Low-Level Control-Flow Manipulation Techniques

Horia V. Corcalciuc
Department of Computational Physics and Information Technologies
Horia Hulubei National Institute for R&D in Physics and Nuclear Engineering (IFIN-HH)

October 15, 2018

IFIN-HH

Motivation

1. **Is the reasoning behind attacks carried out on system-level software applicable to defeating software copy protections?**

2. **Are attacks on software copy protections formalized?**

3. **Do copy protection solutions ultimately work?**

4. **Is privacy abused under the guise of protecting copyright holders?**

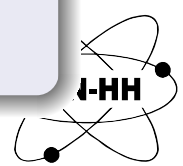
IFIN-HH

Closed Source vs Open Source

Closed source does not provide any added security - closed source *may* make circumventing protections less accessible to a wider audience.

- Open Source software just makes developing an attack more convenient. Many system-level Open Source software such as MTAs, graphics libraries have been attacked successfully ("sendmail", "libpng", etc...).
- Closed source software has either been reverse-engineered in the case of software protections or probed till an attack vector has been found ("IIS", "Outlook", etc...).

All software mentioned for this research had a commercial license and the source code was not provided.



Typical Attack Patterns

Two major categories that seem to be relevant to defeating software protections:

- **Code Injections** - Buffer Overflows, SQL injections
- **Race Conditions** - Timing attacks, Time of Check to Time of Use (TOCTTOU)

Examples include:

- "Rooting" or "Jailbreaking" operating systems - Apple iOS code injection via (**Code Injections**)
- Combined hardware attacks - George Hotz' Playstation 3 "glitching attack" (**Race Conditions**) voltage pulse delivered at the appropriate time, allowing read-write access to memory (**Code Injections**).

IFIN-HH

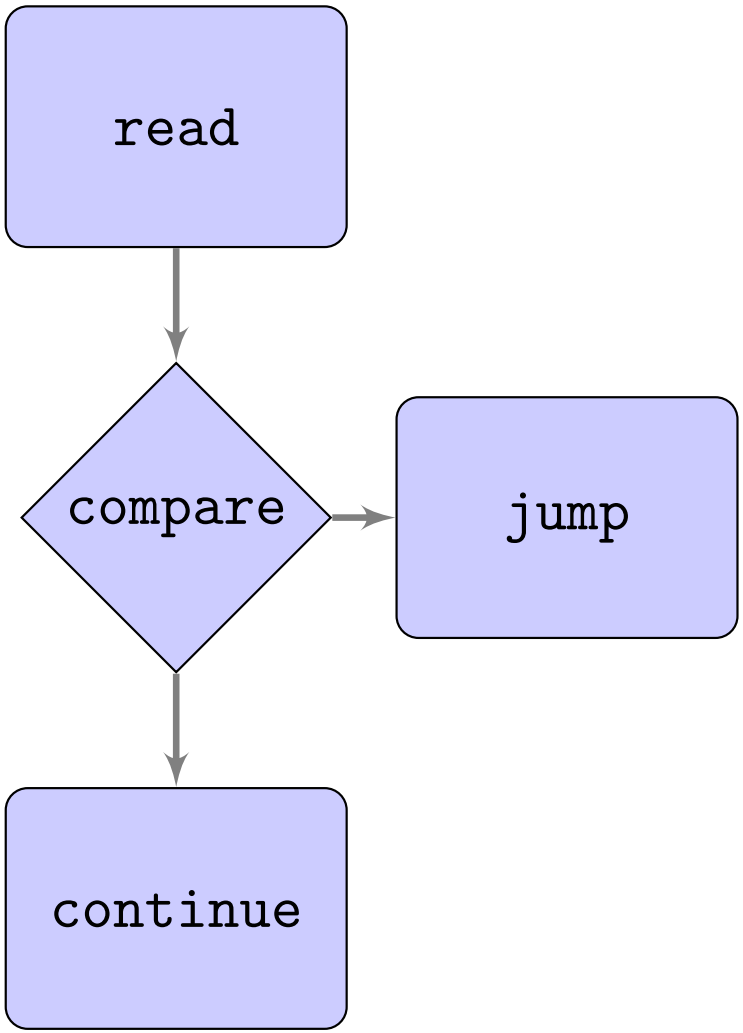
Minimal Attack Patterns

5. Can control-flow manipulation alone yield large returns when defeating copy protections?

Let the minimal set of attack patterns pertaining to control flow be:

1. **Manipulate jump instructions**, ie replace a conditional jump (`jne`, `je`, etc..) by an unconditional jump or invert a conditional, for instance "jump if not equals" (`jne`) replaced by a "jump if equals" (`je`).
2. Eliminate a jump altogether, for instance by replacing a jump instruction by a no-operation `nop`. (**can be derived from 1st pattern**)
3. Remove a function entirely. (**can be derived from 1st pattern**)
4. `nop` sledge from one instruction to a region by padding with `nop` instructions thereby "carrying a predicate". (**can be derived from 1st pattern**)

Manipulate Jump Instructions

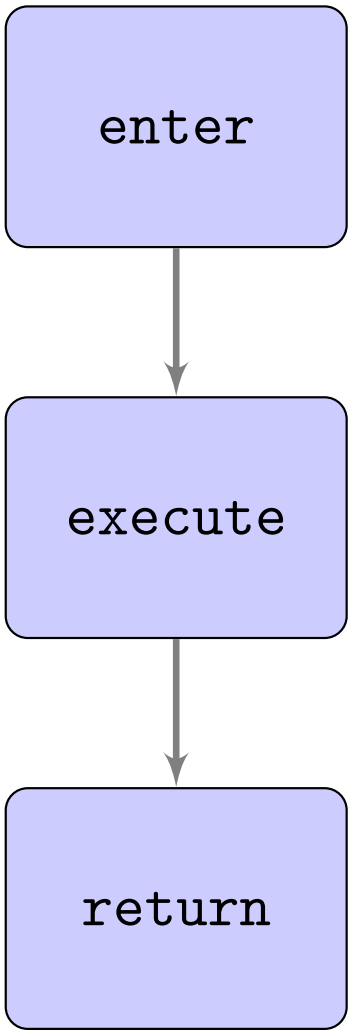



```
int reg = isRegistered(); // read
if(reg != 1) { // compare
  return; // jump
}
start(); // continue
```

```
xor     cl, cl ; read
cmp     rax, rbx ; compare
jle     0x10001FF07 ; jump
mov     cl, 0x1 ; continue
```

```
xor     cl, cl ; read
cmp     rax, rbx ; compare
nop
nop
mov     cl, 0x1 ; continue
```

Function Elimination




```
needsLicenseReminder() { // enter
  ... // execute
  return; // return
}
```

```
_needsLicenseReminder: ; enter
        push    rbp
        mov     rbp, rsp
        push    r15
        push    r14
        push    rbx
        jmp     0x2129          ; jump
...
0x2129  pop     rbx
        pop     r14
        pop     r15
        pop     rbp
        ret
; return
```

nop Sledges

```
int i = isRegistered();
if(i == 1) {
  registered = TRUE; // It's registered!
  printf("You are now registered!");
  return;
}

printf("Will expire in 30 days!");
setTimer();

return;
```

```
mov rbx, 0x1 ; registered = 1
nop          ; sledge to end of method
nop
nop
...
...
...
...
...
nop
ret
```

The effect of building a nop sledge is that a predicate is "carried" to a different region of code. In the aforementioned example, an environment-bound variable registered is set thereby letting other regions of code that **rely** on the variable work under the assumption that the software is registered.

Pattern Equivalence

```
0x0001 nop
0x0002 mov     rdx, r14
```

```
0x0001 jmp     0x0002
0x0002 mov     rdx, r14
```

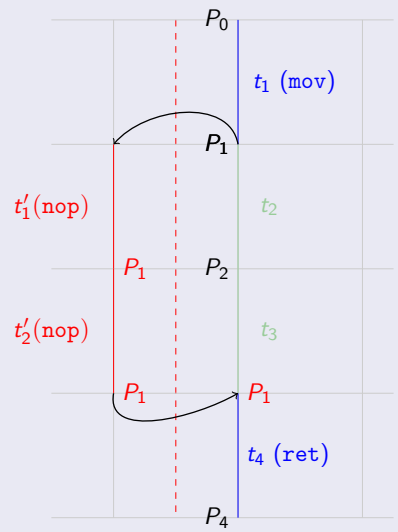
All patterns can be *derived* from manipulating jump instructions:

- A single `nop` instruction can be semantically equivalent to a jump `jmp` one instruction ahead.
- A `nop` sledge can be replaced by a `jmp` instruction to the target region of code.
- A method can be eliminated by either replacing instructions with `nop` (slide) or just jump via `jmp` to the end of the method.

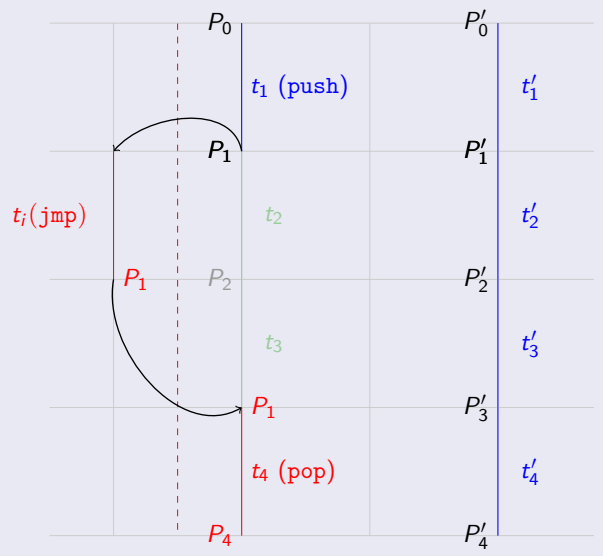
IFIN-HH

Formalizing using Aczel Traces

nop Sledge (Sequential: PocketRSS)



Jump Coercion (Parallel: ShareMouse Timer)



- Given a series of composed and terminating traces traces $t = t_1 \cdot t_2 \cdot \ldots \cdot t_4 \cdot \checkmark$, an execution of the program that respects stability such as $P_1 \triangleright P_2$, $P_2 \triangleright P_3$, ... as part of the **rely** set of conditions can **guarantee** proper termination and is also a legitimate run of the program.
- It is *sufficient* for an attacker that injects traces t_i to *guarantee termination* by making sure that the program state is not invalidated when coercing control flow. In other words, the "carried predicate" (ie: P_1) over predicate P_2 must at least be a subset of the replaced predicate (ie: P_3) such that $P_1 \subseteq P_3$. (**Proof by induction over traces,** $s \models P_1$, $s' \models P_3$)

IFIN-HH

Results

Software with Defeated Copy Protections

"Acorn", "Alfred", "Amnesty", "BBEdit", "CleanGenius", "CornerStone", "DaisyDisk", "Decloner", "DropDMG", "Entropy", "FontAgent Pro", "Grappler", "iGlasses", "IconBox", "iPulse", "iRamDisk", "Latexian", "Leech", "Omnigraph Sketcher", "Omni Plan", "On The Job", "PathFinder", "Perfect Photo Suite", "PhotoSweeper", "ProxyCap", "QPict", "SecuritySpy", "Smasher", "SnapzProX", "Snippets", "SubethaEdit", "TextMate", "Transmit", "VelaClock", "WireTap Studio", "XScope", "Zoom2", etc...

All of the above commercial software packages have been defeated with the following mentions:

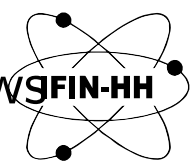
- Only the four described techniques were used to attack the software.
- A lot of the studied software (for instance, "Keyboard Maestro") required a single instruction to be changed.
- Canaries, stack guards and other protections ("Paddle" framework) were sometimes irrelevant since other regions of code could be coerced without tripping over them.

Security vs. Privacy

Studying software packages in order to defeat copyright protections led to the following observations.

- Many software packages are laced with in-line calls to the creators' websites where identifiable information could be stored - the information sent ranges from data such as workstation user-name and up to hardware identifiers such as MAC addresses.
- Cracker groups that defeat copy protections tend to not disable the **dial home device** such that it becomes trivial for a creator to determine whether some person is using an illegitimate copy (ie: a demo application that still dials home even after the trial period has long expired).

It is uncertain on what legal basis the identifying information is collected and whether creators are compliant with new privacy laws (such as the European **GDPR**).



Any Questions?

IFIN-HH