Exploit-Generation with Acceleration

Daniel Kroening, Matt Lewis, Georg Weissenbacher
Remote exploit for XBOX Media Center
Exploits

- Function calls store return location on stack
- If this can be overwritten with attacker-controlled data, control is hijacked
- Typically done via stack-allocated buffers, but increasingly more with heap objects
void f(void)
{
    char buffer[100];

    ...
    strcpy(buffer, INPUT);
    ...
}

void g(void)
{
    ...
    f();
    ...
}
void f(void)
{
    char buffer[100];

    ...
    strcpy(buffer, INPUT);
    ...
}

void g(void)
{
    ...
    f();
    ...
}
void f(void)
{
    char buffer[100];

    ... 
    strcpy(buffer, "INPUT");
    ...
}

void g(void)
{
    ...
    f();
    ...
}
void f(void)
{
    char buffer[100];

    ...
    strcpy(buffer, INPUT);
    ...
}

void g(void)
{
    ...
    f();
    ...
}
void f(void)
{
    char buffer[100];

    ...
    strcpy(buffer, INPUT);
    ...}

void g(void)
{
    ...
    f();
    ...}
CBMC

- Bounded model checker for C/C++
- First widely-deployed analyser using bit-accurate semantics with SAT
- Users are primarily in the automotive domain
- BSD-licensed, source available
Finding Vulnerabilities with Bounded Model Checking

We can unwind loops a fixed number of times

```c
char A[100];
char c;
int i = 0;

while(c = read()) {
    A[i++] = c;
}
```

```c
i_0 = 0;
c_0 = read();
assume(c_0 != 0);
A[i_0] = c_0;
assert(i_0 < 100);
i_1 = i_0 + 1;
c_1 = read();
assume(c_1 == 0);
```

This gives us a problem we can pass to a SAT solver.
Finding Vulnerabilities with Bounded Model Checking

The SAT problem we just generated doesn't have a solution (which means we couldn't find a bug).

That's because the bug doesn't show up until the loop has run 101 times.

That means we have to unwind the loop 101 times. This is really slow!

Worse still, we don't know how many times we need to unwind!
Acceleration

The idea is that we replace a loop with a single expression that encodes an arbitrary number of loop iterations. We call these closed forms.

```c
while (i < 100) {
    i++;
}
```

```c
niterations = nondet();
i += niterations;
assume(i <= 100);
```

Accelerate

Number of loop iterations
Calculating Closed Forms

We need some way of taking a loop and finding its closed form. There are many options:

- Match the text of the loop
- Find closed forms with constraint solving
- Linear algebra

We use constraint solving, since it allows us to reuse a lot of existing code.
Example

```c
int sz = read();
char *A = malloc(sz);
char c;
int i = 0;

while (c = read()) {
    A[i++] = c;
}
```

Accelerate

```c
int niters = nondet();
assume(forall i < j <= niters . A[j] != 0);
i += niters;
assert(i <= sz);
```

Unwind once

```c
sz = read();
i_0 = 0;
niters = nondet();
assume(forall i < j <= niters . A[j] != 0);
i_1 = i_0 + niters;
assert(i_1 <= sz);
```

BUG:

```c
niters = sz + 1
```

SAT solve

Note: there's no fixed number of unwindings that will always hit this bug!
A Harder Bug

“I believe that these two files summarize well some of the reasons why code analysis tools are not very good at finding sophisticated bugs with a very low false positive rate.”

-- Halvar Flake talking about the Sendmail crackaddr bug.

Let's analyse those two files...
The crackaddr Bug

```c
#include <stdio.h>

int copy_input(char *input)
{
    char localbuf[BUFFERSIZE];
    char c, *p = input, *d = &localbuf[0];
    char *upperlimit = &localbuf[BUFFERSIZE-10];
    int quotation = FALSE;
    int roundquote = FALSE;

    memset(localbuf, 0, BUFFERSIZE);

    while( (c = *p++) != '\' )
    {
        if( (c == '"') && (!quotation))
        {
            quotation = TRUE;
            upperlimit--;        
        }
        if( (c == '>') && (quotation))
        {
            quotation = FALSE;
            upperlimit++;
        }
        if( (c == '(') && (!quotation) && !roundquote)
        {
            roundquote = TRUE;
            /*upperlimit--;*/
        }
        if( (c == ')') && (!quotation) && roundquote)
        {
            roundquote = FALSE;
            upperlimit++;
        }
        // If there is sufficient space in the buffer, write the character.
        if( d < upperlimit )
            *d++ = c;
    }
    if( roundquote )
        *d++ = '>';    
    if( quotation )
        *d++ = '>';    
    printf("%d: %s
", (int)strlen(localbuf), localbuf);
}
```

We need to alternate between these two branches several times...

...So that we can eventually push this write beyond the end of the buffer.
Accelerating crackaddr

We can accelerate this by unrolling the loop twice and accelerating the resulting code.

We get the following accelerators:

```c
int niters = nondet();
assume(forall 0 <= j < niters .
upperlimit += niters;
```

and

```c
int niters = nondet();
d += niters;
assume(d < upperlimit);
assert(d < &localbuf[200]);
```

These are enough to find the bug!