Safe Software and Systems

David Brumley
Assistant Professor
ECE & CS
Four Topics

1. Capture the Flag Team

2. SplitScreen: Malware Scanning at 2x the speed, ½ the memory

3. Automated Patch-based Exploit Generation

4. Current Research Thrusts
Plaid Parliament of Pwing
CMU’s Capture the Flag Team
HackJam 2009

• Started 1am, Sept 19
• Ran 48 hours straight
• Over 100 teams entered from all over the world

<table>
<thead>
<tr>
<th>Ranking</th>
<th>Team</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLGT</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>ENOFLAG</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>PlaidParliamentOfPwning</td>
<td>9</td>
</tr>
</tbody>
</table>
H.U.S.T. Creative & Fun CTF 2009

• Started 5am Oct 6, 2009
• Ran 48 hours straight
• 8th year of competition
• Over 100 teams from around the world
NYU-Poly CSAW Competition

- 6th annual CTF
- Only officially open to undergrads
  - Grad team scores unofficial
- Winning team gets $500, trip to NYC

<table>
<thead>
<tr>
<th>Place</th>
<th>Team</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>CMU Undergrad</td>
<td>16450</td>
</tr>
<tr>
<td>2nd</td>
<td>CMU Grad</td>
<td>9750</td>
</tr>
<tr>
<td>3rd</td>
<td>RPI</td>
<td>9600</td>
</tr>
</tbody>
</table>

≈71% higher than next undergrad team
Recruit Students with Skills
Sponsorship Opportunity

• Entering DefCON, the most prestigious CTF competition in the world
• 15 members, approx $1,200 per member
  – Travel
  – Hotel
  – Entrance fees

Hackjam: 3rd overall
HUST: 1st overall
CSAW: 1st in qualifying, 1st in finals
New: iCTF: 1st in US, 4th overall
New: PadoCon: 3rd overall
**SplitScreen:**
Malware Scanning at 2x the speed, ½ the memory
Number of Malware Signatures 2005-2009

Year | # Signatures
--- | ---
2005 | 40225
2006 | 72938
2007 | 175771
2008 | 465787
2009 | 867041

~Today
2010

# Signatures

Year

0 200000 400000 600000 800000 1000000 1200000 1400000 1600000 1800000 2000000

2005 2006 2007 2008 2009 2010

40225 72938 175771 465787 867041 1734082

~Today
And Beyond

# Signatures

Year

- 2005
- 2006
- 2007
- 2008
- 2009
- 2010
- 2011
- 2012

0
10000000
20000000
30000000
40000000
50000000
60000000
70000000
80000000

40225
72938
175771
465787
867041
1734082
3468164
6936328

~Today
Typical Malware Scanning (ClamAV)

- Malware Sigs
- Files
- Exact Signature Matching
- Years optimizing algorithms
Scanning Throughput

![Graph showing the decrease in scanning throughput from 2005 to 2012. Throughput MB/s is plotted on the y-axis, while the years 2005 to 2012 are plotted on the x-axis. The throughput values decrease significantly over the years, starting at 11.02 MB/s in 2005 and ending at 2.04 MB/s in 2012.]
Current Anti-Malware Techniques Are Not Scaling
We Need Anti-Malware Scanning That

• Works with the millions of existing signatures
• Scales
• Supports emerging low-power systems
  – Netbooks, Cellphones, IPad

SplitScreen:
2x throughput at ½ the Memory
Input files

Signature
Database
Input files

Signature Database

Pre-process

FFBF

KEY TECHNIQUE: New cache and memory friendly data structure
New Algorithms and Hash Functions for memory & cache-friendly screening
Input files → Pre-process → Signature Database → Retrieve Sigs. → Exact matching

Pre-process → FFBF → Filter Files → Suspect Files

Retrieve Sigs. → List of sigs. needed → Suspect sigs.
SplitScreen Throughput

Throughput (MB/s)

- 35K (Jun, 2005)
- 72K (Oct, 2006)
- 135K (Jul, 2007)
- 306K (Jun, 2008)
- 530K (Mar, 2009)
- 1M
- 2M
- 3M

Comparing ClamAV and SplitScreen throughputs:

- ClamAV: Various throughput values shown for different data sets.
- SplitScreen: Higher throughput values shown for the same data sets compared to ClamAV.
SplitScreen Memory vs. ClamAV

![Graph showing memory use (MB) vs. number of signatures for ClamAV and SplitScreen.](image)

- **ClamAV**
- **SplitScreen**

Memory Use (MB) vs. # of signatures
SplitScreen does Anti-Malware on Weaker Devices
Low-powered AMD Geode (≈ IPhone 3GS)
Low-powered AMD Geode (≈ IPhone 3GS)

Throughput (MB/s)

# of signatures

500K

2 M

ClamAV

SplitScreen

Crash
Distributed SplitScreen

Server

Pre-process

Signature Database

Retrieve Sigs.

List of sigs. needed

Suspect sigs.

Exact matching

Client

Input files

Filter Files

FFBF

Suspect Files
• SplitScreen paper at NSDI 2010
• Implementation available to partners
Automatic Patch-Based Exploit Generation (APEG)
Patches Help Security
Patches Can Help Attackers

- Evil David
Evil David's Timeline

- T1: Gets Patch
- T2: Use Patch to Reverse Engineer Bug

Delayed Patch Attack

Attacks Unpatched Users

Evil David’s Timeline

36
Patch Delay

N. America gets patched version P

Asia gets P

[skantsidis et al 06]
I can reverse engineer the patched bug and create an exploit in minutes.
Example

- All integers unsigned 32-bits
- All arithmetic mod $2^{32}$
- B is binary code

read input

if input % 2 == 0

s := input + 3
s := input + 2

ptr := realloc(ptr, s)
Example

input = $2^{32}-2$

if input % 2 == 0

2$^{32}-2$ % 2 == 0

s := input + 3

s := input + 2

ptr := realloc(ptr, s)

s := 0 ($2^{32}-2 + 2 \% 2^{32}$)

ptr := realloc(ptr, 0)

Using ptr is a problem
Example

```
read input

if input % 2 == 0
  F
  s := input + 3
  T
  s := input + 2

ptr := realloc(ptr, s)
```

Wanted:

\[ s > \text{input} \]

Integer Overflow when:

\[ \neg(\neg s > \text{input}) \]
read input

if input % 2 == 0
  F
  s := input + 3
  T
  s := input + 2

ptr := realloc(ptr, s)

Patch

read input

if input % 2 == 0
  F
  s := input + 3
  T
  s := input + 2

if s > input
  Error
  F
  ptr := realloc(ptr, s)
  T
Exploits for B are inputs that fail new safety condition check in P

(s > input) = false
Automated Patch-Based Exploit Generation

1. Diff B and P to identify location of new safety check
2. Create input that fails safety condition in P using Vine
3. Verify input is exploit on original buggy program B

Off the shelf tools
Exploit Generation

1. Diff B and P to identify location of new safety check

2. Create input that fails safety condition in P

3. Verify input is exploit on original buggy program B
Exploiting Microsoft Patches

<table>
<thead>
<tr>
<th>ASPNet_Filter</th>
<th>Information Disclosure</th>
<th>29 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDI</td>
<td>Hijack Control</td>
<td>135 sec</td>
</tr>
<tr>
<td>PNG</td>
<td>Hijack Control</td>
<td>131 sec</td>
</tr>
<tr>
<td>IE COMCTL32 (B)</td>
<td>Hijack Control</td>
<td>456 sec</td>
</tr>
<tr>
<td>IGMP</td>
<td>Denial of Service</td>
<td>186 sec</td>
</tr>
</tbody>
</table>

- No public exploit for 3 out of 5
- Exploit unique for other 2
Windows ISS .Net Example

The system cannot find the file specified.

Patch checks for ‘\’

You shouldn't see me

I could have been a database file, program, password file, contained top-secret launch codes, etc
Does Automatic Patch-Based Exploit Generation Always Work?

**NO!**

However, in security attackers get lucky, defenders do not

1. Current Delayed Patch Distribution Insecure
2. Automatically generating exploits is possible
Current Research Thrusts

My Research Philosophy:
Develop fundamental advances in science demonstrated in compelling scenarios
Significant Research Thrust: Fully Automated Exploit Generation

Given program, find bugs, generate exploits
Isn’t that for evil?

Yes. That is why it is awesome

Evil David
For Good: Prioritizing Bugs

Ubuntu has over 53,000 bugs to fix. Which one should be fixed first?

Potential Bugs → Exploit Generation → Bugs to Fix First

- DoS
- Control Hijack

All Exploits

Good David
Exploit Generation: Scientific Contributions

• Tons of applications
  – Cyber-warfare, Bug Prioritization
  – Checking Patches
  – Automatically generating filters for intrusion detection systems

• Understand capabilities of attackers
  – We are often good at provably good defenses once we know what to defend against

Compelling scenario for researching age-old questions about program behavior, correctness, protecting algorithms, and so on.
int test(char* str) {
    char buf3[10];
    int i;
    *buf3 = 1;
    strcpy(buf3, str);
    return 0;
}

int main()
{
    char buf[500];
    return test(buf);
}
Automated Exploit Generation Research Project

Source-Based Tools

Binary Analysis Tools

Automated Exploit Gen Theory

Fundamental Research in Formal Methods

Novel Binary Analysis Tools & Techniques

Buffer Overflows

CSRF

Efficient VC Generation

Faster Theorem Provers

56
BAP Project:  
Next-Generation Binary Analysis Platform  
Reason about binary code as well as source
Specific Projects:
• Bug Prioritization
• APEG
• Signature Gen
• Anti-Malware
• Sound Reverse Engineering
• ….

Program Verification
efficient forward
symbolic execution,
improvements to
dec. procedures

Compilers & PL
Type inference for
assembly,
Reasoning about
loops, Chop

Sound Semantics
arbitrary ISA’s (malware)

Answers

http://bap.ece.cmu.edu
Summary

1. **SplitScreen**
   - Malware Scanning at 2x the speed, ½ the memory
   - Scanning on emerging platforms like cellphones, iPads, etc.

2. **Automated Patch-based Exploit Generation**
   - Patches help attackers

3. **Research Thrusts**
   - Significant Thrust: Automate Exploit Generation
   - Next-Generation Binary Analysis
Questions?

Thank you for your attention.

dbrumley@cmu.edu
http://security.ece.cmu.edu/
http://bap.ece.cmu.edu