Testing for Security

Tips and Tools for keeping security in mind during functional testing.

Marge Shinkle, Oracle Corp.

About Me

- Degree in Hardware and Systems Engineering
- About 15 years in Unix-based IT, Network Support, QE
- Currently working in Systems QE for new Hardware Platforms, Security Features

Objective

- To present various approaches and concepts
- Think of security throughout the product

Myths and "Myth"-conceptions of Security Testing

- "It's not my area"
- There is a "Security" person or team who specializes in Security testing
- The great terror of hearing "security"
- Our software is "behind the firewall" (a.k.a. The Great Denial)

It's Everyone's Area

- Security concerns are pervasive throughout every product, not limited to one function or feature
- Affects you, your company and, ultimately, your paycheck
- Errors potentially affect us all

What is Secure Information?

- Personally Identifiable Information
- Sensitive Information
- Configuration Information

Think CIA

- Confidentiality
- Integrity
- Availability

Where can I begin?

- Clear Text
- File permissions/ACL
- Shared Memory permissions
- Fuzz testing

Fuzz Testing

- input massive amounts of random data into system
- uses a variety of inputs
  - random characters
  - wildcards
  - improper input

Useful Tools (UNIX)

- Truss/trace - follow commands as executed
- lsof, netstat - show open ports
- find, ls - check permissions
- Ipcs - show perms on resources like shared memory
- readelf (linux) - check if binaries are built correctly
- strings, grep - check for clear text in files
- snoop - check network packets for clear text

Useful Tools (Windows)

- Attack Surface Analyzer
- Binscope - check Windows Executables are built correctly (freely available from Microsoft)

Commonly Overlooked Areas

- Demo files
- Installation Tools
- Late features (Feature creep)
- File permissions
- Third party

"You can't trust code that you did not totally create yourself."

- Ken Thompson

"Reflections on Trusting Trust"

1984

The Weird Machine - Linux ELF symbol table

- Rebecca Shapiro at Darthmouth College modified the symbol table to implement a "Weird Machine"
- She wrote a BF program to su to a root shell and inserted it into "ping"'s symbol table, using 8 instructions (inc, dec, (inc), (dec), jmp forward, jump back, print) and 3 registers
- A signature that excludes the symbol table (metadata) will not catch this modification
- This exploit is not possible in Solaris, but it is a proof of concept

Weird Machine - cont'd

- Lesson: any input is a program, any parser is an interpreter
- (Rebecca “bx” Shapiro, Sergey Bratus, Sean W. Smith)

http://www.cs.dartmouth.edu/~bx/elf-bf-tools/slides/ELF-WOOT-2013.pdf

QE Lesson

- Be aware of interaction between your software and outside software
- Stay within Scope - don't overreach by testing outside software, but understand boundaries and look for exploits when exchanging data

More Low-hanging Fruit

- Documentation!!!!!
  - Installation Notes
  - Manuals
  - Readme's
  - Help pages
  - Logs
  - Debug output

What are we looking for?

- Post-install “instructions” to change passwords, permissions, etc
- Back door or factory defaults
- Any passwords, keys, secrets
- Sample data

Digging Deeper

- Compare to other companies (BSIMM)
- Risk Analysis
- Data Flow
- Attack Trees

BSIMM

- Building Security in Maturity Model
- www.bsimm.com
- Study of real-world existing software security initiatives
- 112 activities
- 12 initiatives

BSIMM Measures

- Strategy and Metrics
- Compliance and Policy
- Training
- Attack Models
- Security Features and Design
- Standards and Requirements
- Architecture Analysis
- Code Review
- Security Testing
- Penetration Testing
- Software Environment
- Configuration Management and Vulnerability Management

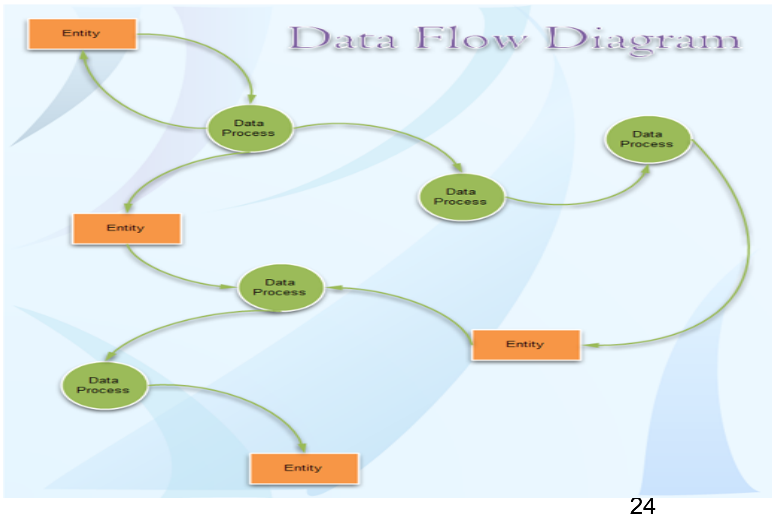
Takeaways

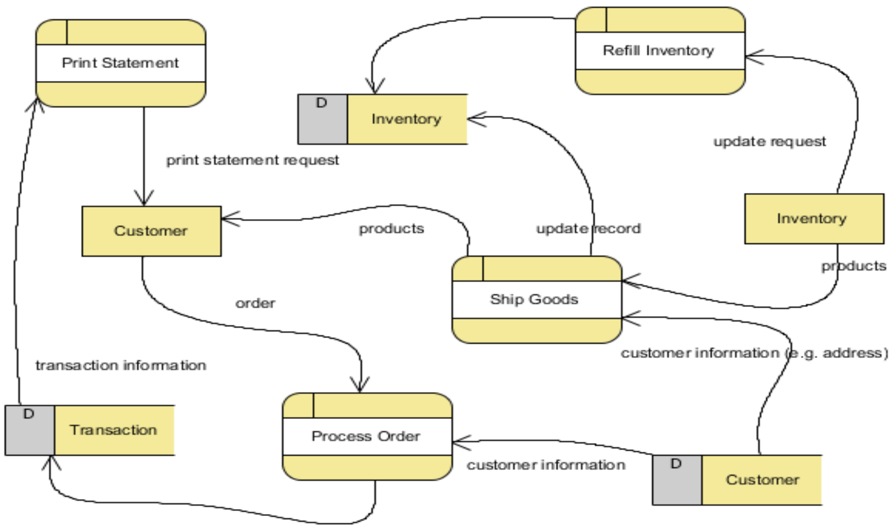
- Fuzz Testing
- Attack Modeling
- Edge cases and boundary conditions

Other Stuff

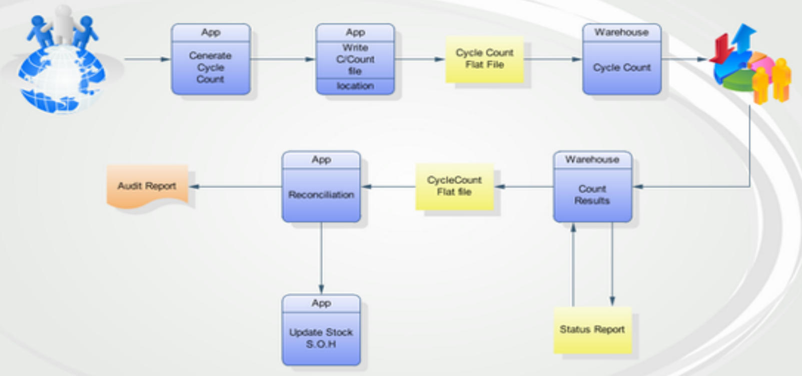
- Risk Analysis
- Data Flow
- Attack Trees

Example Data Flow



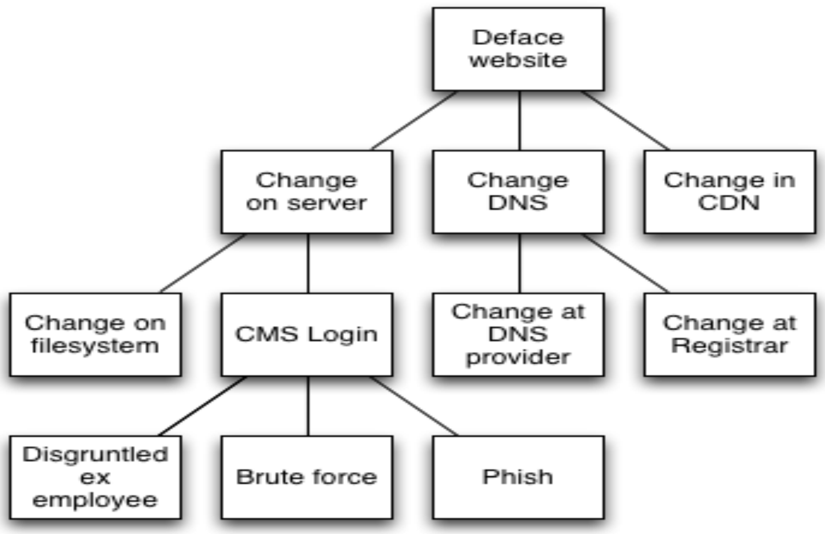


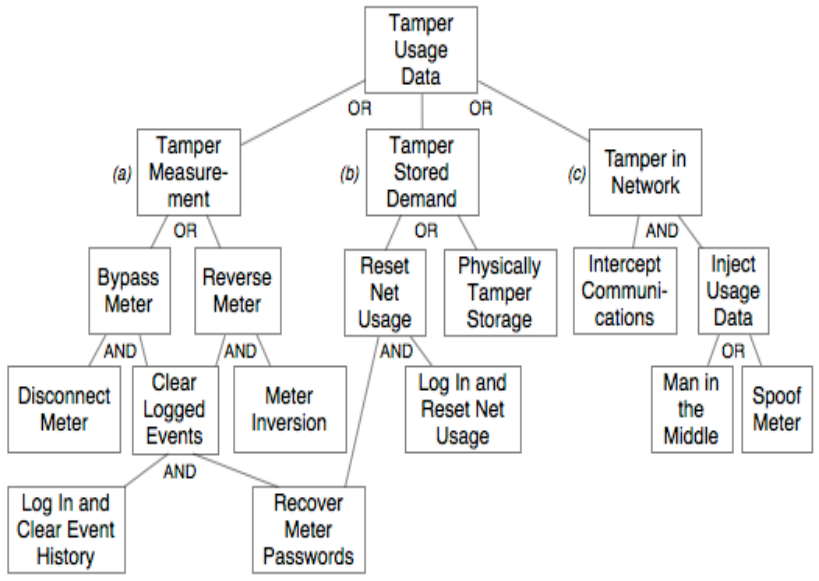
Data Flow Model Diagram



Attack trees

- Attack trees are multi-leveled diagrams consisting of one root, leaves, and children. From the bottom up, child nodes are conditions which must be satisfied to make the direct parent node true; when the root is satisfied, the attack is complete. Each node may be satisfied only by its direct child nodes.





Summary

- No single solution
- Security is part of functional tests, not an outside area
- Look for any place where protections change
- Look for places where data is being moved or altered or stored
- Remember that Security belongs to all of us

References

- USENIX Workshop on Offensive Technologies

https://www.usenix.org/conference/woot13

- www.bsimm.com