Fuzz By Number
More Data About Fuzzing Than You Ever Wanted To Know

Charlie Miller
Independent Security Evaluators
cmiller@securityevaluators.com
March 28, 2008
Who Am I?

- Former NSA security guy
- Break stuff: iPhone, SecondLife
- Give talks
- Write books
  - “Open Source Fuzzing Tools” (co-author)
  - “Fuzzing for Software Testing and Quality Assurance”
- Due out in June
Agenda

- Fuzzing, why we care
- How do you test fuzzers?
- My testing
- Results
- Why some bugs are harder to find than others
- Analysis and fun facts
Fuzzing

- Send invalid/semi-valid data into a system
  - If data is too valid, might not cause problems
  - If data is too invalid, might be quickly rejected
- Monitor system for faults
- Not the best tool, but finds lots of bugs
- Better at finding some classes of bugs than others
  - i.e. buffer overflows versus race conditions
Generating Test Cases

- Mutation-based approach
  - Take valid data and add anomalies
  - Only as good as the quality of valid data
  - Easy: requires no knowledge of protocol

- Generation-based approach
  - Generate test cases from protocol specification
  - Hard: need to represent all possibilities of inputs
I Heard Fuzzing Is Useful...

- Which fuzzer do I use?
Fuzzing Lifecycle

- Identifying interfaces
- Input generation  <-- This is all we test
- Sending inputs
- Target monitoring
- Exception analysis
- Reporting
How To Test Fuzzers?

- Retrospective testing
- Simulated vulnerability discovery
- Code coverage analysis
Retrospective Testing

- Time period is selected, say 6 months
- All security bugs in the products under study that emerged during the testing period are identified
- 6 month old fuzzers are run against 6 month old products
- We see if the “new” bugs are found
Retrospective Testing (Cont.)

- **Positives**
  - Measures how well fuzzers find real bugs in real programs

- **Negatives**
  - In good products, not many bugs come out in 6 months
  - Small sample size - hard to draw conclusions
  - Old versions of fuzzers are being tested
Simulated Vulnerability Discovery

- Experienced security researcher adds bugs to a product
- Bugs should be representative of the types of bugs found in this product in the past
- Each bug is verified to be reachable from an external interface
- Another researcher uses fuzzers to try to find these “fake” bugs
Fake Bugs

- Positives
  - Large sample size - add as many bugs as you want
  - The fuzzers still has to actually find the bugs

- Negatives
  - Bugs aren’t “real” - depend on the prejudices of the person adding them
Code Coverage Analysis

- Instrument the target application to measure the amount of code each fuzzer executes.
- Absolute numbers are meaningless, but relative numbers can be used.
- Lines not executed by a fuzzer indicate the fuzzer will not find bugs in those lines (if they exist).
- Measure “opportunity” of finding bugs.
Code Coverage

- Positives
  - Easy to obtain

- Negatives
  - Doesn’t actually measure “bug finding” ability
    - Measures what isn’t tested
  - Covered does not necessarily mean fuzzed
    - Think non-security regression tests
Our Testing

- Three network protocols
- Two servers, one client
- A handful of fuzzers
- Simulated vulnerability discovery and code coverage used
Caveats

- In real life, choice of fuzzer will depend heavily on your particular project
- Funding can be an issue - commercial fuzzers are expensive!
- Fuzzing an obscure or proprietary protocol may limit your choices
- This testing was only 3 protocols and relied heavily on the placement of the fake bugs - buyer beware
Introducing The Fuzzers

- General Purpose Fuzzer (GPF)
- The Art of Fuzzing (Taof)
- ProxyFuzz
- Mu-4000
- Codenomicon
- beSTORM
- Application specific fuzzers: FTPfuzz, PROTOS
GPF

- Open source
- Mutation based (requires packet capture)
- Parses packet capture and adds anomalies
- Can do this automatically or with a custom written “tokAid”
  - Custom tokAids can take many hours to write
- SuperGPF: a mode which modifies packet capture, adds anomalies, and launches many GPF instances
  - Only works for text based protocols
Taof

- Open source, mutation based
- GUI based
- User dissects the captured packets and identifies length fields, etc.
  - Effort comparable to writing a GPF tokAid
- Types of anomalies added are configurable
- Currently cannot handle length fields within length fields
  - Limits effectiveness in many binary protocols
ProxyFuzz

- Open source, mutation based
- Sits in the middle of traffic and randomly injects anomalies into live traffic
- Can set up and run in a matter of seconds
- Completely protocol unaware
Mu-4000

- Commercial fuzzer from Mu Security
- Generation based
  - Understands 55+ protocols
- Easy to use
- Can only fuzz protocols it knows
- Can only fuzz servers
- Sophisticated target monitoring
Codenomicon

- Commercial, generation based fuzzer
- Understands 130+ protocols
- Can only fuzz these protocols
- Fuzz client, server, and file parsing applications
- Limited or no monitoring capabilities
beSTORM

- Commercial, generation based fuzzer
- Understands 50+ protocols
- Can be used to fuzz arbitrary protocols
  - Configured through GUI
- Sophisticated monitoring capabilities
Application Specific Fuzzers

- FTPFuzz
  - GUI driven, open source, generation based
  - Only fuzzes FTP servers
- PROTOS SNMP test suite
  - Generation based
  - Java command line application fires off SNMP packets
- Found all those ASN.1 bugs a few years ago
What’s Missing?

- What about SPIKE, Sulley, Peach, etc...
- These are fuzzing frameworks, not fuzzers
- Their effectiveness is based solely on the quality of the protocol description they are given
  - We wouldn’t be testing the frameworks, but the specification files
- We’d have to write the protocol descriptions - I’m too lazy to do that!
Targets

- FTP Server - ProFTPD
  - Uses common ASCII based protocol
- SNMP Server - Net-SNMP
  - Uses binary based protocol
- DNS client - dig from BIND
  - Uses binary based protocol
The Bugs

- 17 bugs added to each application - Thanks Jake Honoroff!
- Half were buffer overflows
- A fourth were format strings
- A fourth were others types of issues: command injection, double free, wild writes, etc.
- Not detectable with normal client (not THAT obvious)
- Prefaced with logging code
- Not necessarily “exploitable” - but probably
Example: FTP Bug #0

MODRET xfer_type(cmd_rec *cmd) {
...
    if (strstr(get_full_cmd(cmd), "%")!=NULL) {
        BUGREPORT(0);
    }
    char tempbuf[32];
    snprintf(tempbuf, 32, "%s not understood", get_full_cmd(cmd));
    pr_response_add_err(R_500, tempbuf);

- This is a format string bug because
  pr_response_add_err() expects a format string for the
  second argument
Results!
<table>
<thead>
<tr>
<th>Bug</th>
<th>0</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>9</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPF Partial</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPF Full</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Super GPF</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Taof Partial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Taof Full</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ProxyFuzz Partial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProxyFuzz Full</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mu-4000</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>FTPfuzz</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Codenomicon</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FTP - Summary
<table>
<thead>
<tr>
<th>Bug</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPF Generic</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPF SNMP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProxyFuzz</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mu-4000</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROTOS</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Codenomicon</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beSTORM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SNMP Summary

% bugs found

% code coverage

Random
GPF Generic
GFP SNMP
ProxyFuzz
Mu-4000
PROTOS
Codenomicon
beSTORM
<table>
<thead>
<tr>
<th>Bug</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPF Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPF Generic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProxyFuzz</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Codenomicon</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>beSTORM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
DNS Summary

- Random
- GPF Generic
- ProxyFuzz
- Codenomicon
- beSTORM

Bar chart showing:
- % Bugs found
- % Code coverage

Values:
- 0
- 15
- 30
- 45
- 60
A Closer Look
FTP Oddities

- Bugs 9, 12, and 13 were found by GPF but no other fuzzers.
- Bugs 14 and 16 were found by Taof and ProxyFuzz but no other fuzzers.
- Bugs 4, 5, and 15 were found by the generational based fuzzers, but not the mutation based ones.
FTP Bug 9

Generation based fuzzers didn’t run SIZE verb - not in RFC

Likewise, other 2 bugs are in EPSV
FTP Bug 16

```c
MODRET core_eprt(cmd_rec *cmd) {
    char delim = '\0', *argstr = pstrdup(cmd->tmp_pool, cmd->argv[1]);
    ...
    /* Format is <d>proto<d>ip address<d>port<d> (ASCII in network order),
      * where <d> is an arbitrary delimiter character.
     */
    delim = *argstr++;
    ...
    while (isdigit((unsigned char) *argstr))
        argstr++;
    ...
    if (*argstr == delim)
        argstr++;
    ...
    if ((tmp = strchr(argstr, delim)) == NULL) {
        char tempbuf[64];
        if(strstr(cmd->argv[1], "%")!=NULL){
            BUGREPORT(16);
        }
        snprintf(tempbuf, 64, "badly formatted EPRT argument: '%s'", cmd->argv[1]);
        pr_response_add_err(R_501, tempbuf);
        return ERROR(cmd);
    }
    ...
```
FTP Bug 16 (Cont.)

- Need to not have enough delimiters
- The data after the second one needs to have a format string specifier
- Generation based fuzzers did not issue EPRT
- GPF was not random enough
FTP Bug 4

char *dir_canonical_path(pool *p, const char *path) {
char buf[PR_TUNABLE_PATH_MAX + 1] = {'\0'};
char work[256 + 1] = {'\0'};

if (*path == '~') {
    if (strlen(path) > 256 + 1) {
        BUGREPORT(4);
    }
    if (pr_fs_interpolate(path, work, strlen(path)) != 1) {
        if (pr_fs_dircat(work, sizeof(work), pr_fs_getcwd(), path) < 0)
            return NULL;
    }
}

- Need a long path path that starts with a ‘~’.
FTP Bug 4 (Cont.)

- Generation based fuzzers got this one
- Mutation based did not - never began a path with a ‘~’

```c
char *dir_canonical_path(pool *p, const char *path) {
    char buf[PR_TUNABLE_PATH_MAX + 1] = {'\0'};
    char work[256 + 1] = {'\0'};
    ...
    if (*path == ' ~') {
        if (strlen(path) > 256 + 1) {
            BUGREPORT(4);
        }
        if (pr_fs_interpolate(path, work, strlen(path)) != 1) {
            if (pr_fs_dircat(work, sizeof(work), pr_fs_getcwd(), path) < 0) {
                return NULL;
            }
        }
    } else {
        if (pr_fs_dircat(work, sizeof(work), pr_fs_getcwd(), path) < 0) {
            return NULL;
        }
    }
    pr_fs_clean_path(work, buf, sizeof(buf));
```
int snmp_pdu_parse(netsnmp_pdu *pdu, u_char * data, size_t * length)
{
  ...
  data = asn_parse_sequence(data, length, &type, (ASN_SEQUENCE | ASN_CONSTRUCTOR), "varbinds");
  if (data == NULL)
    return -1;
  ...
  while ((int) *length > 0) {
    ...
    switch ((short) vp->type) {
    ...
    case ASN_OCTET_STR:
    case ASN_IPADDRESS:
    case ASN_OPAQUE:
    case ASN_NSAP:
      if (vp->val_len < sizeof(vp->buf)) {
        vp->val.string = (u_char *) vp->buf;
      } else {
        vp->val.string = (u_char *) malloc(200);
        if (vp->val_len > 200)
          {
            BUGREPORT(4);
          }
      }
    ...
    asn_parse_string(var_val, &len, &vp->type, vp->val.string, &vp->val_len);
    break;
SNMP Bug #4 (Cont.)

- Bug is reached with a particular type of packet and a large length and corresponding long string.
- GPF executes the function but doesn’t even make it to the switch statement (i.e. its too random).
- ProxyFuzz and Mu-4000 sent the right kind of packet, but not with a long enough string.

```c
3292:     case ASN_OBJQ02,
3292:       case ASN_NSAP:
3292:         if (vp->val_len < sizeof(vp->buf)) {
3292:             vp->val.string = (u_char *) vp->buf;
3292:         } else {
3292:             vp->val.string = (u_char *) malloc(vp->val_len);
3292:         }
3292:         if (vp->val.string == NULL) {
3292:             return -1;
3292:         }
3292:         asn_parse_string(var_val, &len, &vp->type, vp->val.string,
3292:                        &vp->val_len);
3292:         break;
```
General Conclusions
The More Fuzzers The Better

# bugs found: all fuzzers
# bugs found: best fuzzer

FTP
SNMP
DNS
Generation Based Approach
Most Effective

# bugs found: mutation based
# bugs found: generation based

FTP
SNMP
DNS
Initial Test Cases Important

- **# FTP bugs found - partial capture**
- **# FTP bugs found - full capture**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Partial Capture</th>
<th>Full Capture</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPF</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Taof</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>ProxyFuzz</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Protocol Knowledge Is Good

% SNMP bugs found

ProxyFuzz

GPF Generic

GPF w/tokAid

Mu/Code.
Does Code Coverage Predict Bug Finding?
More Code Coverage...
More Code Coverage...
Statistics Says “Yes”

Dep Var: BUGS   N: 11   Multiple R: 0.716   Squared multiple R: 0.512

Adjusted squared multiple R: **0.458**   Standard error of estimate: 9.468

<table>
<thead>
<tr>
<th>Effect</th>
<th>Coefficient</th>
<th>Std Error</th>
<th>Std Coef</th>
<th>Tolerance</th>
<th>t</th>
<th>P(2 Tail)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-5.552</td>
<td>8.080</td>
<td>0.000</td>
<td>.</td>
<td>-0.687</td>
<td>0.509</td>
</tr>
<tr>
<td>CC</td>
<td><strong>0.921</strong></td>
<td>0.300</td>
<td>0.716</td>
<td>1.000</td>
<td>3.074</td>
<td>0.013</td>
</tr>
</tbody>
</table>

Analysis of Variance

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum-of-Squares</th>
<th>df</th>
<th>Mean-Square</th>
<th>F-ratio</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>847.043</td>
<td>1</td>
<td>847.043</td>
<td>9.449</td>
<td><strong>0.013</strong></td>
</tr>
<tr>
<td>Residual</td>
<td>806.813</td>
<td>9</td>
<td>89.646</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A 1% increase in code coverage increases the percentage of bugs found by .92%
How Long To Run Fuzzers?

Time to discovery in minutes, ProxyFuzz versus DNS
A Real Bug

- All this fuzzing with different fuzzers against a real program might have actually found a real bug.
- It is possible that some were found but were lost in the “noise.”
- One Net-SNMP bug was found (DOS):
  - Only found by Codenomicon
  - Reported and fixed
Conclusions

- Verified a lot of what intuition tells us
- Incorporate as much protocol specific knowledge as possible
- Commercial fuzzers are good (if you can afford them)
- Multiple fuzzers are better than one
- Run fuzzers for a very long time (longer than you’d think)
- Code coverage in fuzzers is useful as a measurement
Special Thanks To:

- Commercial fuzzer vendors who let me use their product - very cool!
- Open source fuzzer developers who helped me find/fix bugs in their fuzzers

Thanks!
Questions?

- Buggy programs will be made available
- Contact me at: cmiller@securityevaluators.com