Oracle Solaris 11.x / 10 whodo / w Buffer Overflow

Authored by Marco Ivaldi

Posted Apr 17, 2020

A difficult to exploit heap-based buffer overflow in setuid root whodo and w binaries distributed with Solaris allows local users to corrupt memory and potentially execute arbitrary code in order to escalate privileges.

tags | exploit, overflow, arbitrary, local, root
systems | solaris
advisories | CVE-2020-2771
SHA-256 | b4fd5ab59754c50d0a4004387d6ef82f58b1de0dc8f81de2438e2e8a8dd7f4fb Download | Favorite | View

Related Files

Change Mirror Download

```
@Mediaservice.net Security Advisory #2020-07 (last updated on 2020-04-15)

         Title:  Heap-based buffer overflow in Solaris whodo and w commands
   Application:  Setuid root whodo and w binaries distributed with Solaris
     Platforms:  Oracle Solaris 11.x (confirmed on 11.4 X86)
    Oracle Solaris 10 (confirmed on 10 1/13 X86)
    Other platforms are potentially affected (see below)
   Description:  A difficult to exploit heap-based buffer overflow in setuid
    root whodo and w binaries distributed with Solaris allows
    local users to corrupt memory and potentially execute arbitrary
    code in order to escalate privileges
        Author:  Marco Ivaldi <marco.ivaldi@mediaservice.net>
 Vendor Status:  <secalert_us@oracle.com> notified on 2019-08-23
      CVE Name:  CVE-2020-2771
   CVSS Vector: CVSS:3.0/AV:L/AC:H/PR:L/UI:R/S:C/C:L/I:N/A:N (Base Score: 2.5)
    References: https://github.com/0xdea/advisories/blob/master/2020-07-solaris-whodo-w.txt
    https://www.oracle.com/security-alerts/cpuapr2020.html
    https://www.oracle.com/technetwork/server-storage/solaris11/
    https://github.com/illumos/illumos-
gate/blob/61aaa916808c601f9ee36d96c05ee9dac211d09e/usr/src/cmd/whodo/whodo.c
    https://github.com/illumos/illumos-gate/blob/61aaa916808c601f9ee36d96c05ee9dac211d09e/usr/src/cmd/w/w.c
    https://www.mediaservice.net/
    https://0xdeadbeef.info/

1. Abstract.

A difficult to exploit heap-based buffer overflow in setuid root whodo and w
binaries distributed with Solaris allows local users to corrupt memory and
potentially execute arbitrary code in order to escalate privileges.

2. Example Attack Session.

In order to reproduce this bug, the following commands can be used:

raptor@stalker:~$ cat /etc/release
                             Oracle Solaris 11.4 X86
  Copyright (c) 1983, 2018, Oracle and/or its affiliates.  All rights reserved.
                            Assembled 16 August 2018
raptor@stalker:~$ uname -a
SunOS stalker 5.11 11.4.0.15.0 i86pc i386 i86pc
raptor@stalker:~$ id
uid=100(raptor) gid=10(staff)
raptor@stalker:~$ cp /usr/bin/sleep
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
raptor@stalker:~$ exec -a '-
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
./AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
256 &
[switch to another shell]
raptor@stalker:~$ whodo -l # or w
 12:43pm  up 5 day(s), 20 hr(s), 36 min(s)  5 user(s)
User     tty           login@  idle   JCPU   PCPU  what
raptor   vt/7         Tue 2pm 6days   1:49   1:49  /usr/lib/tracker-miner-apps
Segmentation Fault

3. Discussion.

A detailed analysis of the buffer overflow in whodo follows. The w binary is
also affected by this bug, because the two programs share a large portion of
their codebase. Therefore, similar considerations apply to w.

The overflow happens as follows (the Illumos source code available on GitHub
has been used as a reference for this analysis, even though it doesn't exactly
match the code of the binaries shipped with commercial Solaris versions):

* The psinfo structure info is populated by reading /proc/<pid>/psinfo
* The char array info.pr_fname[16] is copied into the char array
  up->p_comm[80+1]
* As a side note, the call to strncpy() at lines 344-345 incorrectly uses the
  size of the source buffer instead of the size of the destination buffer, but
  in this case this programming mistake doesn't cause a problem, because the
  source buffer is always smaller than the destination buffer:
  (void) strncpy(up->p_comm, info.pr_fname,
    sizeof (info.pr_fname));
* The char array up->p_args[80+1] is then populated at line 418 based on the
  char array info.pr_psargs[80] as follows:
  (void) strcpy(up->p_args, info.pr_psargs);
* If up->p_args begins with "?" or "- " (or, more correctly, with "-" followed
  by any byte <= 0x20), the following code branch at lines 423-425 is taken:
  (void) strcat(up->p_args, " (");
  (void) strcat(up->p_args, up->p_comm);
  (void) strcat(up->p_args, ")");
* In detail, the following chars are appended to the string:
  " (" + up->p_comm [maximum size excluding NULL-terminator is 15] + ")" + NULL
* Therefore, it is possible to overflow the up->p_args buffer at most as
  follows:
  * Buffer is 81 bytes: "- " + "B"x77 + " ("
  * Overflow is 17 bytes: "A"x15 + ")" + NULL

The uproc structure is declared at lines 106-119:
struct uproc {
  pid_t  p_upid;      /* user process id */
  char  p_state;    /* numeric value of process state */
  dev_t  p_ttyd;      /* controlling tty of process */
  time_t  p_time;      /* ticks of user & system time */
  time_t  p_ctime;    /* ticks of child user & system time */
  int  p_igintr;    /* 1=ignores SIGQUIT and SIGINT */
  char  p_comm[PRARGSZ+1];  /* command */
  char  p_args[PRARGSZ+1];  /* command line arguments */
  struct uproc  *p_child,  /* first child pointer */
      *p_sibling,  /* sibling pointer */
      *p_pgrplink,  /* pgrp link */
      *p_link;  /* hash table chain pointer */
};

A 17 bytes overflow past the p_args buffer is not large enough to reach
critical control structures and directly take control of the program flow.
However, we are able to overflow into the p_child and p_sibling members of the
uproc structure up, assuming 64-bit addressing. With 32-bit addressing we
should be able to corrupt additional pointers, i.e. p_pgrplink and p_link.

A skilled attacker might be able to leverage the corruption of these pointers
to obtain arbitrary code execution. However, he or she would face a number of
additional challenges:

* The target program uses privilege bracketing with the PRIV_PROC_OWNER
  privilege. This privilege allows a process to send signals to other
  processes, inspect, and potentially modify (with some additional
  restrictions) the process state in other processes, regardless of ownership.
  Therefore, it's theoretically possible to write a shellcode that activates
```


File Archive: December 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Top Authors In Last 30 Days

Red Hat 150 files
Ubuntu 68 files
LiquidWorm 23 files
Debian 16 files
malvuln 11 files
nu11secur1ty 11 files
Gentoo 9 files
Google Security Research 6 files
Julien Ahrens 4 files
T. Weber 4 files

File Tags

ActiveX (932)
Advisory (79,754)
Arbitrary (15,694)
BBS (2,859)
Bypass (1,619)
CGI (1,018)
Code Execution (6,926)
Conference (673)
Cracker (840)
CSRF (3,290)
DoS (22,602)
Encryption (2,349)
Exploit (50,359)
File Inclusion (4,165)
File Upload (946)
Firewall (821)
Info Disclosure (2,660)
Intrusion Detection (867)
Java (2,899)
JavaScript (821)
Kernel (6,291)
Local (14,201)
Magazine (586)
Overflow (12,419)
Perl (1,418)
PHP (5,093)
Proof of Concept (2,291)
Protocol (3,435)
Python (1,467)
Remote (30,044)
Root (3,504)
Ruby (594)
Scanner (1,631)
Security Tool (7,777)
Shell (3,103)
Shellcode (1,204)
Sniffer (886)

File Archives

December 2022
November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
Older

Systems

AIX (426)
Apple (1,926)
BSD (370)
CentOS (55)
Cisco (1,917)
Debian (6,634)
Fedora (1,690)
FreeBSD (1,242)
Gentoo (4,272)
HPUX (878)
iOS (330)
iPhone (108)
IRIX (220)
Juniper (67)
Linux (44,315)
Mac OS X (684)
Mandriva (3,105)
NetBSD (255)
OpenBSD (479)
RedHat (12,469)
Slackware (941)
Solaris (1,607)

```
  the privilege and dumps the memory of a privileged process (e.g. "passwd")
  via /proc/<pid>/mem, without ever executing an actual shell. However, this
  must be done before privileges are relinquinshed at line 455. This leaves
  only a limited amount of code paths to leverage our corrupted structure
  (namely, the main loop through /proc starting at line 315 and ending at line
  452).
* The char array info.pr_psargs[80] is cleaned up by the clnarglist() function
  at line 417: non-printable ASCII chars (c < 0x20 and c > 0x7e) get replaced
  with a "?" and must therefore be considered badchars. Luckily this
  restriction does not apply to the part of the buffer that causes the actual
  overflow, but only bytes that are valid in file names can be used in our
  malicious buffer.
* The ")" + NULL chars at the end of the evil buffer might cause unforeseen
  problems during exploitation.
* Additional security measures such as Address Space Layout Randomization
  (ASLR) might get in the way of reliable exploitation.

Based on this analysis, our conclusion is that this bug not exploitable on
Solaris 11.x and 10 in order to escalate privileges. That said, as a rule of
thumb all memory corruption issues have the potential to become serious
security vulnerabilities until otherwise proven. For instance, it might very
well be possible to exploit this bug on systems that don't implement privilege
bracketing, such as Solaris 9 and earlier. Therefore, we recommend to treat
this bug as a potential security vulnerability and to fix it as such.

4. Affected Platforms.

This bug was confirmed on the following platforms:

* Oracle Solaris 11.x (confirmed on 11.4 X86)
* Oracle Solaris 10 (confirmed on 10 1/13 X86)

Other Oracle Solaris versions (including those that run on the SPARC
architecture) and Illumos distributions are also likely affected.

5. Fix.

Oracle has assigned the tracking# S1199548 and has released a fix for all
affected and supported versions of Solaris in the Critical Patch Update (CPU)
of April 2020.

As a temporary workaround, it is possible to remove the setuid bit from whodo
and w executables as follows (note that this might prevent them from working
properly):

bash-3.2# chmod -s /usr/sbin/whodo /usr/bin/w

Copyright (c) 2020 Marco Ivaldi and @Mediaservice.net. All rights reserved.
```


Spoof (2,166)
SQL Injection (16,102)
TCP (2,379)
Trojan (686)
UDP (876)
Virus (662)
Vulnerability (31,136)
Web (9,365)
Whitepaper (3,729)
x86 (946)
XSS (17,494)
Other

SUSE (1,444)
Ubuntu (8,199)
UNIX (9,159)
UnixWare (185)
Windows (6,511)
Other

Site Links
News by Month
News Tags
Files by Month
File Tags
File Directory

About Us
History & Purpose
Contact Information
Terms of Service
Privacy Statement
Copyright Information

Hosting By
Rokasec

Follow us on Twitter
Subscribe to an RSS Feed