Bug Hunting with Static Code Analysis

Nick Jones

6th June 2016
Bug Hunting with Static Code Analysis

++

The Problem

+ Software developers make mistakes
+ Mistakes = bugs = vulnerabilities
+ Our goal is fewer bugs
Who am I?

Nick Jones

+ Security Consultant at MWR InfoSecurity
+ Web application security, infrastructure assessments
+ Previous experience doing commercial software development
+ Developed bespoke analysis tools for clients
What will we be covering?

- The problem of applications security
- Regular Expressions
- Parsers
- Control Flow Graphs
- Case study: bug hunter
- Case study: software developer
What will we be covering?

- The problem of applications security
- Regular Expressions
- Parsers
- Control Flow Graphs
- Case study: bug hunter
- Case study: software developer
Bug Hunting with Static Code Analysis

++

A Case Study

+ MWREvents has developed a new online events planning platform – website and mobile apps
+ Their developers are of average quality
+ No in–house security experts
+ Want to find and fix all their security issues
How Do We Find Bugs?

Static Analysis

+ Analysing an application without executing it
+ Code review, binary analysis, reverse engineering

Dynamic Analysis

+ Analysing by monitoring and interacting with the application as it executes
+ Fuzzing, tampering, functional testing
How Do We Find Bugs?

Static Analysis

+ Analysing an application without executing it
+ Code review, binary analysis, reverse engineering

Dynamic Analysis

+ Analysing by monitoring and interacting with the application as it executes
+ Fuzzing, tampering, functional testing
Bug Hunting with Static Code Analysis

++

How Do We Find Bugs?

Static Analysis

+ Analysing an application without executing it
+ Code review, binary analysis, reverse engineering

Dynamic Analysis

+ Analysing by monitoring and interacting with the application as it executes
+ Fuzzing, tampering, functional testing
How Do We Code Review?

Manual
+ Give code to smart security experts
+ They read, understand and spot bugs

Automated
+ Pass code to tool
+ Tool parses code, hunts for known issues
Bug Hunting with Static Code Analysis

++

Code Review – Examples

```c
void echo ()
{
    char buf[8];
    gets(buf);
    printf("%s\n", buf);
}
```
Code Review – Examples

webView.getSettings().setJavaScriptEnabled(true);
Bug Hunting with Static Code Analysis

**Manual Code Review – The Downsides**

- Manual code review is expensive

~45 Million LOC  ~86 Million LOC  ~24 Million LOC
How Many Bugs Is That?

- Steve McConnell (Code Complete) says 10–20 defects per 1000 lines of code

~675,000 bugs

~1,290,000 bugs

~360,000 bugs
Bug Hunting with Static Code Analysis

++

Static Code Analysis

Automated searching of source code for issues

+ Higher up front costs
+ ‘Free’ security once built and configured
+ Catch low hanging fruit automatically
To best use tools, you need to understand them.

- Language types
- Automata
- Parsers
Languages

- “[A] set of strings of symbols that may be constrained by rules that are specific to it”
- Defined by a grammar
Bug Hunting with Static Code Analysis

Chomsky’s Language Hierarchy

- recursively enumerable
- context-sensitive
- context-free
- regular
What will we be covering?

- The problem of applications security
- Regular Expressions
- Parsers
- Control Flow Graphs
- Case study: bug hunter
- Case study: software developer
Bug Hunting with Static Code Analysis

++

Chomsky’s Language Hierarchy
Regular Expressions

Regular expressions can parse any regular language

- Act as a finite automata
- List of states, list of transitions between them
- Process input until accept or error state is reached

In practice, modern regexes are far more powerful than the definition given here, but the key limitations remain
Bug Hunting with Static Code Analysis

++

Regular Expressions
++

Bug Hunting with Static Code Analysis

++

Bug Hunting with Regular Expressions

Match code snippets that look like known problems

+ Quick and easy to write, so low cost
+ “Does my code match this very specific known issue?”
+ Bad imports
+ Calls to known dangerous functions
+ Known security misconfigurations
Bug Hunting with Static Code Analysis

++

Code Review – Examples

Code:

webView.getSettings().setJavaScriptEnabled(true);

Regex:

`setJavaScriptEnabled\(true\)`
Bug Hunting with Static Code Analysis

++

Code Review – Examples

Code:
webView.getSettings().setJavaScriptEnabled(true);

Regex:
'setJavaScriptEnabled\((true)\)'
Bug Hunting with Static Code Analysis

++

Regular Expressions – Example

Code:

```c
if (DEBUG) {
    printf('Debug statement 1: %s', var1);
    printf('Other stuff: %s', var1);
    printf('Finally: %s', var1);
}
```

Regex:

`printf\((.*)\)`
Bug Hunting with Static Code Analysis

++

Regular Expressions – Example

Code:

```c
if (DEBUG) {
    printf('Debug statement 1: %s', var1);
    printf('Other stuff: %s', var1);
    printf('Finally: %s', var1);
}
```

Regex:

`'printf\(.*\)'`
Bug Hunting with Static Code Analysis

++

Regular Expressions – Example

Code:

```c
if (DEBUG) {
    printf('Debug statement 1: %s', var1);
    printf('Other stuff: %s', var1);
    printf('Finally: %s', var1);
}
```

Regex:

`
printf\((.*)\)`
Bug Hunting with Static Code Analysis

++

Regular Expressions – Example

Code:

```c
if (DEBUG) {
    printf('Debug statement 1: %s', var1);
    printf('Other stuff: %s', var1);
    printf('Finally: %s', var1);
}
```

Regex:

`printf\((.*)\)`
Regular Expressions – The Disadvantages

- Regular expressions can’t ‘count’
- No way to maintain state
- Cannot back trace
++

Regular Expressions – The Disadvantages

Two options to check for debug guard:

+ Check backwards line by line until you reach beginning of file – inefficient

+ Check X many previous lines – lots of false positives

Three alerts generated for the same missing guard
++

Regular vs Context-Free Languages

+ Regular expressions only match regular languages*
+ Programming languages usually context-free

*mostly
Chomsky’s Language Hierarchy

- Regular
- Context-free
- Context-sensitive
- Recursively enumerable
What will we be covering?

+ The problem of applications security
+ Regular Expressions
+ Parsers
+ Control Flow Graphs
+ Case study: bug hunter
+ Case study: software developer
Bug Hunting with Static Code Analysis

++

Context-Free Languages

* Superset of regular languages
* Anything that can be accepted by a pushdown automata
Bug Hunting with Static Code Analysis

++

Pushdown Automata

+ Finite State Machines with stacks
+ Decide transition based on both input and top of stack
+ Can push/pop to stack as needed
Bug Hunting with Static Code Analysis

++

Pushdown Automata

finite control

\[ \delta \]

state

\[ p \]

input tape

\[ \cdots \quad a \quad \cdots \]

stack

\[ \cdots \]

\[ A \]
Bug Hunting with Static Code Analysis

++

**Parsers**

- Converts text into a hierarchical data structure
- Several different types, depending on what you’re parsing
- TL;DR: Construct a Parse Tree or Abstract Syntax Tree (AST) from the source code
Bug Hunting with Static Code Analysis

++

**Parsers**

Two separate stages

+ Lexer splits input text into tokens (strings with an understood meaning)
+ Parser constructs AST or similar from list of tokens

Can combine both – scannerless parsing
Bug Hunting with Static Code Analysis

++

Lexer Example

Code:
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
    printf(...);
}

Lexed Code:
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
}

Lexer Example

Code:
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
    printf(...);
}

Lexed Code:
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
}
++

Lexer Example

**Code:**

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
    printf(...);
}
```

**Lexed Code:**

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
}
```
Bug Hunting with Static Code Analysis

++

Parser Example

Code:

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
    printf(...);
}
```
Bug Hunting with Static Code Analysis

Parser Example

Code:

```c
if (DEBUG) {
    printf(...);
    printf(...);
    printf(...);
    printf(...);
}
```
Bug Hunting with Static Code Analysis

++

Parser Example

Code:

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
    printf(...);
}
```
Bug Hunting with Static Code Analysis

**Parser Example**

Code:
```c
if (DEBUG) {
    printf(...);
    printf(...);
    printf(...);
}
```
++

Parser Example

Code:

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
}
```

Diagram:

- If()
- Code Block
- Printf()
- Printf()
Bug Hunting with Static Code Analysis

++

Parser Example

Code:

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
}
```

Diagram:

- If()
  - Code Block
    - Printf()
    - Printf()
    - Printf()
Bug Hunting with Static Code Analysis

++

Parser Example

Code:

```c
if (DEBUG)
{
    printf(...);
    printf(...);
    printf(...);
}
```
Bug Hunting with Static Code Analysis

++

We’ve got an AST, now what?

Basic:

+ Search AST for dodgy function calls, check for guards
+ Check for questionable imports
+ Same as before, fewer false positives

Advanced:

+ Control Flow Graphs (CFGs)
+ Taint Analysis
What will we be covering?

+ The problem of applications security
+ Regular Expressions
+ Parsers
+ Control Flow Graphs
+ Case study: bug hunter
+ Case study: software developer
**Control Flow Graphs**

“a representation, using graph notation, of all paths that might be traversed through a program”

- Each basic block represented as a graph node
- Jump targets start block, jumps end block
- Jumps represented as directed edges
Bug Hunting with Static Code Analysis

++

Control Flow Graphs
Why Should I Care About Control Flow Graphs?

- Allows tracing of execution dependant on given inputs without running the application
- Trace data sinks back to original source
- Data sanitized several function calls ago? Trace the graph back and find it
Why Should I Care About Control Flow Graphs?

```php
$result = login($_POST['user'], $_POST['password']);

function login(user, password) {
    return login_query(user, password);
}

function login_query(user, password) {
    return mysqli_query('select * from user where user=' + $user + ' and password=' + $password + ';');
}
```
Bug Hunting with Static Code Analysis

++

Why Should I Care About Control Flow Graphs?

```
$result = login($_POST['user'], $_POST['password']);

function login(user, password) {
    return login_query(user, password);
}

function login_query(user, password) {
    return mysqli_query('select * from user where user=' + $user + ' and password=' + $password + ';');
}
```
Bug Hunting with Static Code Analysis

Why Should I Care About Control Flow Graphs?

```php
$result = login($_POST['user'], $_POST['password']);

function login(user, password) {
    return login_query(user, password);
}

function login_query(user, password) {
    return mysqli_query('select * from user where user=' + $user + ' and password=' + $password + ';');
}
```
Why Should I Care About Control Flow Graphs?

$result = login($_POST[‘user’], $_POST[‘password’]);

function login(user, password) {
    return login_query(user, password);
}

function login_query(user, password) {
    return mysqli_query(‘select * from user where user=’ + $user + ‘ and password=’ + $password + ‘;’);
Why Should I Care About Control Flow Graphs?

```
$result = login($_POST['user'], $_POST['password']);

function login(user, password) {
    return login_query(user, password);
}

function login_query(user, password) {
    return mysqli_query(
        'select * from user where user=' + $user + ' and password=' + $password + ';');
}
```
Why Should I Care About Control Flow Graphs?

```php
$result = login($_POST['user'], $_POST['password']);

function login(user, password) {
    return login_query(user, password);
}

function login_query(user, password) {
    return mysqli_query('select * from user where user=' + $user + ' and password=' + $password + ';');
}
```
Bug Hunting with Static Code Analysis

++

**Parsers**

Downsides:

+ Higher upfront cost to develop
+ More computationally intensive
The Bigger Picture

These tools all fit into a larger picture, all of which needs to work together

- Static code analysis
- Manual code review
- Fuzzing
- Functional testing
What will we be covering?

+ The problem of applications security
+ Regular Expressions
+ Parsers
+ Control Flow Graphs
+ Case study: bug hunter
+ Case study: software developer
Case Studies

Two primary categories of people:

+ Bug hunters – security consultants, people doing bug bounties or looking for 0-days

+ Developers – people building applications who care about security
I’m a bug hunter, why do I care?

- Target identification – pick a project to go after
- Find low hanging fruit
- Identify ropey parts of the codebase
Bug Hunting with Static Code Analysis

++

Target Identification

+ Download source for a bunch of projects
+ Run analyser on all of them, look at the outputs
## Target Identification – Example

<table>
<thead>
<tr>
<th></th>
<th>OpenSSL</th>
<th>LibreSSL</th>
<th>GnuTLS</th>
<th>mbedTLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flawfinder</td>
<td>1794</td>
<td>1389</td>
<td>1228</td>
<td>1381</td>
</tr>
</tbody>
</table>
Bug Hunting with Static Code Analysis

++

**Target Identification – Example**

Bug Hunting with Static Code Analysis

++

Low Hanging Fruit

+ SQL Injection
+ XSS
+ Buffer Overflows
+ Some Use after Frees
Low Hanging Fruit

SQL Injection, XSS, Buffer Overflows

- Look for data sinks – SQL queries, user-provided data rendering etc
- Trace input to data sinks back up CFG to source
- If no sanitisation on user-provided data, probably an attack vector
Bug Hunting with Static Code Analysis

++

Low Hanging Fruit

Use after frees

+ Track allocation/deallocation of pointers through CFG
+ UAF where pointer referenced after deallocation
Example Tools

+ Flawfinder (C/C++)
+ Graudit (ASP/C/.NET/JSP/Perl/PHP/Python)
+ Find Security Bugs (Java, FindBugs Plugin)
+ RATS (C/C++/Perl/PHP/Python)
+ RIPS (PHP)
+ Brakeman (Ruby/Rails)
Bug Hunting with Static Code Analysis

++

Example Libraries/Platforms

For building your own:

+ Clang Analyzer
+ PLY and libraries that build on it (PLYJ for Java)
+ Pyparsing
+ ANTLR
+ Coco/R
What will we be covering?

+ The problem of applications security
+ Regular Expressions
+ Parsers
+ Control Flow Graphs
+ Case study: bug hunter
+ Case study: software developer
Bug Hunting with Static Code Analysis

++

Static Analysis for Developers

+ Catch security issues before the penetration tests
+ One developer builds it, everyone can use it
+ Can be built into existing toolchains and development lifecycles
Bug Hunting with Static Code Analysis

++

Static Analysis and CI

+ CI: Continuous Integration

+ Continuously integrating new features as they’re developed

+ Periodic automated compilation and testing
Bug Hunting with Static Code Analysis

++

CI Tooling Examples

+ Hudson
+ Jenkins
+ Travis CI
+ Bamboo
+ Team Foundation Server
Bug Hunting with Static Code Analysis

++

**CI Workflow**

+ Developer checks in code
+ Server compiles code
+ Test suites are automatically run
Bug Hunting with Static Code Analysis

CI Workflow
Bug Hunting with Static Code Analysis

++

CI Advantages

+ Automated security testing
+ Catch issues as they are introduced to the codebase
+ Catch regressions in code before it hits production
+ Runs automatically, no developer interaction required
Case study – M&S data breach, Oct 2015

+ Developer error led to users being presented with other people’s data on login
+ Personal details and partial card numbers exposed
+ Automated regression testing as part of CI would likely catch this
Bug Hunting with Static Code Analysis

++

Commercial Static Analysis Tools

+ Veracode
+ Coverity
+ Fortify
+ Checkmarx
+ Klocwork
Bug Hunting with Static Code Analysis

Commercial Tools
Bug Hunting with Static Code Analysis

++

Where Security Expertise Can Help

+ Identifying where security risks are likely to lie in their codebase

+ Writing custom rules for existing static analysis engines

+ Developing bespoke analysis tools

+ Advising on integrating automated security testing into development lifecycles
Bug Hunting with Static Code Analysis

Conclusions

- Static analysis can provide low-cost security checks once configured
- ASTs and CFGs let you do all kinds of awesome things
- Automated code analysis complements traditional manual assessments
Thanks for listening!

Questions?