Fireaxe: The DHS Secure Design Competition Pilot

Yevgeniy Vorobeychik, **Michael Z. Lee**, Adam Anderson, Mitchell Adair, William Atkins, Alan Berryhill, Dominic Chen, Ben Cook, Jeremy Erickson, Steve Hurd, Ron Olsberg, Lyndon Pierson, and W. Owen Redwood

Affiliations (in author order): Sandia National Labs, The University of Texas at Austin, University of Nebraska at Omaha, The University of Texas at Dallas, Sandia, University of California Berkeley, Arizona State University, Sandia, Sandia, Sandia, Sandia, Sandia, Florida State University.

Secure Systems

- Critically important
- Difficult to build
- Learn through practice

Secure Systems

- Critically important
- Difficult to build
- Learn through practice

But how do you teach it?

Competitions

- Provides motivation
- Builds skills
- Already successful
  - Tracer FIRE
  - DEFCON
  - Pwn2Own

Competitions

- Provides motivation
- Builds skills
- Already successful
  - Tracer FIRE
  - DEFCON
  - Pwn2Own

But what about securing systems?

Fireaxe

- Build a secure system
- Understand different attack vectors
- Simulate both attacker and defender
- Learn how to apply secure design ideas

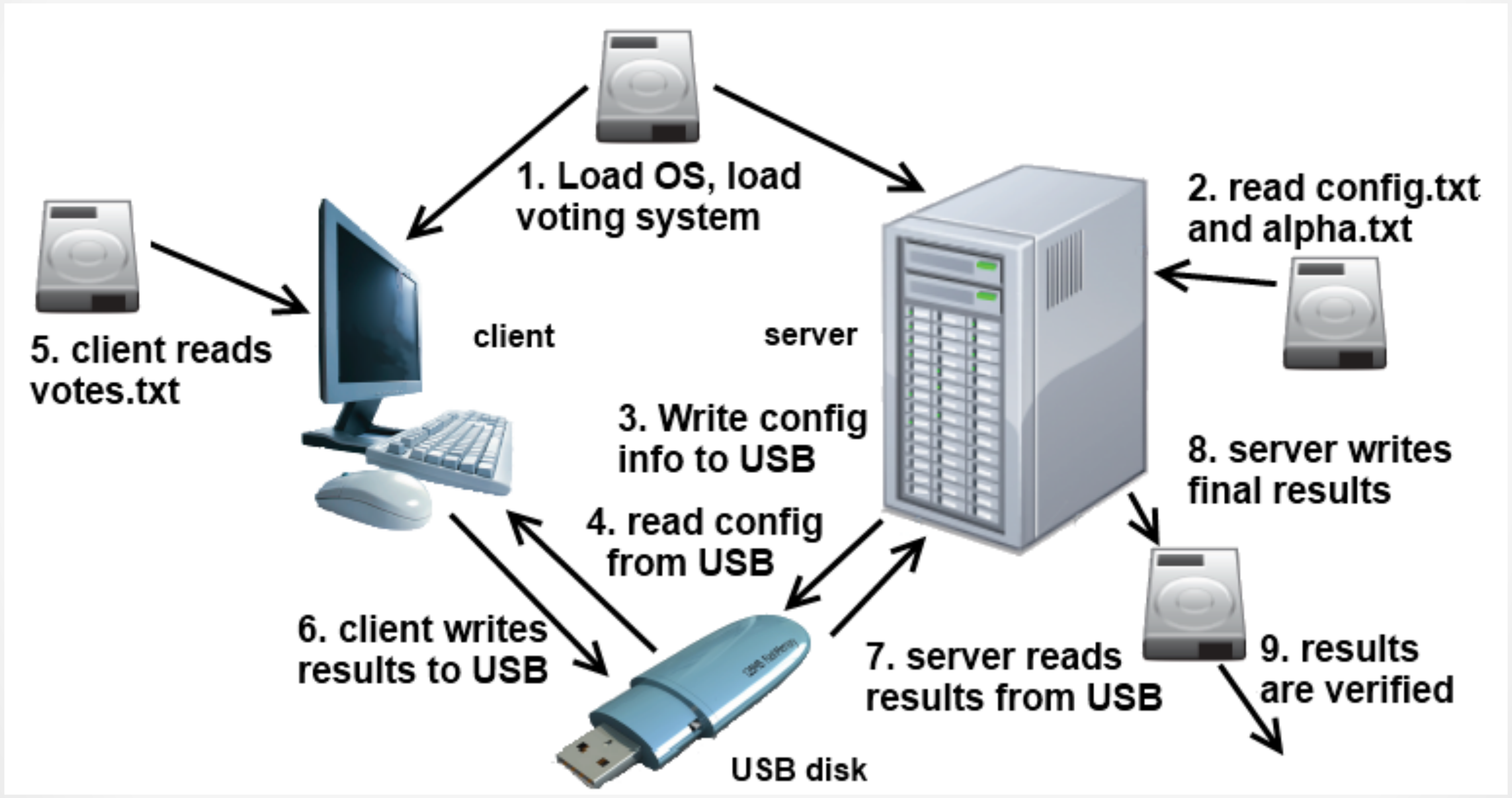
- The general structure
- Our competition
  - Defenses
  - Attacks
- Principles and guidelines

- The general structure
- Our competition
  - Defenses
  - Attacks
- Principles and guidelines

Competition Structure

- n-teams
- Simplified system
- Multiple rounds of build and attack
- Multiple attack scenarios

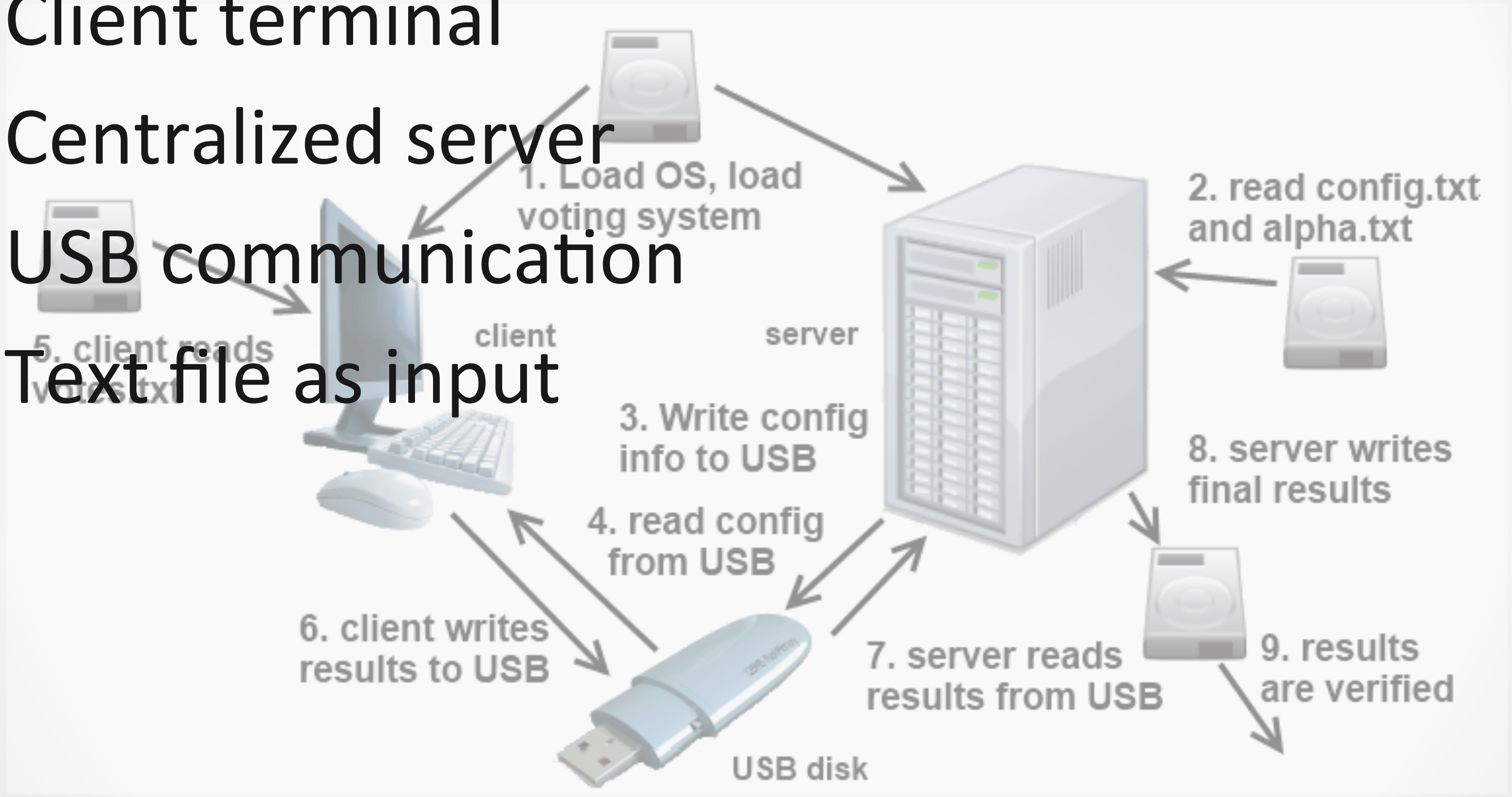
Electronic Voting System



1. Load OS, load voting system
2. read config.txt and alpha.txt
client
server
5. client reads votes.txt
3. Write config info to USB
8. server writes final results
4. read config from USB
6. client writes results to USB
7. server reads results from USB
9. results are verified
USB disk

Electronic Voting System

- Client terminal
- Centralized server
- USB communication
- Text file as input


1. Load OS, load voting system
2. read config.txt and alpha.txt
client
server
3. Write config info to USB
4. read config from USB
5. client reads votes.txt
6. client writes results to USB
7. server reads results from USB
8. server writes final results
9. results are verified
USB disk

- The general structure
- Our competition
  - Defenses
  - Attacks
- Principles and guidelines

The Trial Run

- Two teams
  - Myself, Dominic Chen, Adam Anderson
  - Owen Redwood, Mitch Adair, Alan Berryhill
- Two rounds of build and attack
  - Weeks for development phase
  - Days for attack phase

The Trial Run

- Four adversarial scenarios
  - Client breach
  - Server breach
  - Client root-level attack
  - Malicious USB/communication channel

The Trial Run

- Blue Team
  - 2 points for selecting the correct winners
  - 2 points for tallying all of the votes correctly
  - 1 point for detecting a denial of service attempt
- Red Team
  - 1 point for causing a bad selection
  - 1 point for causing a bad tally

The Systems

NM Team

- EVS written in C
- System built from Buildroot
- grsecurity and PaX
- Custom kernel modifications

CA Team

- EVS written in Python
- System built from Debian
- grsecurity and PaX
- Custom kernel modifications

Defenses

- Many different tactics
  - Code obfuscation
  - System customization
  - Kernel modification

Defenses

Obfuscation => delay the other team

- Hand obfuscation
- Code misdirection
- Filesystem packing

Defenses

System customization => stable environment

- Custom user shell
- Limit binaries
- Remove extra libraries

Defenses

Kernel modifications => the lowest level

- Standard security patches
- Restrict keyboard input
- Filter system call arguments

Attacks: Round 1

- NM team allowed access to /bin/sh
  - `/bin/sh -nv <filename>` to `` `cat` ``
  - `while true` allows looping without `` `[` ``
  - `mount` gave reliable feedback
- CA team created their own crypto
  - Did not check input integrity
- Root attacks went completely unchecked

Attacks: Round 2

- CA Team added an audio CAPTCHA
  - This effectively became a DoS
  - Return from `raw_input` turned into a 2 to 5 second delay
- CA Team still did not sufficiently protect root
- Maliciously crafted directories and files caused both teams to fail

Lessons Learned

- Restrict access when possible
  - Custom user shell
  - Packed filesystem
- Use well tested tools
  - Crypto libraries
- Reduce the attack surface
- Enforce policies and protections at the lowest level

Expanding

- Automate deployment and validation
- Limit system modification
- Remove user interaction

Conclusion

- Competitions can teach
- Provides an environment to experiment
- Spur students’ creativity