Τεχνικές εκμετάλλευσης τρωτοτήτων ασφάλειας πυρήνα

Πάτροκλος Αργυρούδης

census, inc
IT security research, development and services
argp@census.gr

University of Piraeus Software Libre Society
Event #16: Computer Security

Περιεχόμενα

Γιατί στον πυρήνα;

- Παράκαμψη τεχνολογιών αποτροπής εκμετάλλευσης
  - Non-executable stack/heap, ASLR, προστασία της διεύθυνσης επιστροφής
- Δυνατότητα για απόλυτα αθόρυβες επιθέσεις και κερκόπορτες
- Πολύ μεγαλύτερο ενδιαφέρον

Γιατί (όχι) στον πυρήνα;

- Δυσκολίες
  - Ο εντοπισμός σφαλμάτων κατά την εκμετάλλευση
  - Η δημιουργία αξιόπιστου κώδικα εκμετάλλευσης
  - Το βασικότερο πρόβλημα αποτελεί η συνέχιση του πυρήνα μετά την εκμετάλλευση η οποία αφήνει το σύστημα σε ασταθή κατάσταση
- Καλή γνώση του πυρήνα του λειτουργικού συστήματος που διερευνούμε
- Καλή γνώση x86 Assembly

Γιατί FreeBSD;

- Οι υπάρχουσες προγενέστερες εργασίες αφορούν
  - Είτε άλλα συστήματα (Windows, Linux, OpenBSD)
  - Είτε αρκετά παλαιότερες εκδόσεις του FreeBSD (4.0-4.1.1)
  - Είτε ιδιαίτερα συγκεκριμένες περιπτώσεις εκμετάλλευσης οι οποίες δεν είναι γενικά εφαρμόσιμες
- Θα διερευνήσουμε την τελευταία έκδοση του FreeBSD (7.0) σε x86

Προγενέστερες εργασίες (1)

- Exploiting kernel buffer overflows FreeBSD style (2000)
  - Αφορά τις εκδόσεις 4.0 εώς 4.1.1
  - Η τρωτότητα βρίσκεται στη κλήση συστήματος jail(2)
  - Παρουσιάζεται κατά τη ανάγνωση της κατάστασης του jail από το procfs όταν το εν λόγω jail έχει δημιουργηθεί με υπερβολικά μεγάλο hostname
  - Μη επαναχρησιμοποιήσιμος αυθαίρετος κώδικας (shellcode) πυρήνα
  - Μη επαναχρησιμοποιήσιμος αλγόριθμος εκμετάλλευσης
  - Μετά από οχτώ χρόνια οι τεχνικές είναι μη εφαρμόσιμες

Προγενέστερες εργασίες (2)

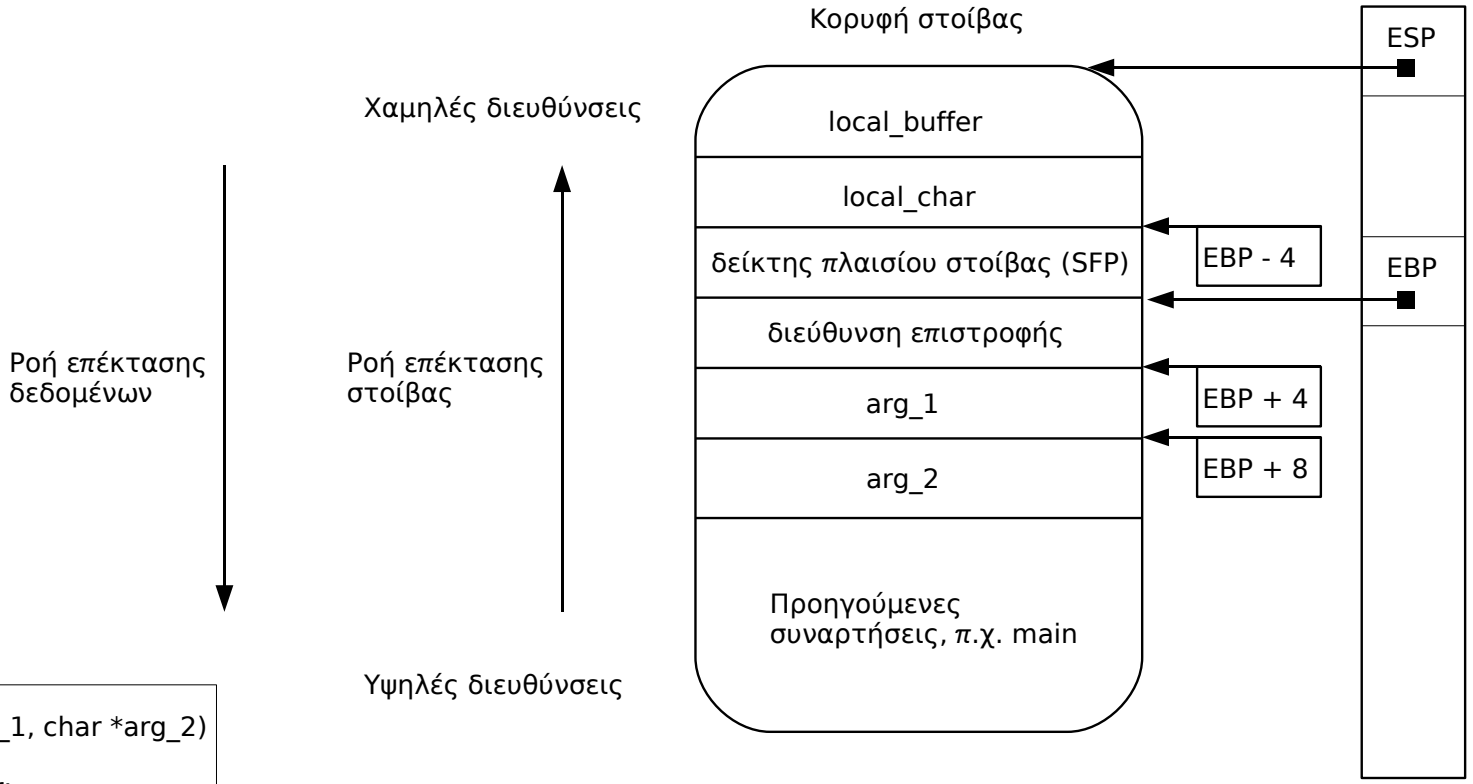
- Kernel level vulnerabilities, 2001
  - Solaris 2.7-2.8 (x86)
- Smashing the kernel stack for fun and profit, 2002
  - OpenBSD 2.x-3.x (x86)
  - Κύρια συνεισφορά: τεχνική για τη συνέχιση του πυρήνα
- Kernel wars, 2007
  - Συγκεκριμένες περιπτώσεις εκμετάλλευσης τρωτοτήτων σε Windows, {Free, Net, Open}BSD (x86)
  - Δεν δημοσιεύθηκαν λεπτομέρειες και κώδικας
- Attacking the core: kernel exploiting notes, 2007
  - Linux (x86, AMD64), Solaris (UltraSPARC)
  - Κύρια συνεισφορά: ανάλυση τρωτοτήτων σωρού (heap) στον πυρήνα

Κατηγορίες τρωτοτήτων πυρήνα

- Εκτέλεση αυθαίρετου κώδικα με στόχο την
  - Κλιμάκωση προνομίων
  - Διαφυγή από περιορισμένα περιβάλλοντα τύπου jail, chroot, κτλ.
- Διαρροή πληροφοριών από τη μνήμη του πυρήνα
  - Ιδιωτικά SSH κλειδιά από τη περιοχή μνήμης του sshd
  - Περιεχόμενα ευαίσθητων αρχείων (/etc/master.passwd)
  - Δομή και περιεχόμενα της στοίβας ή/και του σωρού

Υπερχείλιση στοίβας

- Πολύ γνωστό και κατανοητό πρόβλημα, υπερχείλιση τοπικών μεταβλητών από αντιγραφή δεδομένων χωρίς έλεγχο ορίων




```
void func(int arg_1, char *arg_2)
{
  char local_char;
  char local_buffer[512];
  strcpy(local_buffer, arg_2);
}
```

census

Στοίβες πυρήνα

- Ο ιδεατός χώρος διευθύνσεων χωρίζεται σε δύο τμήματα, ένα για τον πυρήνα και ένα για τις διεργασίες (σε x86)
- Κάθε διεργασία έχει τη δική της αποκλειστική στοίβα στον πυρήνα
- Όταν η διεργασία χρησιμοποιεί υπηρεσίες του πυρήνα (π.χ. int $0x80) ο ESP δείχνει στην αντίστοιχη στοίβα πυρήνα
- Οι στοίβες του πυρήνα έχουν σταθερό μέγεθος και δεν μεγαλώνουν δυναμικά, σε x86 το μέγεθος είναι δυο σελίδες
- Ο κύριος λόγος ύπαρξης τους είναι ότι πρέπει να παραμένουν πάντα στη μνήμη ώστε να μπορούν να εξυπηρετούν πιθανά σφάλματα σελίδων
- Κατά την υλοποίηση εφαρμογών που πρόκειται να εκτελεστούν στον πυρήνα πρέπει να αποφεύγονται μεγάλες τοπικές μεταβλητές

Υπερχείλιση στοίβας του πυρήνα

- Δυο διαφορετικές περιπτώσεις υπερχείλισης
  - Τοπικής μεταβλητής με αποτέλεσμα την επικάλυψη α) της διεύθυνσης επιστροφής της συνάρτησης, ή β) του δείκτη του πλαισίου στοίβας, ή γ) κάποιου δείκτη συνάρτησης που έχει οριστεί ως τοπική μεταβλητή
  - Της στοίβας συνολικά από διαδοχικές ένθετες συναρτήσεις
- Σε περίπτωση υπερχείλισης προκαλείται σφάλμα προσπέλασης και ο πυρήνας του FreeBSD ``κρεμάει''
  - Θα μπορούσε απλά να τερματίζεται η διεργασία (Linux)
  - Το FreeBSD (ορθώς) τα αντιμετωπίζει ως θεμελιώδη σφάλματα

Αλγόριθμος εκμετάλλευσης πυρήνα

1. Εκτροπή της ροής εκτέλεσης του πυρήνα με στόχο την εύρεση του αυθαίρετου κώδικά μας
2. Αυθαίρετος εκτελέσιμος κώδικας μηχανής πυρήνα
3. Κλιμάκωση προνομίων της τρέχουσας διεργασίας (ή του γονέα της)
4. Συνέχιση πυρήνα

Εκτροπή της ροής εκτέλεσης του πυρήνα

- Κώδικας που εκτελείται σε πλαίσιο διεργασίας (process context) μπορεί να ανατρέξει στον ιδεατό χώρο διευθύνσεων του χρήστη
  - Πλαίσιο διεργασίας είναι το πλαίσιο κατά το οποίο ο πυρήνας εκτελείται για λογαριασμό κάποιας διεργασίας (π.χ. κλήση συστήματος)
- Συνεπώς μπορούμε να στείλουμε τη ροή εκτέλεσης στην περιοχή μνήμης που χρησιμοποιείται ως προέλευση στην προβληματική συνάρτηση
  - Σε αντίθεση με την εκμετάλλευση απλών εφαρμογών όπου συνήθως χρησιμοποιούμε την περιοχή μνήμης που χρησιμοποιείται ως προορισμός
  - Κύριο πλεονέκτημα: γνωρίζουμε ακριβώς πού βρίσκεται ο στόχος μας (δεν υπάρχει ανάγκη για λύσεις τύπου ελκήθρου από nop εντολές)

Αυθαίρετος κώδικας πυρήνα

- Συνήθως δεν είναι απαραίτητος στη μορφή που γνωρίζουμε από την εκμετάλλευση απλών εφαρμογών
- Μπορούμε να κάνουμε την υλοποίηση σε C και να επιστρέψουμε στην περιοχή μνήμης όπου βρίσκεται
- Εμείς θα παρουσιάσουμε αυθαίρετο κώδικα εξολοκλήρου σε Assembly
- Η γνωστή μέθοδος όπου αντικαθιστούμε την τρέχουσα διεργασία με αυτή που παρουσιάζει το σφάλμα ασφάλειας χρησιμοποιώντας συναρτήσεις της οικογένειας execve(2) προφανώς δεν λειτουργεί
- Ο κώδικάς μας πρέπει να
  1. Εντοπίσει τη δομή του πυρήνα της τρέχουσας διεργασίας (ή του γονέα της)
  2. Τροποποιήσει τα προνόμιά της (effective και real UID) στα ανώτερα δυνατά (0)
  3. Διαφυγή από jail κτλ.

Εντοπισμός της δομής διεργασίας

- Τα προνόμια είναι αποθηκευμένα στη δομή proc
  - http://fxr.watson.org/fxr/source/sys/proc.h?v=FREEBSD70#L484
- Η διεύθυνση της οποίας βρίσκεται στη δομή kinfo_proc
  - http://fxr.watson.org/fxr/source/sys/user.h?v=FREEBSD70#L115
- Εύρεση της διεύθυνσης της proc μέσω του sysctl(9) interface του πυρήνα και της δομής kinfo_proc

sysctlnametomib(3)

- Η καταχώρηση kern.proc.pid του MIB παρέχει τη δομή kinfo_proc

```
u_long
get_proc(void)
{
        int mib[4];
        size_t len;
        struct kinfo_proc kp;

        len = 4;
        sysctlnametomib("kern.proc.pid", mib, &len);

        mib[3] = getppid();
        len = sizeof(struct kinfo_proc);

        if(sysctl(mib, 4, &kp, &len, NULL, 0) == -1)
        {
                perror("sysctl");
                exit(1);
        }

        return (u_long)kp.ki_paddr;
}
```

Κλιμάκωση προνομίων

- Τα προνόμια είναι αποθηκευμένα σε δομή τύπου ucred η οποία υπάρχει στη δομή proc
  - http://fxr.watson.org/fxr/source/sys/ucred.h?v=FREEBSD70#L45

```
# η διεύθυνση της δομής proc στον ECX
mov $0x12345678, %ecx
# EBX = ucred
mov 0x30(%ecx), %ebx
# EAX = 0
xor %eax, %eax
# effective user ID = cr_uid = 0
mov %eax, 0x4(%ebx)
# real user ID = cr_ruid = 0
mov %eax, 0x8(%ebx)
```

Συνέχιση πυρήνα

- Το βασικότερο πρόβλημα κατά την εκμετάλλευση τρωτοτήτων του πυρήνα
- Η στοίβα, οι καταχωρητές και συνεπώς το σύστημα γενικότερα βρίσκονται σε ασταθή κατάσταση μετά την επικάλυψη της στοίβας για την αντικατάσταση της διεύθυνσης επιστροφής
- Για να αποφύγουμε κατάρρευση του συστήματος πρέπει να διορθώσουμε την κατάσταση της στοίβας και των καταχωρητών
- Και να επιστρέψουμε τη ροή εκτέλεσης σε κατάλληλο σημείο
- Δεν είναι πάντα απαραίτητο, εξαρτάται από την τρωτότητα και το σύστημα
    - Σε Linux συνήθως μπορούμε απλά να αγνοήσουμε αυτό το βήμα αφού ο πυρήνας απλά τερματίζει τη διεργασία

Τεχνικές συνέχισης πυρήνα

- Εξομοίωση της διαδικασίας επιστροφής από τον πυρήνα
  - Δεν μετατρέπεται εύκολα σε αξιόπιστο αυθαίρετο κώδικα μηχανής
  - http://fxr.watson.org/fxr/source/i386/i386/exception.s?v=FREEBSD70#L186
  - Αλλαγή δομών πυρήνα, εκκρεμή σήματα/κλειδώματα
- Μπορούμε ``απλά'' να επιστρέψουμε τη ροή της εκτέλεσης στο σημείο που θα επέστρεφε εάν δεν υπήρχε η εκμετάλλευση της τρωτότητας (και ο αυθαίρετος κώδικάς μας)
- Η σύμβαση κλήσης κάθε αρχιτεκτονικής ορίζει ποιοι καταχωρητές πρέπει να επαναφερθούν στις αρχικές τους τιμές (σε x86: EBX, ESI, EDI)
- Επίσης πρέπει να δώσουμε κάποια έγκυρη τιμή στον EBP ώστε μετά την επιστροφή του αυθαίρετου κώδικά μας να δείχνει στο προηγούμενο πλαίσιο στη στοίβα πυρήνα

Επιστροφή από τον πυρήνα

```
push %ebx
push %esi
push %edi

# αυθαίρετος κώδικας, π.χ. κλιμάκωση προνομίων

pop %edi
pop %esi
pop %ebx

# αποκατάσταση %ebp

xor %eax, %eax
push $διεύθυνση_επιστροφής_πυρήνα
ret
```

Αποκατάσταση EBP

- $EBP = ESP + VAL => VAL = EBP \quad ESP$
  - Ο EBP πρέπει να δείχνει στο προηγούμενο πλαίσιο στη στοίβα πυρήνα
  - Καταγράφουμε την τιμή του EBP κατά τον πρόλογο της προβληματικής συνάρτησης
  - και από αυτή αφαιρούμε την τιμή του ESP μετά την επιστροφή της προβληματικής συνάρτησης στην προηγούμενη συνάρτηση
- Συνεπώς για να αποκαταστήσουμε τον EBP στη σωστή τιμή του μπορούμε να προσθέσουμε την τιμή *VAL* που υπολογίσαμε στον ESP
  - lea 0x*VAL*(%esp), %ebp

Αποφυγή στατικής διεύθυνσης επιστροφής

- Κατά την εκτέλεση του αυθαίρετου κώδικά μας πρέπει να υπολογίσουμε δυναμικά τις διευθύνσεις των
  - syscall
  - και στη συνέχεια του σημείου στη συνάρτηση syscall στο οποίο πρέπει να επιστρέψει η ροή εκτέλεσης

Εύρεση της syscall

- Ο στόχος μας είναι να υπολογίσουμε δυναμικά τη διεύθυνση του χειριστή διακοπών int $0x80 syscall
- Το Interrupt Descriptor Table (IDT) χρησιμοποιείται για την διαχείριση διακοπών από το λειτουργικό σύστημα
- Όταν έχουμε διακοπή τύπου int $0x80 αρχικά καλείται ο χειριστής διακοπών Xint0x80_syscall, ο οποίος στη συνέχεια καλεί τον πραγματικό χειριστή syscall
- Η εντολή sidt παρέχει τη διεύθυνση του IDT (idt0 στο FreeBSD)
  - Εξαιτίας του τρόπου με τον οποίο υλοποιείται σε συστήματα εικονικοποίησης δεν λειτουργεί πάντα όπως περιμένουμε
- idt0 => Xint0x80_syscall => syscall

Εύρεση της διεύθυνσης επιστροφής

- Στόχος μας είναι να αποφύγουμε τι στατική μετατόπιση των 0x335 bytes
- Στο FreeBSD 7.0 η syscall διεκπεραιώνει κλήσεις συστήματος με την εντολή call *%edx
  - Σε δεκαεξαδική κωδικοποίηση: ffd2
- Συνεπώς εντοπίζουμε τη διεύθυνση της syscall μέσω του idt0
- και στη συνέχεια από αυτή τη διεύθυνση ως σημείο εκκίνησης αναζητούμε την εντολή call *%edx
- Αφού την εντοπίσουμε απλά αυξάνουμε την τιμή της κατά ένα για να πάρουμε τη διεύθυνση επιστροφής του πυρήνα

Ολοκληρωμένος αλγόριθμος εκμετάλλευσης (1)

1. Εκτροπή της ροής εκτέλεσης του πυρήνα
   - Οπουδήποτε στον ιδεατό χώρο διευθύνσεων του χρήστη
   - Περιοχή μνήμης προέλευσης, μνήμη από mmap(2), μεταβλητές περιβάλλοντος, ορίσματα γραμμής εντολών, ...
   - Εάν η τρωτότητα επιτρέπει αυθαίρετη επικάλυψη μνήμης: δείκτη συνάρτησης (ioctl), καταχώρηση του πίνακα sysent
     - http://fxr.watson.org/fxr/source/kern/init_sysent.c?v=FREEBSD70#L31
2. Αυθαίρετος κώδικας πυρήνα
   - Μπορεί να υλοποιηθεί σε C ή εξολοκλήρου σε Assembly
3. Απενεργοποίηση μέτρων ασφάλειας, αλλαγή securelevel, κτλ.
4. Διαφυγή από jail, chroot, κτλ.

Ολοκληρωμένος αλγόριθμος εκμετάλλευσης (2)

5. Κλιμάκωση προνομίων
   - Εντοπισμός της δομής διεργασίας: sysctlnametomib(3) kern.proc.pid, mov %fs:0x0, %ecx, allproc
     - http://fxr.watson.org/fxr/source/sys/proc.h?v=FREEBSD70#L803
   - Εντοπισμός της δομής ucred και αλλαγή των effective και real user ID
6. Εγκατάσταση κερκόπορτας
7. Συνέχιση πυρήνα
   - Δεν είναι πάντα απαραίτητο βήμα
   - Εξομοίωση της διαδικασίας επιστροφής από τον πυρήνα
   - Επιστροφή της ροής εκτέλεσης στο κατάλληλο σημείο, αποκατάσταση καταχωρητών

Πραγματικές τρωτότητες πυρήνα (1)

- Mac OS X 10.4 -- bsd/netat/ddp_rtmptable.c (Tobias Klein)

```
zt_add_zone(name, length)
char *name;
short length;
{
        at_nvestr_t zname;
        bcopy(name, &zname.str, length);
        zname.len = length;
        return (zt_add_zonename(&zname));
}
```

- Linux 2.4.9 -- drivers/isdn/act2000/capi.c (Dawson Engler)

```
isdn_ctrl cmd;
...
while((skb = skb_dequeue(&card->rcvq)))
{
        msg = skb->data;
        ...
        memcpy(cmd.parm.setup.phone,
                msg->msg.connect_ind.addr.num,
                msg->msg.connect_ind.addr.len - 1);
```

Πραγματικές τρωτότητες πυρήνα (2)

- Linux 2.2.16 -- arch/i386/kernel/mtrr.c (Silvio Cesare)

```
static ssize_t mtrr_write(struct file *file, const char *buf,
        size_t len, loff_t *ppos)
{
        int i, err;
        ...
        memset(line, 0, LINE_SIZE);
        if(len > LINE_SIZE) len = LINE_SIZE;
        if(copy_from_user(line, buf, len - 1)) return -EFAULT;
```

- Madwifi 0.9.2 -- net80211/ieee80211_wireless.c (Laurent Butti)

```
static void giwscan_cb(void *arg,
        const struct ieee80211_scan_entry *se)
{
        struct iwscanreq *req = arg;
        ...
        memset(&iwe, 0, sizeof(iwe));
        memcpy(buf, se->se_wpa_ie, se->se_wpa_ie[1] + 2);
        iwe.cmd = IWEVGENIE;
```

Όχι τέλος αλλά αρχή...

- Ανάλυση πραγματικών τρωτοτήτων
- Υλοποίηση προγραμμάτων εκμετάλλευσης για δημοσιευμένα προβλήματα
- Εύρεση τρωτοτήτων σε πυρήνες ανοικτού κώδικα
- Υπερχείλιση μεταβλητών στο σωρό του πυρήνα
  - BSD -- zone allocator
  - Linux -- buddy allocator
- Wargame/CTF

Παραπομπές

Kirk McKusick and George Neville-Neil

The design and implementation of the FreeBSD operating system
Addison-Wesley, 2004

Esa Etelavuori

Exploiting kernel buffer overflows FreeBSD style
fbsdjail.txt, 2000

Last Stage of Delirium research group

Kernel level vulnerabilities
5th Argus Hacking Challenge, 2001

Sinan "noir" Eren

Smashing the kernel stack for fun and profit
Phrack, Volume 0x0b, Issue 0x3c, 2002

Bitsec team

Kernel wars
Black Hat Europe, 2007

sgrakkyu and twiz

Attacking the core: kernel exploiting notes
Phrack, Volume 0x0c, Issue 0x40, 2007