ROP is still dangerous: breaking modern defenses

Simon Iser

November 27, 2016
- ROP in general
- Existing Defenses
- Attacks on these Defenses
- Comparison of these Attacks
In general

- A vulnerability to gain control over the instruction pointer is needed (e.g. buffer overflow)
- Knowledge over the binaries is needed or must be gained to mount the attack
- Gadgets are chained together to issue actions the attacker wants

ROP is still dangerous: breaking modern defenses
With defenses enabled

- A normal ROP must be possible, if the defenses were not there
- At least one binary of sufficient size must not be randomized or there must be a information disclosure vulnerability
Policy

- Call preceded
  - Each indirect jump is directly preceded by a call
- Short gadgets
  - Gadgets are only short instructions as they do not have unwanted side effects
Uses indirect branch tracing

Pausing the program execution, inspecting recent behavior and deciding whether the process should be terminated

Inspection started when system-call is issued, inspection is using Intel CPU feature

Last 16 indirect branches are checked

If there are more than 8 gadget-like instructions in a row, an alarm is raised

If a non gadget-like instruction is reached, the search is terminated
ROP attacks

Defenses

Attacks

kBouncer

ROPecker

ROPecker

- Uses a set of executable pages
  - If a instruction-call causes a page fault, a check is issued
- Like kBouncer, a check is issued when a syscall is invoked
- Alarm is raised when a long chain of gadget-like instructions is detected
- It predicts the future behavior of the program
  - If a violation of the policy is found in future instructions a alarm is raised and the process is terminated

Simon Iser

ROP is still dangerous: breaking modern defenses
A sequence of instructions is called a gadget if six or fewer instructions lead to an indirect jump, with no direct or conditional jump in the path.

A sequence of gadget-like instructions is called an attack if there are eleven or more sequences classified as gadget in a row.

NOTE: eleven is just the recommended number, as a lower number could heavily influence the execution of regular programs.
- Only 6% of the gadgets are call preceded
  - Like last week, 6% are still a large amount of possible gadgets
  - About 70KB of binary code are sufficient to mount a fully call-preceded ROP-attack
  - As an example, even 'apt-get' has more than 40KB of binary code
Usually defenses rely heavily on length based classification of gadgets.

As an attacker you could choose a mix of long and short gadgets.

→ Defensive systems classify these attacks falsely as normal execution.
Some defenses check only a certain amount of instructions back

→ An attacker could perform some "innocent" actions to overwrite that limited storage

→ The defensive system will never raise an alarm
Defeating kBouncer

- kBouncer can look 16 instruction-sets back in the past
- kBouncer uses its conservative threshold of 8 or less gadget-like instructions
- kBouncer is invoked at every syscall
Perform a typical ROP attack on the victims system

Before doing the syscall, the history has to be flushed and made valid

→ Using flushing gadgets and termination gadgets
→ Restore changed registers

→ Issue syscall
Flush Gadget

- Call-preceded
- Performs a `ret`-operation
- Does not change any register ideally

```
a) ...
call xyz ← call-preceded
pop ebp ← return here
ret
...
b) ...
call xyz
jmp A
...
A: mov ebx,35
ret
...
```
Termination Gadget

- Call preceded
- Long enough to not be classified as gadget
- Might change the registers

```
add  [esp+17Ch], ebx
mov  ebx, [esp+17Ch]
sub  ebx, ebp
jmp  A
...
A: add  [esp+64h], ebx
    jmp  B
    ...
B: mov  esi, [esp+1C0h]
    lea  eax, [esi+8-4]
    sub  eax, [esp+64]
    and  eax, 7h
    mov  edi, [esp+64]
    lea  eax, [edi+eax+4]
    shr  eax, 3
    cmp  eax, esi
    jbe  C
    ...
C: mov  eax, [esp+1C0h]
    add  esp, 19Ch
    pop  ebx
    pop  esi
    pop  edi
    pop  ebp
    ret
```
Push all registers before using the flushing and termination gadgets and pop them afterwards.

- Chaining pop-operations with indirect jumps
- Jump Oriented Programming
- Use gadgets, that jump to non call-preceded gadgets
- Call Oriented Programming
Use arbitrary gadgets to push needed values before using the flushing and termination gadgets

After entering a valid state only 8 gadgets can be called without causing a alarm

→ These gadgets are visible to kBouncer, therefore they must be call preceded

→ 8 gadgets are sufficient to POP all needed values from the stack
Without ret Operations

- Use arbitrary gadgets to push needed values before using the flushing and termination gadgets.
- After entering a valid state only 8 gadgets can be called without causing an alarm.
- These gadgets are less common, therefore this method is less common.
Use register indirect jumps to chain gadgets together
After entering a valid state only 8 gadgets can be called without causing an alarm
For each restore operation, a dispatcher-gadget is needed for chaining
→ There are only 4 pairs of gadgets left to restore the registers
→ Might be difficult to execute successfully
Non call-preceded gadgets

- If there is need to use non call-preceded gadgets
- A call-preceded gadget (reflector gadget) is executed and jumps to any desired gadget
- Rarely used, but in some cases easier to process
Call Oriented Programming

- Gadgets do chain together via memory-indirect calls
  - The locations where to jump to can be initialized in advance
- COP gadgets are common due to the dynamically linked libraries and object oriented code uses these type of calls
- An attacker needs more control over the victims system to use COP, therefore it might not be possible every time or a combination of these two types of attacks are more effective or practical.

ROP is still dangerous: breaking modern defenses
The function is most likely not call-preceded
→ Use a reflector gadget
→ Look for a program that directly calls the desired function
→ Return to a call-preceded point in the function after preparing the register state
Assumption: kBouncer can take the whole history in consideration

→ Use only call-preceded gadgets

→ No need for a flushing gadget
Defeating ROPecker

- Invokes a check when a page-fault occurs or a syscall is issued
- Checks future executions
- Enforces a similar policy compared to kBouncer
  - History must be hidden whenever a check is about to be issued
  - Attack consists of three phases
    1.: Loading the page
    2.: Attack using only gadgets from that page
    3.: Flush the history

ROP is still dangerous: breaking modern defenses
Loading Phase

- Start with a termination gadget to prevent a too long chain of short gadgets
- Use any call-preceded gadget on a wanted page to load the page
- Use a termination gadget to prevent ROPecker to look any further

⇒ No attack will be noticed
Use any available gadget to do necessary computing

→ Ignore the defenses

• The attack might be split into parts, each with its own three phases

→ Splitting makes each single attack phase easier, as different gadgets can be loaded between the parts
Insert flushing gadgets like the ones used against kBouncer.

Use a termination gadget at the end to prevent further inspection and to screen the following loading phase.
If a attack needs to be splitted, the values need to be stored and loaded again.

Often, each attack phase is used to compute one needed value.

After all values are computed and loaded back into registers, the syscall is issued.

**ROP still dangerous: breaking modern defenses**
There must be at most 11 gadgets including the syscall
The used gadgets can be taken out of any page as the last termination gadget will be at most 11 gadgets away
Evasion Attack

- If ROPecker would check at arbitrary points
- Used if ROPecker would only allow one executable page
After every ten gadgets a termination gadget is used

There will never be a sequence of gadgets long enough to trigger ROPecker

The attack might take longer as after every 10 gadgets some registers will be overridden by the termination gadget

Effectively, every call preceeded gadget can be used no matter if it is in the executable page
Repeated History Hiding Attack vs. Evasion Attack

Repeated History Hiding Attack

- Better if checks are only invoked at certain points
- Only gadgets of loaded page available
- Unbound amount of gadgets can be used before the history must be flushed

Evasion Attack

- Better if the whole process is monitored
- All call-preceded gadgets provided by the binary available
- Every 11th gadget must be a termination gadget