Wormen en virussen

Ongedierte op het net

Wormen en virussen richten veel schade aan en de kosten daarvan lopen in de miljarden. De auteur beschrijft hoe wormen en virussen werken, wat de kenmerken zijn en gaat in op bestrijdingsmethoden.

Herbert Bos

De gemiddelde levensduur van een onbeschermde Windows-pc is minder dan tien minuten. Soms korter, soms langer, maar denk niet dat u een volle dag op internet zonder kleerscheuren doorkomt zonder een elektronische slotgracht, dikke muren en strenge bewaking aan de poort. Wie het zelf wil proberen, moet zijn Windows 2000- of XP-machine – zonder firewall, service pack of security-updates – maar eens aansluiten op internet en wachten tot het ding wordt geïnfecteerd. Na het experiment zes keer herhaald te hebben, met een gemiddelde infectietijd van acht minuten, werd het vervelend om steeds weer de machine op te starten en geloofde ik het wel. De kans besmet te raken tijdens een willekeurig half uurtje op internet is groter dan tijdens een maandenlange griepepidemie.

De directe schade die wordt aangericht door wormen en virussen loopt in de miljarden: onbruikbare computers, verloren gegevens en de kosten van reparatie en systeembeheer. Daarnaast is er indirecte schade, want met het binnendringen van een computer is voor veel wormen en virussen de kous niet af: ze installeren zogenoemde *backdoors* die worden gebruikt voor nieuwe, vaak criminele, activiteiten. Zo worden deze computers gebruikt voor het versturen van spam of voor het lanceren van massale denial-of-service (dos)-aanvallen.

Het gaat ook al lang niet meer alleen om pubers op zolderkamers die indruk willen maken op vrienden. Steeds vaker is het een kwestie van geld, waarbij de handelswaar bestaat uit onze machines die inmiddels dankzij wormen en virussen onder controle staan van hackers. Spamverspreiders betalen hackers voor de toegang tot deze machines, waarna ze gebruikt worden om grote hoeveelheden ongewenste e-mail te versturen – onder onze neus, vanuit onze eigen huiskamers!

Hoe werken ze?

Wormen en virussen maken gebruik van kwetsbaarheden in programma's. Hiervoor worden zogenaamde *exploits* geschreven die een hacker controle geven over de machine. Een van de meest gebruikelijke technieken hiervoor is gebaseerd op *bufferoverflows*. Het idee is eenvoudig, maar een precieze uitleg vergt enige kennis van hoe software werkt in moderne computers (zie kader).

In grote lijnen komt het hierop neer: een programma leest van het netwerk data in een buffer ter grootte van X bytes. De aanvaller levert echter meer dan X bytes aan, waardoor de buffer 'overloopt': er worden data geschreven voorbij de buffergrens. Als de aanvaller in staat is een geheugenlocatie te overschrijven waarin zich het adres bevindt van een instructie die moet worden uitgevoerd, kan hij ervoor zorgen dat het programma springt naar de instructies die hij wil uitvoeren, bijvoorbeeld instructies om uw harde schijf te formatteren of om uw computer te veranderen in een advertentiebureau voor penisverlenging.

Ongedierte in het wild

Een nieuw verschijnsel dat steeds populairder

Samenvatting

Er zijn diverse bestrijdingsmethoden voor wormen en virussen, bijvoorbeeld deep scans en honeypots, maar geen daarvan is afdoende tegen alle soorten. De hoeveelheid valse meldingen van sommige oplossingen en het polymorfisme van toekomstige wormen zijn waarschijnlijk de grootste problemen. Zolang er geen adequate bestrijdingsmethoden zijn, rest niets anders dan met allerlei maatregelen zoveel mogelijk gaten af te dekken.

wordt, is de organisatie van besmette machines in een gecoördineerd gedistribueerd systeem dat kan bestaan uit soms wel honderdduizenden computers: een *botnet*. Een botnet maakt vaak gebruik van openbare internetchatkanalen (irc) om te communiceren. Zodra een machine besmet is, logt hij in op een chatkanaal van een openbare server en volgt daar samen met alle andere gehackte machines (*drones*) wat er zoal aan conversatie plaatsvindt op het kanaal. Sommige teksten die voorbijkomen bevatten codewoorden, die voor de drones dienen als commando's. Zo kan een hacker drones opdracht geven andere machines te besmetten, een dos-aanval te beginnen of spam te versturen.

De verscheidenheid aan wormen en virussen is eveneens verontrustend. Sommige verspreiden zich razendsnel. De Slammer-worm verspreidde zich in 2003 in nog geen 10 minuten naar 90 procent van alle kwetsbare computers op internet en verdubbelde zich elke 8 seconden. En passant werd ook het veiligheidscontrolesysteem van een Amerikaanse kerncentrale uitgeschakeld. Geen wonder dat de angst voor een zogenaamde *flash*-worm er goed in zit. Schattingen van Amerikaanse onderzoekers over hoe lang het duurt voordat de 'The Perfect Worm' een miljoen servers zou kunnen infecteren, werden recentelijk bijgesteld van 30 naar 0,5 seconden.

Laat u zich echter niet bang maken, want zo'n vaart zal het – letterlijk – niet lopen. Deze scenario's gaan uit van *state-sponsored attacks*, waarbij een

Mijn eerste exploit, een spoedcursus

Als we een programma beschouwen als een reeks instructies in het geheugen van een pc, dan bestaat die reeks zelf weer uit een verzameling functies die elkaar over en weer aanroepen. Wanneer functie f functie g aanroept, dan wordt gesprongen naar de instructies die bij g horen. Als g is uitgevoerd, wordt teruggesprongen naar de instructie in f direct volgend op de instructie die de aanroep deed. Er wordt daarom door de computer bijgehouden welke instructie dat is, door middel van een *stack* (stapel).

Het 'server'-programma met een beveiligingslek 1

```
1 // Functie die een file leest in een buffer. Bij een webserver lees
2 // je niet van file maar bijvoorbeeld van een netwerkverbinding. In
3 // dat geval zult u deze functie moeten vervangen door iets dat van
4 // het netwerk leest.
5 void lees_in_buf (char *buf)
6 {
7   int fd = open ("hello.exploit", O_RDONLY); // open de file
8
9 // Lees data in buffer 'buf'. Er worden maximaal 64 bytes gelezen.
10 // Maar de buffer is maar 36 bytes lang. Als de file langer is dan
11 // 36 bytes (zoals bij 'hello.exploit), dan stroomt de buffer over.
12   read (fd, (void*)buf, 64);
13   close (fd);
14 }
15 int kwetsbare_functie () // deze functie bevat een kwetsvare buffer
16 {
17   char buf[36];          // deze buffer gaan we overstromen
18   lees_in_buf(buf);      // m.b.v. deze functie
19   printf ("klaar met lezen\n");
20   return 0;
21 }
22 int main ()
23 {
24   kwetsbare_functie ();  // roep de functie aan
25   return 1;
26 }
```

land dat beschikt over een moderne netwerkinfrastructuur alle middelen inzet. Welke landen zijn dit? Rijke, veelal westerse landen! Misschien schat u het anders in, maar ik verwacht van België of Zwitserland in de nabije toekomst niet veel narigheid. Dat neemt niet weg dat ook een iets minder snelle flash-worm nog gevaarlijk genoeg kan zijn. Een worm die zich binnen tien minuten over de hele wereld verspreidt, kan verschrikkelijke gevolgen hebben.
Maar misschien schuilt het echte gevaar wel niet in flash-wormen. Er zijn namelijk ook wormen die zich juist langzaam verspreiden, maar daardoor moeilijker zijn te traceren. We hebben het dan over *stealth*-wormen. Tegen de tijd dat zo'n worm ontdekt wordt, kan hij al enorme hoeveelheden machines besmet hebben, met grote schade als gevolg.
Welke worm erger is, is zoiets als vragen welke ziekte erger is, Spaanse griep of aids. De ene ziekte verspreidde zich razendsnel met een groot aantal onmiddellijke slachtoffers, terwijl de besmetting bij de ander langzamer is en minder zichtbaar, maar met even catastrofale gevolgen. Het voordeel van een flash-worm is dat je in ieder geval onmiddellijk merkt dat er iets aan de hand is. Het voordeel van een stealth-worm is dat je meer tijd hebt om te ontdekken wat er aan de hand is. De implicatie is dan ook dat een flash-worm zo'n snelle respons nodig heeft dat menselijke interventie niet werkt. Daar staat tegenover dat het makkelijker is om een automatisch responssysteem te ontwikkelen, omdat de aanval makkelijker te identificeren is. Voor een stealth-worm geldt het omgekeerde: het mag dan moeilijk zijn om de aanval te identificeren, er is in elk geval voldoende *tijd* om dat te doen. Daarbij zal een belangrijke rol weggelegd zijn voor de intelligentie en ervaring van menselijke experts.
Een belangrijke conclusie is dan ook dat we *niet* hoeven te zoeken naar een waterdichte oplossing voor *alle* wormen en virussen, de heilige graal in internetbeveiliging. En wie op dit moment zo'n oplossing te koop aanbiedt, is een kwakzalver en een handelaar in schijnveiligheid. In plaats daarvan kunnen we ons beter richten op specifieke oplossingen voor specifieke problemen. In de reguliere geneeskunde is dat natuurlijk niet anders. Een ziekte als tbc wordt nu eenmaal op geheel andere wijze bestreden dan aids of sars. In de computerbeveiliging kunnen we nog veel opsteken van onze medische vakbroeders en -zusters.
Om goed beschermd het donkere internet op te gaan is het dan ook raadzaam meerdere veiligheidsmaatregelen te nemen. Regelmatige updates, een firewall en een virusscanner zijn al snel onontbeerlijk. Wie op vakantie gaat naar donker Afrika, doet dat tenslotte ook niet zonder een half dozijn vaccinaties en een rugzak vol malariapillen. En wie seks wil, schuift een condoom om zijn interface.

Kenmerken van wormen en virussen

Voor verschillende aanvallen worden verschillende bestrijdingsmethoden gebruikt. Net als in de reguliere geneeskunde geldt ook hier dat de beste bestrijdingsvorm een afweging is van de meest *gewenste* bestrijdingsvorm (in termen van kosten, acceptatie, juridische mogelijkheden, enzovoort) en de meest *effectieve* bestrijdingsvorm voor de aanval die moet worden bestreden. Wat betreft het laatste zijn de volgende kenmerken van een worm of virus van groot belang: is het een nieuwe worm of een oude bekende?; is de manier waarop de worm of het virus verpakt zit steeds anders of blijft deze constant?; weet de worm of het virus wie moet worden aangevallen of wordt er in het wilde weg geschoten?
Een 'nieuwe' worm is een aanval waarvan we nog geen signatuur hebben en die wellicht een beveiligingslek exploiteert waarvan we het bestaan niet eens kenden. Dit betekent dat we eerst moeten detecteren dat er iets niet in de haak is, bepaald geen triviale operatie. Een aanval waarvan geen signatuur bekend is, wordt wel een *zero-day*-aanval genoemd.
De verpakking van een worm is hoe de datapakketten waaruit de worm bestaat er op het netwerk uitzien. Net als in normale politiezaken is het moeilijker om een boef te pakken die steeds een

»We hoeven niet te zoeken naar een waterdichte oplossing voor alle wormen en virussen«

nieuwe vermomming draagt dan iemand die makkelijk herkend kan worden, bijvoorbeeld aan de hand van een politiefoto. De internetvariant van de vermomming noemen we *polymorfisme*: een polymorfe aanval is dus een worm die er steeds anders uitziet en daardoor niet eenvoudig kan worden herkend aan de hand van een patroon (signatuur). In het meest extreme geval gaat de worm helemaal versleuteld over het netwerk, waardoor het vrijwel onmogelijk wordt om de aanval te herkennen door simpelweg te kijken naar de bits op de kabel.

Het laatste kenmerk kent eveneens een equivalent in opsporingskringen. Een effectieve manier om een fietsendief te pakken is om een nieuwe fiets neer te zetten bij het Centraal Station in Amsterdam en te wachten tot die wordt meegenomen. Je lokt de crimineel dan in een val en je kunt tevens zien hoe zo'n dief in de praktijk te werk gaat. Helaas werkt dat alleen voor dieven die *in het algemeen* op zoek zijn naar een nieuwe fiets, maar niet voor de boef die het alleen voorzien heeft op *jouw* fiets en verder alle andere fietsen links laat liggen. Als de dief bovendien een lijst bij zich heeft van alle adressen waarop een fiets te vinden is die hij wil hebben en waarvan hij weet dat hij het slot openkrijgt, is hij ook nog eens veel sneller omdat hij niet elke woning hoeft te proberen. In termen van computercriminaliteit spreken we dan van een *hitlist*-worm.

Een uitbraak van een polymorfe zero-day hitlist-worm is een van de grootste uitdagingen voor wormbestrijders. Op dit moment zijn zulke wormen nog niet echt in het wild waargenomen, maar het is bekend hoe ze gemaakt moeten worden en in de internetwereld betekent dat, helaas, dat ze ook gemaakt *zullen* worden.

Bestrijding

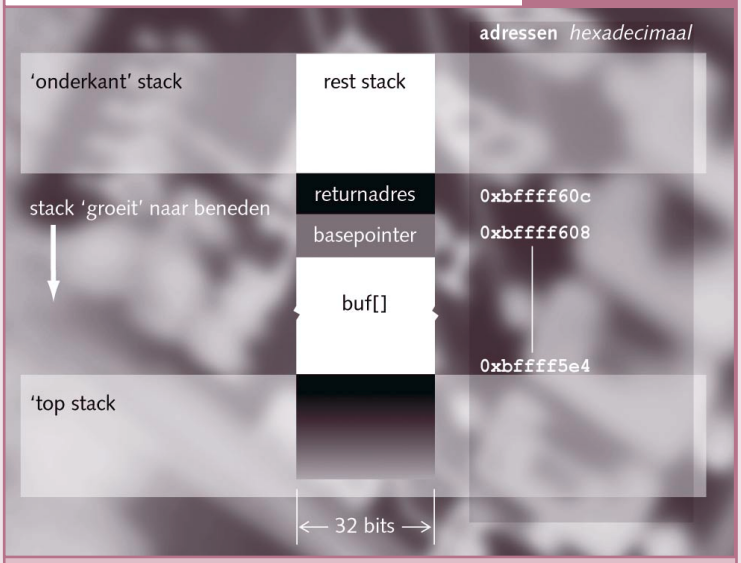
Bestrijding van wormen en virussen gaat aan de hand van *intrusion-detection*- en *intrusion-prevention*-systemen (meestal afgekort tot ids en ips). Een ips verschilt van een ids enkel daarin dat er niet alleen alarm wordt geslagen, maar dat de aanval ook effectief wordt gestopt. Als we nu eens een aantal bestrijdingsmethoden op een rij zetten, kunnen we vervolgens bepalen welke methoden passen bij welke wormen. Daarbij kijken we niet alleen naar wat vandaag in de winkel te vinden is, maar ook naar wat in de toekomst valt te verwachten.

Wereldwijd wordt enorm veel onderzoek verricht naar de bestrijding van wormen en virussen. In Nederland is er op de Vrije Universiteit in

Neem bijvoorbeeld het C-programma van figuur 1. Zoals alle C-programma's begint deze code in de functie `main()`. Vandaaruit wordt een functie `kwetsbare_functie()` aangeroepen. In deze functie worden data gelezen in een buffer genaamd `buf`. Hiervoor wordt de functie `lees_in_buf()` gebruikt. Normaal zou deze functie wellicht data van het netwerk lezen, maar voor het gemak lezen we nu uit een bestand.

De stack bij een functieaanroep: array `buf[]` zit onder het returnadres

2



De stack bevindt zich boven in het geheugen en groeit naar beneden. Figuur 2 illustreert de stack zoals die eruitziet net na aanroep van de functie `kwetsbare_functie()`. Het eerste dat gebeurt is dat het *returnadres* op de stack wordt gezet. Als de functie klaar is en het `return`-commando in regel 20 wordt uitgevoerd, wordt het returnadres op de stack gebruikt om naartoe te springen. Normaliter wordt dan gesprongen naar de machinecode die hoort bij regel 25 in `main()`.

Andere dingen die op de stack worden gezet, zijn de basepointer en de lokale variabelen. De basepointer is niet van belang voor deze discussie en wordt buiten beschouwing gelaten. De lokale variabele `buf[]` is echter de spil waarom de exploit draait. De buffer is 36 bytes groot, en er wordt op de stack dan ook keurig ruimte gereserveeerd voor deze 36 bytes. Helaas verzuimt de functie `lees_in_buf()` bij het lezen rekening te houden met de grootte van buffer `buf`. Als meer dan 36 bytes worden gelezen, stroomt de buffer over en wordt geschreven over de velden die onder `buf` liggen op de stack (dus op hogere adressen, omdat de stack van boven naar beneden groeit). Met andere woorden, over `basepointer` en `returnadres`!

Stel dat we weten dat het returnadres zich bevindt op adres 0xbffff60c (decimaal 3221222924). Het begin van `buf[]` bevindt zich dan op adres 0xbffff60c – 4 – 36 = 0xbffff5e4 (3221222884). Wat we nu kunnen doen, is een programma van precies 44 bytes laden in `buf`. Het programma begint op 0xbffff5e4 (dat wil zeggen, daar komt de eerste instructie van het programma). In de laatste 4 bytes

Amsterdam bijvoorbeeld een groot onderzoeksprogramma gestart dat zich met name richt op worminfectie. Hierbij worden zoveel mogelijk relevante aspecten in het onderzoek betrokken (zoals besturingssysteem, netwerk, encryptie en informatie-uitwisseling met internationale partners). Maar kleinere of grotere securityprojecten zijn aan vrijwel elke universiteit te vinden. Ik zal proberen de belangrijkste bestrijdingsmethoden te karakteriseren aan de hand van kenmerken en de verschillende verschijningsvormen te illustreren aan de hand van bestaande (onderzoeks)projecten.

Verdediging in het netwerk of bij de gebruiker?

De bekendste verdedigingsmechanismen zijn virusscanners en firewalls. Een firewall is een tamelijk eenvoudig systeem dat bepaalt welk verkeer naar binnen mag en welke pakketten buiten moeten blijven. Veelal geschiedt dit aan de hand van adressen en poortnummers. Het is dan ook niet echt een bestrijdingsmethode en we laten deze hier verder buiten beschouwing.
Een virusscanner daarentegen is een zeer actieve en belangrijke verdediging in de strijd tegen virussen. In principe werkt de virusscanner op het niveau van een gebruikersapplicatie. Een scanner op een pc thuis en een scanner op een centrale mailserver zijn daarin niet wezenlijk verschillend: beide scannen ze e-mail op virussen. Hoewel moderne virusscanners beschikken over functionaliteit waardoor ze in beperkte mate zero-day-aanvallen kunnen detecteren, zijn ze toch vooral geschikt voor niet-polymorfe virussen waarvan een signalement bekend is.
Andere verdedigingsmechanismen voor op de pc van de eindgebruiker richten zich nadrukkelijk ook op polymorfe zero-day-aanvallen. Deze projecten bevinden zich veelal nog in de onderzoeksfase, maar zijn waarschijnlijk een voorbode van wat in de toekomst gebruikt gaat worden. Zo zijn er projecten die zoeken naar ongebruikelijke patronen in het gedrag van applicaties. Als het normale gedrag van een applicatie bijvoorbeeld is dat het een beperkt aantal systeemaanroepen in min of meer dezelfde volgorde uitvoert, dan gaan alarmbellen rinkelen wanneer de applicatie ineens heel afwijkend gedrag vertoont.
Een andere pc-benadering die wordt gebruikt in een van de projecten aan de Vrije Universiteit, is bijhouden wat er gebeurt met data die van het netwerk komen. Het idee is als volgt: bij de geheugenlocaties waar deze potentieel besmette data worden opgeslagen, zetten we een kruisje. Als potentieel besmette data worden gekopieerd naar een andere geheugenlocatie, zetten we daar eveneens een kruisje. Op het moment dat een poging wordt gedaan om instructies uit te voeren die zich bevinden op een locatie waar een kruisje achter staat, weten we dat iets wordt uitgevoerd dat van buiten komt. Op dat moment wordt een alarm gegenereerd.

In het netwerk

Net als op de pc kunnen we in het netwerk eveneens onderscheid maken tussen technieken die speuren naar afwijkend *gedrag* en mechanismen die kijken naar de *inhoud* van de data. Voor het gemak noemen we ze *flow-based* en *deep scan*.
Het idee achter een flow-based methode is dat gekeken wordt naar afwijkend gedrag van grote hoeveelheden data, bijvoorbeeld de hoeveelheid verkeer die naar webservers gaat. Is die een orde van grootte meer dan gebruikelijk, dan zou dat kunnen duiden op de aanwezigheid van een worm.
Zulk afwijkend gedrag kan natuurlijk ook andere oorzaken hebben, zoals de start van een populair evenement (denk aan het wk voetbal). Gedrag dat wordt geduid als een aanval terwijl het legitiem gedrag is, is een valse melding. Het is een van de belangrijkste obstakels voor volledig geautomatiseerde verdedigingssystemen. Stel dat het verdedigingssysteem besluit om automatisch bepaalde (legitieme) datapakketten te weren op basis van een valse melding, dan gaat dat ten koste van de beschikbaarheid. In wezen veroorzaakt het systeem zelf een denial-of-service (dos)-aanval. De schade die aangericht kan worden door een dos-aanval kan snel oplopen. Vandaar dat een lage ratio van valse meldingen een van de belangrijkste criteria is voor een geautomatiseerd verdedigingssysteem.
Een voorbeeld van een commercieel verkrijgbaar flow-based ids is de VirusThrottle-module in moderne HP edge switches. De module slaat alarm als via een van de inkomende lijnen een ongebruikelijk aantal verbindingen wordt opgezet naar een grote verscheidenheid aan bestemmingen. Dit is namelijk typisch gedrag voor een worm. De switch kan verschillende maatregelen

nemen, van het attenderen van de beheerder tot het uitzetten van de betreffende poort waar deze verbindingen vandaan komen. Om nog even terug te komen op het probleem van de valse meldingen: het is niet altijd verstandig om over te gaan tot het uitzetten van de switchpoort, want het genoemde gedrag is niet alleen typisch voor wormen, maar eveneens voor peer-to-peerapplicaties! Nogmaals, het nemen van drastische maatregelen op basis van een valse melding leidt tot vermindering van de *beschikbaarheid* van het systeem.
Het idee achter deep scans is dat je kijkt naar de inhoud van de pakketten op het netwerk. Een bekend voorbeeld is NetSifts EarlyBird-systeem, dat recent werd opgekocht door Cisco. EarlyBird vergelijkt de inhoud van de datapakketten die op hoge snelheid voorbij flitsen in het netwerk. Als vaak dezelfde patronen langskomen en de pakketten afkomstig zijn van verschillende bronnen, dan wordt dit beschouwd als een worm. Een ander voorbeeld van deep scan is de implementatie van wormscanners op netwerkkaarten die momenteel in verschillende laboratoria worden ontwikkeld.

Honeypot

Een benadering die tussen host-based en network-based in zit, is de *honeypot*. Een honeypot is vaak een gewone pc met een aantal ip-adressen die niet actief worden gebruikt. De pc doet zich voor als een gewone machine waarop bijvoorbeeld bekende diensten worden gedraaid of gesimuleerd, zoals een webserver. De machine wordt echter niet geadverteerd en omdat zijn ip-adressen onderdeel zijn van ongebruikte adresruimte, zou er in principe nooit verkeer naartoe gestuurd moeten worden. Als de honeypot dan toch verkeer ontvangt, is dat per definitie verdacht. Opnieuw geldt dat het niet altijd om een aanval gaat en het probleem van valse meldingen is hier eveneens levensgroot aanwezig. Als we herhaaldelijk en op verschillende honeypots dezelfde datapakketten zien, worden de aanwijzingen sterker dat het om een aanval gaat. In dat geval wordt gepro-

zetten we dan het adres waarop het programma begint, dus 0xbffff5e4. Deze 4 bytes overschrijven dan precies het returnadres van de functie `kwetsbare_functie()`. Met andere woorden, we schrijven een klein programmaatje en sturen dit naar het programma van figuur 1 en tegelijkertijd overschrijven we het returnadres zodat er naar dit programma gesprongen wordt. Op het moment dat de `return`-instructie wordt uitgevoerd in regel 20 in figuur 1, wordt dan niet meer gesprongen naar de instructie die behoort bij regel 26, maar naar onze eigen instructie op adres 0xbffff5e4.
Als triviaal voorbeeld bekijken we een programmaatje dat 'hello' afdrukt op het scherm. Maar wacht, we kunnen natuurlijk niet zomaar een C-programma sturen! In plaats daarvan moeten we een in machinetaal programmaatje genereren dat direct kan worden uitgevoerd. We noemen zulke programmaatjes die ons controle geven over de aan te vallen machine *shellcode*. Deze code wordt meestal geschreven in assembly. Assembly is een stuk ingewikkelder in het gebruik dan C, maar wie een beetje assembly leert, kan al snel heel elegante shellcode schrijven.
Het 'hello'-programmaatje in assembly is weergegeven in figuur 3. De details zijn niet belangrijk, enkel de functionaliteit: het drukt 'hello' af op het scherm. Vervolgens halen we het programmaatje door een assembler en wat we overhouden is een binair blokje data, bestaande uit de instructies in machinetaal. Voor het programma van figuur 3 is de binaire vertaling weergegeven als hexadecimale getallen, bijvoorbeeld:

```
eb 19 31 c0 31 db 31 d2 31 c9 b0 04 b3 01 59 b2 05 cd 80 31 c0 b0
01 31 db cd 80 e8 e2 ff ff ff 68 65 6c 6c 6f
```

Dit moeten we nu in de variabele `buf` plaatsen. Maar eerst moeten we het nog uitbreiden, want in deze code zit nog niet het adres 0xbffff5e4 verweven waarmee we het returnadres op de stack willen overschrijven. Dit plakken we er eenvoudig achteraan. Als we het aantal bytes tellen in de bovenstaande shellcode, komen we op 37. We moeten in totaal 44 bytes hebben, waarvan de laatste 4 bytes gevormd worden door het adres. De tussenliggende 3 bytes

'Hello'-programma in assembly — 3

```
[SECTION .text]
global _start
_start:
        jmp short ender
        starter:
        xor eax, eax     ;clean up the registers
        xor ebx, ebx
        xor edx, edx
        xor ecx, ecx
        mov al, 4        ;syscall write
        mov bl, 1        ;stdout is 1
        pop ecx          ;get the address of the string from the stack
        mov dl, 5        ;length of the string
        int 0x80
        xor eax, eax
        mov al, 1        ;exit the shellcode
        xor ebx,ebx
        int 0x80
        ender:
        call starter     ;put the address of the string on the stack
        db 'hello'
```

maken we voor het gemak nul. De uiteindelijk shellcode is nu:

```
eb 19 31 c0 31 db 31 d2 31 c9 b0 04 b3 01 59 b2 05 cd 80 31 c0 b0
01 31 db cd 80 e8 e2 ff ff ff 68 65 6c 6c 6f 00 00 00 e4 f5 ff bf
```

We zien dat de oorspronkelijke shellcode nu gevolgd wordt door 3 nul-bytes en een 4-bytes adres. Misschien vraagt u zich af waarom het adres in omgekeerde volgorde staat opgeschreven: e4 f5 ff bf in plaats van bf ff f5 e4. Daar zit verder niets achter. Het is nu eenmaal de wijze waarop in een Pentium-pc een 32-bits getal wordt gepresenteerd. Deze omgekeerde volgorde wordt wel Little Endian genoemd. Andere machines maken gebruik van Big Endian en daar hoeven de bytes dus niet omgedraaid te worden.
Nu zijn we klaar. We kunnen de shellcode in binaire vorm wegschrijven in een bestand hello.exploit en het 'server'-programma starten met dit bestand als invoer. Het resultaat is dat 'hello' wordt afgedrukt op het scherm.

beerd er een signatuur van te genereren die vervolgens bijvoorbeeld door een host-based of network-based intrusion-preventionsysteem kan worden gebruikt. Hiervoor is het nuttig om actief signaturen uit te wisselen en verschillende projecten houden zich hiermee bezig, zowel onderzoeksprojecten (bijvoorbeeld het Europese NoAH-project) als commerciële projecten (bijvoorbeeld de volledig geautomatiseerde Fingerprint Sharing Alliance).

Wat heeft u nodig?

Bestrijdingsmethoden die kijken naar bekende wormsignaturen (zoals het populaire Snort en veel virusscanners) zijn erg geschikt om bekende aanvallen af te weren, maar staan machteloos tegenover zero-day-aanvallen, zeker wanneer sprake is van polymorfisme. Deep scans in het netwerk, zoals EarlyBird, zijn eveneens niet opgewassen tegen polymorfisme. Methoden die kijken naar het gedrag van aanvallen op hoger niveau (zoals HP's VirusThrottle in het netwerk en systemen die kijken naar afwijkend gedrag van applicaties op de eindgebruikers pc), zijn weliswaar in staat bepaalde vormen van polymorfe zero-daywormen en -virussen te detecteren, maar resulteren nogal eens in grote aantallen valse meldingen. Honeypots kunnen worden gebruikt voor de detectie van (zero-day-)aanvallen en kunnen soms signaturen genereren, maar opnieuw duikt hier het probleem van de valse meldingen op. Bovendien kunnen ze vaak slecht omgaan met polymorfisme en hitlists. Erger, voor volledig polymorfe wormen en virussen is het tot op heden met geen enkele methode mogelijk een betrouwbare signatuur te genereren. Dit is verontrustend, want deze aanvallen komen er zeker aan.
De hoeveelheid valse meldingen van huidige oplossingen en het polymorfisme van toekomstige wormen zijn waarschijnlijk de grootste problemen waar we mee te kampen hebben. Zolang we geen adequate bestrijdingsmethoden hebben, rest ons niets anders dan met een waaier aan maatregelen zoveel mogelijk gaten af te dekken.

Welke oplossingen *wenselijk* zijn, hangt af van de organisatie. De beheerder ziet zich daarbij al snel geplaatst voor een duivels dilemma: wat weegt zwaarder, beschikbaarheid of veiligheid?

Literatuur
Honeypot Project & Research Alliance, The (2005). Know your Enemy: Tracking Botnets, www.honeynet.org/papers/bots.
HP (2005). Press release: HP "Throttles" Viruses from the Network to the Desktop with New Security Software and Promising Research. 11 februari 2005. ProCurve Networking by HP, www.hp.com/rnd/news/virus_throttle_software.htm.
Singh, S. e.a. (2005). Automated Worm Fingerprintig, www.cs.ucsd.edu/~savage/papers/OSDI04.pdf.

Links
www.arbor.net/fingerprint-sharing-alliance.php: Fingerprint Sharing Alliance
www.fp6-noah.org: Noah (Network of Advanced Honeypots).

Herbert Bos
is universitair docent aan de Vrije Universiteit Amsterdam.
E-mail: herbertb@cs.vu.nl.