Vulnerability Discovery with Time Constraints

Kaveh Razavi, Hamid Ebadi, Mehdi Shajari, Babak Sadeghian

Amirkabir University of Technology CERT

Contents

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A New Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

CERT and Vulnerability Discovery

- Vulnerabilities are one of the main reasons for security incidents
- Prevention is better than cure, even when it comes to security
- CERTs normally have a VA team (Vulnerability Analysts)
- One of the tasks of VA is VD service in time of need
- Time of need
  - An incident has happened on the **most recently updated** software and we need to know how (for future prevention)
  - Deployment of a software in an **insecure environment**
  - A **3rd party company** may require some security assurance of the API they are using
  - …

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

A Brief Overview of Software Security

- Security must be insured in all the steps of software development
- Software development cycle
  - Requirements
  - Analysis
  - Design
  - Implementation
  - Test
    - Software testing in software quality assurance
    - Software security assurance
    - Vulnerability discovery as a part of SQA

A Brief Overview of Vulnerabilities

- A software security vulnerability might cause:
  - Unauthorized Access
  - Information Disclosure
  - Data Invalidation
  - Denial of Service
- Causes of software vulnerabilities
  - Memory Access Violations
    - Buffer Overflows (Stack, Heap, BSS)
  - Improper I/O Handling
    - Format Strings
    - Injections (SQL Injection, Code Injection)
    - Directory Traversal
  - Race Conditions
    - TOCTOU
  - Logical Errors

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A New Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

Vulnerability Discovery Methodology

- A lot of contributions from academia
  - White-Box
  - Black-Box
  - Gray-Box
- Most of these methods are stayed out of technology
  - They need a lot of time and time is costly in software development
  - Security analysts are expensive

The White, The Black and The Gray

	White	Black	Gray
Implementation Time	Low ~ Medium	Low	Software
Source code	Required	Not Required	Both!
Complexity	Low ~ Medium	Low ~ Medium	Medium ~ High
Automation	Difficult	Easy	Software
Patch	Easy	Difficult	Software

- Choosing an appropriate method
  - Time
  - Automation
  - Flexibility
  - Effectiveness

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

Our Proposed VD Procedure I

1. Gathering Information
2. Designing/Implementing Test Case Generator
3. Vulnerability Analyzing
4. Patching and Reporting

Our Proposed VD Procedure II

- Backgrounds of this proposed VD procedure
- Discussed with two real world example showing the effectiveness of this procedure
  - You all know PHP! It has shown very risk induced when used in shared hosting environments
  - xrdp is a remote desktop server with a lot of features
- Software source code independency

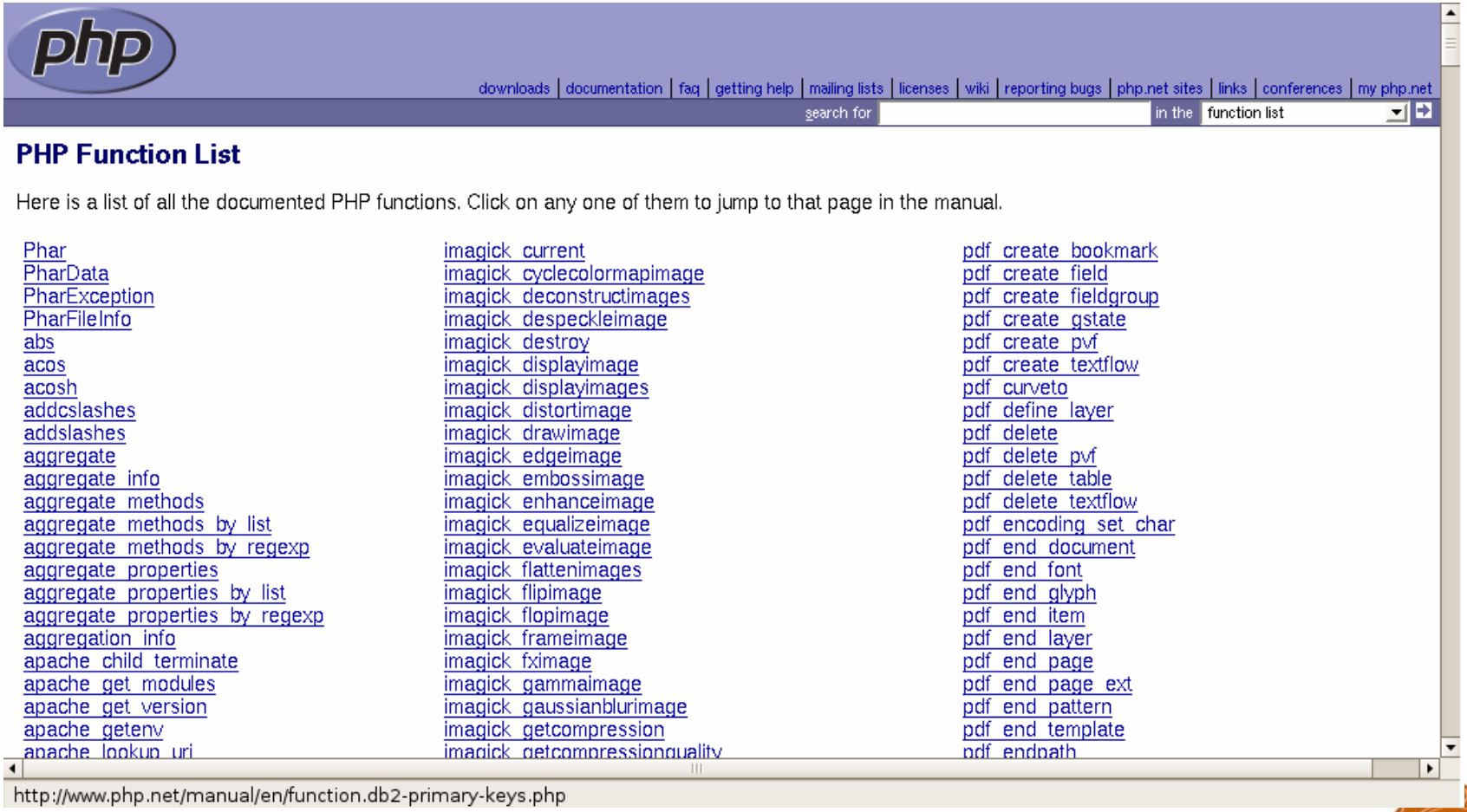
Gathering Information

- Determining software functionality
- Using software engineering diagrams
- Software documentations
- Comparing with other similar software
- Interaction of software with environment
  - Network activities
  - Input Files
  - The external API
  - etc
- Tools to gather information

Gathering Information in Action I

- PHP code is externally exposed via its APIs
- We would like to have a list of PHP APIs
- PHP APIs are also very easy to audit (loosely typed variables)
- Checking the PHP website we found the information we were looking for

Gathering Information in Action II

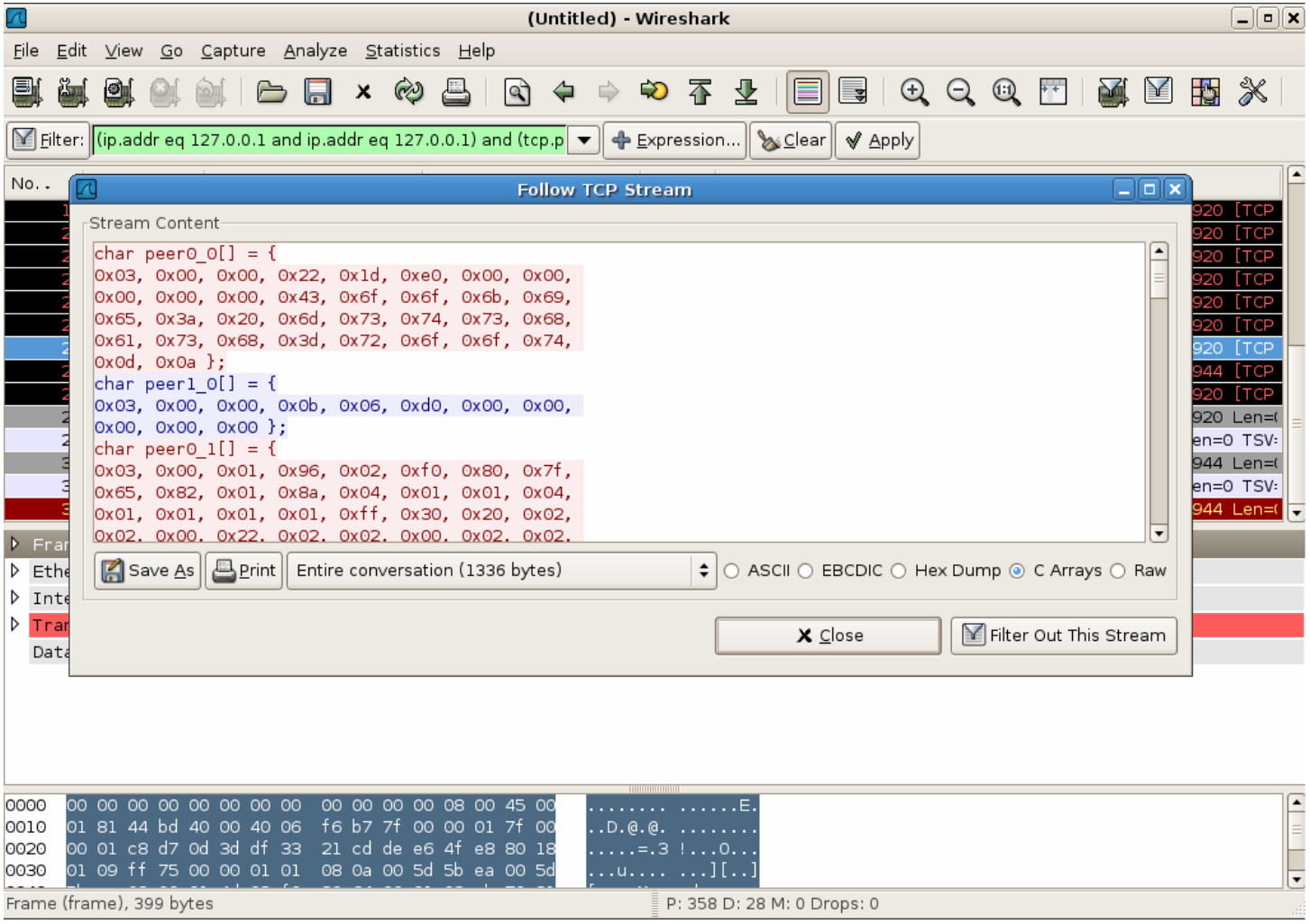


Gathering Information in Action III

Gathering Information in Action IV

- Gathering proper data for auditing xrdp would be a bit trickier
  - xrdp deploys a complex binary protocol
- Some samples from xrdp network traces is required
  - Wireshark
    - Beautiful GUI
    - Powerful filters
    - TCP stream following
  - Tcpdump
    - Fast setup
    - No GUI

Gathering Information in Action V



Designing/Implementing Test Case Generator

- Designing an effective test case generator is difficult and depends on the previous steps
  - Designing a set of powerful test cases for a specific software could be time consuming (effective)
  - Sending (executing) raw random data proven to be ineffective in many cases (easy to implement)
- Since time is an important factor, the design should be effective and easy to implement
- It should also be easy to trace what input causes a fault
- Randomness is powerful, make use of it whenever possible in the design of your test case generator

Designing/Implementing Test Case Generator in Action I

- All the functions (APIs) name from the PHP website are retrieved with a simple regular expression
- Since PHP variables are loosely typed, a variable could be anything
  - A long string (buffer overflow)
  - A negative number (integer overflow)
  - A zero or a very big number (division by zero, null pointer dereferences)
  - Some streams (required by some functions)
  - Some random characters (random behavior)
- The general method that we used for PHP is a combination of random testing and basic fuzzing

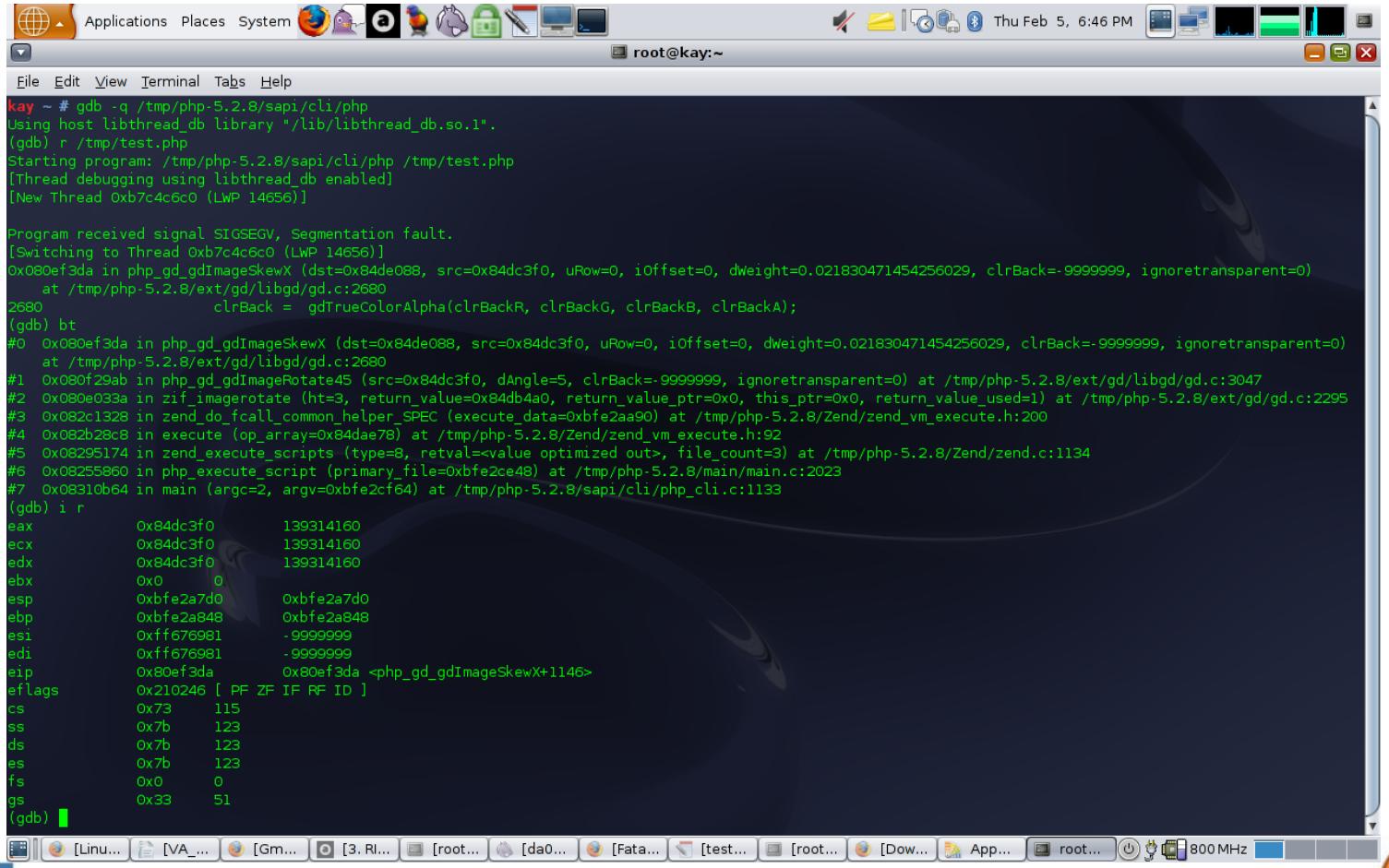
Designing/Implementing Test Case Generator in Action II

- We did not try to find the exact prototypes of PHP APIs
  - Needs more run time (less effective)
  - Faster implementation time
- It is a trade-off you might like to consider when designing test case generators
- Each function was executed multiple times
  - a random number of arguments
  - A random combination of the mentioned malformed variables
- Monitoring PHP during runtime
  - UNIX signals and signal handlers

Designing/Implementing Test Case Generator in Action III

- Packing it all together, we have
  - A list of functions
  - Some good variables that might crash PHP
  - It is not important for us how many arguments each function has
- 10,000 php files created containing
  - An include to the variables file
  - A call to a random function name
- A simple debugger was created to run /usr/bin/php with each of these files
  - The name of files that caused suspicious signals were logged
- It took us some (< 5h) hours to audit PHP
  - Retrieve the function names
  - Code a program to generate the php files
  - Code a simple debugger which ran the php binary with signal monitoring
- The run-time was less than a minute for generating the function.php files, we let each run take two seconds and the total run-time was about 8 hours
- PHP crashed a lot of times. Some of those were new security bugs (CVE-2008-5498)

Designing/Implementing Test Case Generator in Action IV



Designing/Implementing Test Case Generator in Action V

- For xrdp the story was shorter
- We decided to change the normal traces a bit and send them to xrdp
- This method happens to be very effective when auditing a software which works with a binary protocol (explain why)
- This method is generally known as mutation based fuzzing
- Wireshark did us a nice work and the network packets are ready to be pasted to our little fuzzer

Designing/Implementing Test Case Generator in Action VI

- We started from the first packet, changed some bytes from some random places to some other random bytes
- Then the first and the second packet and after that the first three packets and so on
- We used gdb (The GNU Debugger) to monitor xrdp
- We let the auditing process take about an hour
- xrdp also crashed several times, almost all of them were new high risk vulnerabilities (CVE-2008-5902, CVE-2008-5903, CVE-2008-5904)

Vulnerability Analyzing

- The time required for debugging a vulnerability is based on:
  - Type of the vulnerability
  - Experience of the one who is responsible for this task
  - familiarity with the software under question
- Tools
  - Linux
    - gdb (or ddd)
    - IDA Pro
  - Windows
    - windbg
    - ollygdb
    - IDA Pro

Vulnerability Analyzing in Action

- It took us almost a day to understand the nature and impact of the vulnerabilities discovered in PHP which we finally came to categorize as information disclosure (e.g. read SSL private key of apache)
- Debugging xrdp vulnerabilities did not take as long, the cause was mainly the unchecked null pointers returned from malloc() and some buffer overflows (e.g. DoS or Unauthorized access)

Patching and Reporting

- Patching is normally a straight forward process when the nature of a vulnerability is discovered
- Sometimes a major change to the source code is required
- For formal reporting of the vulnerability, you should then inform the vendor (ask MITRE for a CVE)
- Normally vendors come with a newer version or a patch and ask if it is still vulnerable (and sometimes it still is!)
- Side effects might be introduced by patching a vulnerability

Patching and Reporting in Action

- In our cases we just added some checks to the program when handling input data
- On UNIX you can use diff –u source.c source_patched.c > source.patch and then distribute source.patch
- PHP guys were very cooperative and the vulnerability got patched in CVS 30 minutes later, they also used the same patch we sent to them
- xrdp developers were a bit harder to work with, after the formal reporting of the vulnerabilities to security sources, the major distributions patched it in a case basis.

Contents

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A New Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

Conclusion

- There could be a lot of security bugs in a software which could be discovered and patched with a little effort
- Normally the produced testing tools could be used again with minor modifications
- Following the procedure we gave in this presentation, effective VD is possible even when there are time constraints
- It may not discover in depth vulnerabilities but the tested software could be given the minimums of security assurance

Q & A