Quantifying Information Leaks in Software

Jonathan Heusser, Pasquale Malacaria

Queen Mary University of London

11.10.2016
Introduction

- High complexity associated with quantifying precise leakage quantities
- Technique to decide if a program conforms to a quantitative policy
- Applied to a number of officially reported information leak vulnerabilities in Linux Kernel and authentication routines in SRP and IMSP
- 'When is there an unacceptable leakage?’ and 'Does the applied software patch solve it?’
- First demonstration of QIF addressing real world industrial programs
Tools able to quantify leakage of confidential information

Example:

```python
if(password==guess) access=1 else access=0
```

Unavoidable leakage - Attacker observing value of access

If leakage is unavoidable, the real question is not whether or not the programs leak, but 'How much?'

For example, how much information about password can be obtained by the attacker who can read/write guess

If the amount leaked is very small, the program might as well be considered secure
A precise QIF analysis for secret > few bits is computationally infeasible.

Involves computation of entropy of a random variable whose complexity is the same as computing all possible runs of the program.

Even when abstraction techniques and statistical sampling are used, useful analysis of real code through this method is problematic.

Hence to address computational feasibility, shift the focus from *How much does it leak?* to *Does it leak more than k?*

Off-the-shelf symbolic model checkers like CBMC are able to efficiently answer the second question.
CBMC

- Makes it easy to parse and analyse large ANSI C based projects
- It models bit vector semantics of C accurately - detect arithmetic overflows
- Nondeterministic choice functions to model user input - efficient solving due to the symbolic nature
- Though bounded, can check whether enough unwinding of the transition system was done - no deeper counterexamples
Quantification + nature of leak

Counter examples → causes of leak

For example, we can extract a public user input from the counter example that triggers a violation

Prove whether official patch eliminated the information leak

Four main technical contributions
C function where inputs are formal arguments and outputs are either return values or pointer arguments

- $P$ is the function taking inputs $h, \ell$
- Consider $o = (h \mod 4) + 1$
- $h \rightarrow 4$ bits, $1 \rightarrow 1$ bit, observable $o$ takes values from 0..4 and $\ell$ is the low input
- $P$ is modelled as transition system $TS = (S, T, I, F)$
Successor function for $s \in S$:

$$Post(s) = \{ s' \in S | (s, s') \in T \}$$

(1)

- A state $s$ is in $F$ if $Post(s) = \emptyset$
- A path is a finite sequence of states $\pi = s_0 s_1 s_2 ... s_n$ where $s_0 \in I$ and $s_n \in F$
- A state is a tuple $S = S_H \times S_L$
- Input/output pairs of states of a path denoted as $\langle (h, l), o \rangle$ where $o$ is produced by final state drawn from some output alphabet $O$
A distinction on the confidential input through observations $O$ exists when at least two paths through $P$, that leads to different $o$ for different $h$ but constant $\ell$

An equivalence relation $\simeq_{P,\ell}$ on the values of the high variables is defined as follows: $h \simeq_{P,\ell} h'$ iff:

if $\langle (h, \ell), o \rangle, \langle (h', \ell), o' \rangle$ are input/output pairs in $P$, then $o = o'$

That is, two high values are equivalent if they cannot be distinguished by any observable.

For the modulo program example, and equivalence class in $\simeq_{P,\ell}$ would be $\{1, 5, 9, 13\}$
Let $\mathcal{I}(X)$ be the set of all possible equivalence relations on a set $X$.

Define on $\mathcal{I}(X)$ the order:

$$\approx \sqsubseteq \sim \iff \forall s_1, s_2 (s_1 \sim s_2 \Rightarrow s_1 \approx s_2)$$

(2)

- $\approx, \sim \in \mathcal{I}(X)$
- $s_1, s_2 \in X$
- $\sqsubseteq$ defines a complete lattice over $X$ (*Lattice of Information*)
Characterization of Non-Leaking Programs

PROPOSITION 1: \( P \) is non-interfering iff for all \( \ell, \simeq_{P,\ell} \) is the least element in \( I(S_H) \)

PROPOSITION 2: \( \simeq_p \sqsubseteq \simeq_{p'} \) iff for all probability distributions \( H(R_P) \leq H(R_{P'}) \)

PROPOSITION 3:
1. \( P \) is non-interfering iff \( \log_2(|\simeq_P|) = 0 \)
2. The channel capacity of \( P \) is \( \log_2(|\simeq_P|) \)
3. If for all probability distributions \( H(R_P) \leq H(R_{P'}) \) then \( |\simeq_P| \leq |\simeq_{P'}| \)
Encoding Distinction-Based Policies

- A program violates a policy if it makes more distinctions than what is allowed by the policy.
- Use assume-guarantee reasoning to encode such a policy in the driver function.
- Triggers violation producing a counterexample of the policy.

```c
int h1, h2, h3;
int o1, o2, o3;
h1 = input(); h2 = input(); h3 = input();
o1 = func(h1);
o2 = func(h2);
assume(o1 != o2); // (A)
o3 = func(h3);
assert(o3 == o1 || o3 == o2); // (B)
```
Bounded Model Checking

- ANSI-C program into propositional formula
- Tool can check if unwinding bound is sufficient and ensure that no longer counterexample exists
- $C \land \neg P$ where $C$ is constraint and $P$ is accumulation of assumptions
- If $E_1$ and $E_2$ are two assume statements and $Q$ is expression of assert statement, then $P$ is $P \equiv E_1 \land E_2 \implies Q$
Template to syntactically generate a driver for $N$ distinction policy has been given.

If the driver template is successfully verified up to bound $k$, then $func$ does not make more than $N$ distinctions on the output within $k$.

It implies the validity of the following implication:

$$o_1 \neq o_2 \land o_1 \neq o_3 \land \ldots \land o_{n-1} \neq o_n$$

$$\implies o_{n+1} = o_1 \lor \ldots \lor o_{n+1} = o_n$$

Three claims on the result of model checking process.
Modelling Low Input

typedef long long loff_t;
typedef unsigned int size_t;
int underflow(int h, loff_t ppos) {
    int bufsz;
    size_t nbytes;
    bufsz=1024;
    nbytes=20;

    if(ppos + nbytes > bufsz) // (A)
        nbytes = bufsz - ppos; // (B)
    if(ppos + nbytes > bufsz) {
        return h; // (C)
    } else{
        return 0;
    }
}
Environment

- Library functions or data structures that have no implementation, need to be modelled in a way for the property to be verified
- CBMC replaces function calls with no implementations with non-deterministic values
- Example: `strcmp` and `memcmp` returning 0 or non-zero

```c
int memcmp(char *s1, char *s2, unsigned int n){
    int i;
    for(i=0; i<n; i++){
        if(s1[i] != s2[i]) return -1;
    }
    return 0;
}
```
Experimental Results
Linux Kernel

- Parts of kernel memory gets mistakenly copied to user space
- Kernel memory modelled as non-deterministic values
- Syscalls - arguments and return value (*Data structure and single values*)

AppleTalk

```
struct sockaddr_at {
    u_char sat_len, sat_family, sat_port;
    struct at_addr sat_addr;
    union{
        struct netrange r_netrange;
        char r_zero[8];
    }sat_range;
};
#define sat_zero sat_range.r_zero
```
int atalk_getname(struct socket *sock,  
    struct sockaddr *uaddr, int *uaddr_len, int peer) {  
    struct sockaddr_at sat; 

    //Official Patch. Comment out to trigger leak  
    //memset(&sat.sat_zero, 0, sizeof(sat.sat_zero)); 
    . 
    . 
    //sat structure gets filled  
    memcpy(uaddr, &sat,sizeof(sat));  
    return 0;  
}
tcf_fill_node:

struct tcmsg *tcm;
...
nlh=NLMSG_NEW(skb, pid, seq, event, sizeof(*tcm), flags);
tcm=NLMSG_DATA(nlh);
tcm->tcm_family = AF_UNSPEC;
tcm->tcm__pad1 = 0;
tcm->tcm__pad1 = 0; // typo, should be tcm__pad2 instead.
sigaltstack.

Structure with padding:

typedef struct sigaltstack{
  void __user *ss_sp;
   int ss_flags; //4 bytes padding on 64-bit
   size_t ss_size;
} stack_t;
**Copying whole structures:**

```c
int do_sigaltstack (const stack_t __user *uss,
                  stack_t __user *uoss, unsigned long sp){
    stack_t oss;
    ... // oss fields get filled
    if (copy_to_user(uoss, &oss, sizeof(oss)))
        goto out;...
```

**Calculation:**

```c
    pad = ALIGN - (sizeof(oss) % ALIGN);
    if(pad==ALIGN)
        padding=0;
    else
        padding = ((unsigned int) nondet_int())%  
            (1 << (pad*8))
```
if (*ppos + nbytes > ctr->bufsz)
nbytes = ctr->bufsz - *ppos;
if (copy_to_user(buf, ctr->buf + *ppos, nbytes))
return -EFAULT;

- Way out of actual buffer and thus disclose kernel memory
- Requires too much manual intervention
- Modify CBMC to return non-deterministic values for out-of-bound memory accesses
SRP

_TYPEx( int ) t_getpass (char* buf, unsigned maxlen,
    const char* prompt) {

    DWORD mode;

    GetConsoleMode( handle, &mode );
    SetConsoleMode( handle, mode & ~ENABLE_ECHO_INPUT );
    if(fputs(prompt, stdout) == EOF ||
        fgets(buf, maxlen, stdin) == NULL) {
        SetConsoleMode(handle,mode);
        return -1;
    }
}...
int login_plaintext(char* user, char* pass, char* reply
    struct passwd* pwd = getpwnam(user);
    if (!pwd) return 1;
    if (strcmp(pwd->pw_passwd, crypt(pass,
            pwd->pw_passwd))!=0){
        *reply = "wrong password"
            return 1;
    }
    return 0;
<table>
<thead>
<tr>
<th>Description</th>
<th>CVEBulletin</th>
<th>LOC</th>
<th>k</th>
<th>Proof</th>
<th>$\log_2(N)$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>appletalk</td>
<td>2009-3002</td>
<td>237</td>
<td>64</td>
<td>✓</td>
<td>&gt;6bit</td>
<td>1.39h</td>
</tr>
<tr>
<td>tcffillnode</td>
<td>2009-3612</td>
<td>146</td>
<td>64</td>
<td>✓</td>
<td>&gt;6bit</td>
<td>3.34m</td>
</tr>
<tr>
<td>sigaltstack</td>
<td>2009-3612</td>
<td>199</td>
<td>128</td>
<td>✓</td>
<td>&gt;7bit</td>
<td>49.5m</td>
</tr>
<tr>
<td>cpuset</td>
<td>2007-2875</td>
<td>63</td>
<td>64</td>
<td>x</td>
<td>&gt;6bit</td>
<td>1.32m</td>
</tr>
<tr>
<td>SRP</td>
<td>-</td>
<td>93</td>
<td>8</td>
<td>✓</td>
<td>≤1bit</td>
<td>0.128s</td>
</tr>
<tr>
<td>login_unix</td>
<td>-</td>
<td>128</td>
<td>8</td>
<td>-</td>
<td>≤2 bit</td>
<td>8.364s</td>
</tr>
</tbody>
</table>
Conclusion

- Combined model checking with theoretical work on Quantitative Information Flow
- Proof for whether official patches fix the problem
- Leaks are not synonymous with security breach
- Quantitative is better equipped than qualitative