Ed Clarke Symposium

David Brumley
Carnegie Mellon University
Ed’s mentorship and help when I was a student, and later when I was a professor, has been invaluable.

Thank you.

Dawn Song
UC Berkeley
Model Checking for Security Applications

• Athena: an automatic checker for security protocol analysis
  – Work under Ed’s mentorship

• BitBlaze: automatic security analysis of program binaries
  – E.g., Blitz: Compositional Bounded Model Checking for Real-World Programs

• WebBlaze: automatic security analysis and construction for web applications
  – E.g., first step towards building a formal foundation of web security
An epic battle

White Hat

VS.

Black Hat
Exploit bugs
Bug Fixed!

White Hat

Black Hat
Fact:
Windows, Mac, and Linux all have 100,000’s of known bugs
Which bugs are exploitable?
Highly Trained Experts
Automatically Check the World’s Software for Exploitable Bugs
1.2 Advantages of Model Checking

Model Checking has a number of advantages compared to other verification techniques such as automated theorem proving or proof checking. A partial list of some of these advantages is given below:

- No proofs! The user of a Model Checker does not need a correctness proof. In principle, all that is needed is a description of the circuit or program to be checked and press the “return” key. The user may save time working in interactive mode.
- Fast. In practice, Model checking is fast enough to be practical, such as the use of a proof checker, which may require months of the user’s time working in interactive mode.
- **Diagnostic counterexamples.** If the specification is not satisfied, the Model Checker will produce a counterexample execution trace that shows why the specification does not hold (Figure 2). It is impossible to overestimate the importance of the counterexample feature. The counterexamples are invaluable in debugging complex systems. Some people use Model Checking just for this feature.
Automated Exploit Generation [*]

Program → Verification

Correctness Property

Un-exploitability Property

Correct Safe paths

Incorrect Exploit

* Automatic Exploit Generation, NDSS 2011, CACM 2014
A brief history

<table>
<thead>
<tr>
<th>Year</th>
<th>Title</th>
<th>Authors / Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Automatic Discovery of API-Level Exploits</td>
<td>Ganapathy et al., Conference on Software Engineering</td>
</tr>
<tr>
<td>2008</td>
<td>Automatic Patch-Based Exploit Generation</td>
<td>Brumley et al., IEEE Security and Privacy Symposium</td>
</tr>
<tr>
<td>2010</td>
<td>Automatic Generation of Control Flow Hijack Exploits for Commodity Software</td>
<td>Heelan, MS Thesis</td>
</tr>
<tr>
<td>2011</td>
<td>Automatic Exploit Generation</td>
<td>Avgerinos et al., Network and Distributed System Security Symposium</td>
</tr>
<tr>
<td>2011</td>
<td>Q: Exploit Hardening Made Easy</td>
<td>Schwartz et al., USENIX Security Symposium</td>
</tr>
<tr>
<td>2012</td>
<td>Unleashing Mayhem on Binary Code</td>
<td>Cha et al., IEEE Security and Privacy Symposium</td>
</tr>
</tbody>
</table>

And >150 papers on symbolic execution
Basic Execution

- **Code**
- **Stack**
- **Heap**

Processor:
- **EIP**
- Fetch, decode, execute

Process Memory:
- read and write

**Un-exploitability**
Attackers cannot inject into EIP
checking Debian for exploitable bugs

37,000 programs

16 billion verification queries

~$0.28/bug
~$21/exploit

2,606,000 crashes

14,000 unique bugs

152 new exploits

* [ARCB, ICSE 2014, ACM Distinguished Paper], [ACRSWB, CACM 2014]
Q: How long do per-path queries take on average?
A: 3.67ms on average with 0.34 variance

Q: Should I optimize hard or easy formulae?
A: 99.99% take less than 1 second and account for 78% of total time

optimize fast queries
Path Merging[*]

\[ \Pi' = (\Pi \land s \neq 42) \lor (\Pi \land s == 42) \]

* Veritest, ICSE 2014
Execution Profile (Analysis Completes)

Vanilla Symbolic Execution (e.g., KLEE)
- Vanilla: 19%
- Symbolic Execution: 81%

With Path Merging
- Vanilla: 19%
- Symbolic Execution: 81%
- Path Merging: 64%
- 17.8× less time for same results

SMT Solver
Rest
Vision: *Automatically* Check the World’s Software for *Exploitable* Bugs

We’re in the age of automated reasoning. It seems wrong not to try.
Thank You Ed!

- David & Dawn