

# In Search Of Shotgun Parsers

Katie Underwood

University of Calgary

Michael Locasto

SRI International

May 25, 2016

# Overview

Context

Defining The Shotgun Parser

Tainted Path Length In Android Applications

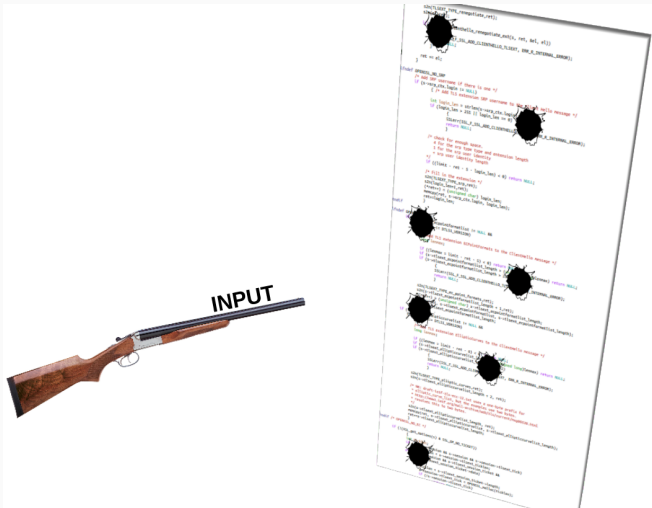
Our Definition In The Wild

Future Work

# WHAT ARE WE LOOKING FOR?

Defining The Shotgun Parser

# Why Shotgun?



Input **use** and **recognition** intermixed throughout!

# What Are We Looking For?

- Before we go searching for shotgun parsers, we need to know what we're looking for!
- How will we know a shotgun parser when we see one?
- We frame our definition in the context of static taint analysis of control flow graphs

# Hallmarks of the Shotgun Parser

## Large Spread Relative To Size

How far does untrusted data propagate through the code?

# Hallmarks of the Shotgun Parser

## Large Spread Relative To Size

How far does untrusted data propagate through the code?

## Use Before Full Recognition

Is input data fully validated before being used?

# Hallmarks of the Shotgun Parser

## Large Spread Relative To Size

How far does untrusted data propagate through the code?

## Use Before Full Recognition

Is input data fully validated before being used?

## Large Number of Variables Involved In Each Tainted Path

How much program state is affected by properties 1 and 2?



## Property 1: Spread Relative To Size

- Consider an application  $\mathcal{A}$ , which reads a set of untrusted inputs  $\mathcal{N}$

# Property 1: Spread Relative To Size

- Consider an application  $\mathcal{A}$ , which reads a set of untrusted inputs  $\mathcal{N}$
- Let  $G$  be the static control-flow graph which describes  $\mathcal{A}$

# Property 1: Spread Relative To Size

- Consider an application  $\mathcal{A}$ , which reads a set of untrusted inputs  $\mathcal{N}$
- Let  $G$  be the static control-flow graph which describes  $\mathcal{A}$
- Let  $P_n$  be the connected subgraph induced by the vertices of  $G$  tainted by  $n \in \mathcal{N}$ , where  $d(P_n) \leq d(G)$

# Property 1: Spread Relative To Size

- Consider an application  $\mathcal{A}$ , which reads a set of untrusted inputs  $\mathcal{N}$
- Let  $G$  be the static control-flow graph which describes  $\mathcal{A}$
- Let  $P_n$  be the connected subgraph induced by the vertices of  $G$  tainted by  $n \in \mathcal{N}$ , where  $d(P_n) \leq d(G)$
- Let  $S = \{P_i | 1 \leq i \leq |\mathcal{N}|\}$  be the set of all taint-induced subgraphs on  $G$

# Property 1: Spread Relative To Size

Shotgun parser indicators:

- $d(P_n)$  comparable to  $d(G)$ 
  - Indicates input  $n$  not handled in principled manner

# Property 1: Spread Relative To Size

Shotgun parser indicators:

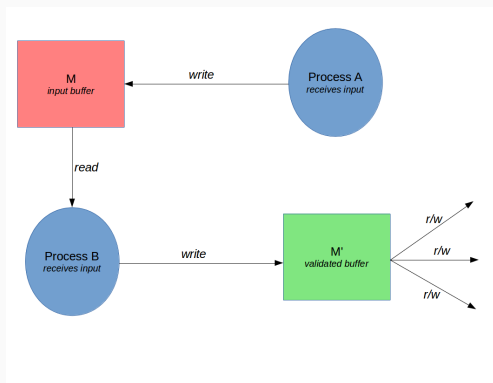
- $d(P_n)$  comparable to  $d(G)$ 
  - Indicates input  $n$  not handled in principled manner
- Large  $|S|$ 
  - Evidence for presence of multiple shotgun parsers in  $\mathcal{A}$

## Property 2: Use Before Full Recognition

- We can't quantify whether arbitrary input to an arbitrary piece of code is "fully recognized"
- We *can* start to define a set of standards for handling of specific data types

# Property 2: Use Before Full Recognition

For example:

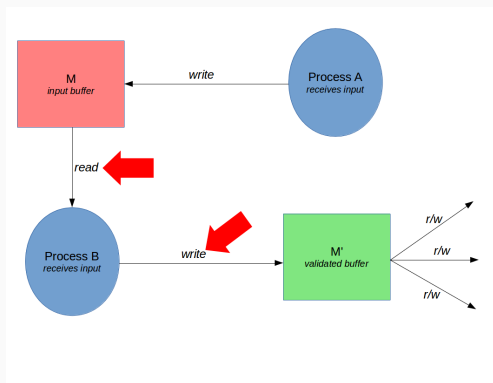


- “For inputs of type 0, you must do 5 reads of 4 bytes each, then write 20 bytes in a specific order”
- Identify read/write memory events which take place after input is received



# Property 2: Use Before Full Recognition

For example:



- “For inputs of type O, you must do 5 reads of 4 bytes each, then write 20 bytes in a specific order”
- Identify read/write memory events which take place after input is received

## Property 3: Number of Tainted Input Variables

- Consider again a tainted subgraph  $P_n$

## Property 3: Number of Tainted Input Variables

- Consider again a tainted subgraph  $P_n$
- Let  $P_n$  now be a weighted graph, where each edge  $E(x, y)$  corresponds to the number of variables tainted by  $n$  after node  $x$

## Property 3: Number of Tainted Input Variables

Shotgun parser indicators:

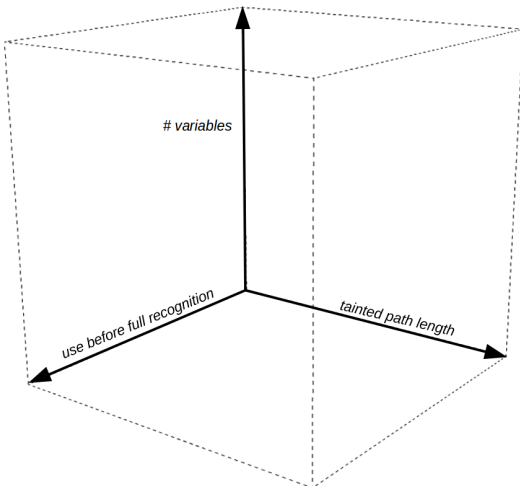
- Large number of tainted variables compared to total number of variables
  - Indicates untrusted input affects significant proportion of program state

## Property 3: Number of Tainted Input Variables

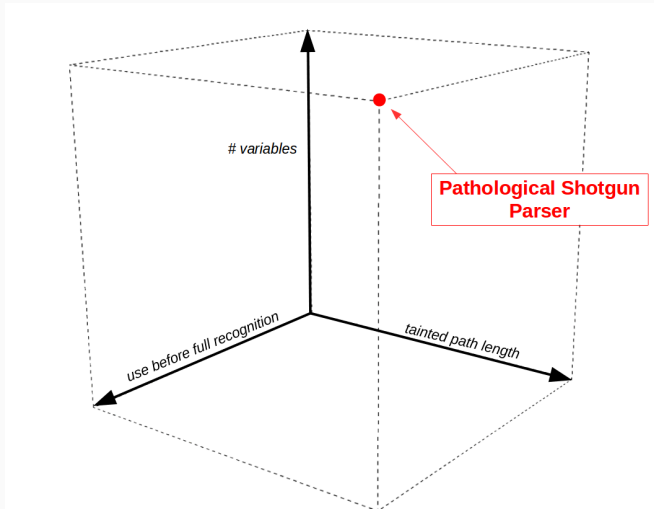
Shotgun parser indicators:

- Large number of tainted variables compared to total number of variables
  - Indicates untrusted input affects significant proportion of program state
- Areas of  $P_n$  where edge weight increases may merit further study
  - Allows us to triage program statements / methods for further analysis

# Definition Summary



# Definition Summary



The “worst case” shotgun parser exhibits all three properties in abundance!

# CASE STUDY: ANDROID

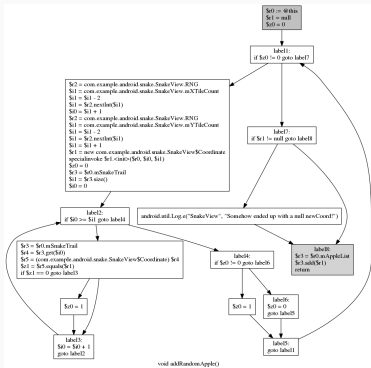
First Steps Towards Automated  
Detection



# Our Goals

- Establish foundation for a recognizer
- First look at “state of affairs” in Android applications
- Start examining a different class of errors through the LangSec lens

# Our Approach



### Simple CFG for one module of the classic game "Snake"

- Static taint analysis of statement-level control flow graphs
- Compute length of tainted path corresponding to each source
- Analysis uses the Jimple intermediate representation

# FlowDroid



- Open-source static analysis framework for Android
- Developed by the Secure Software Engineering Group at Paderborn University/ TU Darmstadt

<https://blogs.uni-paderborn.de/sse/tools/flowdroid/>

## We Add:

- Tracking for *all* tainted paths, not only those terminating in a sink
- Unique identifiers for each taint source
- Specific API call source for each taint
- Taint propagation handler functions to measure input path length

# Our Implementation

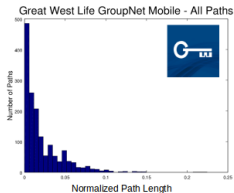
Each time a taint is propagated, our custom handler is invoked:

- Capture incoming flow data object  $F$  and outgoing set of flow data objects  $\mathcal{F}_{\text{out}}$
- If  $F$  has not been seen before:
  - Init  $F.\text{length} = 0$
  - Store original source context of  $F$ .
- For each flow fact  $f \in \mathcal{F}_{\text{out}}$ :
  - $f.\text{length} = F.\text{length} + 1$
  - Store source context information for  $f$

# Workflow



- 56 free apps from 12 different categories

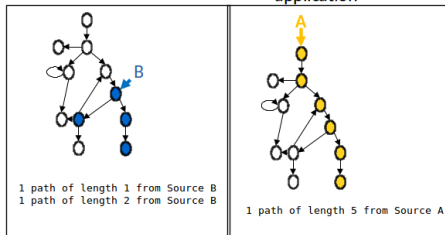


- Histogram of normalized path lengths

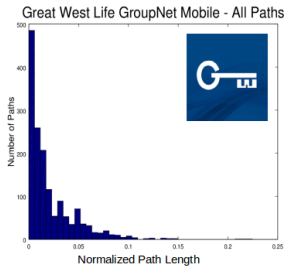
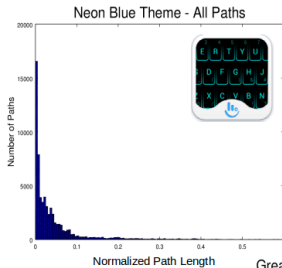


*FlowDroid logo credit - Secure Software Engineering Group at the European Center for Security and Privacy by Design*

- FlowDroid – Android Data Flow Analysis
- Modified this open-source tool to calculate tainted path lengths

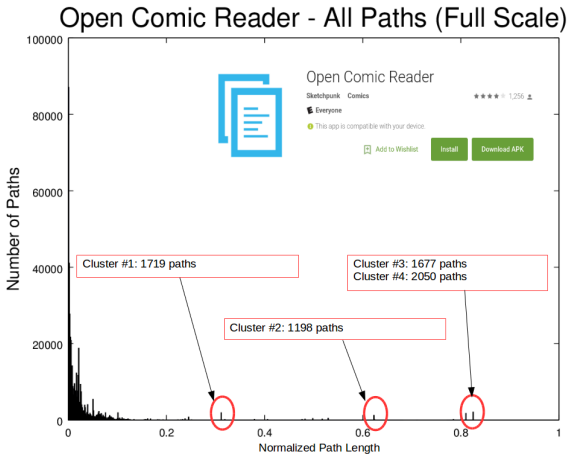


# Initial Results



# Some Thoughts..

- Our tool is:
  - The foundation of a full SGP recognizer
  - A *prioritization method* for app analysis



# OUR DEFINITION IN THE WILD

Let's Look At Real Stuff



# "ImageTragick" (CVE-2016-3714)

```
sanitize_command=SanitizeDelegateCommand(command);
if (asynchronous != MagickFalse)
    (void) ConcatenateMagickString(sanitize_command,"&",MagickPathExtent);
if (message != (char *) NULL)
    *message='\0';
#ifdef MAGICKCORE_POSIX_SUPPORT
if !defined(MAGICKCORE_HAVE_EXECVP)
    status=system(sanitize_command);
#else
    if ((asynchronous != MagickFalse) ||
        (strpbrk(sanitize_command,"&<>|") != (char *) NULL))
        status=system(sanitize_command);
    else
    {
        pid_t
            child_pid;

        /*
         * Call application directly rather than from a shell.
         */
        child_pid=(pid_t) fork();
        if (child_pid == (pid_t) -1)
            status=system(sanitize_command);
        else
            if (child_pid == 0)
            {
                status=execvp(arguments[1],arguments+1);
                _exit(1);
            }
        else
    }
```

# "ImageTragick" (CVE-2016-3714)

```
sanitize_command=SanitizeDelegateCommand(command);  
if (asynchronous != MagickFalse)  
    (void) ConcatenateMagickString(sanitize_command,"&",MagickPathExtent);  
if (message != (char *) NULL)  
    *message='\0';  
#if defined(MAGICKCORE_POSIX_SUPPORT)  
#if !defined(MAGICKCORE_HAVE_EXECVP)  
    status=system(sanitize_command);  
#else  
    if ((asynchronous != MagickFalse) ||  
        (strpbrk(sanitize_command,"&<>|") != (char *) NULL))  
        status=system(sanitize_command);  
    else  
    {  
        pid_t  
        child_pid;  
  
        /*  
         * Call application directly rather than from a shell.  
         */  
        child_pid=(pid_t) fork();  
        if (child_pid == (pid_t) -1)  
            status=system(sanitize_command);  
        else  
            if (child_pid == 0)  
            {  
                status=execvp(arguments[1],arguments+1);  
                _exit(1);  
            }  
        else
```

# "ImageTragick" (CVE-2016-3714)

```
sanitize_command=SanitizeDelegateCommand(command);  
if (asynchronous != MagickFalse)  
(void) ConcatenateMagickString(sanitize_command "%s MagickPathExtent");  
static char *SanitizeDelegateCommand(const char *command)  
{  
    char  
        *sanitize_command;  
  
    const char  
        *q;  
  
    register char  
        *p;  
  
    static char  
        whitelist[] =  
            "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_- "  
            ".@&;<>()|/\\\\'\\":%~`";  
  
    sanitize_command=AcquireString(command);  
    p=sanitize_command;  
    q=sanitize_command+strlen(sanitize_command);  
    for (p+=strspn(p,whitelist); p != q; p+=strspn(p,whitelist))  
        *p='_';  
    return(sanitize_command);  
}
```

# "ImageTragick" (CVE-2016-3714)

```
121 offset=localOffset(MagicCommands[i], client_name, client_name,
122 MagicCommands[i], &act1);
123 if (offset == 0)
124     break;
125 }
126 if (sizeof(MagicCommands)/sizeof(MagicCommands[0])):
127     if (i == 0 && (argc > 1))
128     {
129         for (i=1; i < (sizeof(MagicCommands)/sizeof(MagicCommands[0])); i++)
130         {
131             offset=localOffset(MagicCommands[i], client_name, arg[i]);
132             if (offset == 0)
133             {
134                 argc--;
135                 arg++;
136             }
137         }
138     }
139     if (sizeof(MagicCommands)/sizeof(MagicCommands[0])):
140     {
141         memset(&char, '*', 0);
142         status=MagicCommands[i](image, info, MagicCommands[i].command, arg, argp,
143 MagicCommands[i].use_metadata ? &metadata : (char *) 0, &encrypt);
144 if (&metadata != (char *) 0)
145 {
146     (memset(&char, '*', 0));
147     metadata=decryptString(metadata);
148 }
149 if (MagicCommands[i].command != CompareImagesCommand)
150     exit_sub(status != MagicFalse ? 0 : 1);
151 else
152 {
153     if (status == MagicFalse)
154         exit_sub(0);
155     else
156     {
157         const char
158             *option;
159         option=decryptString(image, info, "compare:discuss");
160         exit_sub(decryptString(option) ? 1 : 0);
161     }
162 }
163 image_info=destroyImageInfo(image, info);
164 encrypt=destroyEncryptContext(encrypt);
165 MagicCommands[i];
166 return(status);
167 }
168
169 #ifdef(MAGICCORE_WITHOUT_SUPPORT) || defined(_CYGWIN_) || defined(_WINDOWS_)
170 {
171     MagicArgv arg;
172     MagicArgv argp;
173 }
174
175 return(MagicArgv(arg, argp));
176
177 }
178
179 int main(int argc, char **argv)
180 {
181     char
182         *utf8;
183     int
184         status;
185     register int
186         i;
187     utf8=argv[0];
188     status=MagicArgv(arg, argp);
189     for (i=1; i < argc; i++)
190         utf8[i]=argv[i];
191     utf8[i]=0;
192     return(status);
193 }
194
195 #endif
```

# "ImageTragick" (CVE-2016-3714)

```

100         if (args[0].equalsIgnoreCase("quit")) {
101             if (client_name != null) {
102                 client_name.close();
103             }
104             return;
105         }
106         if (args[0].equalsIgnoreCase("help")) {
107             printHelp();
108             return;
109         }
110         if (args[0].equalsIgnoreCase("add")) {
111             if (args.length < 2) {
112                 System.out.println("Usage: add <name>");
113                 return;
114             }
115             String name = args[1];
116             add(name);
117             return;
118         }
119         if (args[0].equalsIgnoreCase("delete")) {
120             if (args.length < 2) {
121                 System.out.println("Usage: delete <name>");
122                 return;
123             }
124             String name = args[1];
125             delete(name);
126             return;
127         }
128         if (args[0].equalsIgnoreCase("list")) {
129             list();
130             return;
131         }
132         if (args[0].equalsIgnoreCase("clear")) {
133             clear();
134             return;
135         }
136         if (args[0].equalsIgnoreCase("quit")) {
137             if (client_name != null) {
138                 client_name.close();
139             }
140             return;
141         }
142         if (args[0].equalsIgnoreCase("help")) {
143             printHelp();
144             return;
145         }
146         if (args[0].equalsIgnoreCase("add")) {
147             if (args.length < 2) {
148                 System.out.println("Usage: add <name>");
149                 return;
150             }
151             String name = args[1];
152             add(name);
153             return;
154         }
155         if (args[0].equalsIgnoreCase("delete")) {
156             if (args.length < 2) {
157                 System.out.println("Usage: delete <name>");
158                 return;
159             }
160             String name = args[1];
161             delete(name);
162             return;
163         }
164         if (args[0].equalsIgnoreCase("list")) {
165             list();
166             return;
167         }
168         if (args[0].equalsIgnoreCase("clear")) {
169             clear();
170             return;
171         }
172         if (args[0].equalsIgnoreCase("quit")) {
173             if (client_name != null) {
174                 client_name.close();
175             }
176             return;
177         }
178         if (args[0].equalsIgnoreCase("help")) {
179             printHelp();
180             return;
181         }
182         if (args[0].equalsIgnoreCase("add")) {
183             if (args.length < 2) {
184                 System.out.println("Usage: add <name>");
185                 return;
186             }
187             String name = args[1];
188             add(name);
189             return;
190         }
191         if (args[0].equalsIgnoreCase("delete")) {
192             if (args.length < 2) {
193                 System.out.println("Usage: delete <name>");
194                 return;
195             }
196             String name = args[1];
197             delete(name);
198             return;
199         }
200         if (args[0].equalsIgnoreCase("list")) {
201             list();
202             return;
203         }
204         if (args[0].equalsIgnoreCase("clear")) {
205             clear();
206             return;
207         }
208         if (args[0].equalsIgnoreCase("quit")) {
209             if (client_name != null) {
210                 client_name.close();
211             }
212             return;
213         }
214         if (args[0].equalsIgnoreCase("help")) {
215             printHelp();
216             return;
217         }
218         if (args[0].equalsIgnoreCase("add")) {
219             if (args.length < 2) {
220                 System.out.println("Usage: add <name>");
221                 return;
222             }
223             String name = args[1];
224             add(name);
225             return;
226         }
227         if (args[0].equalsIgnoreCase("delete")) {
228             if (args.length < 2) {
229                 System.out.println("Usage: delete <name>");
230                 return;
231             }
232             String name = args[1];
233             delete(name);
234             return;
235         }
236         if (args[0].equalsIgnoreCase("list")) {
237             list();
238             return;
239         }
240         if (args[0].equalsIgnoreCase("clear")) {
241             clear();
242             return;
243         }
244         if (args[0].equalsIgnoreCase("quit")) {
245             if (client_name != null) {
246                 client_name.close();
247             }
248             return;
249         }
250         if (args[0].equalsIgnoreCase("help")) {
251             printHelp();
252             return;
253         }
254         if (args[0].equalsIgnoreCase("add")) {
255             if (args.length < 2) {
256                 System.out.println("Usage: add <name>");
257                 return;
258             }
259             String name = args[1];
260             add(name);
261             return;
262         }
263         if (args[0].equalsIgnoreCase("delete")) {
264             if (args.length < 2) {
265                 System.out.println("Usage: delete <name>");
266                 return;
267             }
268             String name = args[1];
269             delete(name);
270             return;
271         }
272         if (args[0].equalsIgnoreCase("list")) {
273             list();
274             return;
275         }
276         if (args[0].equalsIgnoreCase("clear")) {
277             clear();
278             return;
279         }
280         if (args[0].equalsIgnoreCase("quit")) {
281             if (client_name != null) {
282                 client_name.close();
283             }
284             return;
285         }
286         if (args[0].equalsIgnoreCase("help")) {
287             printHelp();
288             return;
289         }
290         if (args[0].equalsIgnoreCase("add")) {
291             if (args.length < 2) {
292                 System.out.println("Usage: add <name>");
293                 return;
294             }
295             String name = args[1];
296             add(name);
297             return;
298         }
299         if (args[0].equalsIgnoreCase("delete")) {
300             if (args.length < 2) {
301                 System.out.println("Usage: delete <name>");
302                 return;
303             }
304             String name = args[1];
305             delete(name);
306             return;
307         }
308         if (args[0].equalsIgnoreCase("list")) {
309             list();
310             return;
311         }
312         if (args[0].equalsIgnoreCase("clear")) {
313             clear();
314             return;
315         }
316         if (args[0].equalsIgnoreCase("quit")) {
317             if (client_name != null) {
318                 client_name.close();
319             }
320             return;
321         }
322         if (args[0].equalsIgnoreCase("help")) {
323             printHelp();
324             return;
325         }
326         if (args[0].equalsIgnoreCase("add")) {
327             if (args.length < 2) {
328                 System.out.println("Usage: add <name>");
329                 return;
330             }
331             String name = args[1];
332             add(name);
333             return;
334         }
335         if (args[0].equalsIgnoreCase("delete")) {
336             if (args.length < 2) {
337                 System.out.println("Usage: delete <name>");
338                 return;
339             }
340             String name = args[1];
341             delete(name);
342             return;
343         }
344         if (args[0].equalsIgnoreCase("list")) {
345             list();
346             return;
347         }
348         if (args[0].equalsIgnoreCase("clear")) {
349             clear();
350             return;
351         }
352         if (args[0].equalsIgnoreCase("quit")) {
353             if (client_name != null) {
354                 client_name.close();
355             }
356             return;
357         }
358         if (args[0].equalsIgnoreCase("help")) {
359             printHelp();
360             return;
361         }
362         if (args[0].equalsIgnoreCase("add")) {
363             if (args.length < 2) {
364                 System.out.println("Usage: add <name>");
365                 return;
366             }
367             String name = args[1];
368             add(name);
369             return;
370         }
371         if (args[0].equalsIgnoreCase("delete")) {
372             if (args.length < 2) {
373                 System.out.println("Usage: delete <name>");
374                 return;
375             }
376             String name = args[1];
377             delete(name);
378             return;
379         }
380         if (args[0].equalsIgnoreCase("list")) {
381             list();
382             return;
383         }
384         if (args[0].equalsIgnoreCase("clear")) {
385             clear();
386             return;
387         }
388         if (args[0].equalsIgnoreCase("quit")) {
389             if (client_name != null) {
390                 client_name.close();
391             }
392             return;
393         }
394         if (args[0].equalsIgnoreCase("help")) {
395             printHelp();
396             return;
397         }
398         if (args[0].equalsIgnoreCase("add")) {
399             if (args.length < 2) {
400                 System.out.println("Usage: add <name>");
401                 return;
402             }
403             String name = args[1];
404             add(name);
405             return;
406         }
407         if (args[0].equalsIgnoreCase("delete")) {
408             if (args.length < 2) {
409                 System.out.println("Usage: delete <name>");
410                 return;
411             }
412             String name = args[1];
413             delete(name);
414             return;
415         }
416         if (args[0].equalsIgnoreCase("list")) {
417             list();
418             return;
419         }
420         if (args[0].equalsIgnoreCase("clear")) {
421             clear();
422             return;
423         }
424         if (args[0].equalsIgnoreCase("quit")) {
425             if (client_name != null) {
426                 client_name.close();
427             }
428             return;
429         }
430         if (args[0].equalsIgnoreCase("help")) {
431             printHelp();
432             return;
433         }
434         if (args[0].equalsIgnoreCase("add")) {
435             if (args.length < 2) {
436                 System.out.println("Usage: add <name>");
437                 return;
438             }
439             String name = args[1];
440             add(name);
441             return;
442         }
443         if (args[0].equalsIgnoreCase("delete")) {
444             if (args.length < 2) {
445                 System.out.println("Usage: delete <name>");
446                 return;
447             }
448             String name = args[1];
449             delete(name);
450             return;
451         }
452         if (args[0].equalsIgnoreCase("list")) {
453             list();
454             return;
455         }
456         if (args[0].equalsIgnoreCase("clear")) {
457             clear();
458             return;
459         }
460         if (args[0].equalsIgnoreCase("quit")) {
461             if (client_name != null) {
462                 client_name.close();
463             }
464             return;
465         }
466         if (args[0].equalsIgnoreCase("help")) {
467             printHelp();
468             return;
469         }
470         if (args[0].equalsIgnoreCase("add")) {
471             if (args.length < 2) {
472                 System.out.println("Usage: add <name>");
473                 return;
474             }
475             String name = args[1];
476             add(name);
477             return;
478         }
479         if (args[0].equalsIgnoreCase("delete")) {
480             if (args.length < 2) {
481                 System.out.println("Usage: delete <name>");
482                 return;
483             }
484             String name = args[1];
485             delete(name);
486             return;
487         }
488         if (args[0].equalsIgnoreCase("list")) {
489             list();
490             return;
491         }
492         if (args[0].equalsIgnoreCase("clear")) {
493             clear();
494             return;
495         }
496         if (args[0].equalsIgnoreCase("quit")) {
497             if (client_name != null) {
498                 client_name.close();
499             }
500             return;
501         }
502         if (args[0].equalsIgnoreCase("help")) {
503             printHelp();
504             return;
505         }
506         if (args[0].equalsIgnoreCase("add")) {
507             if (args.length < 2) {
508                 System.out.println
```

# "ImageTragick" (CVE-2016-3714)

```
120 if(!strcmp(cmdname, MagicCommands[i].client_name, client_name,  
121 MagicCommands[i].extn);  
122 if (!isset on 0)  
123 break;  
124  
125 // MagicCommand arg, char *arg;  
126 {  
127 client_name = (char *)cmdname;  
128 {  
129 client_name = (char *)cmdname;  
130 }  
131  
132 MagicCommandType MagicCommandType(MagicCommands[i].magic_type, info,  
133 MagicCommands[i].arg, char *arg, char *extn);  
134  
135 char  
136 client_name(MagicCommandType,  
137 "extn");  
138  
139 double  
140 duration,  
141 serial;  
142  
143 MagicCommandType  
144 concurrent,  
145 request_warnings,  
146 extn);  
147  
148 register_size_t  
149 i;  
150  
151 size_t  
152 iterations,  
153 number_threads;  
154  
155 size_t  
156 m;  
157  
158 (void) setlocale(LC_ALL, "");  
159 (void) setlocale(LC_NUMERIC, "C");  
160 getPerfMonData(target, allPath, client_name);  
161 setPerfMonData(target, allPath, client_name);  
162  
163 concurrentMagicCommand  
164 duration = 0;  
165 iterations = 0;  
166 statusMagicType;  
167 request_warningsMagicType;  
168 for (i=0; i < (sizeof target) / sizeof target; i++)  
169 {  
170 option = (i % 10) + 1; if (i % 10 == 0) && i % 10 == 0  
171 {  
172 continue;  
173 }  
174 if (strcmp(option, "bench", option) == 0)  
175 {  
176 iterations = (int)target[i].iterations;  
177 concurrentMagicCommand;  
178 if (strcmp(option, "concurrent", option) == 0)  
179 {  
180 (void) setlocale(LC_NUMERIC, "C");  
181 if (strcmp(option, "dist", option) == 0, option) == 0)  
182 {  
183 distributeMagicCommand(target[i].iterations, exception);  
184 }  
185 }  
186 if (strcmp(option, "duration", option) == 0)  
187 {  
188 duration = (int)target[i].duration;  
189 if (strcmp(option, "request_warnings", option) == 0)  
190 {  
191 request_warningsMagicType;  
192 }  
193 if (iterations == 1)  
194 {  
195 char  
196 *text;  
197  
198 text = (char *) NULL;  
199 statusMagicType(i, target[i].iterations, exception);  
200 if (exception == 0)  
201 {  
202 if (exception == 0 && exception == 0)  
203 {  
204 request_warnings = MagicCommandType;  
205 statusMagicType;  
206 }  
207 if (test != (char *) NULL)  
208 {  
209 }  
210 }
```

# "ImageTragick" (CVE-2016-3714)

```

100  if (argc < 2) {
101      printf("Usage: %s [filename]\n", argv[0]);
102      return 1;
103  }
104  if (argc > 2) {
105      printf("Error: too many arguments\n");
106      return 1;
107  }
108  // Open the file
109  FILE *fp = fopen(argv[1], "r");
110  if (fp == NULL) {
111      printf("Error: cannot open file\n");
112      return 1;
113  }
114  // Read the file
115  char *buffer = malloc(1024);
116  if (buffer == NULL) {
117      printf("Error: cannot allocate memory\n");
118      return 1;
119  }
120  while (1) {
121      // Read a line of text
122      if (!fgets(buffer, 1024, fp)) {
123          break;
124      }
125      // Parse the line
126      if (parse_line(buffer) != 0) {
127          continue;
128      }
129      // Print the result
130      printf("%s\n", result);
131  }
132  // Close the file
133  fclose(fp);
134  return 0;
135  }
136
137  // Parse a line of text
138  int parse_line(char *line) {
139      // Split the line into tokens
140      char *tokens[10];
141      int ntokens = 0;
142      char *token;
143      while ((token = strtok(line, " \t")) != NULL) {
144          tokens[ntokens++] = token;
145      }
146      // Check the number of tokens
147      if (ntokens < 2) {
148          return 1;
149      }
150      // Parse the tokens
151      if (strcmp(tokens[0], "GET") == 0) {
152          // GET request
153          if (ntokens != 2) {
154              return 1;
155          }
156          // Get the filename
157          char *filename = tokens[1];
158          // Check if the filename is valid
159          if (!is_valid_filename(filename)) {
160              return 1;
161          }
162          // Get the file size
163          off_t filesize = get_file_size(filename);
164          if (filesize < 0) {
165              return 1;
166          }
167          // Get the file content
168          char *content = get_file_content(filename);
169          if (content == NULL) {
170              return 1;
171          }
172          // Print the content
173          printf("%s", content);
174      } else if (strcmp(tokens[0], "POST") == 0) {
175          // POST request
176          if (ntokens != 2) {
177              return 1;
178          }
179          // Get the filename
180          char *filename = tokens[1];
181          // Check if the filename is valid
182          if (!is_valid_filename(filename)) {
183              return 1;
184          }
185          // Get the file size
186          off_t filesize = get_file_size(filename);
187          if (filesize < 0) {
188              return 1;
189          }
190          // Get the file content
191          char *content = get_file_content(filename);
192          if (content == NULL) {
193              return 1;
194          }
195          // Print the content
196          printf("%s", content);
197      } else if (strcmp(tokens[0], "PUT") == 0) {
198          // PUT request
199          if (ntokens != 2) {
200              return 1;
201          }
202          // Get the filename
203          char *filename = tokens[1];
204          // Check if the filename is valid
205          if (!is_valid_filename(filename)) {
206              return 1;
207          }
208          // Get the file size
209          off_t filesize = get_file_size(filename);
210          if (filesize < 0) {
211              return 1;
212          }
213          // Get the file content
214          char *content = get_file_content(filename);
215          if (content == NULL) {
216              return 1;
217          }
218          // Print the content
219          printf("%s", content);
220      } else if (strcmp(tokens[0], "DELETE") == 0) {
221          // DELETE request
222          if (ntokens != 2) {
223              return 1;
224          }
225          // Get the filename
226          char *filename = tokens[1];
227          // Check if the filename is valid
228          if (!is_valid_filename(filename)) {
229              return 1;
230          }
231          // Get the file size
232          off_t filesize = get_file_size(filename);
233          if (filesize < 0) {
234              return 1;
235          }
236          // Get the file content
237          char *content = get_file_content(filename);
238          if (content == NULL) {
239              return 1;
240          }
241          // Print the content
242          printf("%s", content);
243      } else {
244          // Invalid request
245          return 1;
246      }
247      return 0;
248  }
249
250  // Check if a filename is valid
251  int is_valid_filename(char *filename) {
252      // Check if the filename is empty
253      if (filename == NULL || filename[0] == '\0') {
254          return 0;
255      }
256      // Check if the filename contains invalid characters
257      for (int i = 0; i < strlen(filename); i++) {
258          if (filename[i] < ' ' || filename[i] > '~' || filename[i] == '/' || filename[i] == '\\') {
259              return 0;
260          }
261      }
262      return 1;
263  }
264
265  // Get the file size
266  off_t get_file_size(char *filename) {
267      // Open the file
268      FILE *fp = fopen(filename, "r");
269      if (fp == NULL) {
270          return -1;
271      }
272      // Get the file size
273      off_t filesize = ftell(fp);
274      if (filesize < 0) {
275          return -1;
276      }
277      return filesize;
278  }
279
280  // Get the file content
281  char *get_file_content(char *filename) {
282      // Open the file
283      FILE *fp = fopen(filename, "r");
284      if (fp == NULL) {
285          return NULL;
286      }
287      // Get the file content
288      char *content = malloc(1024);
289      if (content == NULL) {
290          return NULL;
291      }
292      // Read the file content
293      if (!fread(content, 1, 1024, fp)) {
294          return NULL;
295      }
296      return content;
297  }
298
299  // Print the content
300  void print_content(char *content) {
301      printf("%s", content);
302  }
303
304  // Main function
305  int main() {
306      // Parse the command line arguments
307      if (argc < 2) {
308          printf("Usage: %s [filename]\n", argv[0]);
309          return 1;
310      }
311      // Open the file
312      FILE *fp = fopen(argv[1], "r");
313      if (fp == NULL) {
314          printf("Error: cannot open file\n");
315          return 1;
316      }
317      // Read the file
318      char *buffer = malloc(1024);
319      if (buffer == NULL) {
320          printf("Error: cannot allocate memory\n");
321          return 1;
322      }
323      while (1) {
324          // Read a line of text
325          if (!fgets(buffer, 1024, fp)) {
326              break;
327          }
328          // Parse the line
329          if (parse_line(buffer) != 0) {
330              continue;
331          }
332          // Print the result
333          printf("%s\n", result);
334      }
335      // Close the file
336      fclose(fp);
337      return 0;
338  }
339
340  // Parse a line of text
341  int parse_line(char *line) {
342      // Split the line into tokens
343      char *tokens[10];
344      int ntokens = 0;
345      char *token;
346      while ((token = strtok(line, " \t")) != NULL) {
347          tokens[ntokens++] = token;
348      }
349      // Check the number of tokens
350      if (ntokens < 2) {
351          return 1;
352      }
353      // Parse the tokens
354      if (strcmp(tokens[0], "GET") == 0) {
355          // GET request
356          if (ntokens != 2) {
357              return 1;
358          }
359          // Get the filename
360          char *filename = tokens[1];
361          // Check if the filename is valid
362          if (!is_valid_filename(filename)) {
363              return 1;
364          }
365          // Get the file size
366          off_t filesize = get_file_size(filename);
367          if (filesize < 0) {
368              return 1;
369          }
370          // Get the file content
371          char *content = get_file_content(filename);
372          if (content == NULL) {
373              return 1;
374          }
375          // Print the content
376          printf("%s", content);
377      } else if (strcmp(tokens[0], "POST") == 0) {
378          // POST request
379          if (ntokens != 2) {
380              return 1;
381          }
382          // Get the filename
383          char *filename = tokens[1];
384          // Check if the filename is valid
385          if (!is_valid_filename(filename)) {
386              return 1;
387          }
388          // Get the file size
389          off_t filesize = get_file_size(filename);
390          if (filesize < 0) {
391              return 1;
392          }
393          // Get the file content
394          char *content = get_file_content(filename);
395          if (content == NULL) {
396              return 1;
397          }
398          // Print the content
399          printf("%s", content);
400      } else if (strcmp(tokens[0], "PUT") == 0) {
401          // PUT request
402          if (ntokens != 2) {
403              return 1;
404          }
405          // Get the filename
406          char *filename = tokens[1];
407          // Check if the filename is valid
408          if (!is_valid_filename(filename)) {
409              return 1;
410          }
411          // Get the file size
412          off_t filesize = get_file_size(filename);
413          if (filesize < 0) {
414              return 1;
415          }
416          // Get the file content
417          char *content = get_file_content(filename);
418          if (content == NULL) {
419              return 1;
420          }
421          // Print the content
422          printf("%s", content);
423      } else if (strcmp(tokens[0], "DELETE") == 0) {
424          // DELETE request
425          if (ntokens != 2) {
426              return 1;
427          }
428          // Get the filename
429          char *filename = tokens[1];
430          // Check if the filename is valid
431          if (!is_valid_filename(filename)) {
432              return 1;
433          }
434          // Get the file size
435          off_t filesize = get_file_size(filename);
436          if (filesize < 0) {
437              return 1;
438          }
439          // Get the file content
440          char *content = get_file_content(filename);
441          if (content == NULL) {
442              return 1;
443          }
444          // Print the content
445          printf("%s", content);
446      } else {
447          // Invalid request
448          return 1;
449      }
450      return 0;
451  }
452
453  // Check if a filename is valid
454  int is_valid_filename(char *filename) {
455      // Check if the filename is empty
456      if (filename == NULL || filename[0] == '\0') {
457          return 0;
458      }
459      // Check if the filename contains invalid characters
460      for (int i = 0; i < strlen(filename); i++) {
461          if (filename[i] < ' ' || filename[i] > '~' || filename[i] == '/' || filename[i] == '\\') {
462              return 0;
463          }
464      }
465      return 1;
466  }
467
468  // Get the file size
469  off_t get_file_size(char *filename) {
470      // Open the file
471      FILE *fp = fopen(filename, "r");
472      if (fp == NULL) {
473          return -1;
474      }
475      // Get the file size
476      off_t filesize = ftell(fp);
477      if (filesize < 0) {
478          return -1;
479      }
480      return filesize;
481  }
482
483  // Get the file content
484  char *get_file_content(char *filename) {
485      // Open the file
486      FILE *fp = fopen(filename, "r");
487      if (fp == NULL) {
488
```

# "ImageTragick" (CVE-2016-3714)

[illegible]



# "ImageTragick" (CVE-2016-3714)

## Observations:

- (Relatively) long path
  - 7 direct function calls between input and (attempted) validation, but input is also passed elsewhere
- *Raw input* is passed between (and used in) 5 different functions before being read into a native data structure
- Input use and validation is intermixed
- Unsuitable validation mechanism

# "Heartbleed" (CVE-2014-0160)

```
tls1_process_heartbeat(SSL *s)
{
    unsigned char *p = &s->s3->rrec.data[0], *pl;
    unsigned short hbtype;
    unsigned int payload;
    unsigned int padding = 16; /* Use minimum padding */

    /* Read type and payload length first */
    hbtype = *p++;
    n2s(p, payload);
    pl = p;

    if (s->msg_callback)
        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
            &s->s3->rrec.data[0], s->s3->rrec.length,
            s, s->msg_callback_arg);

    if (hbtype == TLS1_HB_REQUEST)
    {
        unsigned char *buffer, *bp;
        int r;

        /* Allocate memory for the response, size is 1 bytes
         * message type, plus 2 bytes payload length, plus
         * payload, plus padding
         */
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        bp = buffer;

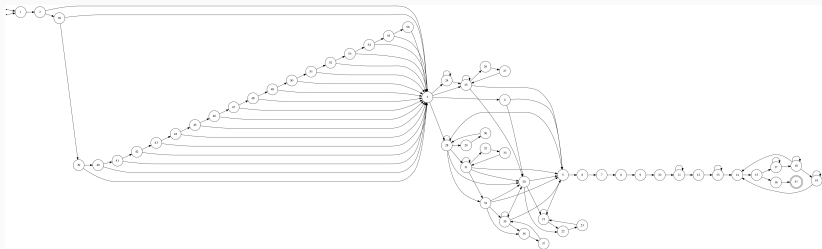
        /* Enter response type, length and copy payload */
        *bp++ = TLS1_HB_RESPONSE;
        s2n(payload, bp);
        memcpy(bp, pl, payload);
        bp += payload;
        /* Random padding */
        RAND_pseudo_bytes(bp, padding);
    }
}
```

# "Heartbleed" (CVE-2014-0160)

## Observations:

- Input passed via several function calls before processing, but not used along the way
- Low degree of input use / validation intermixing, however...
- Almost *total* lack of validation of heartbeat payload!

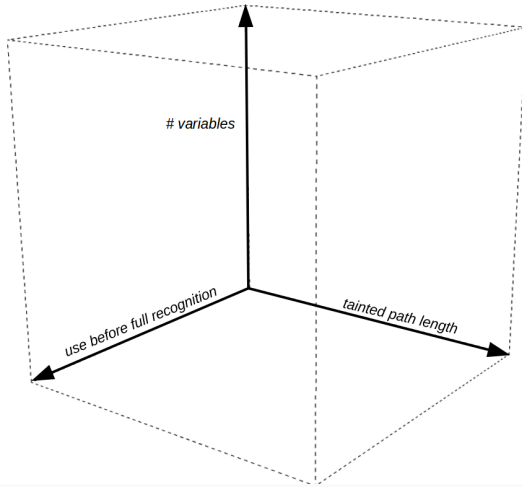
# Mongrel Web Server - HTTP 1.1 Parser



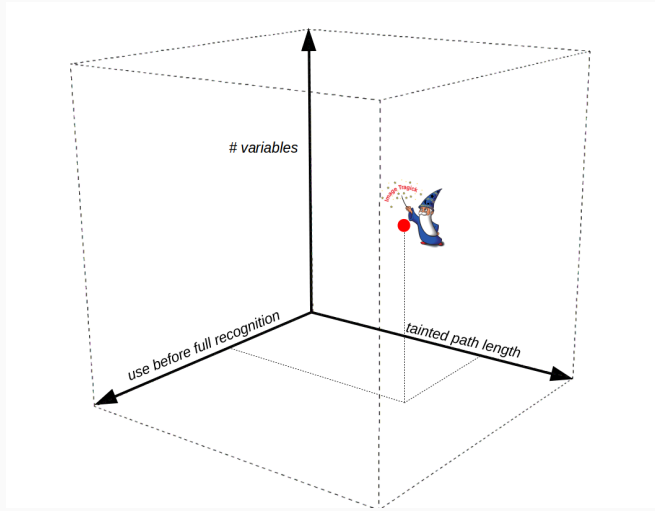
## Parsing Done Right!

- Define a finite state machine for HTTP parsing (uses the Ragel compiler)
- Finite state machine  $\equiv$  regular grammar
- Input language is correctly, formally defined
- Input data is correctly, formally recognized

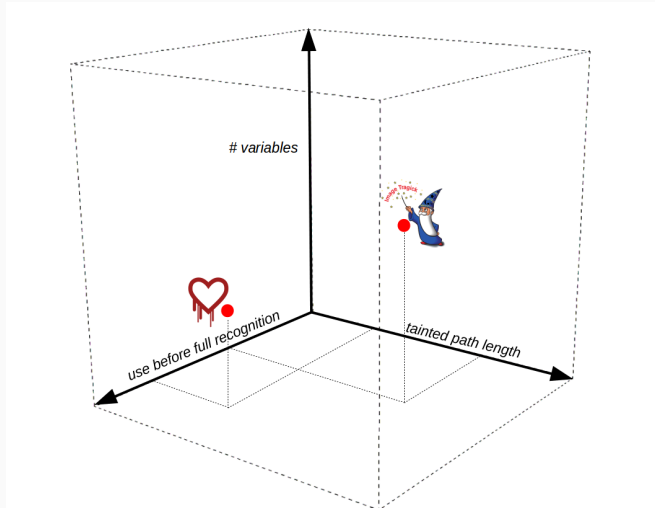
# In The Context Of Our Definition...



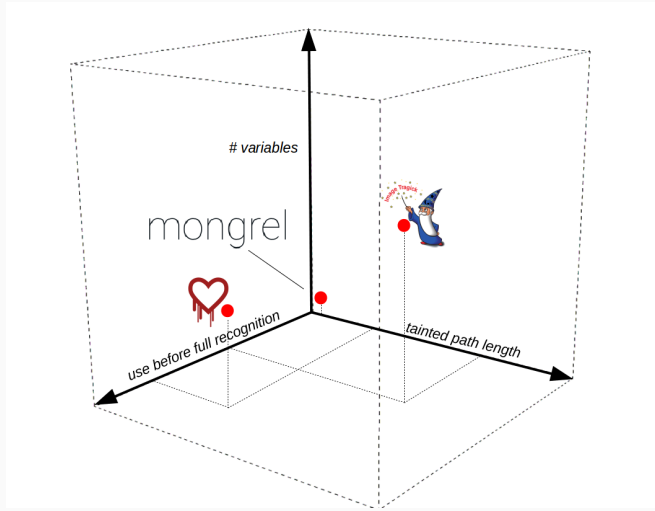
# In The Context Of Our Definition...



# In The Context Of Our Definition...



# In The Context Of Our Definition...





# FUTURE WORK

Where Do We Go From Here...

# Many Roads Lead From Here

- “Climb the hill of Android”
- Develop automated analysis frameworks based on our definition for other software ecosystems
- Develop well-defined input/output patterns for common types (characterize “recognition”)
- Rigorously characterize existing vulnerabilities
- ...

# Acknowledgements

We gratefully acknowledge Steven Arzt from the Secure Software Engineering Group at TU Darmstadt *for his ongoing assistance with technical questions about FlowDroid via the Soot mailing list*

## Other Thoughts...

- Not all vulnerabilities are shotgun parsers...and not all shotgun parsers are necessarily vulnerable
- However:
  - If input data is scattered throughout the code - not just an issue of attack surface, but being error-prone
  - Path length also speaks to how long it takes you to *do* the parsing - why aren't you validating as soon as data enters your software?

# Practical Issues

- Platform specific complications
  - FlowDroid dummy main method - necessary due to Android Lifecycle
- Abstraction level
  - Jimple is an intermediate representation
- Static analysis of real applications is memory intensive!
  - And we had time constraints...