DESENSITIZATION: 
Privacy-Aware and Attack-Preserving Crash Report

Ren Ding, Hong Hu, Wen Xu, Taesoo Kim

Georgia Institute of Technology
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:
Stop code: CRITICAL_PROCESS_DIED
Software Crashes
After a Crash ...

Crashdump

Internet

Developer

client

server
Crash Dump

• A memory dump contains
  • CPU registers
  • Memory snapshot
  • Execution environment

• Formats
  • Coredump, Minidump
What could be wrong?

Privacy leakage
Recent Study

2.5M crash reports contain private user data!

- 20K sessions tokens
- 700 passwords
- 9K emails
We need a Privacy-Aware Crash Reports
Potential Solutions

- Manual annotation
  - Time-consuming
  - Program-dependent

- Pattern-based searching
  - Error-prone

- Input-logging
  - Heavy computation
  - Incompatible
Our Solution: Desensitization

A new crash-reporting framework

- privacy-aware
- attack-/bug-preserving
- practical, extensible

Key Observation
Adopts generalized features from existing analysis techniques!
## Existing Crash Analysis & Triage Techniques

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Callstack</th>
<th>IP</th>
<th>Signature</th>
<th>RevExec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe-CR</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Apport</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrace</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chrome-CR</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREDAL</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>CrashGraphs</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KLEE</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Liblit et al.</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mac-CR</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modani et al.</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POMP</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rebucket</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPT</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETracer</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schroter et al.</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Socorro</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WER</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deanalyze</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **IP-based**
- **Callstack-based**
- **Signature-based**
- **Fault-based**

### DESENSITIZATION
- Yes
- Yes
- Yes
Existing Anomaly Detection Schemes

- Heap metadata
- Program-specific structures
- ROP gadgets
- Shellcode payload

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Heap</th>
<th>Struct.</th>
<th>ROP</th>
<th>Shcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAVER</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DANGNULL</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOTracer</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KOP</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polychronakis et al.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROPecker</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROPMEMU</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROPScan</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBCFI</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHELLOS</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SigGraph</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESENSITIZATION</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Overview

Desensitization

- Pointer
- Heap
- ROP
- Shcode
- FmtStr

client
#define MAX_LEN 64
typedef struct { char *ptr; void (*print)(); } String;
void printString() { ... }
void vuln(char *input) {
    char pwd[MAX_LEN]; load_passwd(pwd);
    char *buf = (char *) malloc(MAX_LEN);
    String *s = (String *) malloc(sizeof(String));
    s->ptr = buf; s->print = &printString;
    strcpy(buf, input);
    s->print();
}
Pointer Identification

- Pointers
  - Scan every 4/8 bytes, **if and only if** points to a valid memory region
- Code ptr vs. data ptr
  - Proper access permission
Heap Structures

- ptmalloc
  - arenas
  - bins
  - chunk metadata
Heap Structures

- ptmalloc
  - arenas
  - bins
  - chunk metadata
- jemalloc
ROP Chains

• Save non-pointer data in-between pointers
  • *if and only if* there are \([>=N]\) code pointers in \([K]\) bytes
## Extensible!

<table>
<thead>
<tr>
<th>Modules</th>
<th>Collected Data</th>
<th>Related Bugs &amp; Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>code pointers</td>
<td>ROP, JOP, COP, GOT.PLT corruption, vtable injection, etc.</td>
</tr>
<tr>
<td></td>
<td>data pointers</td>
<td>DOP, type confusion, UAF, etc.</td>
</tr>
<tr>
<td>Heap</td>
<td>chunk size</td>
<td>heap ovf, overlapping chunks, heap spray, etc.</td>
</tr>
<tr>
<td></td>
<td>chunk status</td>
<td>UAF, double-free, unsafe-unlink, etc.</td>
</tr>
<tr>
<td>ROP</td>
<td>gadgets &amp; args</td>
<td>ROP, JOP, COP, etc.</td>
</tr>
<tr>
<td>Fmtstr</td>
<td>strings &amp; args</td>
<td>format string attack</td>
</tr>
<tr>
<td>Shcode</td>
<td>payloads</td>
<td>shellcode injection</td>
</tr>
</tbody>
</table>
Tools in Action: Experimental Setup

• Hard to find raw crashes
  • Fuzzing comes to rescue!

• 13,390 crashes collected
  • Normal crashes: 7507
  • Attack-relevant crashes: 5883
Tools in Action: Privacy-Awareness

• Coredumps:
  • 80.9% nullified
• Minidumps:
  • 49.0% nullified
• Pointers dominate
Tools in Action: Privacy-Awareness

• Printable strings
• Coredumps:  
  • 95.0% removed
• Minidumps:  
  • 37.2% removed
Case Study: Leakage in Firefox

class nsTAutoStringN : public nsTString<T> {
    public:
        nsTAutoStringN() : string_type(mStorage, ...) 
        static const size_t kStorageSize = N;
    private:
        char_type mStorage[N];
}

nsresult nsAutoCompleteController::EnterMatch( ... ) {
    ...
    nsAutoString value;
    if (selectedIndex >= 0) {
        GetResultValueAt(selectedIndex, true, value);
    } else if (shouldComplete) {
        GetFinalDefaultCompleteValue(value)
    }
    ...
}
Tools in Action: Bug- & Attack-Preservation

• Normal crashes:
  • State-of-the-art classification tools
  • e.g., Socorro, Backtrace

• Attack-relevant crashes:
  • Self-implemented attack detection tools

• Same result before and after desensitization
Tools in Action: File Size Reduction

- Coredumps:
  - 44.2% reduced
- Minidumps:
  - 7.7% reduced
Tools in action: Efficiency

• < 15 seconds
• Size & complexity matters
Summary

• DESENSITIZATION generates privacy-aware and bug-/attack-preserving crash reports
Summary

• DESENSITIZATION generates privacy-aware and bug-/attack-preserving crash reports
  • Program-independent
Summary

• DESENSITIZATION generates privacy-aware and bug-/attack-preserving crash reports
  • Program-independent
  • Resource-saving
Summary

• DESENSITIZATION generates privacy-aware and bug-/attack-preserving crash reports
  • Program-independent
  • Resource-saving
  • Practical, efficient
Summary

• DESENSITIZATION generates privacy-aware and bug-/attack-preserving crash reports
  • Program-independent
  • Resource-saving
  • Practical, efficient

Open-sourced at:
https://github.com/sslab-gatech/desensitization!