Reaktive Sicherheit

Übungsblatt 9

Balduin Binder [s6babind@uni-bonn.de] Charlotte Mädler [s6chmaed@uni-bonn.de]
Bünyamin Sarikaya [s6busari@uni-bonn.de]

Es fehlt: 3.2 # Aufgabe 1

1.1

Eine Umgebungsvaraible sind dynamisch änderbare Werte, die beeinflussen können wie ein Prozess ausgeführt wird. Hierbei sind sie ein Besatndteil der Umgebung in der der Prozess läuft. Sie beinhalten Pfade zu Programmen/Dateien wie auch ggf. Einstellungen für mehrere Programme.

1.2

Man kann die System.Environment class verwenden.

Die beiden Methoden:

```
var value = System.Environment.GetEnvironmentVariable(variable [, Target])
```

und

```
System.Environment.SetEnvironmentVariable(variable, value [, Target])
```

können dies tun. Hier bei ist der Parameter Target (optional) Machine, Process, or User. Wenn Target weggelassen wird ist es per default der aktuelle Prozess.

Alternativ kann man der main methode den char* envp[] übergeben und mittels while Schleife ausgeben.

```
int main (...,char* envp[]){
int i=0;
while(envp[i]!=NULL]{
printf("[%s]", envp[i]);]}
```

1.3

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[], char *envp[]){
    int i=0;
    while(envp[i] != NULL){
        printf("[%s]\n", envp[i]);
        i++;
    }
}
```

Ausgabe:

1.4

```
#include <unistd.h>
int execve(const char *filename, char *const argv[], char *const envp[]);
```

Execve() führt das Programm aus auf den der Pointer Filename zeigt (const char* filename). argv ist ein array von argument strings die dem neues programm übermittelt werden (char* const argv[]) Per convention beginnt es mit dem Filenamen. envp ist ein array von strings die als Umgebung zum neuen Programm übermittelt werden (char* const envp[]) Im normal Fall haben sie die Form key=value. Bei success, returned execve() nicht und returned -1 bei einem error.

1.5

```c
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv) {
    char* argv[] = {"/bin/sh", "-c", "env", 0}; //argumente
    char* envp[] =  //liste neuer env vars
    {
        "RESI=2020",
        0
    };
    execve(argv[0], &argv[0], envp); //ausführung
    fprintf(stderr, "Fehler!\n"); // fehlermeldung
    return -1; // return value
}
```

1.6

```
REAKTIV=Sicherheit #setzt var für diese eine shell
export REAKTIV="Sicherheit" # setzt variable für alle Prozesse ausgehend von dieser Shell
```

1.7

PATH ist die Systemvariable, die das Betriebssystem verwendet, um über die Befehlszeile oder das Terminalfenster nach erforderlichen ausführbaren Dateien zu suchen. Man muss also nicht den geamten Path kennen/eingeben in der command line. In Linux und Mac OS X enthält PATH meist alle bin und sbin directories die für den akutellen nutzer relevant sind. Hier liegen bei UNIXoiden die executable-Dateien. Die varable wird als eine List spezifiziert, wobei die namen mit : getrennt werden. Generel, hat jeder ausgeführte Prozess oder nutzer session eine eigene PATH Einstellung. Sie kann mit dem Systemutility in der Windows-Systemsteuerung bzw. in der Startdatei der Linux-/Solaris-Shell eingerichtet werden. Bei Rechnerrn die auf Windows/Mac OS X laufen sind änderungen an PATH meist nicht erfoderlich.

1.8

Annahme: der Benutzer des Systems Benutzt regelmäßig das Programm firefox.

Beim Aufruf von einem Programm wird mit der $PATH variable überprüft, ob es diese in einem der Ordner gibt, auf die sie zeigt. Firefox liegt für gewöhnlich in /usr/bin/. Wenn wir als Angreifer jedoch die $PATH Variable auf ~/.config/local/cache/ zeigen lassen (und NUR darauf) und in diesem Ordner ein Programm platzieren, was wir firefox nennen (name der Executable muss gleich sein) so können wir dafür sorgen, dass

der ahnungslose Benutzer unser Programm mit seinen Rechten ausführt, obwohl er nur firefox starten wollte. Jetzt haben wir erreicht, dass ein Benutzer unser Programm mit seinen Systemrechten ausführt, das Programm kann jetzt ganz einfach eine Passwortabfrage starten und uns das Passwort im Klartext übermitteln (z.B. indem es eine Datei anlegt in ~/.config/local/cache/ die das Passwort enhält).
Alternativ wäre der gleiche Angriff auch mit dem /tmp Ordner denkbar (Hierauf haben meistens alle Nutzer eines UNIX systems vollen Zugriff), falls man nur lokalen Zugriff auf das System hat und die $PATH variable anderer Benutzer verändern kann.

Aufgabe 3

3.1

spylib.c

```
#include <stdio.h>
static void wrap_init(void) __attribute__((constructor));
static void destruct(void) __attribute__((destructor));

void puts(void) {
   fprintf(stderr,"puts called\n");
}

static void destruct(void) {
   fprintf(stderr,"[*] spyware started.");
}

static void wrap_init(void) {
   fprintf(stderr,"[*] spyware terminated.");
}
```

2

Aufgabe 4

(1) Eine verbreitungsmöglichkeit ist via Spammails. Der Bot verschickt maßenweise emails, wobei versucht wird den Empfänger dazu zu bringen das angehängt Programm asuzuführen, welches dann den Bot installiert. Hierbei wird im Betreff/in der e-Mail häufig vorgegeben sie sei von einem Freund, der Familie, Arbeitskollegen oder es hat mit aktuellen Anlässen wie Weihnachten zu tun. Dieses Prinzip heißt “Social Engeneering”. Ursprünglich wurde Storm nur so verbreitet. Mittlerweise sind auch häufig links in den Emals die auf eine Seite führen, wo man dann die Malware runterlädt. Eine zweite Methode ist es “exploits” also Schwachstellen vom Browser auszunutzen um so den Bot zu installieren. Wenn der Nutzer nicht von alleine auf den Downlaod Trick reinfällt wird ein sogennanter Drive-by download versucht. Hierbei werden ätere Exploits benutzt in der Hoffnung dass wenigerer Erfahrene Benutzer, die ihre Betriebssysteme nicht regelmäßif aktualisieren den Bugfix noch nicht haben.