Acknowledgments: Lecture slides are from the Security Engineering thought by Dan Fleck at George Mason University. When slides are obtained from other sources, a a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.

Fuzzing

Dan Fleck
CS 469: Security Engineering

Sources:
http://www.cse.msu.edu/~cse825//lectures/Fuzzing.pdf
http://www.cs.berkeley.edu/~dawnsong/teaching/f12-cs161/lectures/lec5-fuzzing-se.pdf
http://weis2007.econinfosec.org/papers/29.pdf
http://www.cs.berkeley.edu/~dawnsong/teaching/f12-cs161/readings/toorcon.pdf
http://www.immunityinc.com/downloads/DaveAitel_TheHackerStrategy.pdf
http://www.uninformed.org/?v=5&a=5&t=pdf
http://msdn.microsoft.com/en-us/library/cc162782.aspx

What is Fuzzing?

- A form of vulnerability analysis
- Process:
  - Many slightly anomalous test cases are input into the application
  - Application is monitored for any sign of error

Example

Standard HTTP GET request

- § GET /index.html HTTP/1.1

Anomalous requests

- § AAAAAA...AAAA /index.html HTTP/1.1
- § GET ///////index.html HTTP/1.1
- § GET %n%n%n%n%n%n.html HTTP/1.1
- § GET /AAAAAAAAAAAAA.html HTTP/1.1
- § GET /index.html HTTTTTTTTTTTTTP/1.1
- § GET /index.html HTTP/1.1.1.1.1.1.1.1
- § etc...

User Testing vs Fuzzing

- User testing
  - Run program on many **normal** inputs, look for bad things to happen
  - Goal: Prevent **normal users** from encountering errors
- Fuzzing
  - Run program on many **abnormal** inputs, look for bad things to happen
  - Goal: Prevent **attackers** from encountering **exploitable** errors

Types of Fuzzers

- Mutation Based – “Dumb Fuzzing”
  - mutate existing data samples to create test data
- Generation Based – “Smart Fuzzing”
  - define new tests based on models of the input
- Evolutionary
  - Generate inputs based on response from program

Fuzzing

- Automatically generate random test cases
- Application is monitored for errors
- Inputs are generally either
  - files (.pdf, png, .wav, .mpg)
  - network based (http, SOAP, SNMP)

Mutation Based Fuzzing

- Little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs
- Anomalies may be completely random or follow some heuristics
- Requires little to no set up time
- Dependent on the inputs being modified
- May fail for protocols with checksums, those which depend on challenge response, etc.

- Example Tools:
  - Taof, GPF, ProxyFuzz, Peach Fuzzer, etc.

PEACH FUZZER

Mutation Based Example: PDF Fuzzing

- Google .pdf (lots of results)
- Crawl the results and download lots of PDFs

- Use a mutation fuzzer:

1. Grab the PDF file
2. Mutate the file
3. Send the file to the PDF viewer
4. Record if it crashed (and the input that crashed it)

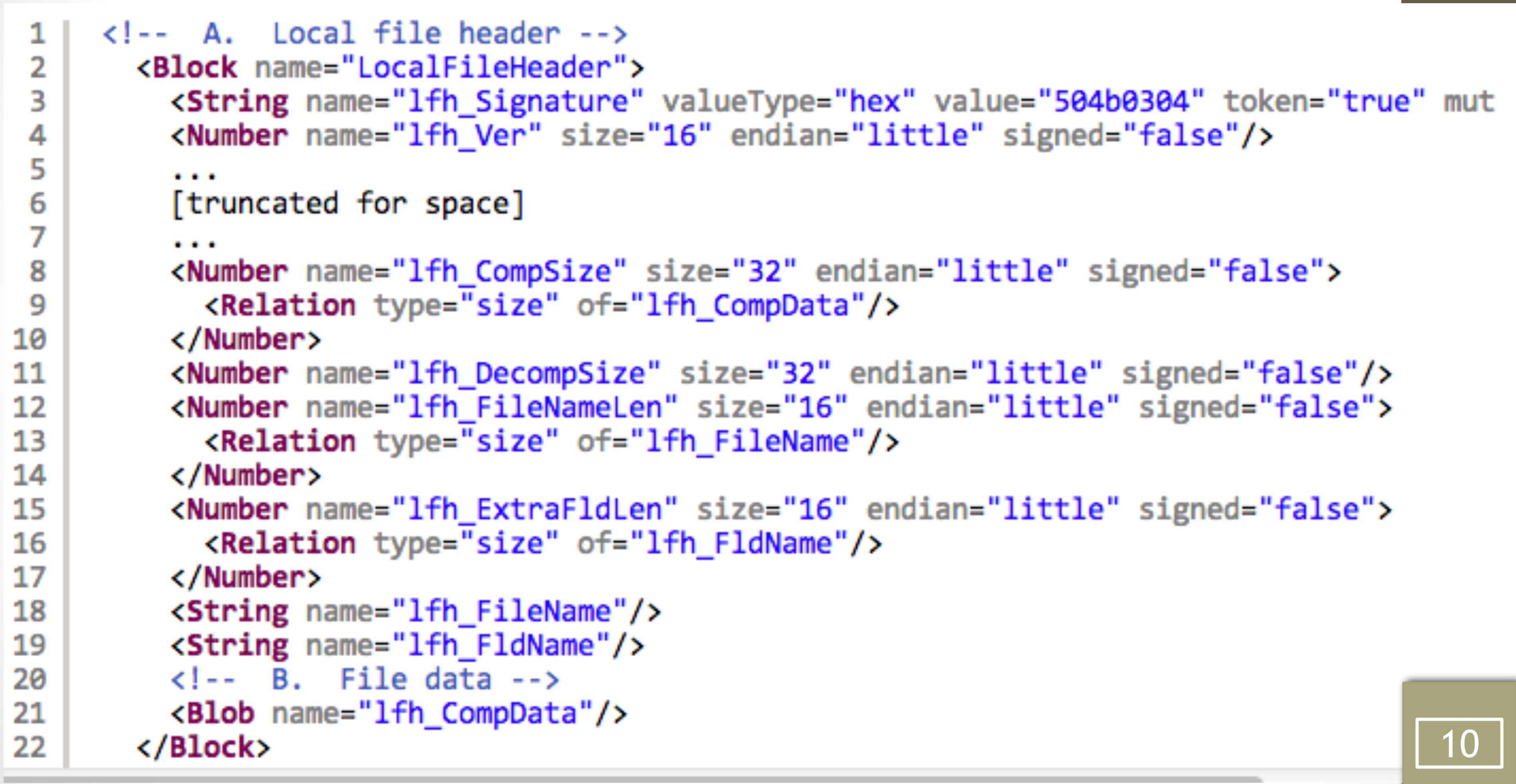
Mutation-based	Super easy to setup and automate +	Little to no protocol knowledge required +	Limited by initial corpus −	May fail for protocols with checksums, or other complexity −

Generation Based Fuzzing

- Test cases are generated from some description of the format: RFC, documentation, etc.
- Anomalies are added to each possible spot in the inputs
- Knowledge of protocol should give better results than random fuzzing
- Can take significant time to set up

- Examples
  - SPIKE, Sulley, Mu-4000,
    Codenomicon,
    Peach Fuzzer, etc…

Example Specification for ZIP file



```
<!--  A.  Local file header -->
  <Block name="LocalFileHeader">
    <String name="lfh_Signature" valueType="hex" value="504b0304" token="true" mut
    <Number name="lfh_Ver" size="16" endian="little" signed="false"/>
    ...
    [truncated for space]
    ...
    <Number name="lfh_CompSize" size="32" endian="little" signed="false">
      <Relation type="size" of="lfh_CompData"/>
    </Number>
    <Number name="lfh_DecompSize" size="32" endian="little" signed="false"/>
    <Number name="lfh_FileNameLen" size="16" endian="little" signed="false">
      <Relation type="size" of="lfh_FileName"/>
    </Number>
    <Number name="lfh_ExtraFldLen" size="16" endian="little" signed="false">
      <Relation type="size" of="lfh_FldName"/>
    </Number>
    <String name="lfh_FileName"/>
    <String name="lfh_FldName"/>
    <!--  B.  File data -->
    <Blob name="lfh_CompData"/>
  </Block>
```

Src: http://www.flinkd.org/2011/07/fuzzing-with-peach-part-1/

Mutation vs Generation

Mutation-based	Super easy to setup and automate ➕	Little to no protocol knowledge required ➕	Limited by initial corpus ➖	May fail for protocols with checksums, or other complexity ➖
Generation-based	Writing generator is labor intesive for complex protocols ➖	have to have spec of protocol (frequently not a problem for common ones http, snmp, etc…) ➖	Completeness ➕	Can deal with complex checksums and dependencies ➕

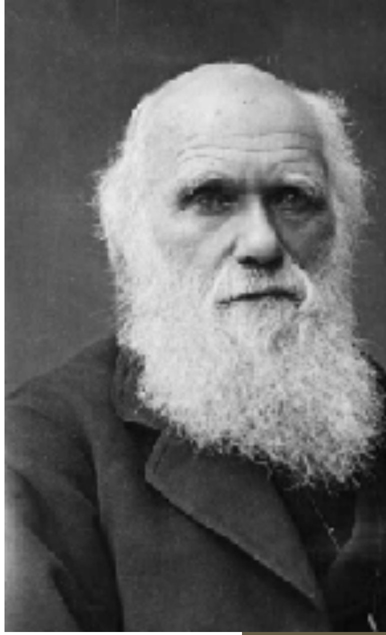
White box vs. black box fuzzing

- Black box fuzzing: sending the malformed input without any verification of the code paths traversed
- White box fuzzing: sending the malformed input and verifying the code paths traversed. Modifying the inputs to attempt to cover all code paths.

Technique	Effort	Code coverage	Defects Found
black box + mutation	10 min	50%	25%
black box + generation	30 min	80%	50%
white box + mutation	2 hours	80%	50%
white box + generation	2.5 hours	99%	100%

Source: http://msdn.microsoft.com/en-us/library/cc162782.aspx

Evolutionary Fuzzing



- Attempts to generate inputs based on the response of the program
- Autodafe
  - Prioritizes test cases based on which inputs have reached dangerous API functions
- EFS
  - Generates test cases based on code coverage metrics
- This technique is still in the alpha stage :)

Challenges

- Mutation based – can run forever. When do we stop?
- Generation based – stop eventually. Is it enough?
- How to determine if the program did something “bad”?

- These are the standard problems we face in most automated testing.

Code Coverage

- Some of the answers to our problems are found in code coverage
- To determine how well your code was tested, code coverage can give you a metric.

- But it’s not perfect (is anything?)

- Code coverage types:
  - Statement coverage – which statements have been executed
  - Branch coverage – which branches have been taken
  - Path coverage – which paths were taken.

Code Coverage - Example

```
if (a > 2)
    a = 2;
if (b > 2)
   b = 2
```

How many test cases for 100% line coverage?
How many test cases for 100% branch coverage?
How many test cases for 100% paths?

Code Coverage Tools

- If you have source: gcov, Bullseye, Emma
- If you don't:
  - Binary instrumentation: PIN, DynamoRIO
  - Valgrind : instrumentation framework for building dynamic analysis tools
  - Pai Mei : a reverse engineering framework consisting of multiple extensible components.

Lots more to discuss on Code Coverage in a Software Engineering class.. but lets move on.

Why does Code Coverage help?

- Lets jump to an example on Page 27 of :
- http://www.cs.berkeley.edu/~dawnsong/teaching/f12-cs161/readings/toorcon.pdf

IPhone Security Flaw: July 2007

Shortly after the iPhone was released, a group of security researchers at Independent Security Evaluators decided to investigate how hard it would be for a remote adversary to compromise the private information stored on the device

acker's Life: Charlie Miller Keeps the
ı World On Its Toes

lacklisted him. Twitter hired him.

nd Early December 28, 2012 1:53 PM

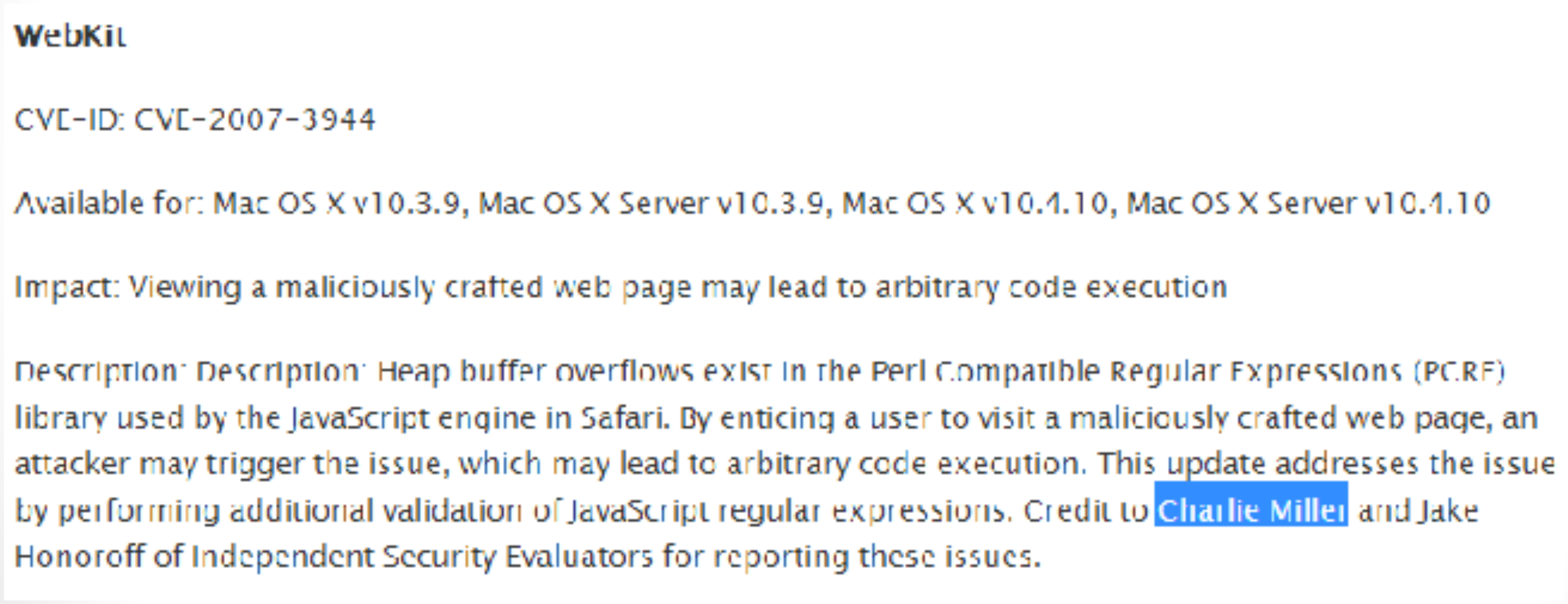
By Rosalind Early

a green military jacket and cap, Charlie Miller explains to a group of hackers and cyber
ow he'd build a cyber army in North Korea. He boasts that his army could infiltrate mil
and power grids, interrupt cellphone service, and take over millions of computers arou
: the bargain-basement price of $49 million." In this presentation, delivered at the 2010
e in Las Vegas, he's included doctored photos of himself standing beside North Korea's
m Jong Il, with the premise that he's been kidnapped.

Success

- Within two weeks of part time work, we had successfully
  - discovered a vulnerability
  - developed a toolchain for working with the iPhone's architecture
  - created a proof-of-concept exploit capable of delivering files from the user's iPhone to a remote attacker
- Notified apple of the vulnerability and proposed a patch.
- Apple subsequently resolved the issue and release and released a patch.

CVE-2007-3944 Issued and Patched



WebKit

CVE-ID: CVE-2007-3944

Available for: Mac OS X v10.3.9, Mac OS X Server v10.3.9, Mac OS X v10.4.10, Mac OS X Server v10.4.10

Impact: Viewing a maliciously crafted web page may lead to arbitrary code execution

Description: Description: Heap buffer overflows exist in the Perl Compatible Regular Expressions (PCRE) library used by the JavaScript engine in Safari. By enticing a user to visit a maliciously crafted web page, an attacker may trigger the issue, which may lead to arbitrary code execution. This update addresses the issue by performing additional validation of JavaScript regular expressions. Credit to Charlie Miller and Jake Honoroff of Independent Security Evaluators for reporting these issues.

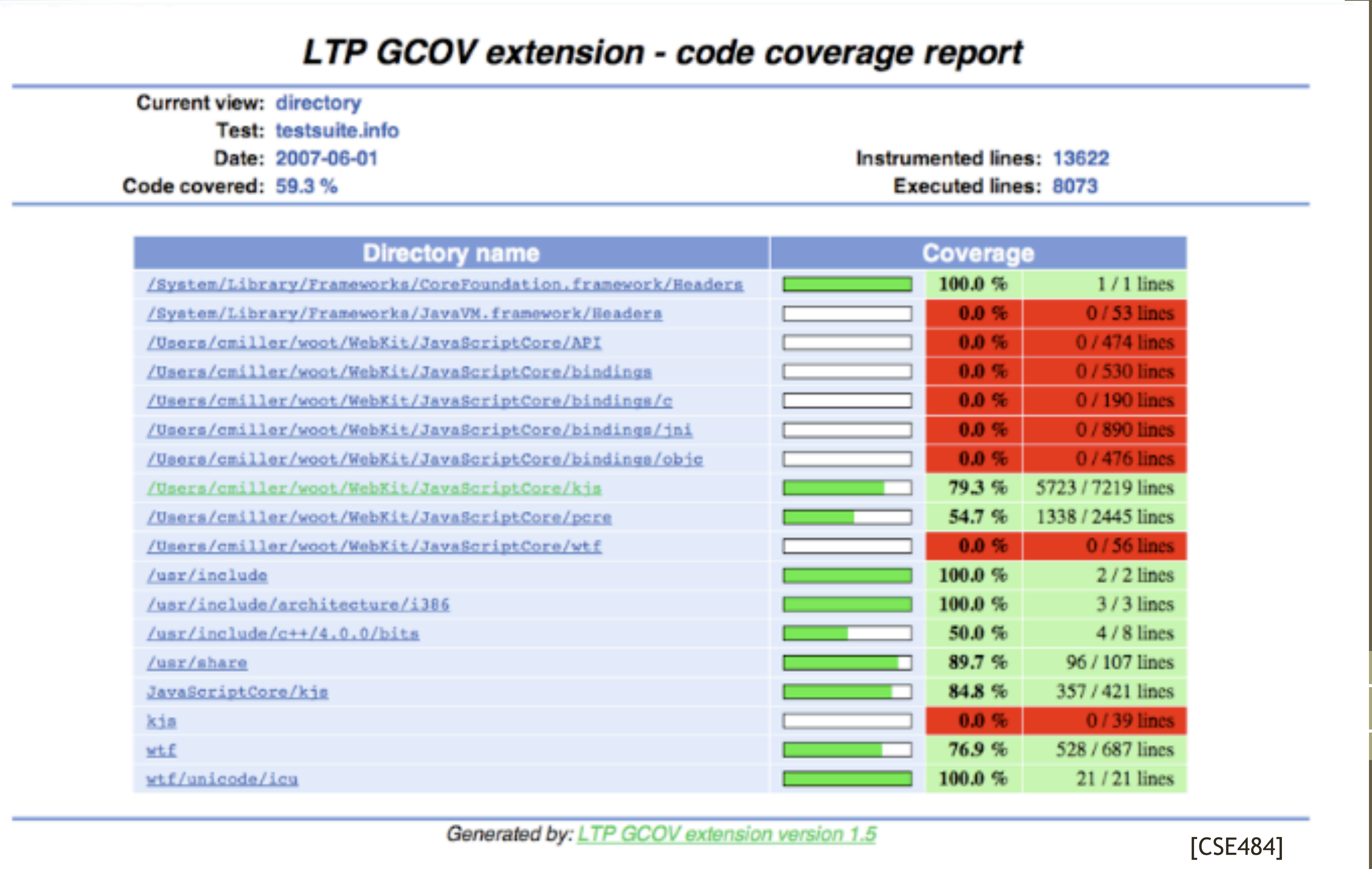
iPhone Attack

- iPhone Safari downloads malicious web page
    - Arbitrary code is run with administrative privileges
    - Can read SMS log, address book, call history, etc.
    - Can transmit collected data to attacker
    - Can perform physical actions on the phone
        - system sound and vibrate the phone for a second
        - could dial phone numbers, send text messages, or record audio (as a bugging device)

How Was This Discovered?

- WebKit is open source
    - “WebKit is an open source web browser engine. WebKit is also the name of the Mac OS X system framework version of the engine that's used by Safari, Dashboard, Mail, and many other OS X applications.”
- So we know what they use for code testing
    - Use code coverage to see which portions of code is not well tested
    - Tools gcov, icov, etc., measure test coverage

Collect Coverage for the Test Suite



LTP GCOV extension - code coverage report

Current view:	directory
Test:	testsuite.info
Date:	2007-06-01
Code covered:	59.3 %
Instrumented lines:	13622
Executed lines:	8073

Directory name	Coverage	
/System/Library/Frameworks/CoreFoundation.framework/Headers	100.0 %	1 / 1 lines
/System/Library/Frameworks/JavaVM.framework/Headers	0.0 %	0 / 53 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/API	0.0 %	0 / 474 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings	0.0 %	0 / 530 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings/c	0.0 %	0 / 190 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings/jni	0.0 %	0 / 890 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/bindings/objc	0.0 %	0 / 476 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/kjs	79.3 %	5723 / 7219 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/pcre	54.7 %	1338 / 2445 lines
/Users/cmiller/woot/WebKit/JavaScriptCore/wtf	0.0 %	0 / 56 lines
/usr/include	100.0 %	2 / 2 lines
/usr/include/architecture/i386	100.0 %	3 / 3 lines
/usr/include/c++/4.0.0/bits	50.0 %	4 / 8 lines
/usr/share	89.7 %	96 / 107 lines
JavaScriptCore/kjs	84.8 %	357 / 421 lines
kjs	0.0 %	0 / 39 lines
wtf	76.9 %	528 / 687 lines
wtf/unicode/icu	100.0 %	21 / 21 lines

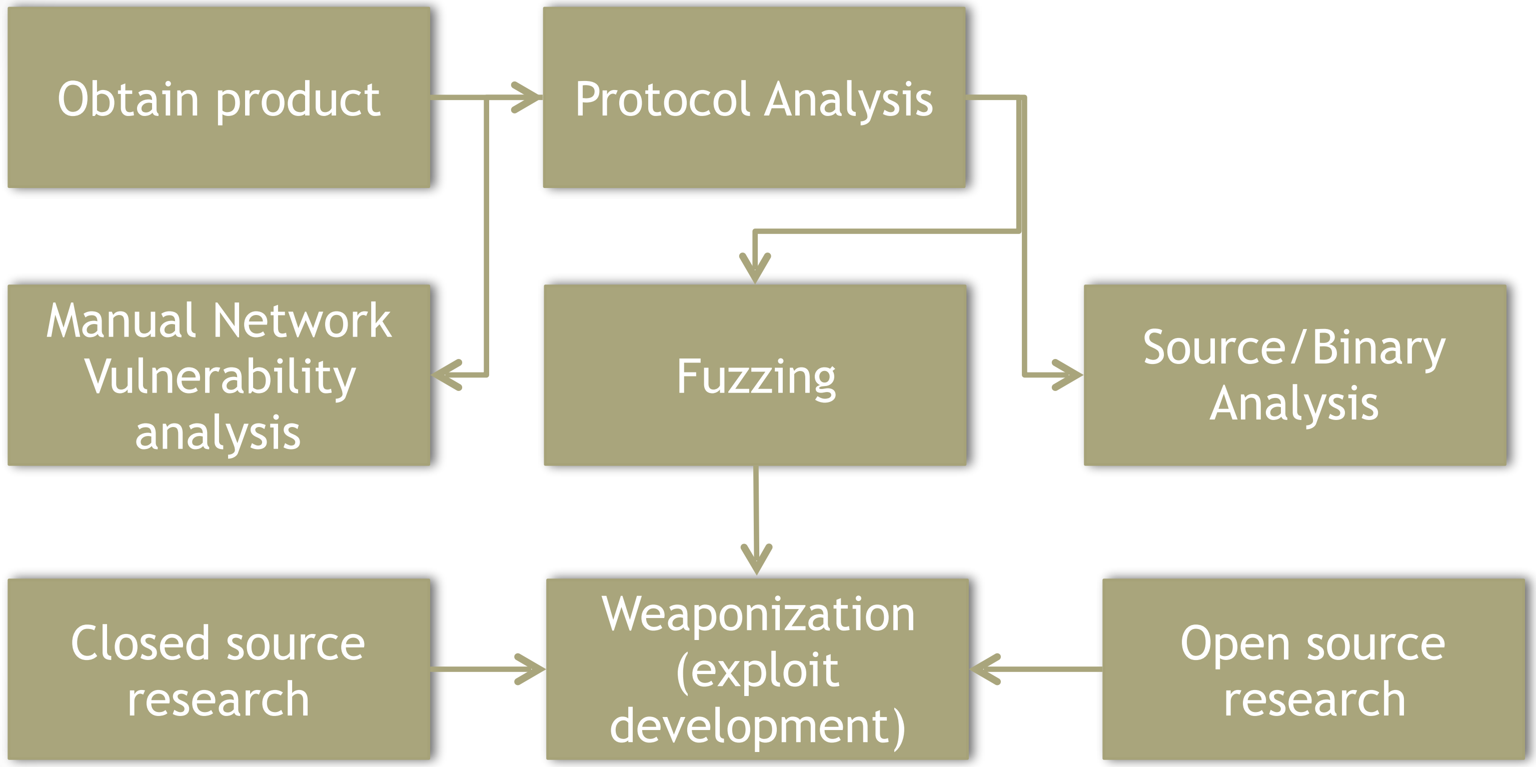
Generated by: LTP GCOV extension version 1.5

[CSE484]

What to Focus on?

- 59.3% of 13,622 lines in JavaScriptCore were covered
  - 79.3% of main engine covered
  - 54.7%ofPerlCompatibleRegularExpression(PCRE)covered
- Next step: focus on PCRE
  - Wrote a PCRE fuzzer (20 lines of perl)
  - Ran it on standalone PCRE parser (pcredemo from PCRE library)
  - Started getting errors: PCRE compilation failed at offset 6: internal error: code overflow
- Evil regular expressions crash mobile Safari

The Attacker Plan



But… why do it?

Last step…Sell it!

- Market for 0-Days ~$10K-100K

ZERO DAY INITIATIVE

eEye Digital Security®

GLEG

VERISIGN®

The Bug Bounty List

Welcome to Bugcrowd's community powered list of bug bounty programs

Lessons about Fuzzing

- Protocol knowledge is helpful
  - Generational beats random, better specification make better fuzzers

- Using more fuzzers is better
  - Each one will vary and find different bugs

- The longer you run (typically) the more bugs you'll find

- Guide the process, fix it when it break or fails to reach where you need it to go

- Code coverage can serve as a useful guide

Acknowledgments

[CSE484] CSE484/CSE584, BLACK BOX TESTING AND FUZZING, Dr. Benjamin Livshits