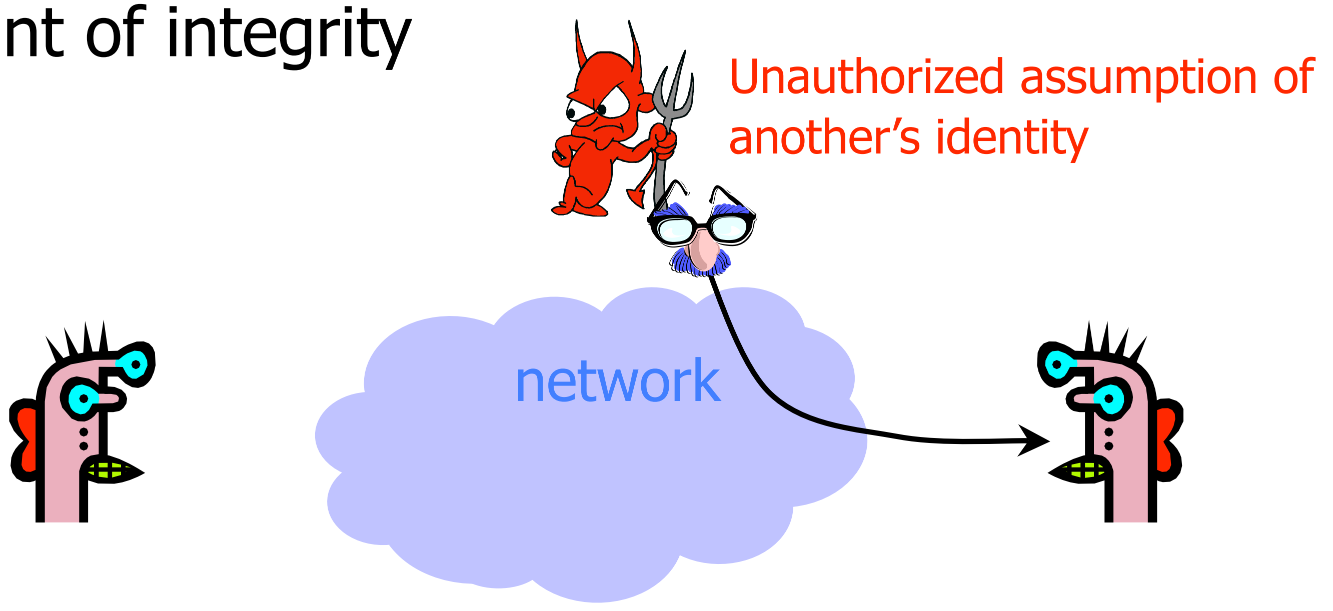
Integrity

- Integrity is prevention of unauthorized changes



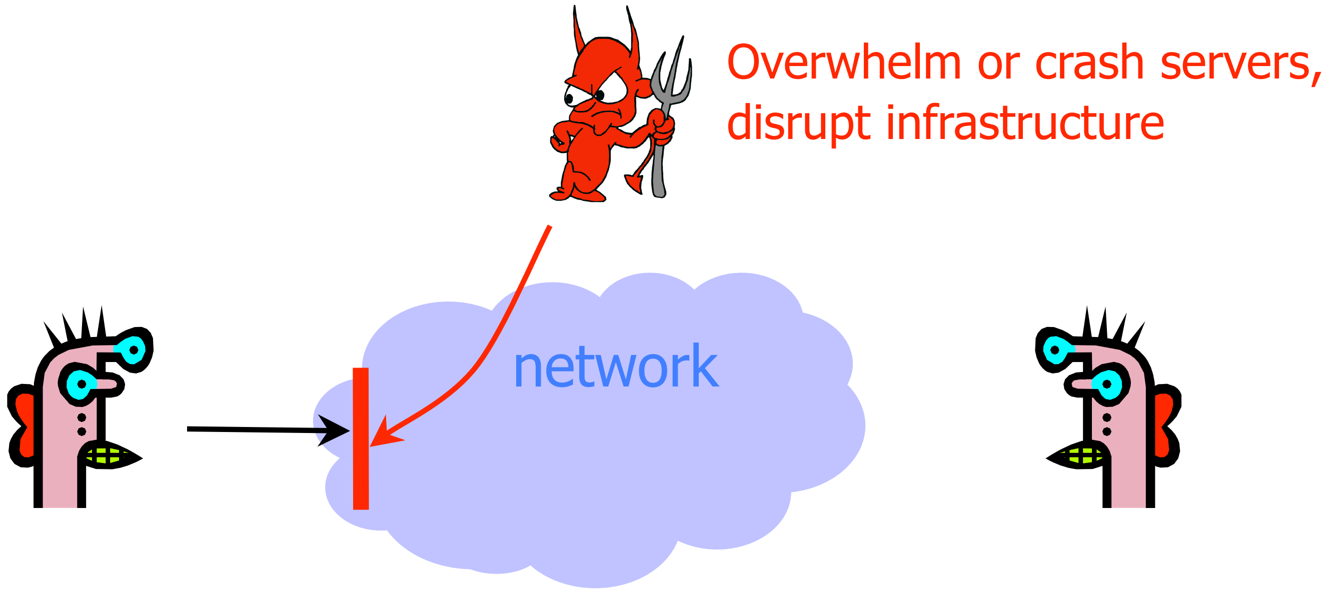
Authenticity

- Authenticity is identification and assurance of origin of information
- Variant of integrity



Availability

- Availability is ability to use information or resources desired



Whole-System is Critical

- Securing a system involves a whole-system view
  - Cryptography
  - Implementation
  - People
  - Physical security
  - Everything in between
- This is because “security is only as strong as the weakest link,” and security can fail in many places
  - No reason to attack the strongest part of a system if you can walk right around it.
  - (Still important to strengthen more than the weakest link)

Analyzing the Security of a System

- First thing: Summarize the system as clearly and concisely as possible
  - Critical step. If you can't summarize the system clearly and concisely, how can you analyze it's security?
- Next steps:
  - Identify the assets: What do you wish to protect?
  - Identify the adversaries and threats
  - Identify vulnerabilities: Weaknesses in the system
  - Calculate the risks

Assets

- Need to know what you are protecting!
  - Hardware: Laptops, servers, routers, PDAs, phones, ...
  - Software: Applications, operating systems, database systems, source code, object code, ...
  - Data and information: Data for running and planning your business, design documents, data about your customers, data about your identity
  - Reputation, brand name
  - Responsiveness
- Assets should have an associated value (e.g., cost to replace hardware, cost to reputation, how important to business operation)

Adversaries

- National governments
- Terrorists
- Thieves
- Business competitors
- Your supplier
- Your consumer
- The New York Times
- Your family members (parents, children)
- Your friends
- Your ex-friends
- …

Threats

- Threats are actions by adversaries who try to exploit vulnerabilities to damage assets
  - Spoofing identities: Attacker pretends to be someone else
  - Tampering with data: Change outcome of election
  - Crash machines: Attacker makes voting machines unavailable on election day
  - Elevation of privilege: Regular voter becomes admin
- Specific threats depend on environmental conditions, enforcement mechanisms, etc
  - You must have a clear, simple, accurate understanding of how the system works!

Threats

- Several ways to classify threats
  - By damage done to the assets
    - Confidentiality, Integrity, Availability
  - By the source of attacks
    - (Type of) insider
    - (Type of) outsider
    - Local attacker
    - Remote attacker
    - Attacker resources
  - By the actions
    - Interception
    - Interruption
    - Modification
    - Fabrication

Vulnerabilities

- Weaknesses of a system that could be exploited to cause damage
  - Accounts with system privileges where the default password has not been changed (Diebold: 1111)
  - Programs with unnecessary privileges
  - Programs with known flaws
  - Known problems with cryptography
  - Weak firewall configurations that allow access to vulnerable services
  - …
- Sources for vulnerability updates: CERT, SANS, Bugtraq, the news(?)

Risks Analyses: Lots of Options

- Quantitative risk analysis
  - Example: $\overset{\text{Risk Exposure}}{\boxed{\text{Risk} =}}\ \overset{\text{Risk Impact}}{\boxed{\text{Asset} \times}}\ \overset{\text{Probability}}{\boxed{\text{Threat} \times \text{Vulnerability}}}$
  - Monetary value to assets
  - Threats and vulnerabilities are probabilities
  - (Yes: Difficult to assign these costs and probabilities)
- Qualitative risk analysis
  - Assets: Critical, very important, important, not important
  - Vulnerabilities: Likely to exit, moderately likely to exist, unlikely to exist
  - Threats: Very likely, likely, unlikely, very unlikely

Helpful Tables

Asset	Confidentiality	Integrity	Availability
Hardware			
Software			
Data			
People			
…			

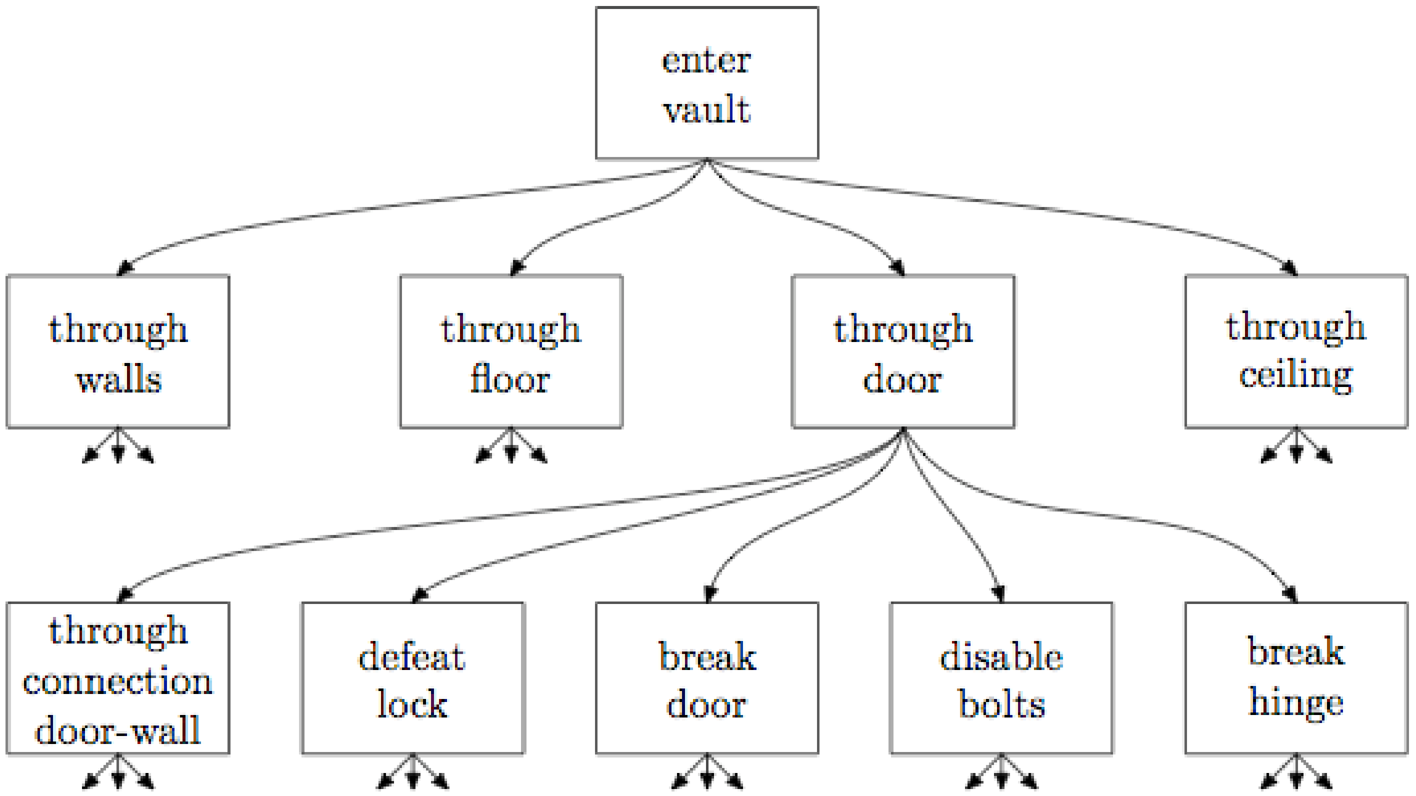
Helpful Tables

	Voter	Election official	…
Privacy of vote			
Integrity of vote			
Availability of voting system			
Confidence in election			
…			

Helpful Tables

	Create New Voter Cards	Decrypt voting record	…
Privacy of vote			
Integrity of vote			
Availability of voting system			
Confidence in election			
…			

Attack Trees



Security is Subtle

- Security attacks can be subtle
- Can’t provably and accurately identify / quantify all risks, vulnerabilities, threats.
- So need to think careful!
  - And keep the whole system in mind
- Phishing one example
  - If attacker can trick user into entering private information, then no protection mechanism will help
  - (So research tries to focus on helping users not be tricked)

On Modularity and Complexity

- Modular design may increase vulnerability
  - Abstraction is difficult to achieve in security: what if the adversary operates below your level of abstraction?
- Modular design may increase security: small TCB
- Complexity may increase vulnerability

One perspective (bad news)

- Security often not a primary consideration
  - Performance and usability take precedence
- Feature-rich systems may be poorly understood
  - Higher-level protocols make mistaken assumptions
- Implementations are buggy
  - Buffer overflows, XSS vulnerabilities, ...
- Networks are more open and accessible than ever
  - Increased exposure, easier to cover tracks
- No matter what technical mechanisms you have, people may circumvent them
  - Phishing, impersonation, write down passwords, ...
- Attackers may be very powerful
  - ISPs, governments, ...
    - –

Better News

- There are a lot of defense mechanisms
  - We'll study some, but by no means all, in this course
- It's important to understand their limitations
  - "If you think cryptography will solve your problem, then you don't understand cryptography… and you don't understand your problem" -- Bruce Schneier
  - Security is not a binary property
  - Many security holes are based on misunderstanding
- Security awareness and user "buy-in" help

Syllabus (Approximate)

- Thinking about security; the "big picture"
  - The hardest part: Getting the "security mindset"
- Software security (including buffer overflow attacks)
- Web security (including XSS attacks)
- Cryptography
- Network security
- Botnets and malware
- The users (including usability)
- Anonymity

Field broad. All parts interconnected, so we will "bounce" around in a methodical way

Security reviews and current events

- Help you develop the “security mindset”
- Best way to learn a foreign language: move to that country and immerse yourself in the language.
- Same thing applies to “security thinking”
- Some homeworks: opportunity to think about security on a regular basis -- outside of “class”
  - When reading current events
  - When hearing about new product announcements
  - While doing regular, day-to-day activities?
    - When you pass a bank, do you start thinking about how you might break in?

Current Events

- Important for computer security practitioners (and all computer scientists) to be able to
  - Reflect on the broader context of technology
  - Guide future development of technology
  - Guide future policy
- For the assignment
  - Summarize current event
  - Discuss why event arose
  - Reflect on what could have been done prior to the event arising (to prevent, deter, or change consequences)
  - Describe broader issues surrounding current event (ethical, societal)
  - How should people respond to the event (policy makers, the public, companies, etc.)
- Why write down? To go through all the steps at least once

Security Reviews

- Summary of system
- Assets
- Adversaries and threats
- Potential weaknesses (OK to speculate, but make it clear that you are speculating)
- Potential defenses.
- Risks
- Conclusions.

Let’s try thinking about security

- Integrated networks on 787s (let’s assume that they are indeed integrated).
- Wireless Picture Frames: http://seattlewireless.net/~casey/?p=13.
- Smart phones
- Recall steps:
  - First thing: Summarize the system as clearly and concisely as possible
  - Identify the assets: What do you wish to protect?
  - Identify the adversaries and threats
  - Identify vulnerabilities: Weaknesses in the system
  - Calculate the risks (we’ll do informally)

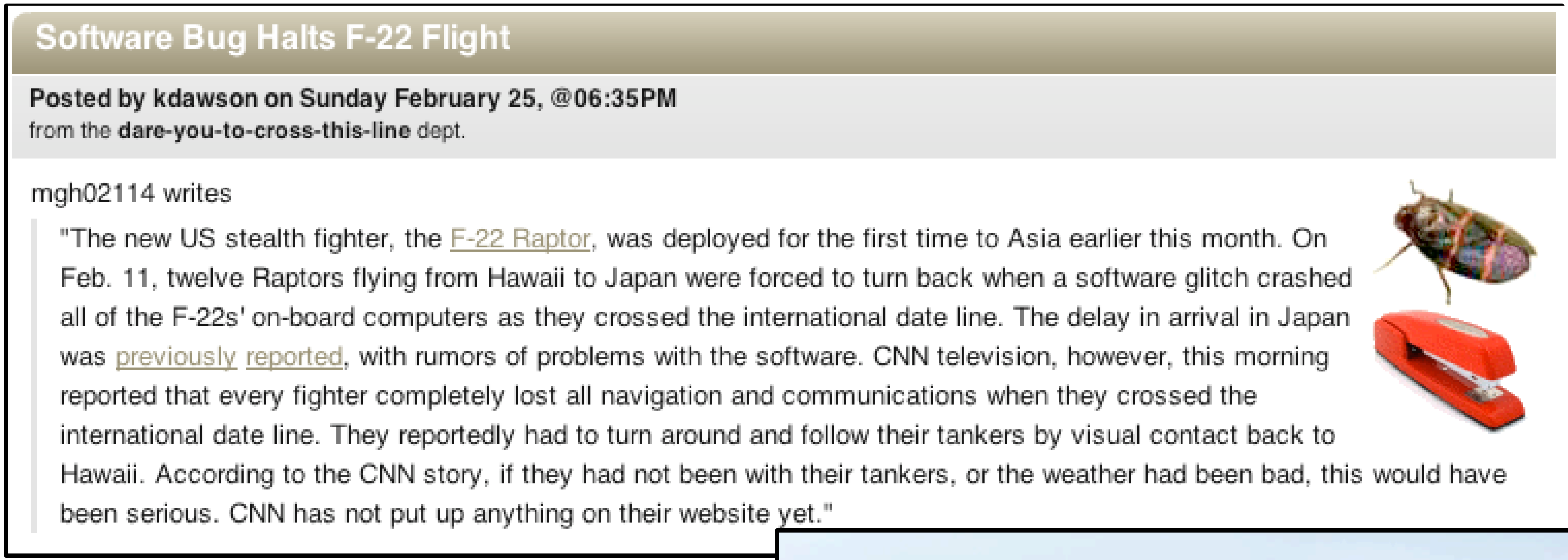
Next

- Software security
  - Software lifecycle
  - Buffer overflow attacks
  - Other software security issues

Software Lifecycle (Simplified)

- Requirements
- Design
- Implementation
- Testing
- Use

Software problems are ubiquitous



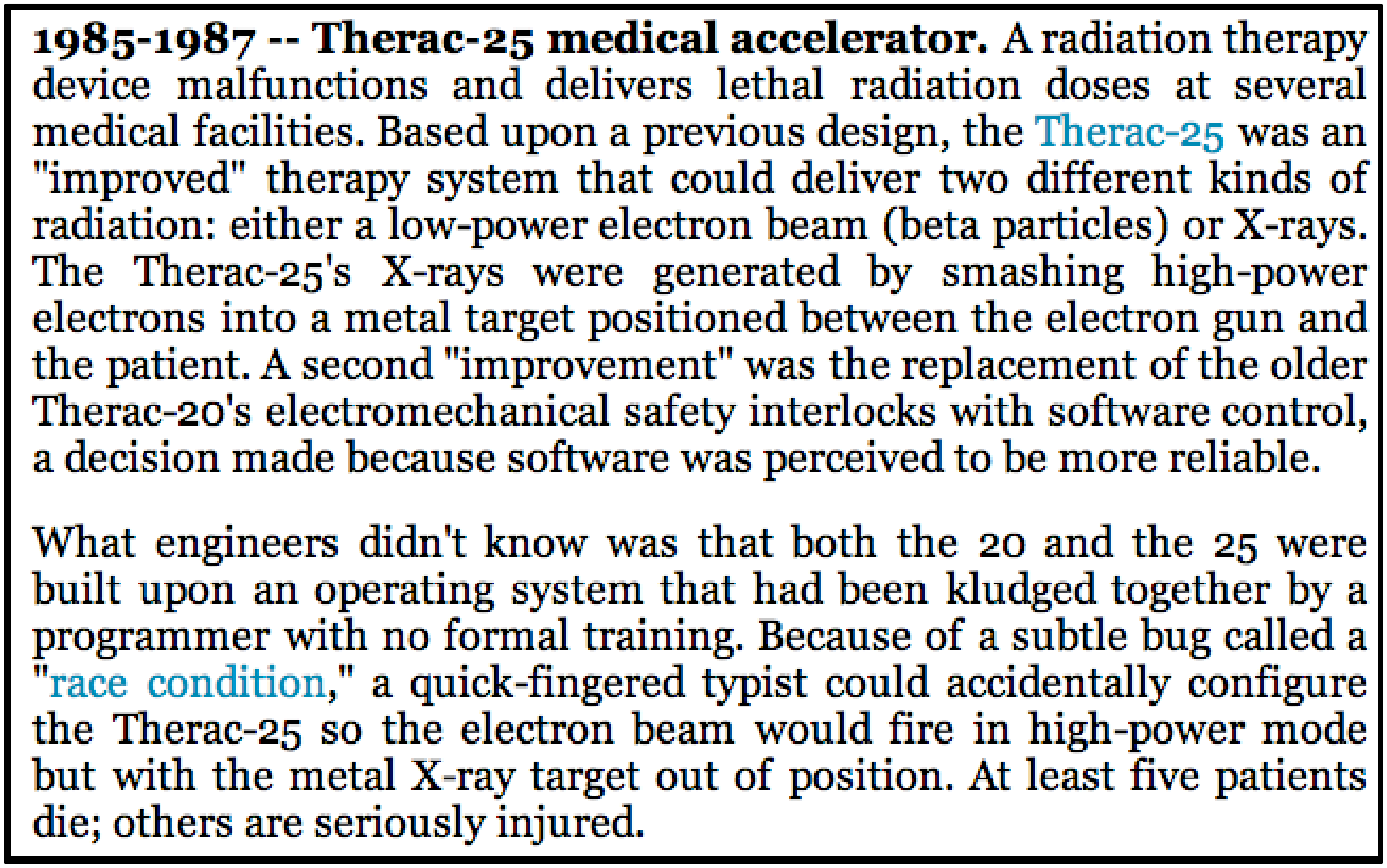
Software Bug Halts F-22 Flight

Posted by kdawson on Sunday February 25, @06:35PM
from the **dare-you-to-cross-this-line** dept.

mgh02114 writes

> "The new US stealth fighter, the F-22 Raptor, was deployed for the first time to Asia earlier this month. On Feb. 11, twelve Raptors flying from Hawaii to Japan were forced to turn back when a software glitch crashed all of the F-22s' on-board computers as they crossed the international date line. The delay in arrival in Japan was previously reported, with rumors of problems with the software. CNN television, however, this morning reported that every fighter completely lost all navigation and communications when they crossed the international date line. They reportedly had to turn around and follow their tankers by visual contact back to Hawaii. According to the CNN story, if they had not been with their tankers, or the weather had been bad, this would have been serious. CNN has not put up anything on their website yet."

Software problems are ubiquitous

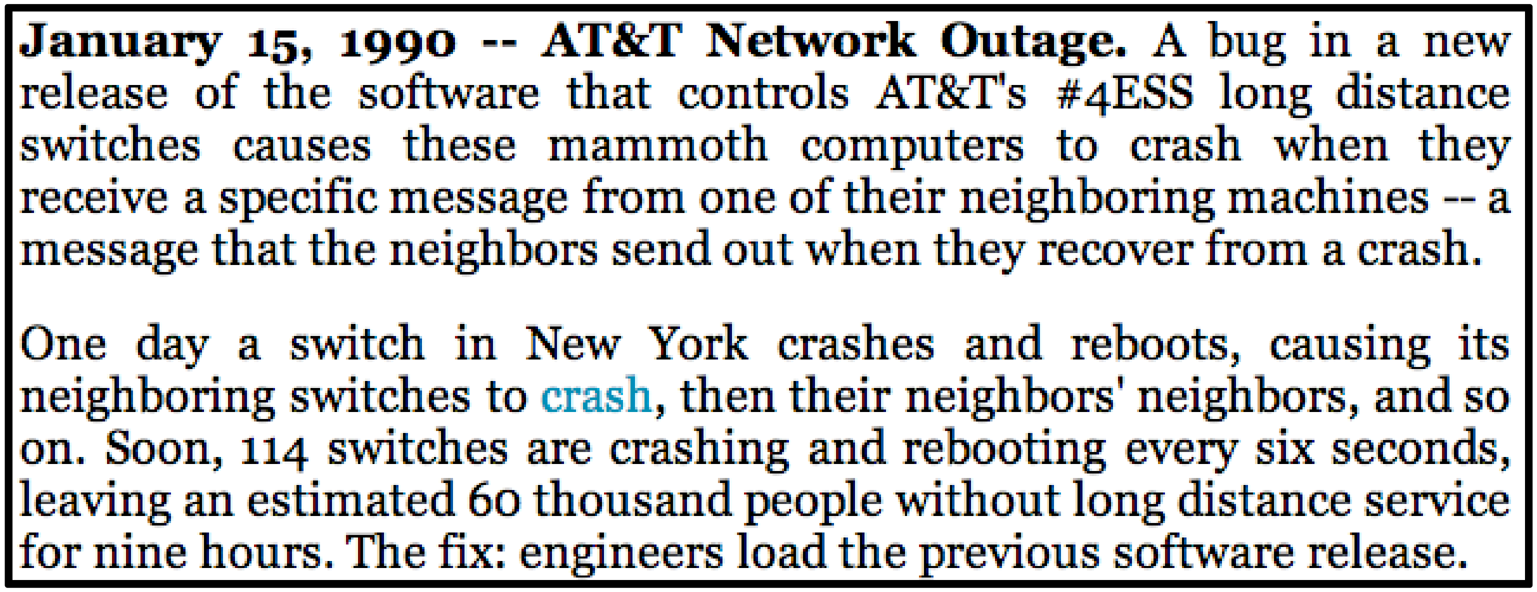


1985-1987 -- Therac-25 medical accelerator. A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured.

http://www.wired.com/software/coolapps/news/2005/11/69355

Software problems are ubiquitous



January 15, 1990 -- AT&T Network Outage. A bug in a new release of the software that controls AT&T's #4ESS long distance switches causes these mammoth computers to crash when they receive a specific message from one of their neighboring machines -- a message that the neighbors send out when they recover from a crash.

One day a switch in New York crashes and reboots, causing its neighboring switches to crash, then their neighbors' neighbors, and so on. Soon, 114 switches are crashing and rebooting every six seconds, leaving an estimated 60 thousand people without long distance service for nine hours. The fix: engineers load the previous software release.

http://www.wired.com/software/coolapps/news/2005/11/69355

Software problems are ubiquitous

- NASA Mars Lander
  - Bug in translation between English and metric units
  - Cost taxpayers $165 million
- Denver Airport baggage system
  - Bug caused baggage carts to become out of “sync,” overloaded, etc.
  - Delayed opening for 11 months, at $1 million per day
- Other fatal or potentially fatal bugs
  - US Vicennes tracking software
  - MV-22 Osprey
  - Medtronic Model 8870 Software Application Card

From *Exploiting Software* and http://www.fda.gov/cdrh/recalls/recall-082404b-pressrelease.html

Adversarial Failures

- Software bugs are bad
  - Consequences can be serious
- Even worse when an intelligent adversary wishes to exploit them!
  - Intelligent adversaries: Force bugs into "worst possible" conditions/states
  - Intelligent adversaries: Pick their targets
- Buffer overflows bugs: Big class of bugs
  - Normal conditions: Can sometimes cause systems to fail
  - Adversarial conditions: Attacker able to violate security of your system (control, obtain private information, ...)

A Bit of History: Morris Worm

- Worm was released in 1988 by Robert Morris
  - Graduate student at Cornell, son of NSA chief scientist
  - Convicted under Computer Fraud and Abuse Act, sentenced to 3 years of probation and 400 hours of community service
  - Now an EECS professor at MIT
- Worm was intended to propagate slowly and harmlessly measure the size of the Internet
- Due to a coding error, it created new copies as fast as it could and overloaded infected machines
- $10-100M worth of damage

Morris Worm and Buffer Overflow

- One of the worm’s propagation techniques was a buffer overflow attack against a vulnerable version of fingerd on VAX systems
  - By sending special string to finger daemon, worm caused it to execute code creating a new worm copy
  - Unable to determine remote OS version, worm also attacked fingerd on Suns running BSD, causing them to crash (instead of spawning a new copy)

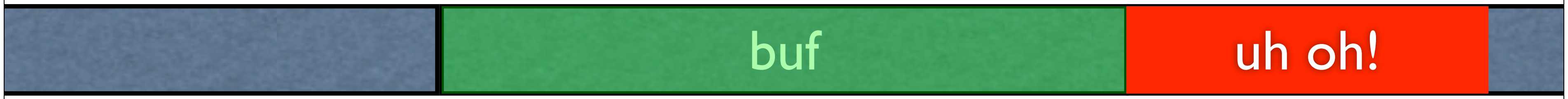
Buffer Overflow These Days

- Very common cause of Internet attacks
  - In 1998, over 50% of advisories published by CERT (computer security incident report team) were caused by buffer overflows
- Morris worm (1988): overflow in fingerd
  - 6,000 machines infected
- CodeRed (2001): overflow in MS-IIS server
  - 300,000 machines infected in 14 hours
- SQL Slammer (2003): overflow in MS-SQL server
  - 75,000 machines infected in **10 minutes** (!!)

Attacks on Memory Buffers

- Buffer is a data storage area inside computer memory (stack or heap)
  - Intended to hold pre-defined amount of data
    - If more data is stuffed into it, it spills into adjacent memory
  - If executable code is supplied as "data", victim's machine may be fooled into executing it – we'll see how
    - Code will self-propagate or give attacker control over machine
- First generation exploits: stack smashing
- Later generations: heaps, function pointers, off-by-one, format strings and heap management structures

Stack Buffers

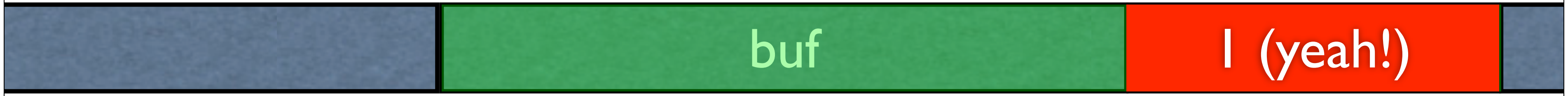


- Suppose Web server contains this function

```
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

Changing Flags



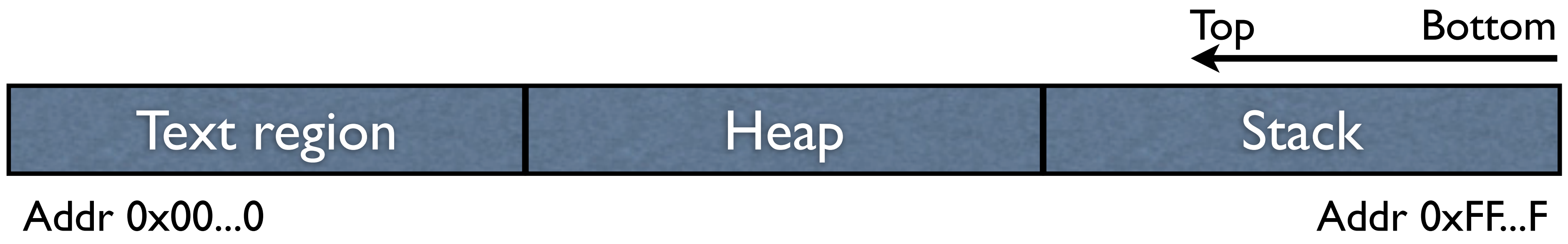
- Suppose Web server contains this function

```
void func(char *str) {
    int authenticated = 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 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

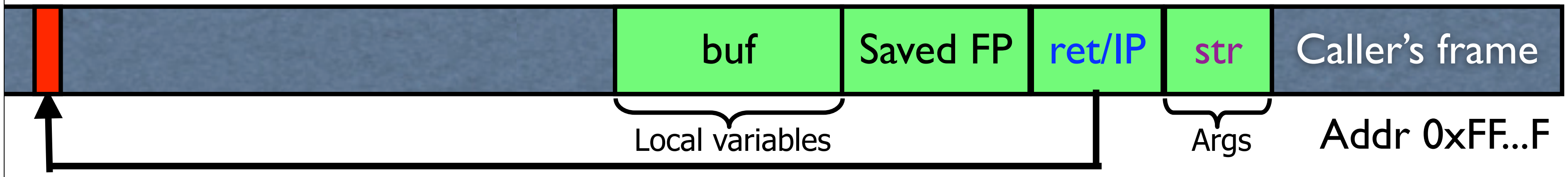
- Suppose Web server contains this function

```
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 with local variables is pushed onto the stack



Execute code at this address after func() finishes

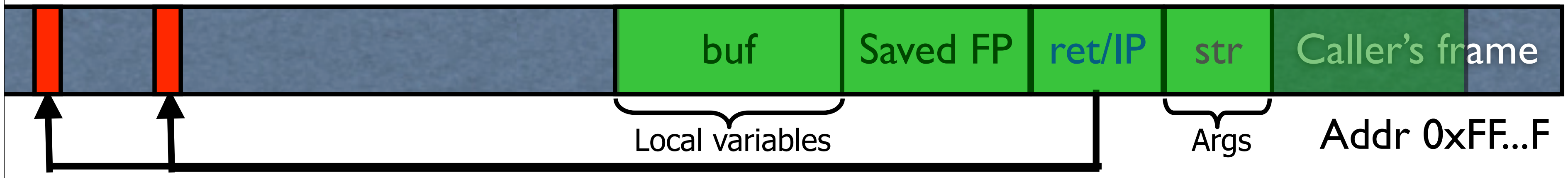
What If Buffer is Overstuffed?

- Memory pointed to by str is copied onto stack…

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

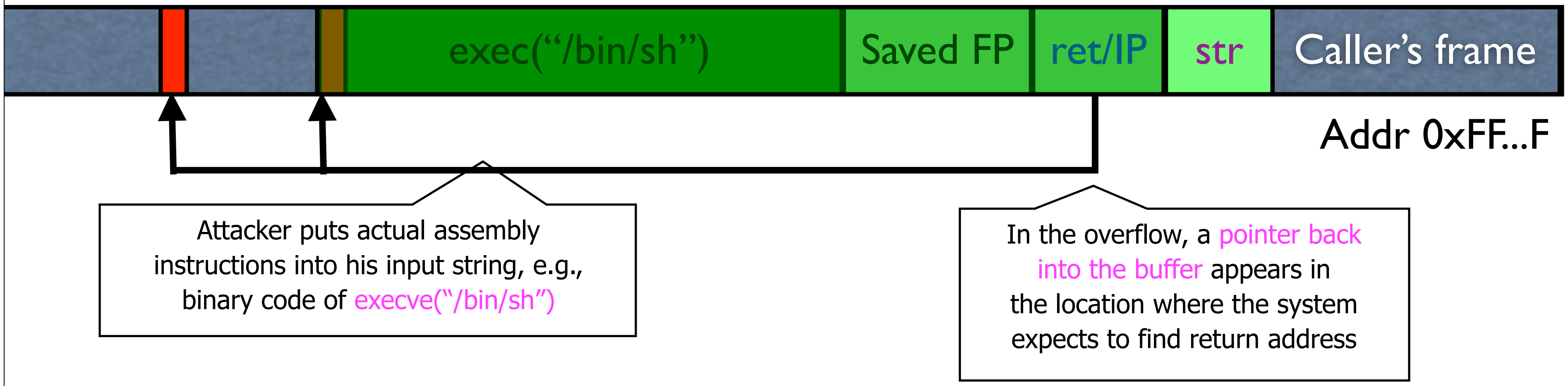
strcpy does NOT check whether the string at *str contains fewer than 126 characters

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



Executing Attack Code

- Suppose buffer contains attacker-created string
  - For example, *str contains a string received from the network as input to some network service daemon



Attacker puts actual assembly instructions into his input string, e.g., binary code of execve("/bin/sh")

In the overflow, a pointer back into the buffer appears in the location where the system expects to find return address

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

Buffer Overflow Issues

- Executable attack code is stored on stack, inside the buffer containing attacker's string
  - Stack memory is supposed to contain only data, but…
- Overflow portion of the buffer must contain correct address of attack code in the RET position
  - The value in the RET position must point to the beginning of attack assembly code in the buffer
    - Otherwise application will (probably) crash with segmentation violation
  - Attacker must correctly guess in which stack position his buffer will be when the function is called

Problem: No Range Checking

- strcpy does <u>not</u> check input size
  - strcpy(buf, str) simply copies memory contents into buf starting from *str until “\0” is encountered, ignoring the size of area allocated to buf
- Many C library functions are unsafe
  - strcpy(char *dest, const char *src)
  - strcat(char *dest, const char *src)
  - gets(char *s)
  - scanf(const char *format, …)
  - printf(const char *format, …)

Does Range Checking Help?

- strncpy(char *dest, const char *src, size_t n)
  - If strncpy is used instead of strcpy, no more than n characters will be copied from *src to *dest
    - Programmer has to supply the right value of n
- Potential overflow in htpasswd.c (Apache 1.3):

```
strcpy(record,user);
strcat(record,":");
strcat(record,cpw); …
```

Copies username ("user") into buffer ("record"), then appends ":" and hashed password ("cpw")

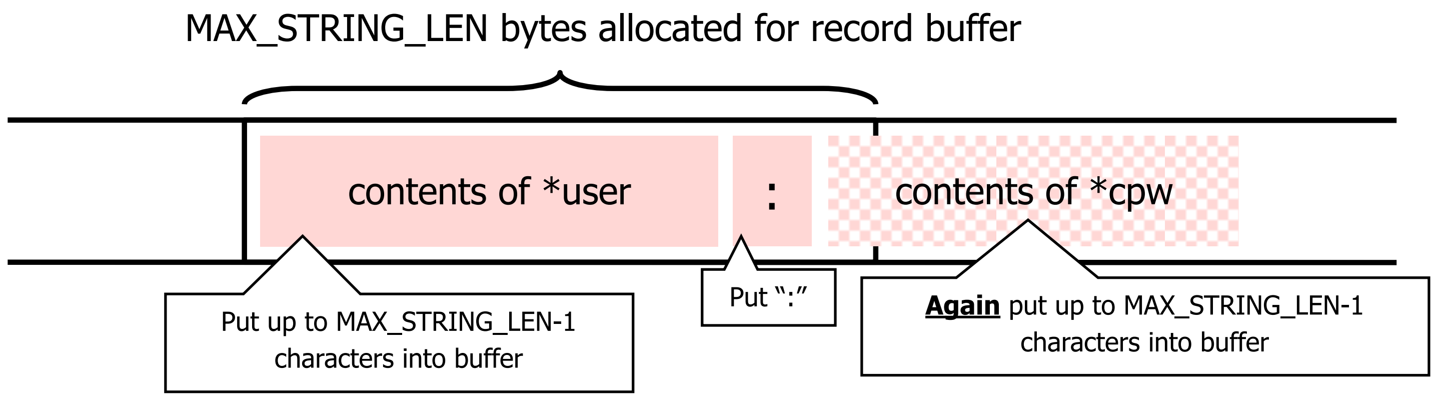
- Published "fix":

```
… strncpy(record,user,MAX_STRING_LEN-1);
  strcat(record,":");
  strncat(record,cpw,MAX_STRING_LEN-1); …
```

Misuse of strncpy in htpasswd “Fix”

- Published “fix” for Apache htpasswd overflow:

```
… strncpy(record,user,MAX_STRING_LEN-1);
  strcat(record,”:”);
  strncat(record,cpw,MAX_STRING_LEN-1); …
```


MAX_STRING_LEN bytes allocated for record buffer
contents of *user
:
contents of *cpw
Put up to MAX_STRING_LEN-1 characters into buffer
Put “:”
Again put up to MAX_STRING_LEN-1 characters into buffer

Off-By-One Overflow

- Home-brewed range-checking string copy

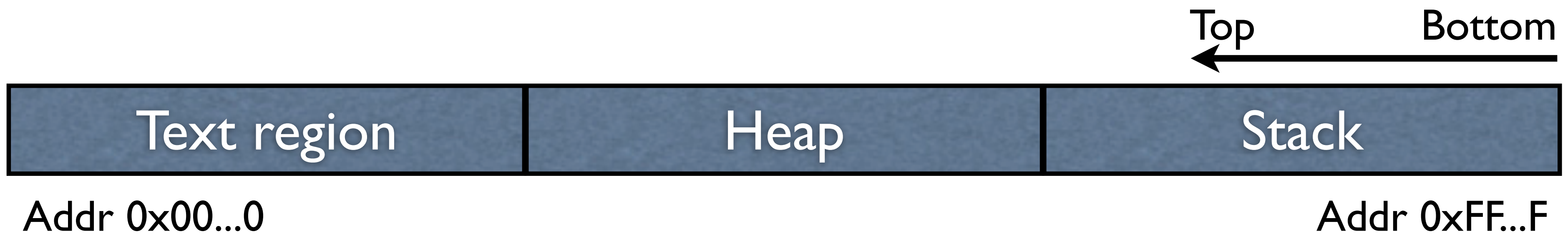
```
void notSoSafeCopy(char *input) {

        char buffer[512]; int i;

         for (i=0; i<=512; i++)
               buffer[i] = input[i];
 }
 void main(int argc, char *argv[]) {
         if (argc==2)
              notSoSafeCopy(argv[1]);
 }
```

This will copy **513** characters into buffer. Oops!

- 1-byte overflow: can't change RET, but can change pointer to previous stack frame
  - On little-endian architecture, make it point into buffer
  - RET for previous function will be read from buffer!

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

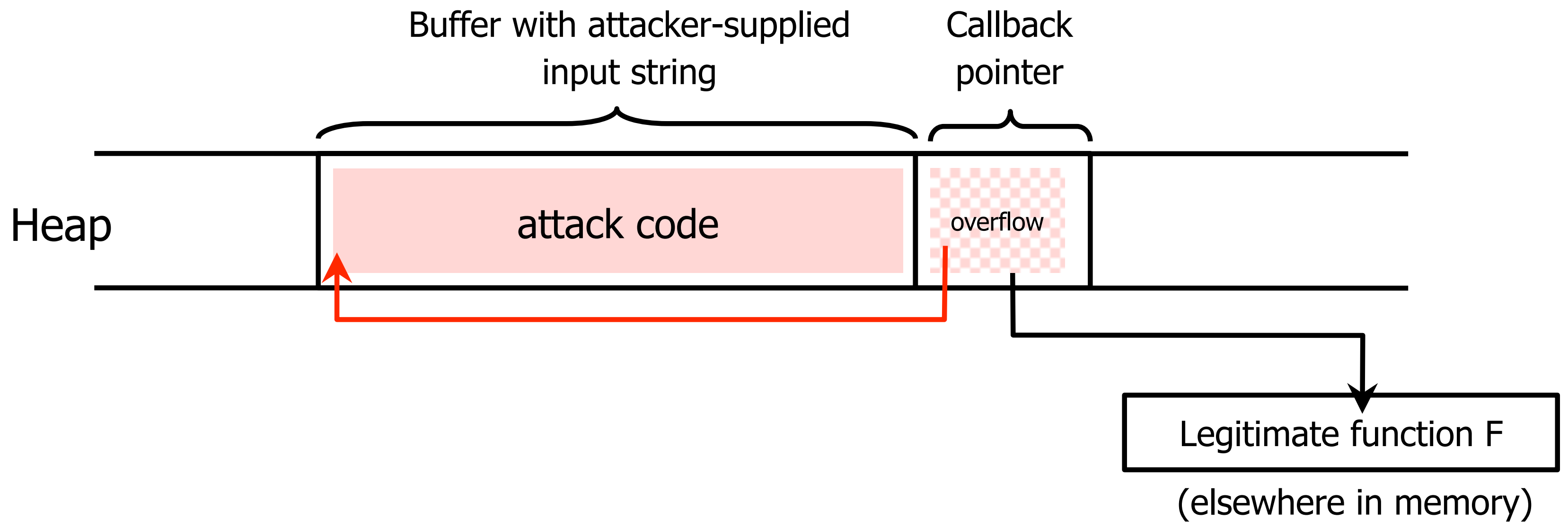


Heap Overflow

- Overflowing buffers on heap can change pointers that point to important data
  - Sometimes can also transfer execution to attack code
  - Can cause program to crash by forcing it to read from an invalid address (segmentation violation)
- Illegitimate privilege elevation: if program with overflow has sysadm/root rights, attacker can use it to write into a normally inaccessible file
  - For example, replace a filename pointer with a pointer into buffer location containing name of a system file
    - Instead of temporary file, write into AUTOEXEC.BAT

Function Pointer Overflow

- C uses function pointers for callbacks: if pointer to F is stored in memory location P, then another function G can call F as (*P)(…)



Format Strings in C

- Proper use of printf format string:

```
… int foo=1234;
  printf("foo = %d in decimal, %X in hex",foo,foo); …
```

  – This will print

```
foo = 1234 in decimal, 4D2 in hex
```

- Sloppy use of printf format string:

```
… char buf[14]="Hello, world!";
  printf(buf);
   // should have used printf("%s", buf); …
```

  – If buffer contains format symbols starting with %, location pointed to by printf's internal stack pointer will be interpreted as an argument of printf. This can be exploited to move printf's internal stack pointer.

Viewing Memory

- %x format symbol tells printf to output data on stack

```
… printf("Here is an int:  %x",i); …
```

- What if printf does not have an argument?

```
… char buf[16]="Here is an int:  %x";
  printf(buf); …
```

  - Stack location pointed to by printf's internal stack pointer will be interpreted as an int. (What if crypto key, password, ...?)

- Or what about:

```
… char buf[16]="Here is a string:  %s";
  printf(buf); …
```

  - Stack location pointed to by printf's internal stack pointer will be interpreted as a pointer to a string

Writing Stack with Format Strings

- %n format symbol tells printf to write the number of characters that have been printed

```
… printf("Overflow this!%n",&myVar); …
```

  - Argument of printf is interpeted as destination address
  - This writes 14 into myVar ("Overflow this!" has 14 characters)

- What if printf does not have an argument?

```
… char buf[16]="Overflow this!%n";
  printf(buf); …
```

  - Stack location pointed to by printf's internal stack pointer will be interpreted as address into which the number of characters will be written.

More Buffer Overflow Targets

- Heap management structures used by malloc()
- URL validation and canonicalization
  - If Web server stores URL in a buffer with overflow, then attacker can gain control by supplying malformed URL
    - Nimda worm propagated itself by utilizing buffer overflow in Microsoft's Internet Information Server
- Aside: Some attacks don't even need overflow
  - Naïve security checks may miss URLs that give attacker access to forbidden files
    - For example, http://victim.com/user/../../autoexec.bat may pass naïve check, but give access to system file
    - Defeat checking for "/" in URL by using hex representation: %5c or %255c.