Εισαγωγή εκτελέσιμου κώδικα σε διεργασίες

Φώτης Λούκος (fotisl)

0x375 - Thessaloniki Tech Talks Sessions
Event 0x2

19 Μαρτίου 2010

Περιεχόμενα

Γιατί;

- Πολλές φορές θέλουμε να εκτελέσουμε κώδικα ο οποίος θα μοιράζεται resources με κάποιο process (vm, file descriptors)
- Ο κώδικας αυτός μπορεί να τρέχει σε ένα ξεχωριστό thread κάντοντας το πιο δύσκολο στον εντοπισμό (ειδικό flag στην εντολή ps, δεν υπάρχει ξεχωριστό directory στο /proc αλλά βρίσκεται κάτω από το /proc/<pid>/task)

Πώς;

- Θα πρέπει να βρούμε έναν τρόπο να εκτελέσουμε δικό μας κώδικα μέσα στα πλαίσια του προγράμματος
- ο κώδικας αυτός πρέπει να φροντίσει για την συνέχιση της εκτέλεσης του νέου κώδικα που θα τρέχει παράλληλα, δεσμεύοντας την κατάλληλη μνήμη και δημιουργόντας το thread
- Τα παραπάνω θα γίνουν με τις κλήσεις συστήματος ptrace και clone

Σύνοψη

Δήλωση

```
#include <sys/ptrace.h>
long ptrace(enum __ptrace_request request, pid_t pid,
            void *addr, void *data);
```

Παράμετροι

- request: Η ενέργεια που θα πραγματοποιήσουμε
- pid: Το process id της διεργασίας
- addr: Εξαρτάται από την ενέργεια
- data: Εξαρτάται από την ενέργεια

Δυνατότητες της ptrace

- PTRACE_PEEKME/PTRACE_ATTACH: Έναρξη ελέγχου διεργασίας
- PTRACE_DETACH: Τερματισμός ελέγχου διεργασίας
- PTRACE_PEEKTEXT/PTRACE_PEEKDATA: Ανάγνωση μίας λέξης από το address space της διεργασίας
- PTRACE_POKETEXT/PTRACE_POKEDATA: Εγγραφή μίας λέξης στο address space της διεργασίας
- PTRACE_GETREGS: Ανάγνωση των καταχωρητών γενικής χρήσης της διεργασίας
- PTRACE_SETREGS: Εγγραφή στος καταχωρητές γενικής χρήσης της διεργασίας

Έναρξη ελέγχου διαδικασίας

Πηγαίος κώδικας

```
ptrace(PTRACE_ATTACH, proc, NULL, NULL);
wait(NULL);
```

Ροή προγράμματος

- Εκτελείται η ptrace από το process που θέλει να προσκοληθεί σε κάποιο άλλο
- Το δεύτερο process γίνεται παιδί του πρώτου και λαμβάνει το σήμα SIGSTOP
- Ο πατέρας περιμένει μέχρι να λάβει ειδοποίηση ότι το παιδί έλαβε το σήμα το οποίο σημαίνει ότι πλέον έχει σταματήσει πλήρως

Ανάγνωση/εγγραφή δεδομένων διαδικασίας

Πηγαίος κώδικας

```
data = ptrace(PTRACE_PEEKDATA, proc, addr, NULL);
data++;
ptrace(PTRACE_POKEDATA, proc, addr, data);
```

Ροή προγράμματος

- Ο πατέρας με την πρώτη κλήση της ptrace διαβάζει ένα word από την διεύθυνση addr του παιδιού και το επιστρέφει
- Αυξάνει την τιμή του και με την δεύτερη κλήση της ptrace γράφει την τιμή αυτή πάλι στην διεύθυνση addr

Ανάγνωση/εγγραφή καταχωρητών διαδικασίας

Πηγαίος κώδικας

```
struct user_regs_struct regs;
ptrace(PTRACE_GETREGS, proc, NULL, &regs);
regs.eax = 0xdeadbeef;
ptrace(PTRACE_SETREGS, proc, NULL, &regs);
```

Ροή προγράμματος

- Ο πατέρας με την πρώτη κλήση της ptrace διαβάζει όλους τους καταχωρητές γενικής χρήσης του παιδιού στην δομή regs
- Θέτει στον eax την τιμή 0xdeadbeef και με την δεύτερη κλήση της ptrace θέτει τις τιμές που είναι αποθηκευμένες στην δομή regs στους καταχωρητές του παιδιού

Εισαγωγή κώδικα σε μία διεργασία

```
void readmem(pid_t proc, unsigned long addr,
        char *buffer, int length);
void writemem(pid_t proc, unsigned long addr,
        char *buffer, int length);

ptrace(PTRACE_GETREGS, proc, NULL, &regs);
readmem(proc, regs.eip, oldcode, len);
writemem(proc, regs.eip, injcode, len);
ptrace(PTRACE_CONT, proc, NULL, NULL);
waitpid(proc, NULL, 0);
writemem(proc, regs.eip, oldcode, len);
ptrace(PTRACE_SETREGS, proc, NULL, &regs);
ptrace(PTRACE_CONT, proc, NULL, NULL);
```

Εισαγωγή κώδικα σε μία διεργασία

- Αρχικά αποθηκεύουμε τις τιμές των καταχωρητών στην δομή regs
- Αποθηκεύουμε τον παλιό κώδικα στην θέση eip στην μεταβλητή oldcode
- Στην θέση του τοποθετούμε τον κώδικα που θέλουμε να εισάγουμε
- Συνεχίζουμε την διεργασία και εκτελείται ο κώδικας μας
- Πρέπει η εκτέλεση να σταματήσει ώστε να αναλάβει ο πατέρας και για τον λόγο αυτό στο τέλος του κώδικα μας εισάγουμε την εντολή `int 3`
- Ο πατέρας επαναφέρει τον αρχικό κώδικα και τις αρχικές τιμές των καταχωρητών
- Η εκτέλεση συνεχίζεται εκτελόντας τον αρχικό κώδικα της διεργασίας

Εισαγωγικά

- Κάθε διεργασία έχει τουλάχιστον ένα νήμα
- Κάθε νήμα αποτελεί μία διαφορετική ροή εκτέλεσης που μπορεί να εκτελείται ταυτόχρονα με κάποια άλλη
- Η δημιουργία γίνεται με την χρήση της κλήσης συστήματος clone
- Το νέο νήμα μπορεί να μοιράζεται την ίδια μνήμη με το νήμα που το δημιούργησε, τα ίδια ανοιχτά αρχεία, το ίδιο αναγνωριστικό (pid) κτλ
- Κάθε νήμα έχει την δική του στοίβα

Σύνοψη της clone

Δήλωση

```
#include <sched.h>
int clone(int (*fn)(void *), void *child_stack, int flags,
          void *arg, ... );
```

Παράμετροι

- fn: η συνάρτηση που θα εκτελέσει το νέο νήμα
- child_stack: η στοίβα του νέου νήματος
- flags: ρυθμίσεις για το νέο νήμα
- arg: όρισμα που θα περάσει στην συνάρτηση fn
- λοιπές παράμετροι: εξαρτώνται από τις ρυθμίσεις

Κλήση της clone από assembly

- Σε αντίθεση με τις υπόλοιπες κλήσεις συστήματος ο wrapper δεν μεταφέρει απλά τα ορίσματα στους καταχωρητές
- Πρέπει πρώτα να βρεθεί χώρος για την στοίβα και να ετοιμαστεί κατάλληλα
- Οι λοιπές παράμετροι δίνονται πάντα αλλά χρησιμοποιούνται μόνο στην περίπτωση που υπάρχουν ειδικές ρυθμίσεις στην μεταβλητή flags
- Μετά την κλήση συστήματος (με χρήση της `int 0x80` ή `sysenter`) πρέπει να γίνει έλεγχος για την τιμή που επιστρέφεται στον καταχωρητή eax και εαν είναι μηδενική πρέπει ο κώδικας να καλέσει την συνάρτηση fn
- Επειδή η συνάρτηση μπορεί να επιστρέψει θα πρέπει να χρησιμοποιηθεί η κλήση συστήματος exit για να σταματήση το νήμα την εκτέλεσή του

Απαιτήσεις για ένα νέο νήμα

- Θα χρειαστεί ειδικός χώρος όπου θα υπάρχει ο κώδικας που θα εκτελεστεί και χώρος για την νέα στοίβα
- Για να μην γίνει overwrite χρήσιμη μνήμη μπορεί να χρησιμοποιηθεί η κλήση συστήματος brk η οποία θα μας επιστρέψει νέα μνήμη
- Η μνήμη η οποία θα επιστραφεί κατά κανόνα δεν βρίσκεται σε μέρος όπου επιτρέπεται η εκτέλεση κώδικα, μόνο η ανάγνωση και εγγραφή
- Μπορούμε να δημιουργήσουμε ένα νέο ανώνυμο mapping όπου θα έχουμε όλα τα απαραίτητα δικαιώματα (rwx) και να βάλουμε στην αρχή τον κώδικα που θα εκτελεστεί και στο τέλος θα αρχίζει η στοίβα

Το πρόγραμμα εισαγωγής κώδικα prez

Το prez αποτελείται από τρία μέρη

- τον injector
- τον thread creator
- τον shellcode

Ο injector

Το κυρίως πρόγραμμα, γραμμένο σε C. Τα βήματα που εκτελεί είναι:

- Attach στο process
- Αποθήκευση του παλιού κώδικα στο σημείο που δείχνει ο eip και των registers
- Εγγραφή του thread creator και του shellcode
- Συνέχιση της εκτέλεσης του process και εκτέλεση του κώδικα που αντιγράψαμε πιο νωρίς
- Ο thread creator μετά την ολοκλήρωση του εκτελεί την εντολή int3 και επιστρέφει πίσω στον injector
- Ο injector επαναφέρει τον παλιό κώδικα και τις παλιές τιμές των registers και κάνει detach από το process

O thread creator

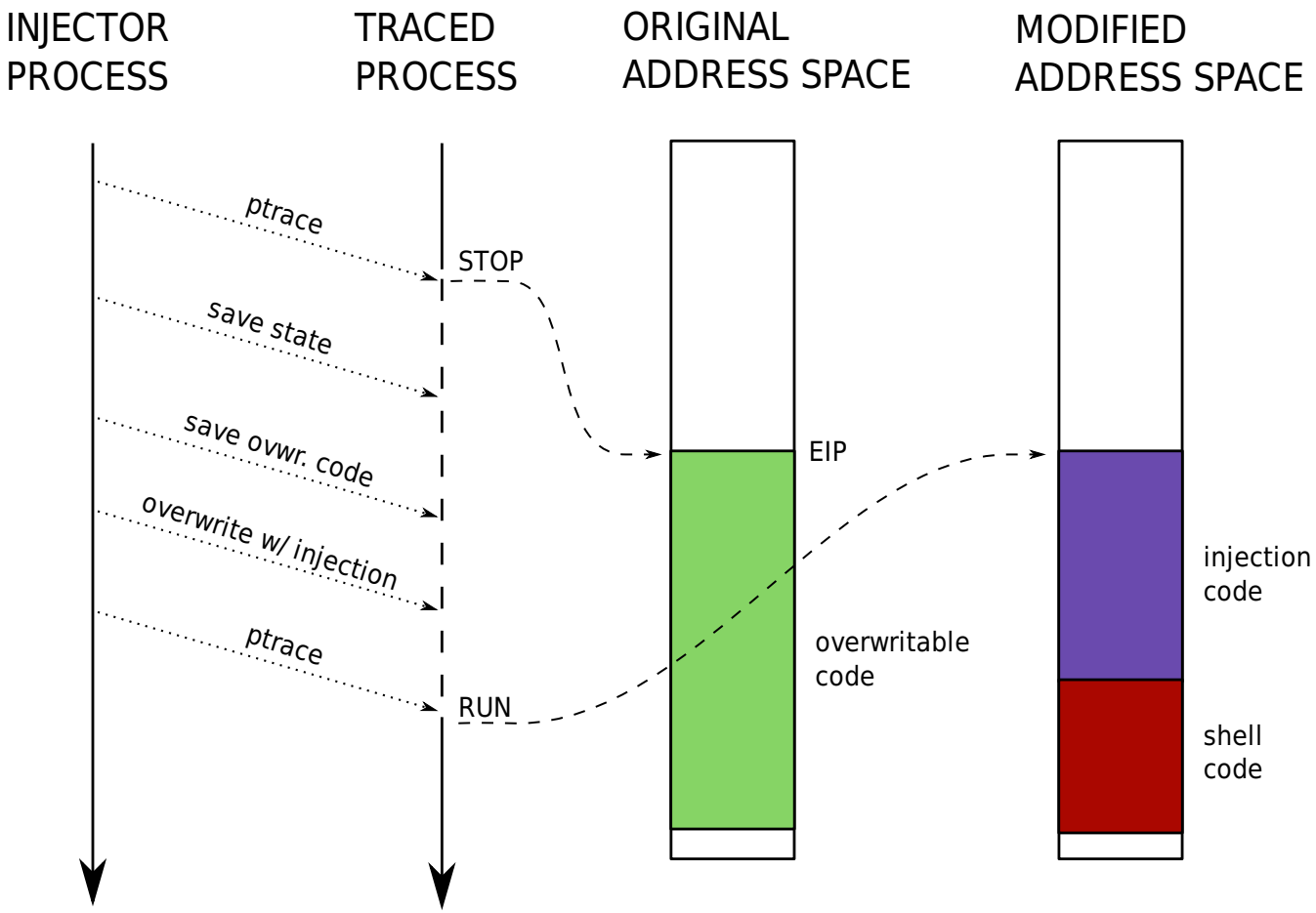
Το μέρος του προγράμματος που αναλαμβάνει το mapping της νέας μνήμης και την δημιουργία του thread. Πλήρως γραμμένο σε assembly. Τα βήματα που εκτελεί είναι:

- Εκτέλεση της mmap για να λάβει ένα νέο μέρος μνήμης με δικαιώματα ανάγνωσης, εγγραφής και εκτέλεσης.
- Αντιγραφή του shellcode στην αρχή του νέου μέρους μνήμης που επιστράφηκε
- Δημιουργία κατάλληλης δομής στο τέλος της μνήμης αυτής που θα χρησιμοποιηθεί ως στοίβα του νέου thread
- Δημιουργία ενός νέου thread με την χρήση της clone
- Εκτέλεση της εντολής int3 για επιστροφή στον injector στον πατέρα ή κλήση του κώδικα που αντιγράφηκε στην νέα μνήμη το παιδί

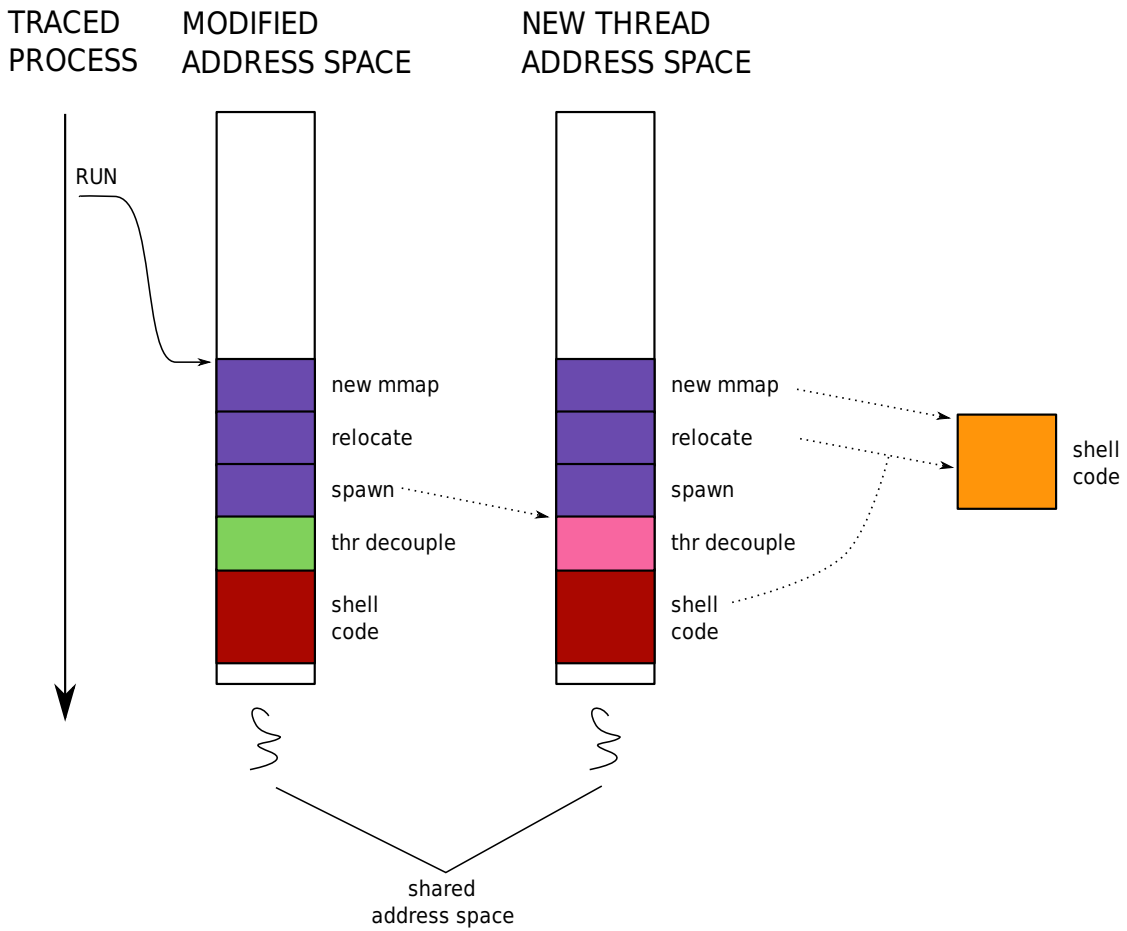
Ο shellcode

Ο κώδικας αυτός μπορεί να εκτελεί όποιες λειτουργίες θέλουμε. Τα χαρακτηριστικά του είναι ότι πρέπει να είναι position independent. Μέσα στο prez υπάρχει ως παράδειγμα ένα τμήμα κώδικα το οποίο δέχεται συνδέσεις σε μία συγκεκριμένη TCP πόρτα και εκτελεί ένα shell για κάθε νέα σύνδεση.

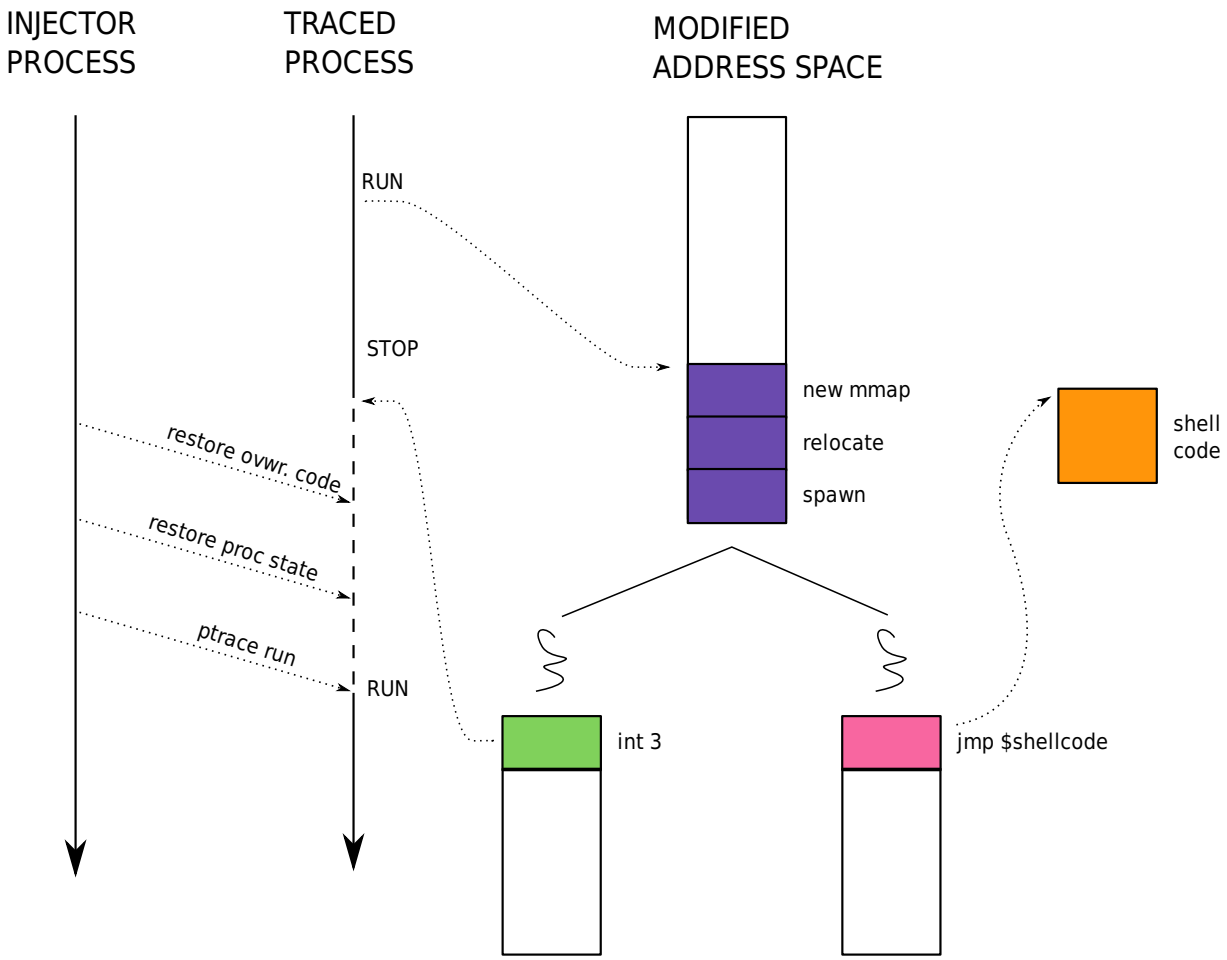
Σχηματικά


INJECTOR PROCESS
TRACED PROCESS
ORIGINAL ADDRESS SPACE
MODIFIED ADDRESS SPACE
ptrace
STOP
save state
save ovwr. code
overwrite w/ injection
ptrace
RUN
EIP
overwritable code
injection code
shell code

Σχηματικά


TRACED
PROCESS
MODIFIED
ADDRESS SPACE
NEW THREAD
ADDRESS SPACE
RUN
new mmap
relocate
spawn
thr decouple
shell
code
new mmap
relocate
spawn
thr decouple
shell
code
shell
code
shared
address space

Σχηματικά


INJECTOR PROCESS
TRACED PROCESS
MODIFIED ADDRESS SPACE
RUN
STOP
new mmap
relocate
spawn
shell code
restore ovwr. code
restore proc state
ptrace run
RUN
int 3
jmp $shellcode

Ιδέες για το μέλλον

- Μεταφορά σε άλλα λειτουργικά συστήματα (ήδη υπάρχει έκδοση για freebsd)
- Shellcode με περισσότερες δυνατότητες (εύρεση passwords αποθηκευμένων στην μνήμη, καταγραφή δεδομένων εισόδου - εξόδου)
- Shellcode ο οποίος πλέον θα κάνει ptrace το αρχικό process και θα μπορεί να το χειρίζεται πλήρως (ήδη έχουν γίνει κάποια βήματα προς αυτή την κατεύθυνση)

Ερωτήσεις

Ευχαριστώ για την προσοχή σας!
Ερωτήσεις;