Μηχανική για τη φθορά της μνήμης

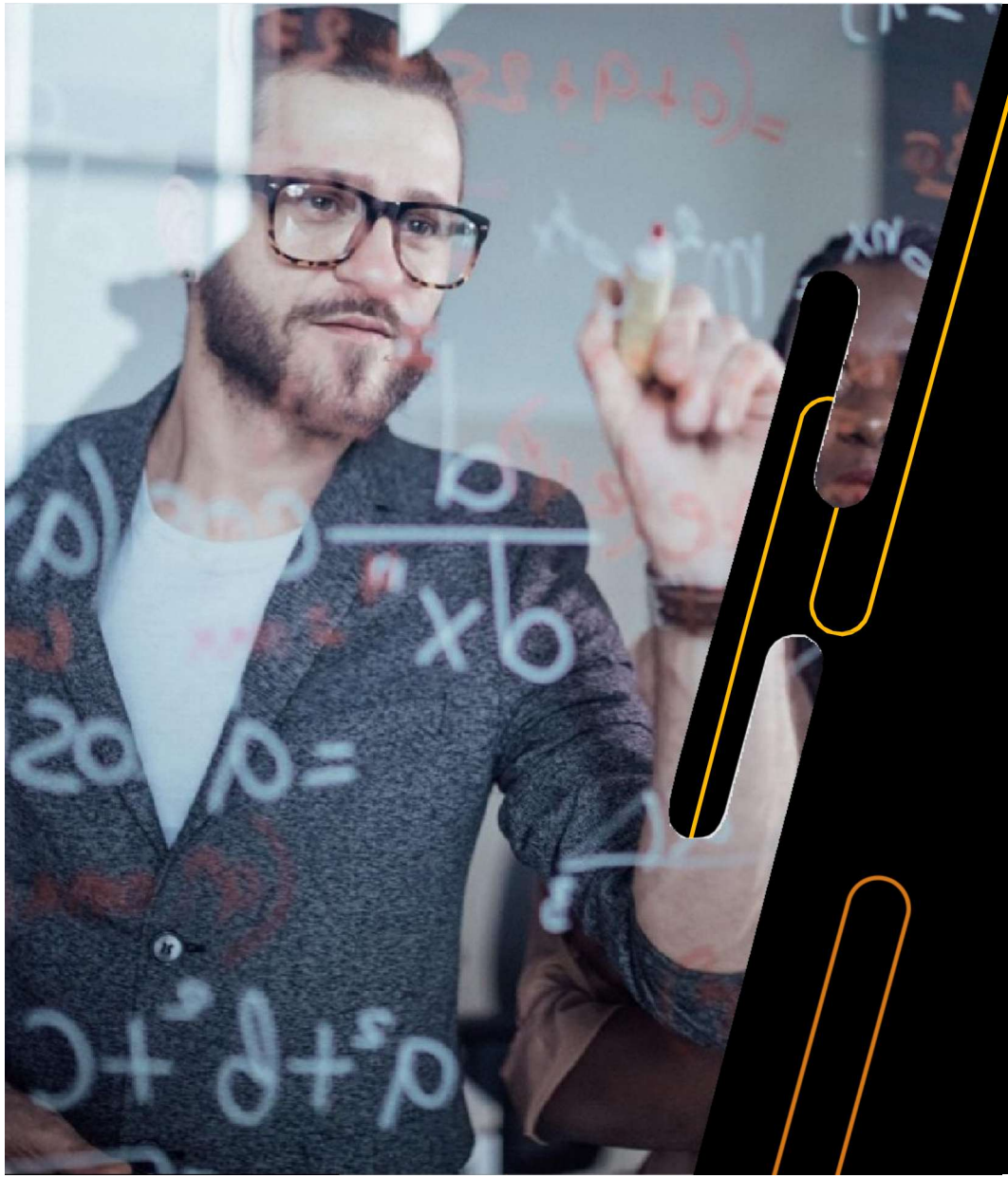
CSP009_S_E

ΠΑΡΟΥΣΙΑΣΗ ΑΠΟ ΤΟΝ:
ΗΛΙΑ ΑΘΑΝΑΣΟΠΟΥΛΟ

Ηλίας Αθανασόπουλος

Προφίλ του εκπαιδευτή

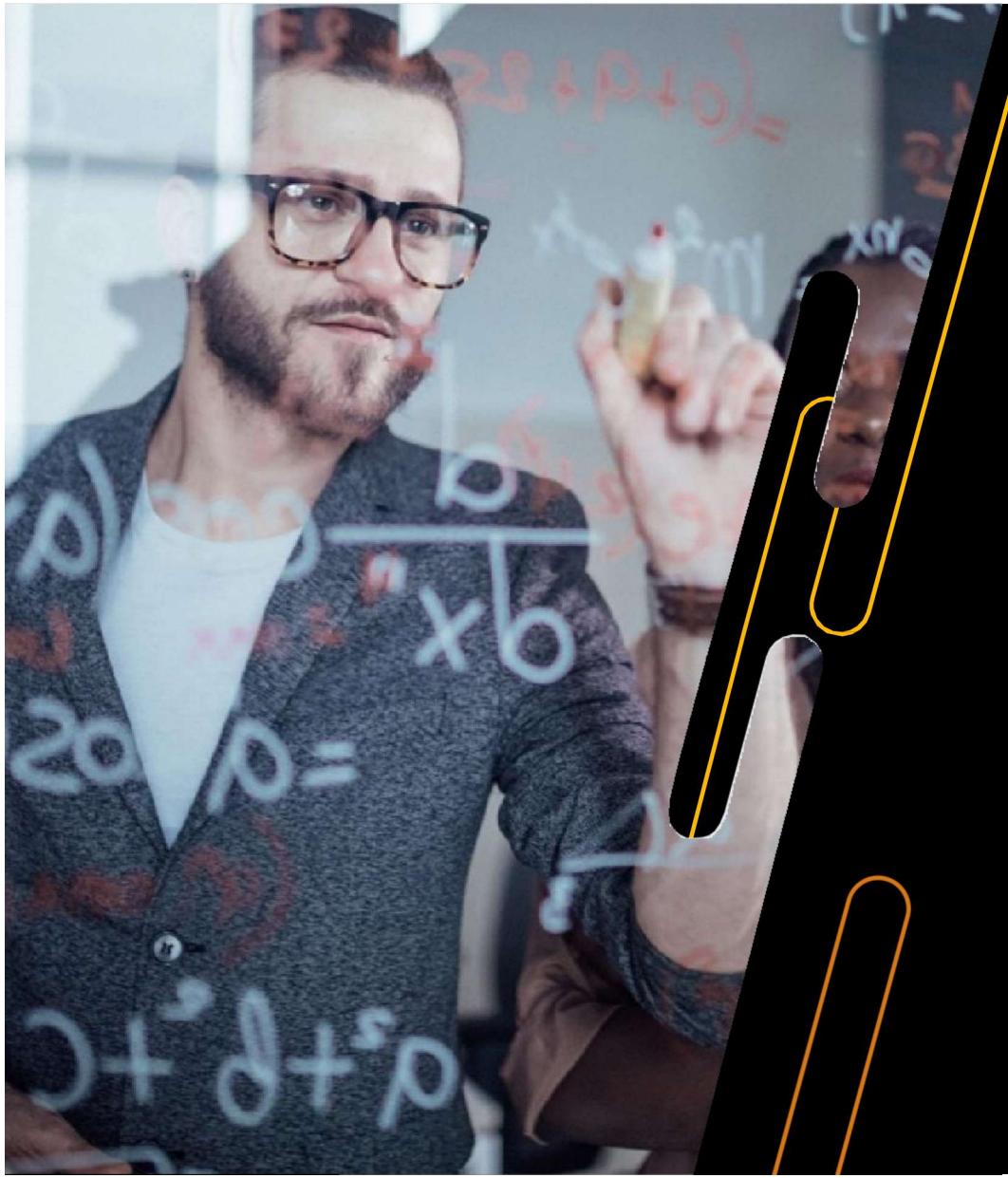
- Αναπληρωτής Καθηγητής, Πανεπιστήμιο Κύπρου
- Έρευνα στην ασφάλεια συστημάτων και την προστασία της ιδιωτικώτητας
- https://elathan.github.io/srec/



Θέμα-1: Εισαγωγή στις ευπάθειες λογισμικού

Θα καλύψουμε αυτές τις δεξιότητες

- Πώς αναπτύσσεται το λογισμικό;
- Ποια είναι η διαφορά μεταξύ των συστημάτων προγραμματισμού με ασφαλή και μη ασφαλή μνήμη;
- Πώς μοιάζουν τα τρωτά σημεία ασφαλείας μνήμης;



Θέμα-2: Εκμετάλλευση ευπάθειας Ευπάθειες στο εγγενές λογισμικό

Θα καλύψουμε αυτές τις δεξιότητες

- Οι μηχανισμοί επίθεσης ροής ελέγχου
- Πώς μπορεί κάποιος να εισάγει νέο κώδικα προγραμματισμού σε ένα ευπαθές προγραμματίζοντας
- Πώς λειτουργεί το λογισμικό ευπάθειας

Πρακτικές ασκήσεις κατάρτισης

Πρακτικές ασκήσεις που απαιτούν τόσο προσωπική όσο και ομαδική προσπάθεια

	Τίτλος	Στόχος της άσκησης
Άσκηση-1 (Ημέρα-1)	Επίθεση ροής ελέγχου	Παραβίαση της ροής ελέγχου ενός προγράμματος με ευπάθεια
Άσκηση-2 (Ημέρα-1)	Έγχυση κώδικα προγραμματισμού	Έγχυση κώδικα σε ένα πρόγραμμα με ευπάθειες
Άσκηση-3 (Ημέρα-1)	Επίθεση ROP	Εκθέτω ένα πρόγραμμα με ευπάθειες

Μέθοδος αξιολόγησης

Περιγράψτε τα στοιχεία αξιολόγησης και τη διαδικασία αξιολόγησης

Στοιχείο αξιολόγησης	Πώς	Σημειώσεις
Εφαρμοσμένες εργασίες (ατομικές)	Η λύση του προβλήματος θα υποβληθεί αργότερα	
Ομαδική εργασία	Ομαδική εργασία που θα υποβληθεί αργότερα	
Ομαδική συζήτηση	Κατά τη διάρκεια του εργαστηρίου	

Προηγούμενη γνώση και προαπαιτούμενα

Γνώσεις υποβάθρου:

Βασική κατανόηση του προγραμματισμού σε C

Βασική κατανόηση των λειτουργικών συστημάτων

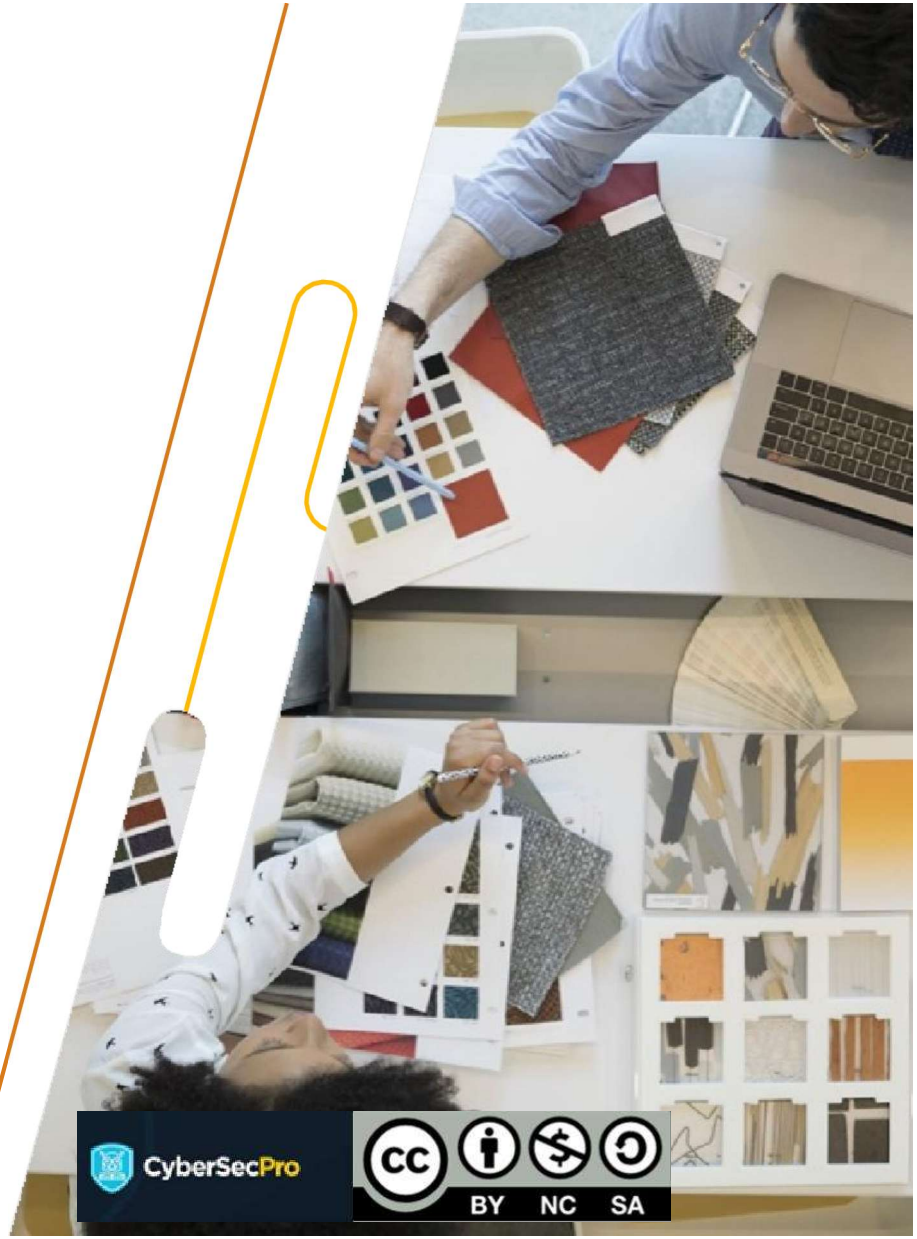
Προαπαιτούμενα:

Κανένα/δεν ορίζεται

Τεχνικά εργαλεία και άλλες απαιτήσεις

Τεχνικά εργαλεία

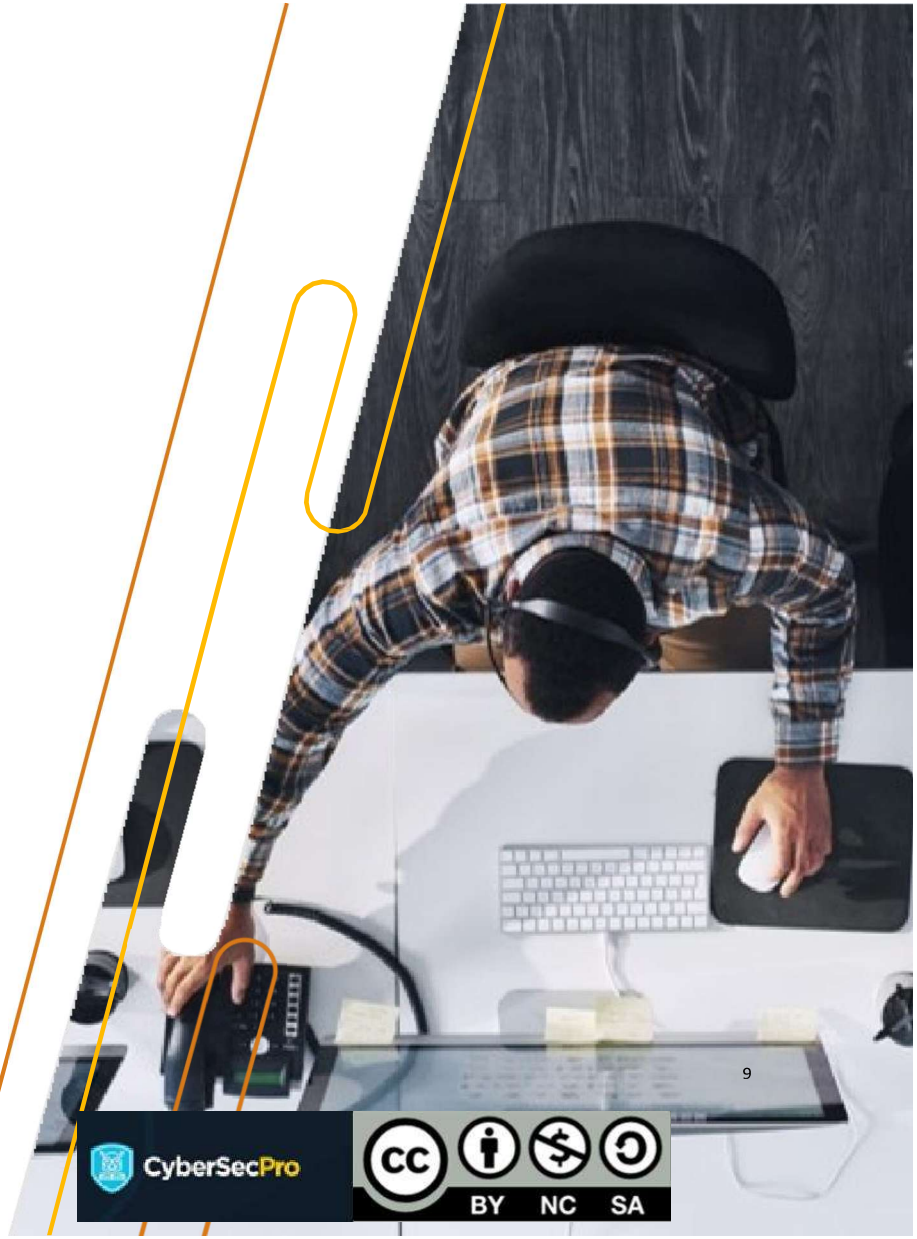
Λειτουργικό σύστημα βασισμένο σε Linux με εργαλειοθήκη μεταγλωττιστή C



Εγγραφή: Πώς να εγγραφείτε και άλλες πρακτικές πληροφορίες

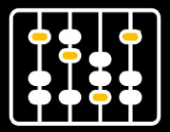
Η συγκεκριμένη διαδικασία εγγραφής για την κατάρτιση Cybersecurity Essentials and Management μπορεί να διαφέρει ανάλογα με τον πάροχο κατάρτισης ή το θεσμικό όργανο. Ωστόσο, τα γενικά βήματα είναι συνήθως απλά και μπορούν να ολοκληρωθούν διαδικτυακά ή αυτοπροσώπως.

1. Διαδικτυακή εγγραφή
2. Προσωπική εγγραφή
3. Πρόσθετες πρακτικές πληροφορίες



Επιθέσεις διαφθοράς μνήμης

Ασφάλεια μνήμης



Συστήματα προγραμματισμού ασφαλή έναντι μη ασφαλούς μνήμης

Επιθέσεις ροής ελέγχου

Πώς μπορεί να παραβιαστεί η ασφάλεια μνήμης

Σύγχρονες επιθέσεις

Πώς λειτουργούν οι σύγχρονες επιθέσεις;

Πρόοδος μαθημάτων

01. Συστήματα προγραμματισμού με ασφάλεια μνήμης

02. Ευπάθειες

03. Επιθέσεις ροής ελέγχου

04. Έγχυση Κώδικα

05. Προγραμματισμός προσανατολισμένος στην επιστροφή ROP)

Λογισμικό

Τα προγραμματισμένα αντικείμενα εγγράφονται σε γλώσσα υψηλού επιπέδου

- C, C++, Java, C#Ruby, Python

Μεταγλωττίζονται για εκτέλεση

- Κώδικας Μηχανής(μη διαχειριζόμενος και μη ασφαλής κώδικας)
- Κώδικας εικονικής μηχανής, όπως η JVM (διαχειρίσιμος και ασφαλής)

Διαφορετικές αρχιτεκτονικές παρουσιάζουν διαφορετικές ιδιότητες κατά την εκτέλεση λογισμικού

- Ορισμένες γενικές έννοιες ισχύουν για όλους

Ασφαλή έναντι μη ασφαλών συστημάτων

Ασφαλή συστήματα προγραμματισμού

- Εκτέλεση κώδικα προγραμματισμού σε εικονική μηχανή (π.χ. Java)
- Αποδίδει στατική ανάλυση και κατά το χρόνο μεταγλώττισης (απόρριψη μη ασφαλούς κώδικα προγραμματισμού) και χρησιμοποιεί ελέγχους κατά το χρόνο εκτέλεσης (π.χ. Rust)
- Περιορισμένη πρόσβαση στη μνήμη

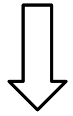
Μη ασφαλή συστήματα προγραμματισμού

- Απεριόριστη πρόσβαση στη μνήμη π.χ. C/C++)
- Καλύτερη επίδοση
- Πρόσβαση χαμηλού επιπέδου(π.χ. οδηγοί προγραμμάτων)
- Παρωχημένο σύστημα κώδικα προγραμματισμού

Μη ασφαλή vs μη ασφαλή συστήματα προγραμματισμού

Μη ασφαλής π.χ. C/C++)

```
a[i] = j;
```



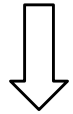
Κώδικας μηχανής

`a` είναι απλώς αρχική διεύθυνση

Δεν υπάρχει τρόπος να ελέγξουμε αν το "`i`" βρίσκεται στα όρια του "`a`"

Μη ασφαλής π.χ. C/C++)

```
a[i] = j;
```



Κώδικας προγραμματισμού εικονικής μηχανής

Το " `a` " είναι μια καλά καθορισμένη οντότητα.

Η VM γνωρίζει τον τύπο, τα όρια,τον αριθμό αναφορών του a, κ.λπ.

Πρόοδος μαθημάτων

01. Συστήματα προγραμματισμού με ασφάλεια μνήμης

02. Ευπάθειες

03. Επιθέσεις ροής ελέγχου

04. Έγχυση Κώδικα

05. Προγραμματισμός προσανατολισμένος στην επιστροφή ROP)

Παράδειγμα προγραμματισμού

```
#include stdio.h>

int main(int argc, char *argv[{

        int ary[5] = {1, 2, 3, 4, 5};
        fprintf(stderr, "The fifth number of ary is: %d\n", ary[5]);
        return 1;
}
```

Σύντομη περιγραφή κώδικα

Πρόκειται για ένα απλό πρόγραμμα C που:

- δηλώνει έναν πίνακα ary πέντε ακεραίων {1, 2, 3, 4, 5}
- επιχειρεί να τυπώσει στο stderr το στοιχείο ary[5] (έκτο — εκτός ορίων, άρα προκαλεί σφάλμα μνήμης / απροσδιόριστη συμπεριφορά)
- επιστρέφει 1

Παράλληλα, ο κώδικας έχει συντακτικά λάθη (λανθασμένο #include, ασύμμετρες αγκύλες στη δήλωση της main).

Πρόσβαση εκτός ορίων!

```
#include stdio.h>

int main(int argc, char *argv[{

        int ary[5] = {1, 2, 3, 4, 5};
        fprintf(stderr, "The fifth number of ary is: %d\n", ary[5]);
        return 1;
}
```

Σύντομη περιγραφή:

- Δημιουργεί πίνακα **ary[5]** και τον αρχικοποιεί με {1, 2, 3, 4, 5}.
- Χρησιμοποιεί fprintf για να τυπώσει στο **stderr** το στοιχείο **ary[5]**.
  - **Προσοχή:** τα έγκυρα indices είναι 0–4, οπότε το ary[5] είναι εκτός ορίων → απροσδιόριστη συμπεριφορά (μπορεί να εμφανιστεί τυχαία τιμή ή σφάλμα).
- Τελειώνει επιστρέφοντας **1** (μη μηδενικός κώδικας εξόδου → συνήθως δηλώνει αποτυχία).

Λεξιλόγιο

Ευπάθεια

- Ένα σφάλμα λογισμικού (επίσης γνωστό ως bug) που μπορεί δυνητικά να επιτρέψει σε κάποιον να εκμεταλλευτεί το ευπαθές προγραμματίζοντας.

Εκμετάλλευση ευπάθειας

- Η διαδικασία ελέγχου ενός προγραμματιστή με την εκμετάλλευση ενός ή περισσότερων Ευπαθειών
- Δεν μπορούν να εκμεταλλευτούν όλες οι ευπάθειες

Λεξιλόγιο

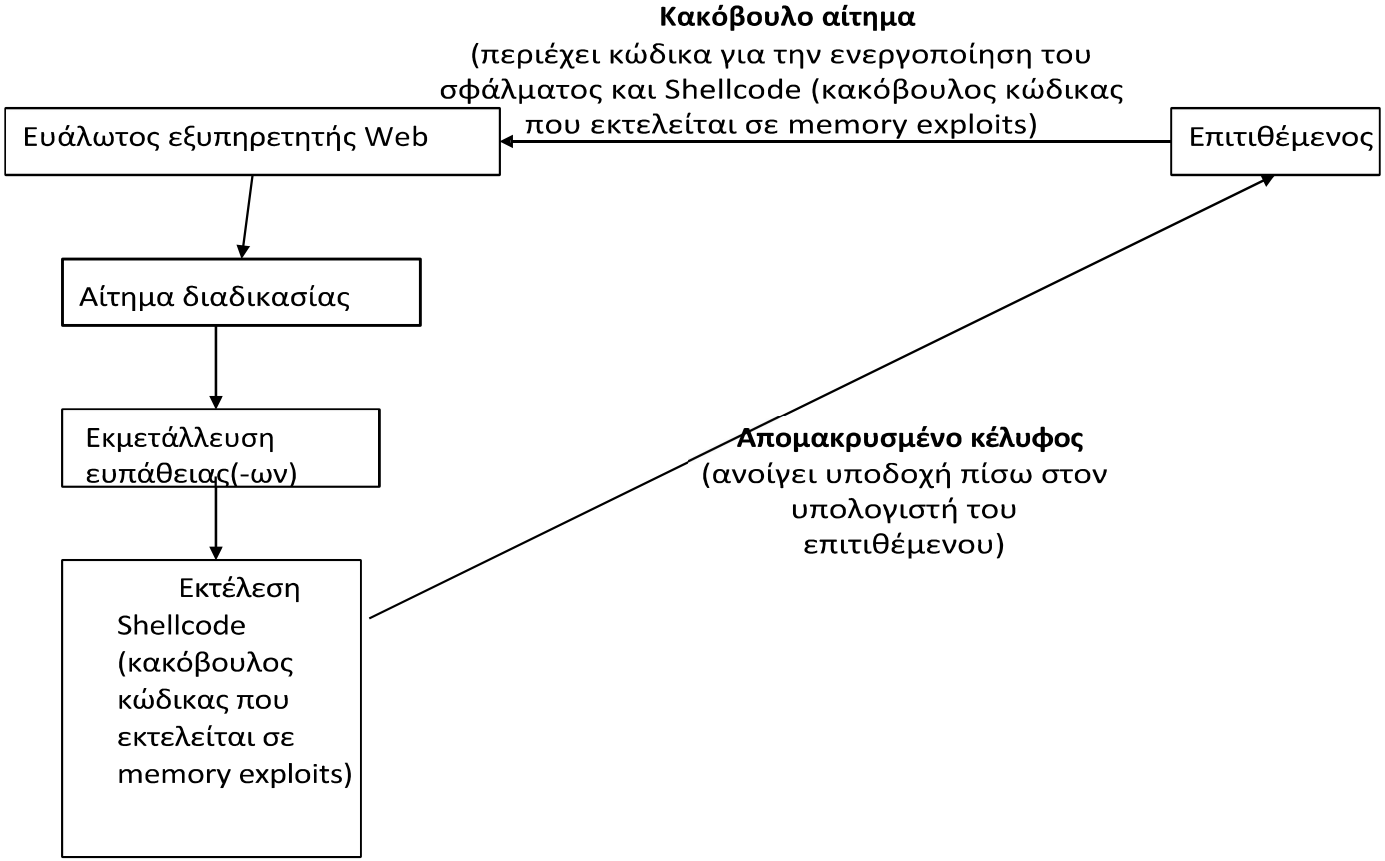
Εκτέλεση αυθαίρετου κώδικα προγραμματισμού

- Η κατάσταση μιας εκμετάλλευσης ευπάθειας όπου ο επιτιθέμενος μπορεί να εκτελέσει ένα προγραμματίζοντάς το

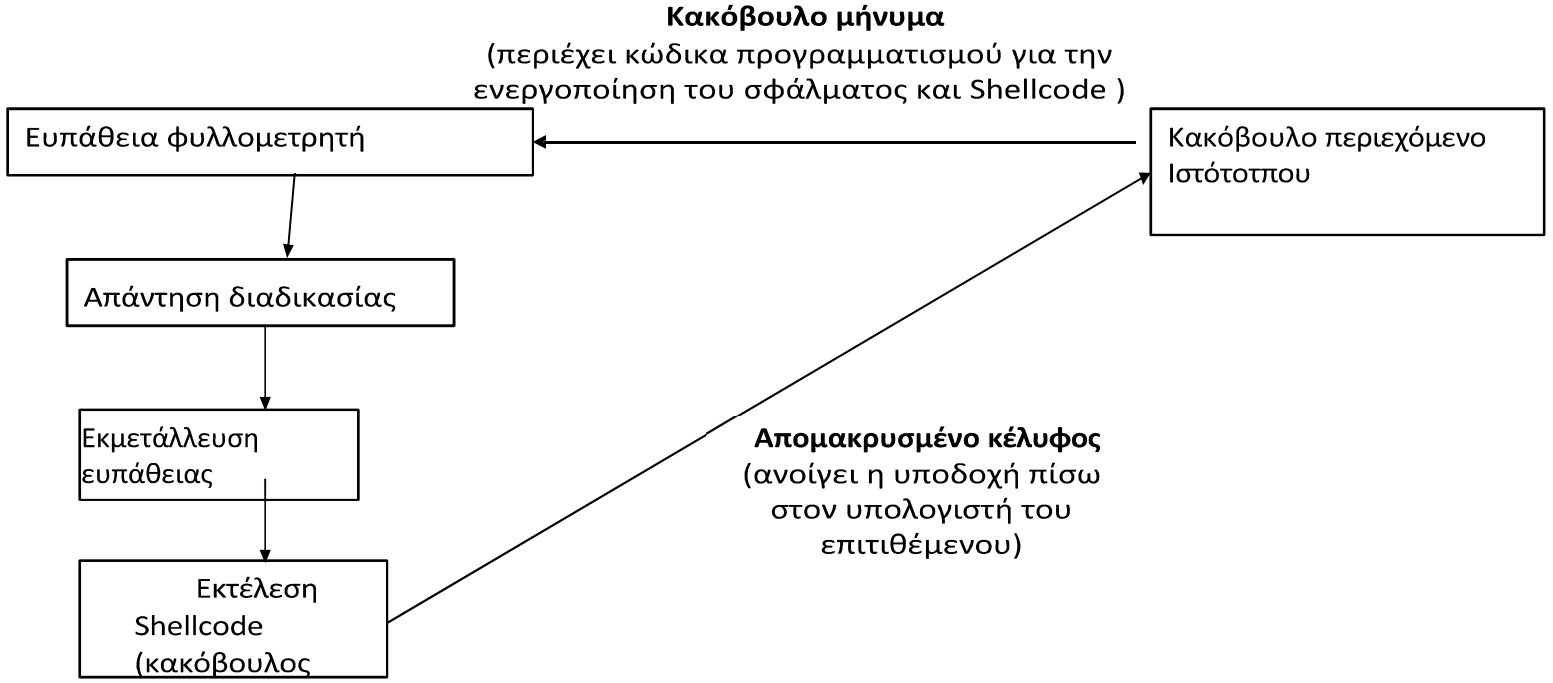
Shellcode (κακόβουλος κώδικας που εκτελείται σε memory exploits)

- Ένας κώδικας μηχανής που εκτελεί ένα ευπαθές πρόγραμμα και εξυπηρετεί τους σκοπούς του επιτιθέμενου.
- Δημιουργία κελύφους(μπορεί να είναι απομακρυσμένο), λήψη κακόβουλου λογισμικού, δημιουργία κρυφού λογαριασμού χειραγώγηση λογισμικού κ.λπ.
- Σε μεγάλο βαθμό εξαρτώμενη από την αρχιτεκτονική

Υψηλού επιπέδου



Υψηλού επιπέδου



Πρόοδος μαθημάτων

01. Συστήματα προγραμματισμού με ασφάλεια μνήμης

02. Ευπάθειες

03. Επιθέσεις ροής ελέγχου

04. Έγχυση Κώδικα

05. Προγραμματισμός προσανατολισμένος στην επιστροφή (ROP)

Συναρτήσεις(Functions)

Το λογισμικό αποτελείται από διάφορες λειτουργίες

- `main(), printf(), malloc(), create_user(), κ.λπ.`

Οι συναρτήσεις επιτρέπουν την επαναχρησιμοποίηση του κώδικα προγραμματισμού

- Κάθε φορά που θέλετε να εμφανίσετε μήνυμα, απλά καλείτε `την printf`

Σε ένα πρόγραμμα μια συνάρτηση μπορεί να καλέσει μια συνάρτηση, και στη συνέχεια μια άλλη συνάρτηση

- Όλη αυτή η αλυσίδα συναρτήσεων ονομάζεται **ροή ελέγχου**

Η ζωή μιας συνάρτησης

Κάθε φορά που καλείται μια συνάρτηση, η ροή ελέγχου του προγράμματος αλλάζει

- Πρέπει να το κάνουμε αυτό με διαφάνεια
- Μόλις ολοκληρωθεί η λειτουργία, η ροή ελέγχου θα πρέπει να συνεχιστεί.

Οι συναρτήσεις μπορούν να λαμβάνουν

ορίσματα

Οι συναρτήσεις μπορούν να επιστρέφουν

δεδομένα

Οι συναρτήσεις μπορούν να δημιουργούν

τοπικά δεδομένα

Λεξιλόγιο

Όταν καλείται μια συνάρτηση `foo`

- Η συνάρτηση "foo" είναι ο καλούμενος
- Η διεύθυνση που κάλεσε τη συνάρτηση καλείται **call site(** ή caller)

Η στοίβα

Οι συναρτήσεις χρειάζονται μνήμη για τις εργασίες τους

- Αυτή είναι η στοίβα

Αυτή η μνήμη είναι για δεδομένα μικρής διάρκειας

- Μόλις ολοκληρωθεί η συνάρτηση μπορούμε να απαλλαγούμε από τα δεδομένα που εμπλέκονται

Εξαρτάται από την αρχιτεκτονική

- Η κύρια ιδέα δεν αλλάζει

Η στοίβα μπορεί να περιέχει διάφορα πράγματα

- Ορίσματα της συνάρτησης, η διεύθυνση επιστροφής, ο παλιός δείκτης πλαισίου, τοπικά ορίσματα

Στοίβα της Intel (32-bit)

Η στοίβα αναπτύσσεται από διευθύνσεις ανώτερης μνήμης σε διευθύνσεις χαμηλότερης μνήμης

- Είναι σαν η στοίβα να είναι αναποδογυρισμένη.

Η κορυφή της στοίβας διατηρείται πάντα σε έναν καταχωρητή υλικού (%esp)

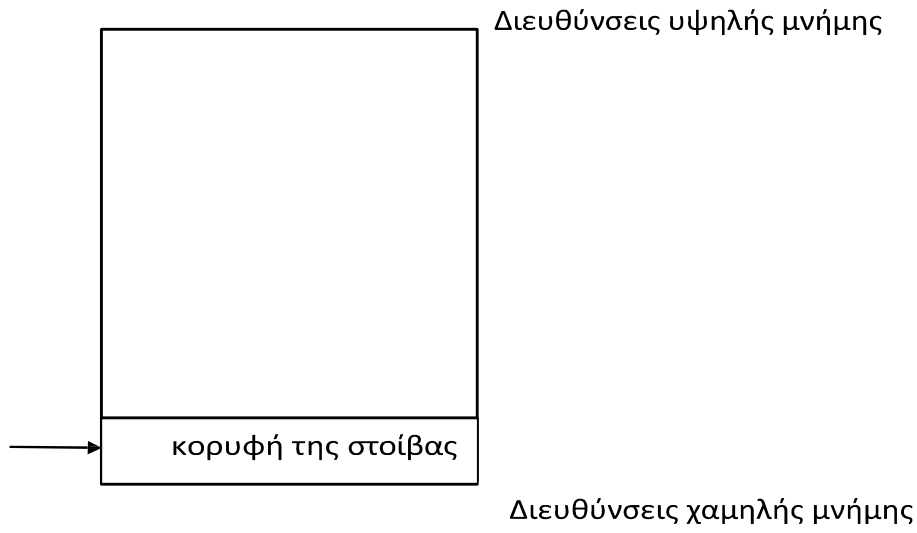
Κάθε συνάρτηση δημιουργεί ένα νέο **πλαίσιο** στοίβας **κατά** την εκτέλεση

- Ένα εικονικό τμήμα μέσα στη στοίβα
- Το πλαίσιο στοίβας καταστρέφεται μόλις ολοκληρωθεί η λειτουργία

Η κορυφή του πλαισίου στοίβας διατηρείται σε καταχωρητή υλικού (%ebp)

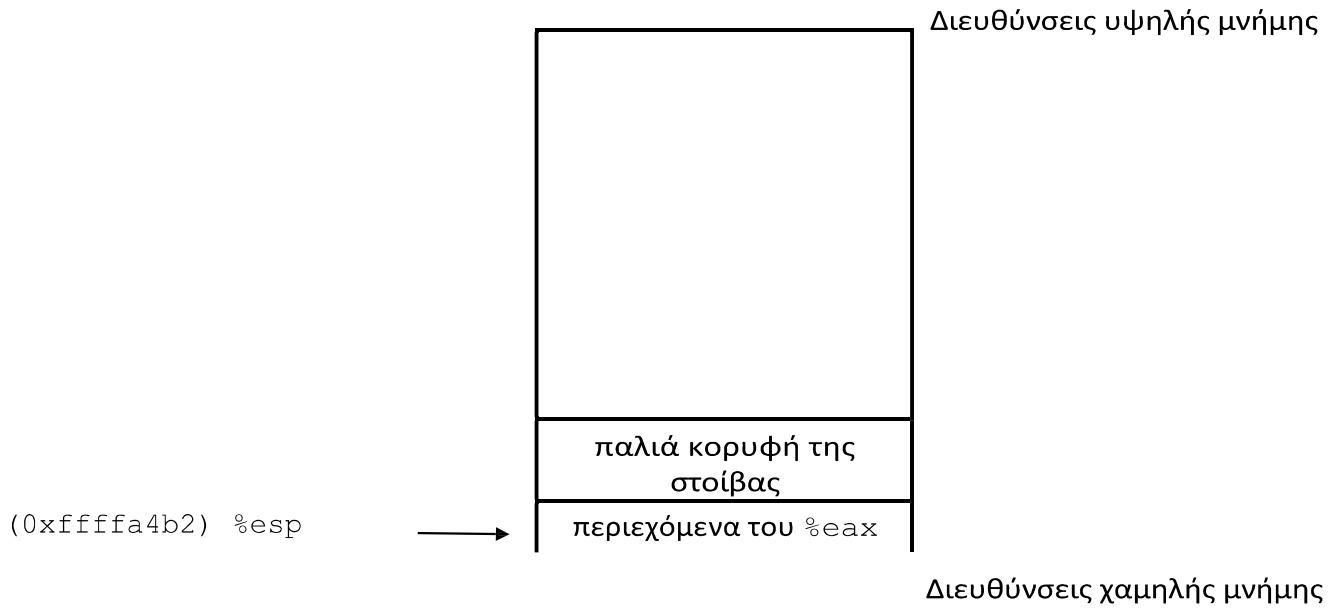
Εισαγωγή στοίβας

```
push %eax
sub 0x4, %esp
 mov %eax, (%esp)
```



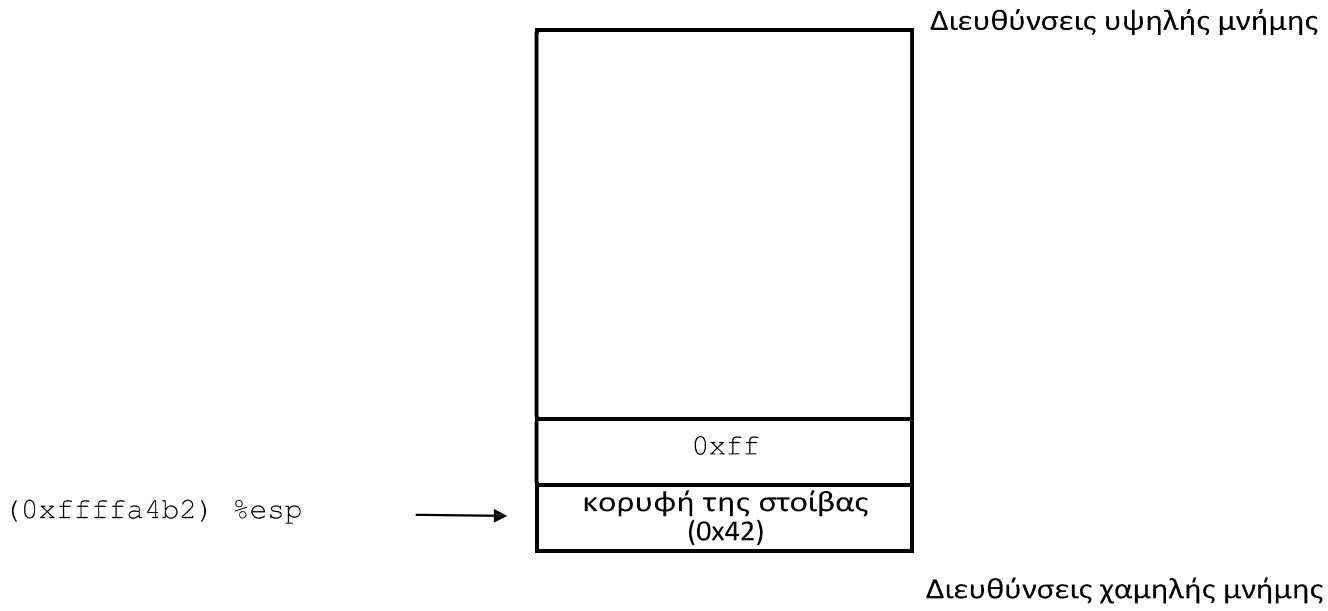
Εισαγωγή στοίβας

```
push %eax
sub 0x4, %esp
mov %eax, (%esp)
```



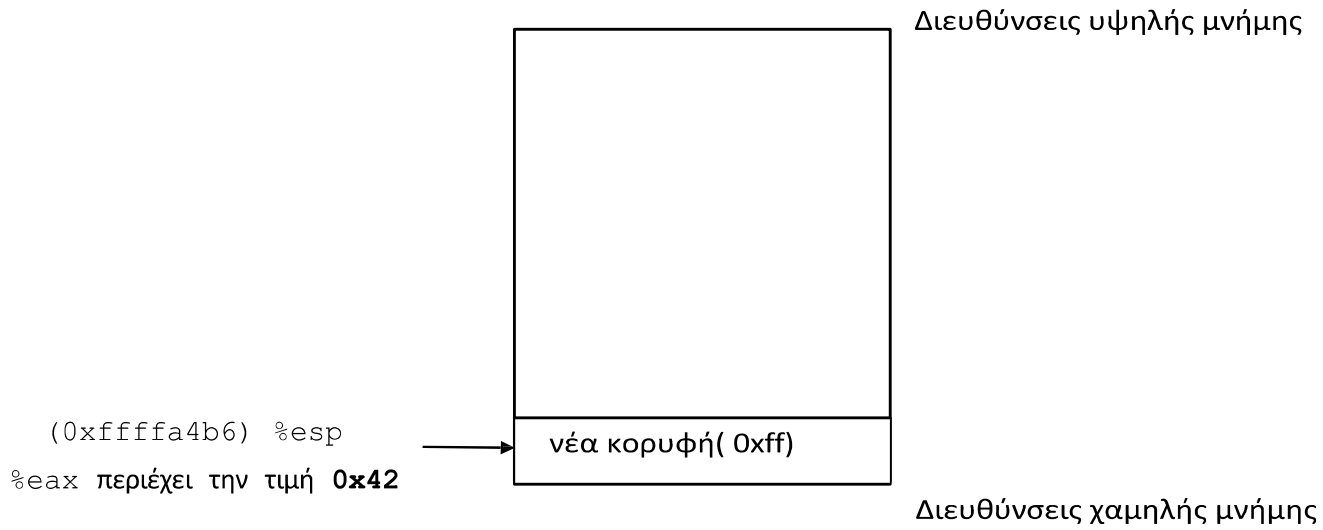
Διαγραφή στοίβας

```
pop %eax
mov %(esp), %eax add 0x4,
%esp
```

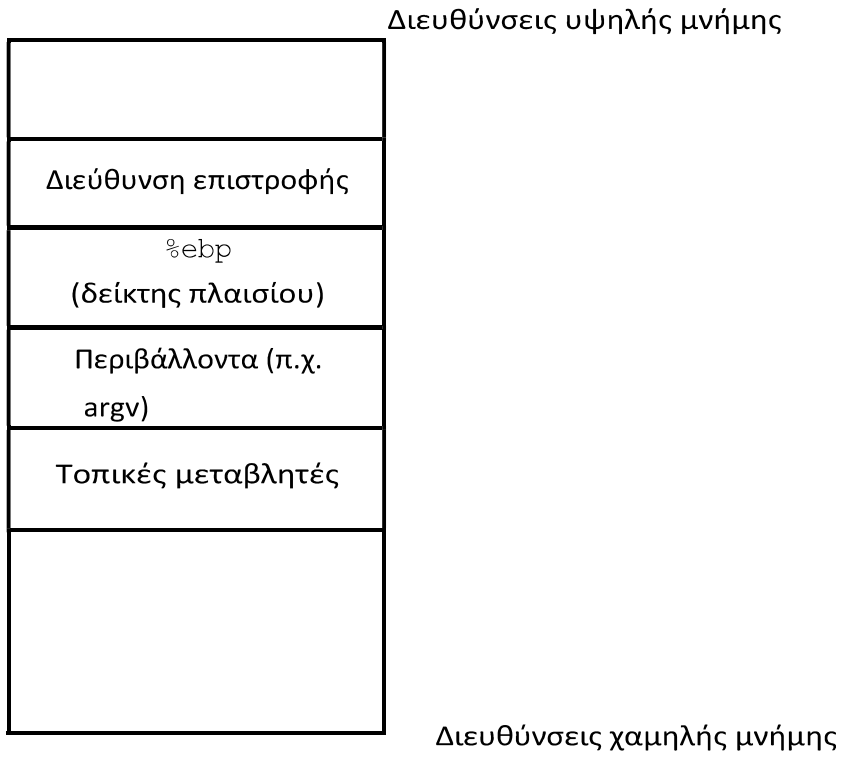


Διαγραφή στοίβας

```
pop %eax
mov %(esp), %eax
add 0x4, %esp
```



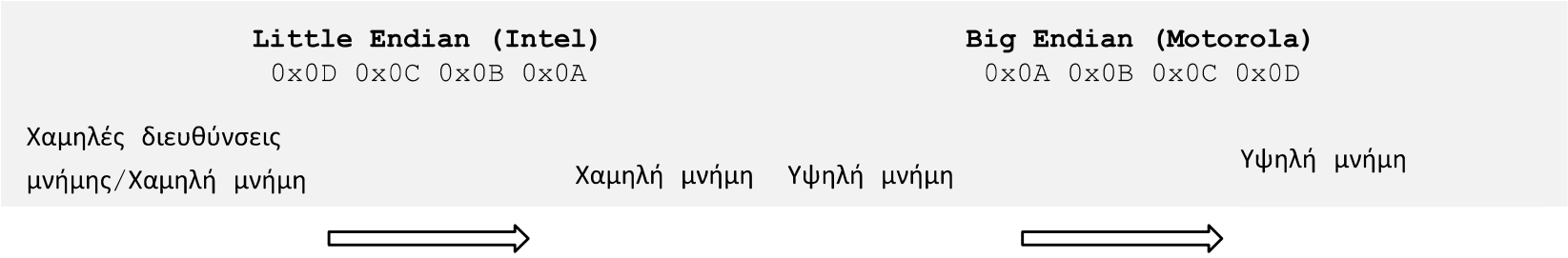
Πλαίσιο στοίβας σε Intel 32-bit



Σειριακή διάταξη bytes(Endianness)

Υποθέστε τη λέξη 32 δυαδικά ψηφία: `0x0A0B0C0D`

Δύο πιθανοί τρόποι αποθήκευσης στη μνήμη



Πρόοδος μαθημάτων

01. Συστήματα προγραμματισμού με ασφάλεια μνήμης

02. Ευπάθειες

03. Επιθέσεις ροής ελέγχου

04. Έγχυση Κώδικα

05. Προγραμματισμός προσανατολισμένος στην επιστροφή (ROP)

Shellcode (κακόβουλος κώδικας που εκτελείται σε memory exploits)

Εκτελέσιμος κώδικας ενσωματωμένος στα δεδομένα εισόδου του χρήστη (κώδικας παρεισφρημένος στην είσοδο χρήστη)

- δημιουργεί ένα (απομακρυσμένο) κέλυφος, κατεβάζει κακόβουλο λογισμικό, δημιουργεί έναν χρήστη, Αποκτά δικαιώματα (γίνεται root), εγκαθιστά κρυφή πρόσβαση σε σύστημα κ.λπ.

Το πρόγραμμα είναι συνήθως μικρό και περιεκτικό

- Ειδικά σε υπερχειλίσεις buffer, τα ευάλωτα buffers συνήθως διαθέτουν περιορισμένο μέγεθος.
- Δεν επιτρέπονται \0s, αλλιώς τα αντίγραφα συμβολοσειράς μπορούν να καταστρέψουν τον Shellcode

Ισχυρή εξάρτηση από τη συγκεκριμένη αρχιτεκτονική
Πολύ ανορθόδοξες τεχνικές προγραμματισμού

- Προσαρμοσμένος κώδικας assembly

Εκκίνηση κελύφους(shell)

```
int execve(const char ,
                char *const argv[],
                char *const envp[]),
```

Παράδειγμα:

- `execve("/bin/sh", NULL, NULL);`
- Θυμηθείτε ,εδώ δεν κάνουμε «σωστό» (κανονικό) προγραμματισμό!

Κλήσεις συστήματος στο Linux

Μπορεί να κληθεί μέσω λογισμικής διακοπής (software interrupt)

- Εντολή assembly: int
- Παράδειγμα: για το execve ο αριθμός διακοπής είναι **0x0b** (ή 11 δεκαδικό)
- Οι παράμετροι περνούν σε καταχωρητές

Το OS (Operating System - λειτουργικό σύστημα) διαθέτει πίνακα κλήσεων συστήματος

- Κάθε κλήση συστήματος καλεί τον κατάλληλο κώδικα προγραμματισμού(ενεργοποιεί τον αντίστοιχο χειριστή)
- `/usr/include/asm/unistd_32.h`:

```
#define NR_execve 11
```

`execve` στο Linux/IA32

- `execve("/bin/sh", NULL, NULL),`

`%eax`: επιστροφή τιμής

`%ebx`

- πρώτο όρισμα, διεύθυνση της μνήμης στην οποία είναι το "/bin/sh" αποθηκευμένο

`%edX` (**πλατφόρμα διαδικτυακών μαθημάτων**)

- 2ο και 3ο όρισμα
- Μπορούμε απλά να τις μηδενίσουμε

Παράκαμψη ασφαλείας

Το "/bin/sh" είναι 7 bytes, θα ήταν καλό να ήταν 8 bytes

Εύκολη (αλλά) πρόχειρη λύση

- /bin//sh

Όχι μηδενικά

- `H strcpy` μπορεί να διαχωρίσει το Shellcode (κώδικας κελύφους) αν περιέχονται μηδενικά
- Εάν πρέπει να μηδενίσουμε έναν καταχωρητή χρησιμοποιούμε **το XOR (Exclusive OR – δυαδική λογική πράξη)**

Push **/bin//sh**

char	h	s	/	/	n	i	b	/
ASCI (hex)	0x68	0x73	0x2f	0x2f	0x6e	0x69	0x62	0x2f
valu e	**0x68732f2f**				**0x6e69622f**			

Η διαφάνεια απεικονίζει πώς «σπάμε» τη συμβολοσειρά
/bin//sh
σε δύο 32-μπιτες λέξεις, έτσι ώστε να τις σπρώξουμε (push) μία-μία στη στοίβα μέσα σε shellcode για execve("/bin//sh", ...):

```
char   h      s      /      /      n      i      b      /
ASCII (hex)   0x68   0x73   0x2f   0x2f   0x6e   0x69   0x62   0x2f
32-bit λέξη   0x68732f2f    0x6e69622f
0x68732f2f ⇒ τα bytes / / s h

0x6e69622f ⇒ τα bytes / b i n
```

Σε έναν little-endian επεξεργαστή (π.χ. x86) η εντολή

```
push 0x68732f2f
push 0x6e69622f
```

τοποθετεί τελικά στη μνήμη, από χαμηλότερη προς υψηλότερη διεύθυνση:

```
/ b i n / / s h \0
```

δηλαδή ακριβώς τη μηδενικά-τερματισμένη συμβολοσειρά που θα περαστεί στο σύστημα με την κλήση execve.

Shellcode (κακόβουλος κώδικας που εκτελείται σε memory exploits)

`execve("/bin/sh"),`

```
.section .text
        .globl  _start
_start:
        /* 1. Εκκαθάριση καταχωρητών */
        xor     %eax, %eax          /* EAX = 0  ---------------------- */

        /* 2. Ωθούμε τη συμβολοσειρά "/bin//sh\0" στη στοίβα         */
        push    %eax                /* NUL τερματισμός                 */
        push    $0x68732f2f         /* "hs//" (λόγω little-endian)     */
        push    $0x6e69622f         /* "/bin"                          */

        /* 3. EBX -> δείχτης στη συμβολοσειρά (argv[0])              */
        mov     %esp, %ebx

        /* 4. Δημιουργούμε τον πίνακα δεικτών argv                   */
        push    %eax                /* argv[1] = NULL                  */
        push    %ebx                /* argv[0] = "/bin//sh"            */
        mov     %esp, %ecx          /* ECX -> argv                     */

        /* 5. EDX = NULL (envp)  -- ήδη 0 από το xor                 */
        /*    EAX = 11 (syscall number για execve)                   */
        mov     $0x0b, %al

        /* 6. System call                                             */
        int     $0x80               /* execve("/bin//sh", argv, NULL) */
```

Πρόοδος μαθημάτων

01. Συστήματα προγραμματισμού με ασφάλεια μνήμης

02. Ευπάθειες

03. Επιθέσεις ροής ελέγχου

04. Έγχυση Κώδικα

05. Προγραμματισμός προσανατολισμένος στην επιστροφή (ROP)

Μη εκτελέσιμη μνήμη

Διάφορα ονομασμένα

- NX-bit (No-eXecute Bit) - μηχανισμός προστασίας από buffer overflow επιθέσεις)
- DEP (Data Execution Prevention) (Πρόληψη εκτέλεσης δεδομένων)
- W^X (Γράψε XOR Εκτέλεσε)

Επιβάλλεται από το υλικό (MMU)

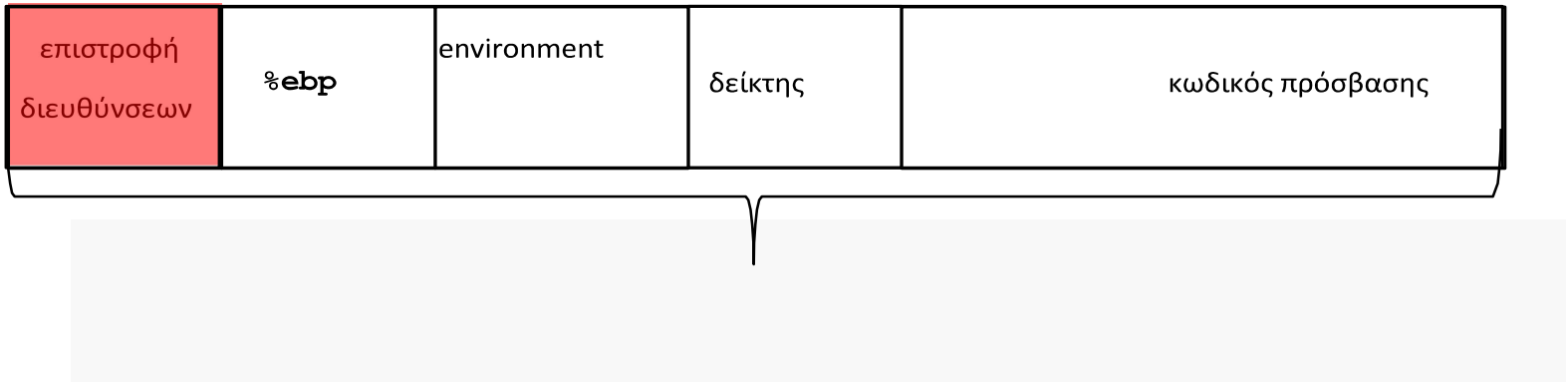
Οι σελίδες μνήμης δεν μπορούν να είναι εκτελέσιμες και εγγράψιμες

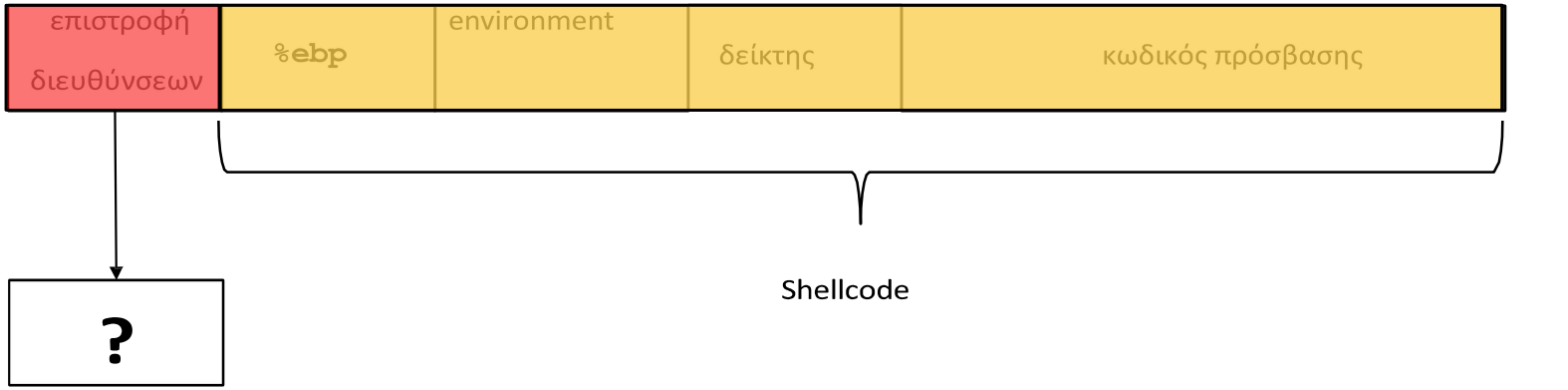
- Η στοίβα και ο σωρός είναι εγγράψιμα αλλά όχι εκτελέσιμα
- Ο κώδικας είναι εκτελέσιμος αλλά όχι εγγράψιμος

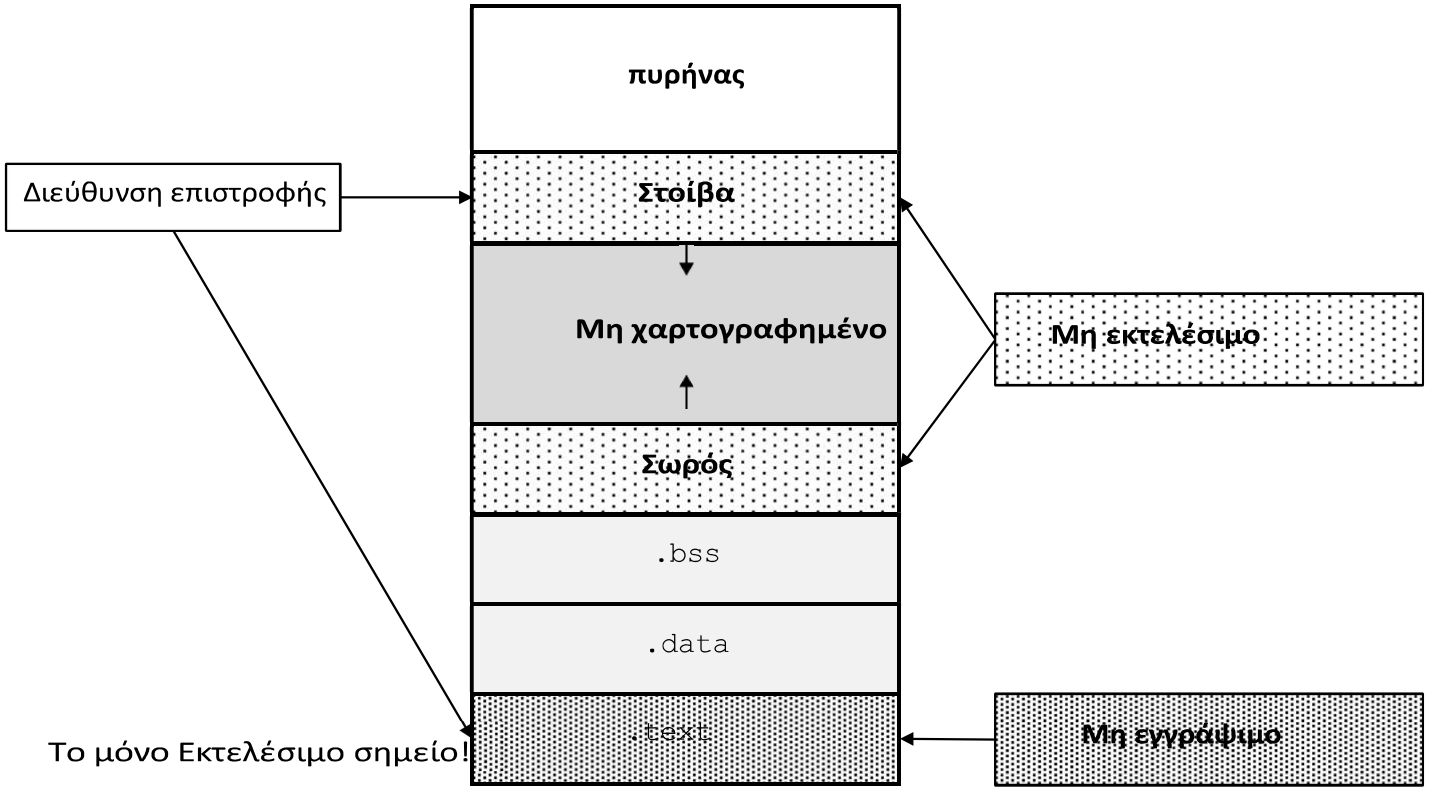
Τα δικαιώματα μπορούν να αλλάξουν χρησιμοποιώντας κλήσεις συστήματος

- `mprotect()` για Linux
- `VirtualProtect()` για Windows

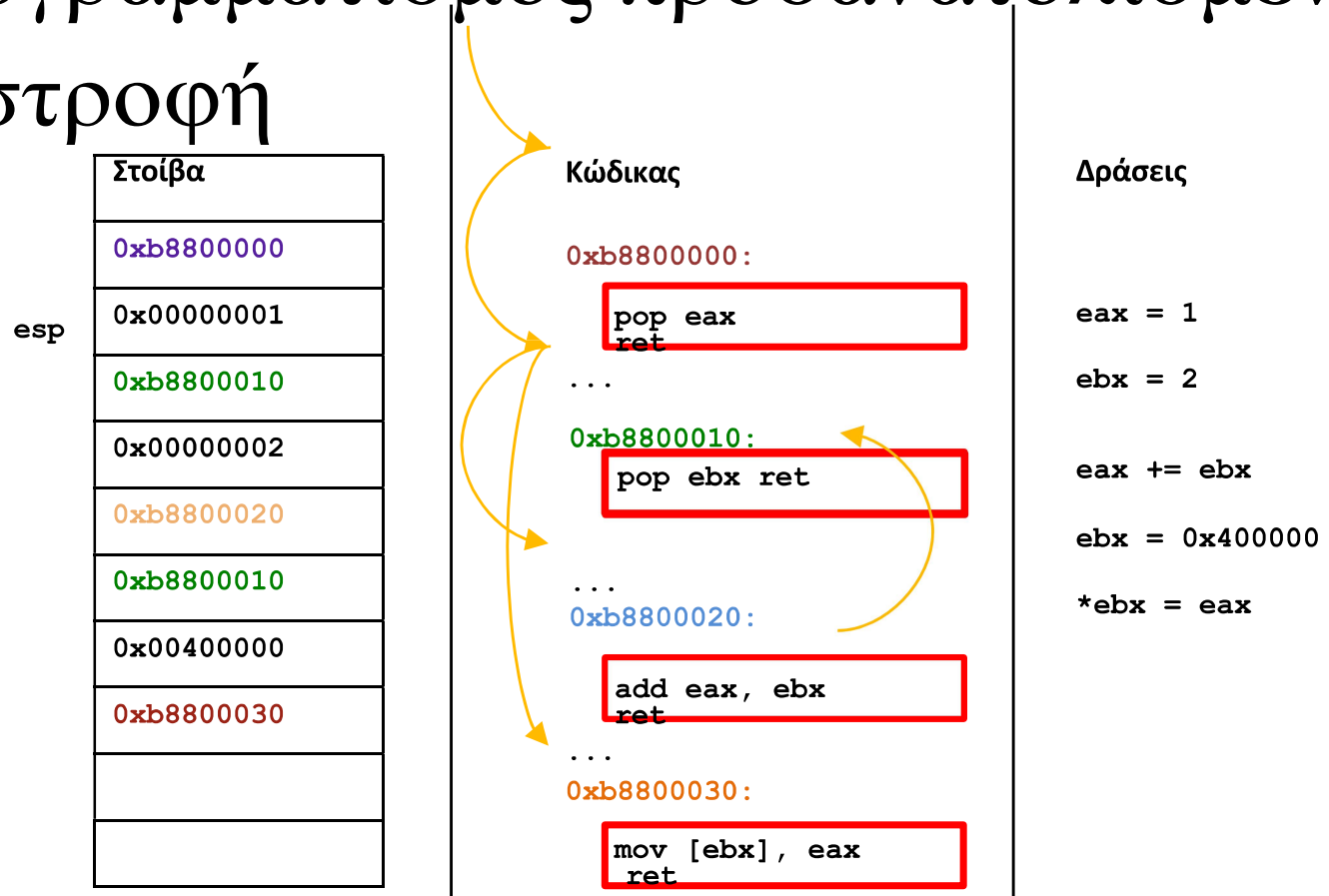
Πού πρέπει να μεταβούμε;






πυρήνας
Διεύθυνση επιστροφής
Στοίβα
Μη χαρτογραφημένο
Μη εκτελέσιμο
Σωρός
.bss
.data
.text
Μη εγγράψιμο
Το μόνο Εκτελέσιμο σημείο!

Προγραμματισμός προσανατολισμένος στην επιστροφή



Διαφάνεια από το http://www.ieee-security.org/TC/SP2012/slides/Smashing%20the%20Gadgets.pptx

Επαναχρησιμοποίηση κώδικα προγραμματισμού

Τα προγράμματα διαθέτουν μεγάλα κομμάτια έτοιμου κώδικα

- αυτά είναι διαθέσιμα στον επιτιθέμενο την ώρα της εκμετάλλευσης.

Μικρές ακολουθίες εντολών που τελειώνουν με `ret`

- Ονομάζονται gadgets
- Εκτελούν κάποιο κώδικα προγραμματισμού και *επιστρέφουν «retutn»(* δηλαδή, διαβάζουν τη στοίβα για το επόμενο gadget)
- Η στοίβα ελέγχεται από τον επιτιθέμενο (χρήση του `esp ως` μετρητή προγραμματισμού)

Συνδέοντας (chaining) πολλά gadget

- Προγραμματισμός προσανατολισμένος στην επιστροφή (ROP)
- ένα Turing-πλήρες μοντέλο
- Στην πράξη χρησιμοποιείται για να παράγει/κάνει μια περιοχή εκτελέσιμη

Gadgets

Είδαμε gadgets όπως

- `add eax, ebx; ret`
- `mov [ebx], eax; ret`
- `pop eax; ret`
- `...`

Είναι αυτός ο κώδικας προγραμματισμού ρεαλιστικός;

Intel και η αρχιτεκτονική CISC

Πυκνό σύνολο εντολών

- Όλες οι τιμές αντιστοιχούν σε έγκυρη εντολή

Μη ευθυγραμμισμένες (unaligned)

Εντολές μεταβλητού μήκους

- Η μετάβαση στη μέση μιας εντολής δημιουργεί νέες εντολές

Υπεράσπιση του ROP

Η ROP βασίζεται στην ακριβή γνώση της διάταξης του κώδικα προγραμματισμού

- Οι διευθύνσεις κώδικα προγραμματισμού είναι η αρχή των ROP gadgets

Τυχαιοποίηση

- Τυχαία διάταξη διευθύνσεων (ASLR)
  (`/proc/sys/kernel/randomize_va_space`)
- Λεπτόκοκκη τυχαιοποίηση(Fine-grained)

κώδικας ανεξάρτητος θέσης (PIC- Position Independent Code)

Διαρροές πληροφοριών

Σφάλματα που επιτρέπουν να διαβαστεί η διάταξη (layout) μιας διεργασίας.

- Μέχρι τώρα, οι υπερχειλίσεις χρησιμοποιούνταν για να γράψουν πάνω σε δεδομένα ελέγχου

ASLR (Τεχνική ασφάλειας μνήμης)

- Η αποκάλυψη της διεύθυνσης στην οποία αντιστοιχίζεται η διεύθυνση μιας χαρτογραφημένης κοινόχρηστη (δυναμική) βιβλιοθήκη(shared library) είναι αρκετή
- Όλα τα gadgets απλώς μετατοπίζονται (σε ένα νέο offset)

Stack canaries/Καναρινιές τιμές (προστασίας) στοίβας(*Μικρές τιμές-φρουροί τοποθετημένες πριν από τον δείκτη επιστροφής· αν αλλοιωθούν από overflow, το πρόγραμμα το αντιλαμβάνεται και τερματίζει.)*

- Η αποκάλυψη του περιεχομένου της στοίβας είναι αρκετή
- Η καναρίνια τιμή αποθηκεύεται στη στοίβα

Σας ευχαριστώ

Παρακαλούμε στείλτε όλες τις ερωτήσεις στη διεύθυνση:
athanasopoulos.elias@ucy.ac.cy

CyberSecPro