Reading memory, without reading memory

Christian W. Otterstad

University of Bergen – Department of Informatics

April 27, 2016

Synopsis

Background

- Purpose and motivation
- What is low-level exploitation?
- Importance and justification
- Exploitation techniques and mitigations
- XnR – Execute-no-Read

Attack model

- Attacker's perspective
- Defender's perspective

Exploitation technique

- Technique overview
- Implementation
- Probe primitive
- Special probes
- Limitations

Possible solutions

- Possible solutions
- Information leak
- Format string exploits
- Sliding window

Conclusion

Purpose and motivation

Reading memory ... without reading memory?

- Specifically for defeating XnR (Execute-no-Read).
- Indirect memory reading, iterable information leakage attack, or other attack is likely to be required.

What is low-level exploitation?

The notion of low-level exploitation can be hard to capture completely in a single defintion.

- Always uses existing functionality.
- Often uses a superset of normal functionality.
- Often sees undefined behavior as a feature.

Importance and justification

What is the goal? Why is this noteworthy?

- Ability to bypass authentication or authorization checks.
- Ability to execute arbitrary code.
- Very powerful: Full control over the target machine.
- Can scale (worms).

Exploitation techniques and mitigations techniques

Several classes of exploitation techniques and mitigation techniques have emerged.

- Control oriented exploits
- Direct execution of attacker supplied data
- Simple code reuse
- Advanced code reuse
- Data oriented exploits

XnR – Execute-no-Read

- Prevents selective pages from being both readable and executable at the same time.
- Strong ASLR (Address Space Layout Randomization).
- Akin to early SEGMEXEC, software only.
- Page fault handler.

Attack model – attacker's perspective

The attacker operates under the following assumptions:

- The attacker possesses a local copy of the target binary.
- The attacker knows the OS type and version.
- The attacker knows the mitigation techniques in place.

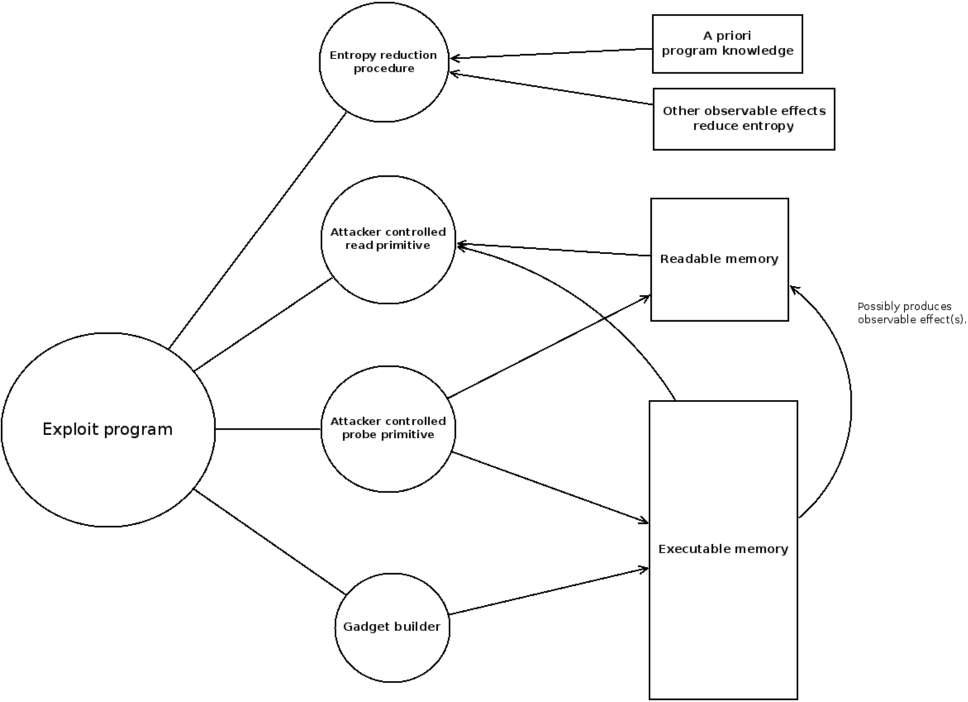
Attack model – defender's perspective

The defender operates with the following assumptions:

- Employs non-standard, strong, fine-grained ASLR (Address Space Layout Randomization).
- Uses standard NX-bit (No-Execute).
- Uses RAP (Return Address Protection) (canaries).
- Uses XnR (Execute-no-Read).
- Is unaware of the vulnerability and the ongoing attack.

Technique overview

- Techniques based on "Hacking Blind" (paper), "Braille" (tool).
- Uses JIT-ROP (Just-In Time ROP) to get enough gadgets to read memory.
- With no read and write permissions, only execution is left.
- Execute memory and observe the behavior.
- Limited ability to observe the behavior in most cases.
- Deal with strong ASLR in some way. Check at page level granularity.


Entropy reduction procedure
A priori program knowledge
Other observable effects reduce entropy
Attacker controlled read primitive
Readable memory
Possibly produces observable effect(s).
Exploit program
Attacker controlled probe primitive
Executable memory
Gadget builder

Implementation

Implementation in C.

- Determine canary, RBP, RIP.
- Arc-injection probes.
- Detect ROP gadgets.
- Detect the BROP gadget (Blind ROP).
- Detect PLT (Procedure Linkage Table).
- Find syscall.
- Miscellaneous.

Note about arc injection.

Implementation

Demo (incomplete).

Probe primitive

First presented in “Hacking Blind”.

- Requires a forking server.
- Probe and observe.
- Stop gadget.

Basic probes

- Stop or halt: [probe]
- pop gadget: [probe] [trap] [stop]
- 2x pop gadget: [probe] [trap] [trap] [stop]

Special probes

- Find syscall: [pop 1] [pause] [pop 2] [pause] $\cdots$ [pop n] [pause] [probe] [trap]
- Find rax: ~~[pop rax] [pause]~~ [pop 2] [pause] $\cdots$ [pop n] [pause] [syscall] [trap]
- Find rdi: [pop rax] [nanosleep] [pop 2] [10^9] $\cdots$ ~~[pop n] [10^9]~~ [syscall] [trap]
- Find rsi: [pop rax] [kill] [pop rdi] [0] $\cdots$ ~~[pop n - 1] [9]~~ [pop n] [9] [syscall] [trap]
- Find rdx: [pop rax] [clock_nanosleep] [pop rdi] [0] [pop rsi] [0] $\cdots$ ~~[pop n - 1]~~ [10^9] [pop n] [10^9] [syscall] [trap]

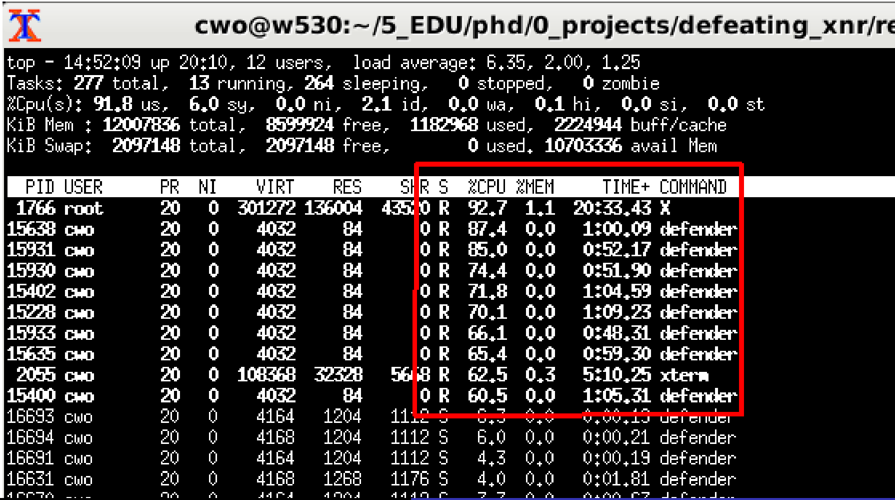
Limitations

At least the following problems are present:

- Noisy attack.
- Performance demanding attack.
- High latency, slow.

However, if the attacker gains a root shell all local evidence will be removed.

Limitations



cwo@w530:~/5_EDU/phd/0_projects/defeating_xnr/re

```
top - 14:52:09 up 20:10, 12 users,  load average: 6.35, 2.00, 1.25
Tasks: 277 total,  13 running, 264 sleeping,   0 stopped,   0 zombie
%Cpu(s): 91.8 us,  6.0 sy,  0.0 ni,  2.1 id,  0.0 wa,  0.1 hi,  0.0 si,  0.0 st
KiB Mem : 12007836 total,  8599924 free,  1182968 used,  2224944 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used. 10703336 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1766 root      20   0  301272 136004  43520 R  92.7  1.1  20:33.43 X
15638 cwo       20   0    4032     84      0 R  87.4  0.0   1:00.09 defender
15931 cwo       20   0    4032     84      0 R  85.0  0.0   0:52.17 defender
15930 cwo       20   0    4032     84      0 R  74.4  0.0   0:51.90 defender
15402 cwo       20   0    4032     84      0 R  71.8  0.0   1:04.59 defender
15228 cwo       20   0    4032     84      0 R  70.1  0.0   1:09.23 defender
15933 cwo       20   0    4032     84      0 R  66.1  0.0   0:48.31 defender
15635 cwo       20   0    4032     84      0 R  65.4  0.0   0:59.30 defender
 2055 cwo       20   0  108368  32328   5668 R  62.5  0.3   5:10.25 xterm
15400 cwo       20   0    4032     84      0 R  60.5  0.0   1:05.31 defender
16693 cwo       20   0    4164   1204   1112 S   6.3  0.0   0:00.13 defender
16694 cwo       20   0    4168   1204   1112 S   6.0  0.0   0:00.21 defender
16691 cwo       20   0    4164   1204   1112 S   4.3  0.0   0:00.19 defender
16631 cwo       20   0    4168   1268   1176 S   4.0  0.0   0:01.81 defender
```

Limitations

Children stuck in infinite loops. Example:

```
(gdb) x/i $pc
=> 0x400db7 <command_parser+53>: add al,ch
(gdb) si
(gdb) x/i $pc
=> 0x400db9 <command_parser+55>: jrcxz 0x400db7 <command_pa
(gdb) si
```

Any solutions?

Possible solutions

- Minimize probes.
- Aligned or unaligned probes?
- Detection of bad probes, avoidance.

Information leak

- Information leak that can be iterated over.
- Would not allow the attacker to read executable pages.
- Would allow the attacker to observe the stack frame.
- Combination of techniques?

Format string exploits

- Would allow an attacker to fetch multiple stack frames.
- Arc injection.
- ROP in some cases.
- Few format string vulnerabilities.

Sliding window

XnR maintains a sliding window as an optimization hack.

- Entails that one page can *always* be read ($n = 1$).
- XnR has reasonable performance at $n = 2$ (2.2% overhead).
- The attacker can easily find the first page by determining RIP.
- Attacker might be able to measure n to speed up the attack?

Conclusion

A method of generalized indirect memory reading has been examined for the specific application of using it to circumvent XnR.

It is clear that some memory reading can be performed. However, it appears that the practical limitations can be quite restrictive.