Oblivious Hashing: A stealthy software verification primitive

Authors: Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. Jakubowski

F.A.P. Petitcolas (Ed.): IH 2002, LNCS 2578, pp. 400-414, 2003.

Presented by Dong Zhang

Summary

- Presented a tamper-resistance primitive that can be used to verify the execution behavior of a program.
- Demonstrated a software implementation
- Discussed unique issues around oblivious hashing

How does it work?

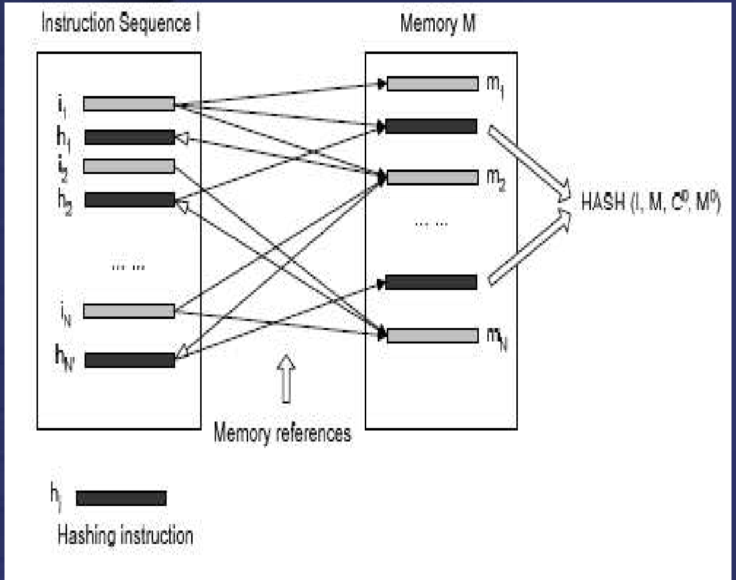


Figure 2: Software-only Oblivious Hashing

- Inject hashing instructions
- Capture memory content
- Produce hash value
- not all are obliviously hashable

"… allows implicit computation of a hash value based on the actual execution."

Appreciative Comments

- Key features of a Software implementation give support to their late arguments.
- Wide application use
- Limitations make the risk known …

Appreciative Comments (limitations)

- Define Unhashable statements
  - Too variable=Unhashable
  - Deterministic functions are hashable statements
- Code coverage for pre-stored hash
  - Reminding us to run through security sensitive execution paths

Critical Comments

Unclear experiment setup

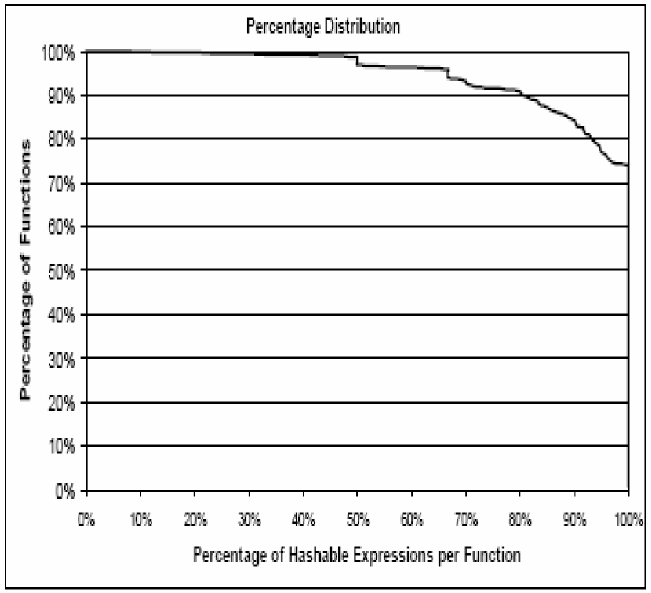
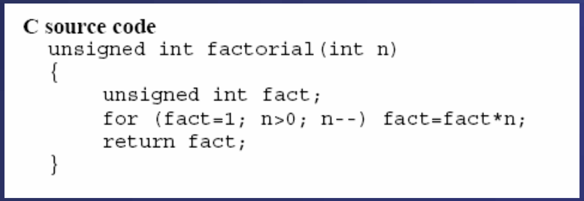


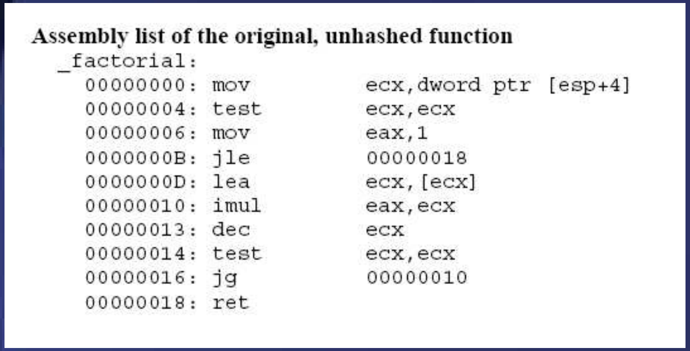
Figure 3: Hashable expressions per function

"We instrumented the program to produce a trace of expression values that we are interested in. We then ran the instrumented program multiple times, in all interesting execution contexts, and post-process the tracing output to determine which expressions were constant across runs."

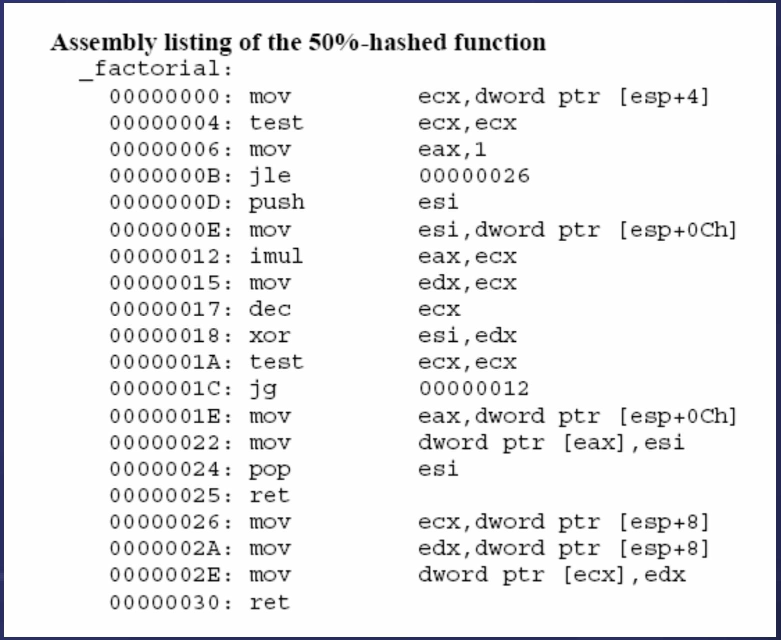
Critical Comments (cont')



C source code
```
unsigned int factorial(int n)
{
      unsigned int fact;
      for (fact=1; n>0; n--) fact=fact*n;
      return fact;
}
```



Assembly list of the original, unhashed function
```
_factorial:
  00000000: mov         ecx,dword ptr [esp+4]
  00000004: test        ecx,ecx
  00000006: mov         eax,1
  0000000B: jle         00000018
  0000000D: lea         ecx,[ecx]
  00000010: imul        eax,ecx
  00000013: dec         ecx
  00000014: test        ecx,ecx
  00000016: jg          00000010
  00000018: ret
```



Assembly listing of the 50%-hashed function
```
_factorial:
  00000000: mov         ecx,dword ptr [esp+4]
  00000004: test        ecx,ecx
  00000006: mov         eax,1
  0000000B: jle         00000026
  0000000D: push        esi
  0000000E: mov         esi,dword ptr [esp+0Ch]
  00000012: imul        eax,ecx
  00000015: mov         edx,ecx
  00000017: dec         ecx
  00000018: xor         esi,edx
  0000001A: test        ecx,ecx
  0000001C: jg          00000012
  0000001E: mov         eax,dword ptr [esp+0Ch]
  00000022: mov         dword ptr [eax],esi
  00000024: pop         esi
  00000025: ret
  00000026: mov         ecx,dword ptr [esp+8]
  0000002A: mov         edx,dword ptr [esp+8]
  0000002E: mov         dword ptr [ecx],edx
  00000030: ret
```

Example from the paper

is this function hashable?

Question

What unhashable code segments can you think of are critical?

Does oblivious hashing affect software update or patching?