Software security - programvarusäkerhet

Software security - programvarusäkerhet

Websäkerhet = saker som körs genom webläsare.

- Säkerhetshål i websidor
- Manipulering av webläsardata
- Skadliga scripts

Programvarusäkerhet = program som körs lokalt eller på en server.

- Lokala säkerhetshål i befintlig programvara
- Malware

Ofta kopplade - malware kan installeras från websidor.

Malware

Nån jävel smusslar in skadlig kod i din dator för att sabba för dig eller stjäla av dig!

Typer av malware (1)

• Adware: Annonser integrerade i programvara (pop-up-annonser eller omdirigering av webläsaren)

• Spyware: Programvara som registrerar din aktivitet, inklusive:

• Keyloggers: Program som registrerar tangenttryckningar

• Spammer programs: Program som används för att skicka stora mängder oönskad E-post

• Trojan: Ett program som verkar användbart men som har dolda potentiellt skadliga egenskaper som undviker säkerhetsmekanismer.

Typer av malware (2)

• Virus: Malware som, när det körs, kopierar sig självt till annan exekverbar kod

• Worm/mask: Ett program som körs oberoende och kan kopiera in sig självt på andra system på nätverket genom att utnyttja säkerhetshål eller stulna lösenord

• Zombie/bot: Program installerade på en infekterad maskin som aktiveras för att göra attacker på andra maskiner

DDoS, Distributed denial of service

Attack för att blockera en websida. Troligen hämnd eller aktivist, men kan också göras för att dölja en annan attack.

Görs genom att accessa den från många noder på en gång.

Förbereds genom att installera malware (zombies) i andras datorer.

IOT-attacken var en sådan.

Hur får man in malware?

- Exploits: Kod som utnyttjar en speciell svaghet
- Backdoor/trapdoor: Mekanismer som går förbi normala säkerhetskontroller
- Drive-by download: Kod på websidor som utnyttjar svagheter i webläsaren för att angripa klienten när sidan öppnas och installera program.
- Installera efter att ha lyckats logga in i systemet.
- Falsk programvara

Falsk programvara

Programvara kan vara något annat än det utger sig för att vara.

- Legitima program som ändrats till att installera malware
- Program som i sig utför skada
- Piratkopior, "kloner" som bryter mot upphovsrätt

Skydd mot falsk programvara

Programvarubutiker kan göra kontroller

Varningar om program som laddats ner från Internet

Program kan ha identifikationsstämplar, riktighetskoder som kräver att programmet är oförändrat

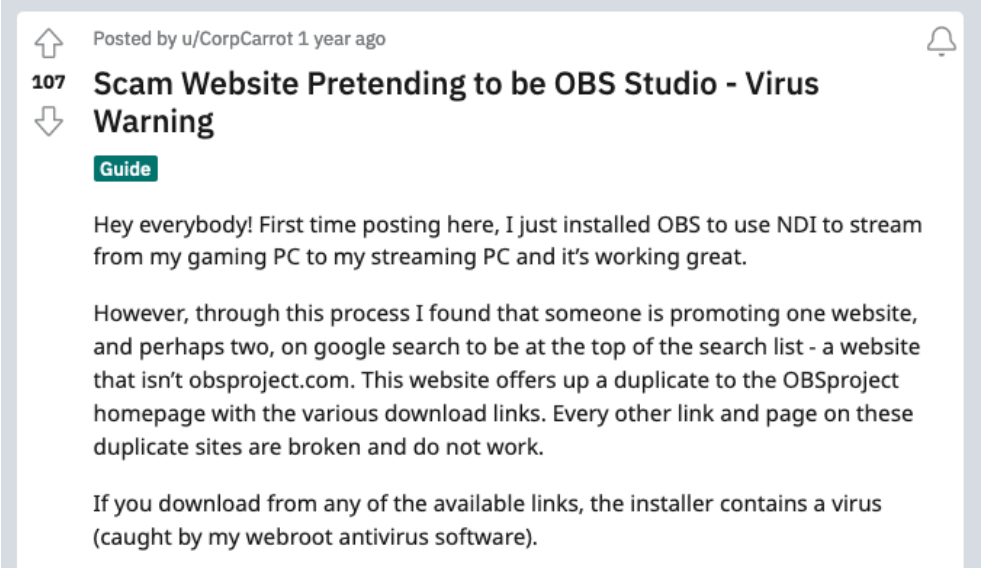
Kontroller tar tid och kan hindra legitim programvara

Risk för konflikter mellan integritet och tillgänglighet!

Se upp med okända websidor,

Falska websidor med infekterade program

Se upp med okända websidor: Bluffsidor med infekterade eller falska program.



Posted by u/CorpCarrot 1 year ago

107

Scam Website Pretending to be OBS Studio - Virus Warning

Guide

Hey everybody! First time posting here, I just installed OBS to use NDI to stream from my gaming PC to my streaming PC and it's working great.

However, through this process I found that someone is promoting one website, and perhaps two, on google search to be at the top of the search list - a website that isn't obsproject.com. This website offers up a duplicate to the OBSproject homepage with the various download links. Every other link and page on these duplicate sites are broken and do not work.

If you download from any of the available links, the installer contains a virus (caught by my webroot antivirus software).

Verktyg för malware-script kiddies

• Attack-kits: Uppsättningar av verktyg för att generera malware automatiskt

• Rootkit: Uppsättning verktyg som används efter att angriparen har kommit in i systemet och har root/admin-rättigheter

Angrepp via inloggning

Angriparen vill komma in som "användare"

Knäck lösenord
Sök igenom systemet
Installera program (malware) som kan hitta läckor

Hur tar sig angripare in?

Ändrar sig över tiden!

1) Hitta mål. Vem vill du angripa? Angriparen identifierar och karakteriserar mål från öppet tillgänglig information, teknisk och icke-teknisk.

2) Första access: Hur kommer den in? Utnyttjar ofta säkerhetsläckor on-line, som svag autenticering eller malware via drive-by-download-attacker.

3) Utökning av privilegier. Väl inne, kan angriparen få administratörsprivilegier?

Hur tar sig angripare in?

4) Samla information eller utforska systemet. Kan göras för att navigera till annat mål.

5) Behålla access. Installera bakdörrar eller annan malware, eller genom att lägga till extra adminstratörskonton.

6) Dölja spåren. Ta bort eller ändra loggfiler som visar vad som hänt.

Säkerhetshål

Hur kan program ha säkerhetshål?

Säkerhetshål

Dålig säkerhet kan läcka information som låter angriparen komma vidare.

Detta kan innebära köra egna program som fiskar information.

Det kan också innebära att köra existerande program som under vissa förutsättningar kan läcka information.

Abstraktion: nödvändighet och risk

Abstraktion eller transparens?

Risker vid förändring och abstraktion

Förändringar skapar problem!

Även om du förstår vad ändringen går ut på så är det lätt att det blir fel.

Abstraktion är användbart för att förstå komplexa system... men kan dölja vad som verkligen händer.

Säkerhetsproblem är ofta så detaljberoede att det är ett problem att dölja detaljer.

Abstraktion vs transparens

Abstraktionsrisker: Heltal

Enkelt exempel: 8-bitars heltal, en byte:

8-bitars heltal "unsigned": 0 till 255

8-bitars heltal "signed": -128 till 127

Unsigned: 255 + 1 = 0

Signed: 127 + 1 = -128

255 unsigned = -1 signed

Demos:
signed-error-2
Integer overflow

Abstraktionsrisker: Heltal

Viktig insikt: Datorns heltal är inte samma sak som matematiskt heltal

$b \geq 0 \nRightarrow a + b \geq a$

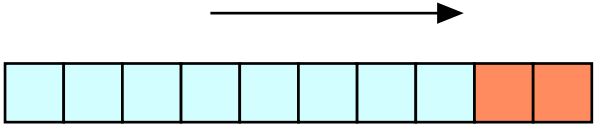
Använd huvudsakligen "unsigned"

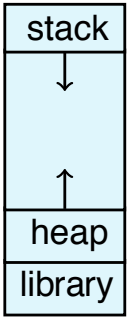
Se upp för overflow

Slå på varningar för operationer mellan signed och unsigned

Minneshantering

Accesser över buffergränser, buffer overruns. Stack overrun och heap overrun. Arraygränser.





Stack overrun

Vad händer när stack och heap tar slut?

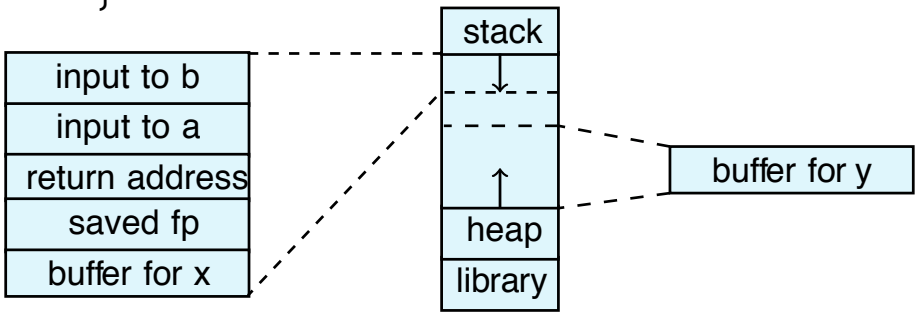
Exempel:

```
void myfunction(int a, int b)
{
        char x[20];

        char *y = malloc(1000);
}
```

Läggs på stack

Läggs på heap



Stack overrun

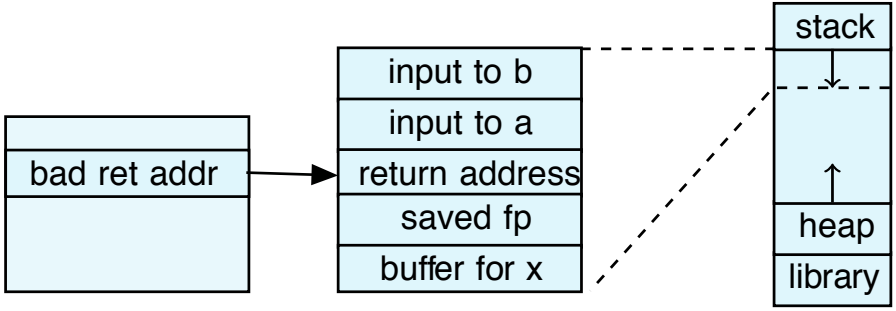
Genom buffern som nu krockar med stacken kan ett program accessa data utanför de egna variablerna. Detta begränsas av minnessaccessrättigheter, men så länge de hålls kan konfidentiell information läsas ut eller farligt information skrivas in (t.ex. en annan returadress).

Det går också att accessa arrayen y utanför arraygränserna med liknande möjligheter.

Stack/buffer overrun

Skriv för mycket data i buffern (här x)

I överlappet skrivs data över returadressen eller andra variabler



Demo:
overflow

Tidigt exempel på buffer overrun

"The Morris worm" kom 1988 och infekterade 5-10% av alla datorer kopplade till Internet!

Utnyttjade buffer overrun i deamonen *fingerd* i BSD Unix på VAX-datorer.

Gärningsmannen avslöjades och bötade $10000 plus 400 timmar samhällsservice.

```
push1  $68732f          push  '/sh, <NUL>'
push1  $6e69622f        push  '/bin'
mov1    sp, r10         save  stackp  in  r10  (string  beginning)
push1  $0               push  0  (arg  3  to  execve)
push1  $0               push  0  (arg  2  to  execve)
push1  r10              push  string  beginning  (arg  1  to  execve)
push1  $3               push  argc
mov1    sp, ap          set  argv  to  stackp
chmk    $3b             perform  'execve'  kernel  call
```

Tidigt exempel #2 på buffer overrun

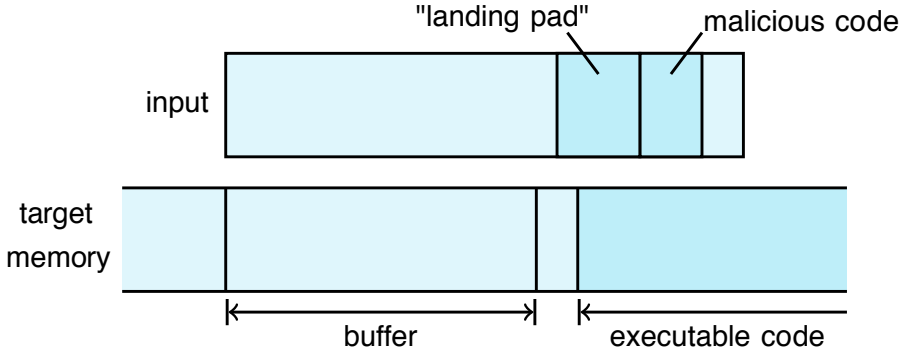
I MS/DOS kunde man ha 8 tecken i filnamn och directorynamn.

Om man skrev in mer än 8 tecken så... kraschade systemet helt!
Filsystemet blev korrupt och man fick formattera om!
(Availabillityproblem.)

Buffer overruns och "smashing the stack"

Svaghet: Programmerare slarvar ofta med att testa storleken på indatasträngar. Angriparen gör en oväntat lång indata som sedan kan tolkas som exekverbar kod.

Utan kunskap om underliggande program ger detta oftast kracher, men kan också resultera i intrång.



Heap overruns

Svårare att utföra eftersom det är svårt att avgöra var buffern befinner sig relativt målet.

Målet är ofta pekare till öppna filer eller funktioner (givet att heapen innehåller exekverbar kod)

Oftast krascher i stället för intrång, men med många försök (många användare) kan attacken till slut resultera i intrång

Free the mallocs!

Tidigare använt minne kan innehålla information när det allokeras igen.

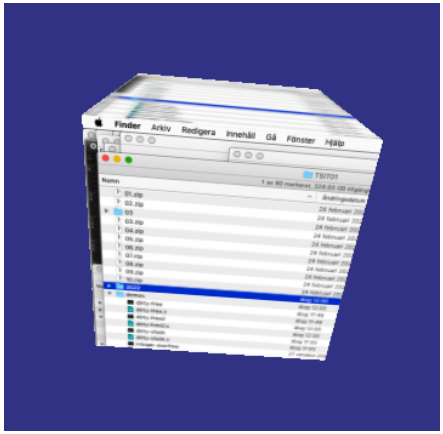
malloc
tilldela minnet hemlig information
free - nu är det borta! Väl...

malloc görs utan att radera.
Nu innehåller den nya buffern det gamlas värde.

Detta kan även ge läckor vid buffer overrun.

Demo:
dirty-free

Även på stacken



Samma sak händer på stacken.

En sträng med information läggs på stacken.

När funktionen avslutas "försvinner den".

Nästa funktion får samma minnesarea.

Även ett problem i grafikminnet på GPUn. Misslyckas en texturladdning så får vi gammal information.

Demo:
dirty-stack

Förebyggande åtgärder: Hårdvara

• Itanium har ett separat register för returadressen.

• Icke exekverbara datablock. AMD kallar detta NX, Microsoft kallar det Data Execution Prevention (DEP), och Intel kallar det eXecution Disable (XD). Kräver stöd av minneshanteraren i CPUn. DEP introducerades i MS Windows i XP SP2.

Förebyggande åtgärder: Programvara

Icke-exekverbar stack stoppar en del attacker.

Programvara som kräver exekverbar stack slutar fungera.

Förutsägbara minnesaccesser är en fördel för angriparen, därför används stack & heap randomization i alla moderns operativsystem. (Med hur påverkar detta prestandan?)

Stack-based overrun detection (för Visual C++) lägger till tester för korrupt stack vid varje gång en funktion avslutas.

Förebyggande åtgärder: Säkrare funktioner

C är ökänt för osäkra strängfunktioner, strcpy, sprintf, getc.

Null-terminerade strängar! Saknas terminering får trängen en okänd längd in i okänt minne! Inget inbyggt test på maxlängd.

T.ex. strncpy innehåller ett argument för maxlängd. Säkrare!

Problem: Säkrare funktioner dåligt standardiserade!

Förebyggande åtgärder: Säkerhet i datatyper

Komplatorer kan testa osäker användning av datatyper, t.ex. att man blandar signed och unsigned.

Dynamisk testning körs i realtid och gör programmet långsammare

Statiskt testning görs vid kompilering. Den är mer komplicerad men kostar inte prestanda.

Vissa språk (Pascal, Java) har strikt testning av datatyper. Andra är släpphänta och tilldelar typer efter användning (JavaScript, Python). Gissa vad som är populärast? :)

Upptäckt

Canaries, minneselement som används för att upptäckta oönskade ändringar i data under exekvering

Kodinspektion för hand är långsamt och du kommer att missa mycket men fungerar ibland.

Automatisk kodinspektion använder sig av expertsystem för att hitta kända svagheter.

Säkerhetstestning med scripts/program som skickar slumpmässiga indata till programmet. Slumpmässiga indata tar tid. Indata fokuseras ofta på förväntade attacker. Bra idé?

Tidigt exempel: "Monkey DA" från Apple.

Begränsa privilegier

Se till att koden inte körs i onödan

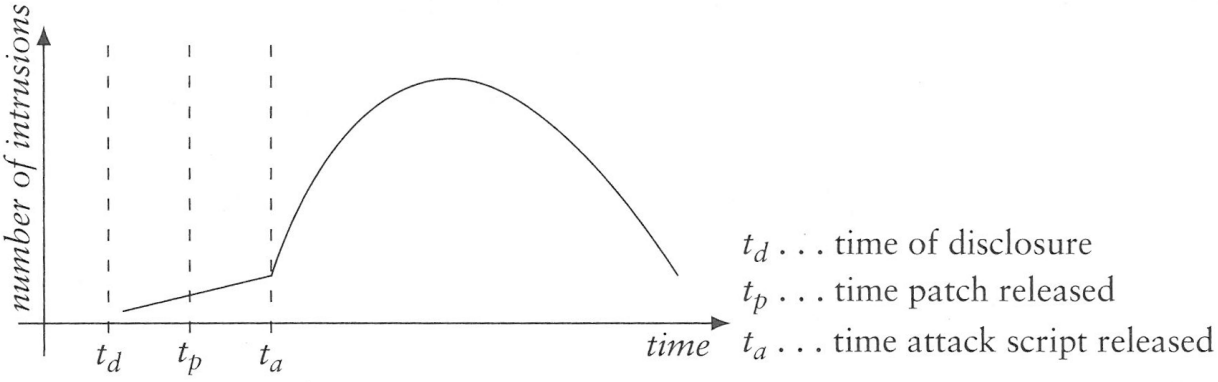
Ge inte användare mer rättigheter än de behöver

Ta bort rättigheterna snarast möjligt

Aktivera inte funktioner som du inte behöver

Reaktion: Uppdatera

Applicera säkerhetsuppdateringar snabbt.



Bugg-fri programvara?

Testning

Testning görs oftast med *förväntad* användning

Angripare gör oväntade saker

I fältet där du skrev "foo" kommer någon annan att skriva "supercalifragilisticexpialidocious"!

Jag kunde krascha en Sun-station (Unix) på en minut genom att skriva en lång rad utan mellanslag i ordbehandlaren!

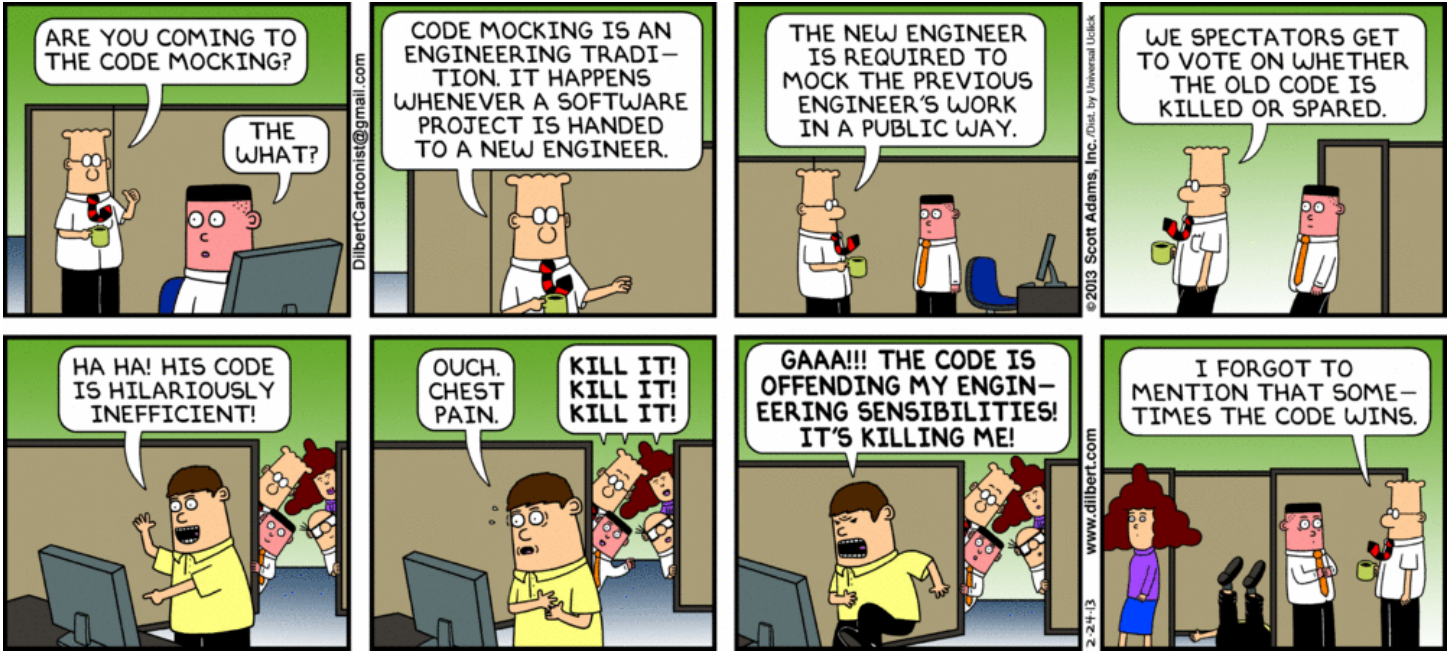
Buggar

Du vill göra koden bugg-fri (så klart).

Antalet buggar påstås minska med tiden. Det gör det inte.
Varför blir det fler buggar fast vi korrigerar dem hela tiden?

- Nya funktioner, mer och mer komplext
- En korrektion ger upphov till nya fel
- Anpassning till nya standarder - skriv om - gör nya fel
- Ny personal

Ny ingenjör, ny kod...



Ny ingenjör, ny kod...

Mycket vanligt problem.

Komplett nytt system vanligt.

När detta görs på mellannivåer så blir kostnaden stor och ger ofta förlorad funktionalitet och genom att allt skrivs om får man lätt förlorad säkerhet.

Konformism som säkerhetsrisk

"Alla andra gör ju så"

Konformism, att alla gör likadant, gör det lättare för angripare.

Skadlig kod riktar sig oftast mot MS Windows, för det är vanligast och bildar ofta stora monokulturer. Stora angrepp har hänt just i sådana.

Angrip de vanligaste OSen, de vanligaste webläsarna, de vanligaste verktygen.

Jfr angrepp av skadedjur, sprider sig snabbast i monokulturer.

https://www.skogsindustrierna.se/hallbarhet/skogsbruk/angreppen-av-granbarkborrar/