FOI-R--0843--SE
Mars 2003
ISSN 1650-1942
Användarrapport

David Lindahl, Mats Persson, Arne Vidström, Mikael Wedlin

Triops, prototyp för IT-vapen

Ledningssystem
581 11 Linköping

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Ledningssystem
Box 1165
581 11 Linköping

FOI-R--0843--SE

Mars 2003

ISSN 1650-1942

Användarrapport

David Lindahl, Mats Persson, Arne Vidström, Mikael Wedlin

Triops, prototyp för IT-vapen

Utgivare Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 Linköping	**Rapportnummer,** FOI-R--0843--SE	**Klassificering** Användarrapport
	Forskningsområde 4. Spaning och ledning	
	Månad, år Mars 2003	**Projektnummer** E7033
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 41 Ledning med samband och telekom och IT-system	

Författare/redaktör David Lindahl Mats Persson Arne Vidström Mikael Wedlin	**Projektledare** Mikael Wedlin
	Godkänd av
	Uppdragsgivare/kundbeteckning
	Tekniskt och/eller vetenskapligt ansvarig David Lindahl

Rapportens titel

Triops, prototyp för It-vapen

Sammanfattning (högst 200 ord)

Projektet IT-vapen i laborativ miljö har utvecklat en prototyp för IT-vapen. Denna döptes till Triops. Vapnet är en modulärt uppbyggd programvara som utvecklats för att testa olika aspekter av datorkrigföring. Genom att bygga programmet i modulär form kan vapnets egenskaper snabbt ändras samt de olika delarna kan utvecklas och testas separat. Dessutom medger konstruktionen att nya delar kan laddas ner under pågående angrepp så att Triopsen kan förändra sina egenskaper för att kunna anpassa sig till olika omgivningar. Triops är tänkt att vara ett system för att kunna få in ett angreppsprogram i en främmande dator. Prototypsystemet består av en leveransdel som sänder in en verkansdel i den främmande datorn. Detta kan betraktas på samma sätt som en konventionell vapenbärare i form av en raket eller robot. Robotkroppen har som uppgift att föra någon form av verkansdel inom räckhåll för ett fientligt vapensystem. Det är sedan verkansdelen som har till uppgift att tillfoga skada.

Nyckelord

IT-krig, Datorkrig, datorvirus,mask, nätverk, IT, Internet, Brandvägg

Övriga bibliografiska uppgifter	**Språk** Svenska
ISSN 1650-1942	**Antal sidor:** 17 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization	Report number,	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--0843--SE	User report
	Programme Areas 4. C4ISR	
	Month year March 2003	**Project no.** E7033
	General Research Areas 5. Commissioned Research	
	Subcategories 41 C4I	

Author/s (editor/s)	Project manager
David Lindahl Mats Persson Arne Vidström Mikael Wedlin	Mikael Wedlin
	Approved by
	Sponsoring agency
	Scientifically and technically responsible David Lindahl

Report title (In translation)

Triops, prototype of an It-weapon

Abstract (not more than 200 words)

The project IT-weapons in a laboratory environment has developed a prototype for IT-weapons. This was named the Triops. The weapon is a series of software modules that can be combined in different ways to test different aspects of computer warfare. By constructing the software in modules the weapons characteristics and capabilities can be changed quickly. Also, the modules can be developed and tested separately. This construction makes it possible for the software to download new modules during an attack in order to adapt to different environments (e.g. OS:s) The Triops is supposed to be a system that delivers an attack program into a specific computer. The prototype system consists of a delivery system that enters the target computer to deliver a payload. This can be viewed as a conventional weapon, e.g. a missile where the missile body houses a payload of explosives. The missile body transports the payload to a certain point and then the payload then causes damage to the target.

Keywords

CNO, CND, CNA, IW, IO, cyberwar, firewall, IT, computer virus, worm

Further bibliographic information	Language Swedish
ISSN 1650-1942	**Pages** 17 p.
	Price acc. to pricelist

Innehållsförteckning

1. Inledning

Denna rapport är tänkt att vara en sammanfattning av de projekterfarenheter som gjorts under framställandet av prototypen Triops. Ett IT-vapen avsett för laborativt bruk som forskningsplattform rörande IT-krigföring. Rapporten behandlar prototypens nuvarande utforming och varför den designats som den gjorts. Rapporten tar vidare upp förbättringsförslag samt i mindre grad alternativa designlösningar. En slutrapport kommer att ges ut senare i projektet vilken kommer att innehålla de generella slutsatser gruppen dragit under arbetets gång.

2. Översiktlig beskrivning av Triops

Triops är tänkt att vara ett system för att kunna få in ett angreppsprogram i en främmande dator. Systemet består av en leveransdel som sänder in en verkansdel i den främmande datorn. Detta kan betraktas på samma sätt som en konventionell vapenbärare i form av en raket eller robot. Robotkroppen har som uppgift att föra någon form av verkansdel inom räckhåll för ett fientligt vapensystem. Det är sedan verkansdelen som har till uppgift att tillfoga skada.

Triopsen är uppbyggd på samma sätt som en robot med separata leverans- och verkansdelar så att man snabbt kan uppdatera och byta ut dem. Den verkansdel vi framställt fungerar genom att den i sin tur försöker lura måldatorn att placera en bit av ett datorprogram på en plats i datorns minne där det kan köra. Detta datorprogram slutligen utför själva angreppet. Prototypen vi har tillverkat öppnar en förbindelse ut från den dator den startas på och kontaktar en annan dator (måldatorn) och angriper denna. Detta kan sägas motsvara scenariot att ett mål finns innanför en brandväg eller på ett separat nätverk. Vi kan med e-post eller en webbsida komma åt någon av datorerna innanför brandväggen och utnyttjar sedan denna dator som en bas för att kunna angripa den specifika måldatorn. En bättre analogi än en robot skulle kunna vara ett transportflygplan som släpper jägarsoldater bakom fiendens linjer vilka sedan infiltrerar fiendens territorium (nätverket bakom brandväggen) och slår mot specifika mål.

Triopssystemet är modulärt uppbyggt så att man kan skräddarsy delar för olika syften och olika tekniska målsystem. Att ha systemet uppbyggt modulärt innebär också att det blir lättare för utvecklarna att arbeta parallellt och att strukturera utvecklingen av framtida prototyper.

3. Leveransdelen

3.1 Översikt

Den nuvarande prototypen av leveranssystem fungerar så att själva leveransdelen består av HTML-kod som skickas i ett e-postmeddelande eller läggs in på en webbsida. Om webbläsaren som används för att visa HTML-koden har en viss säkerhetsbrist kommer ett datorprogram som ligger gömt i meddelandet/webbsidan att köras. Detta program utnyttjar en säkerhetsbrist i Internet Explorer till att placera ett datorprogram och ett antal datafiler i startup-foldern. Datorprogrammet (en så kallad scriptfil) kommer att köras nästa gång Windows startar. Det som då händer är att ett större körbart program skapas av datafilerna. Detta program är verkansdelen som utför någon form av attack.

3.2 Teknisk beskrivning av leveransdelen

Säkerhetsbristen i ActiveX-kontrollen scriptlet.typelib

Säkerhetsbristen i ActiveX-kontrollen scriptlet.typelib som används av Triops fungerar enbart om Internet Explorer 5.0 är installerad och inte har blivit säkerhetsuppdaterad.

Samma brist har använts av andra maskar i "vilt tillstånd". Det finns begränsningar i scriptlet.typelib som gör att det enda man kan skicka med direkt är relativt små WSH-scripts. Det utgör en begränsning då vapnet, eller verkansdelen i de hittills använda maskarna har varit mycket liten för att kunna fungera. Eftersom det var intressant att se hur lätt eller svårt det var att utnyttja denna brist till något större än bara en liten scriptmask så valdes den ut för Triops. Det visade sig möjligt med diverse olika knep komma förbi begränsningen och skicka in relativt kapabla verkansdelar. Tyvärr blev tillförlitligheten hos slutresultatet inte helt perfekt, utan körningen misslyckas vid några av testerna. Anledningen till detta har inte klarlagts, främst eftersom det inte finns så mycket information att tillgå kring hur de olika komponenter som utnyttjas fungerar i den här typen av sammanhang.

Det är ett generellt problem vid framställning av IT-vapen att dokumentation normalt inte ligger på den detaljnivå som utnyttjande av säkerhetsbrister kräver. Detta är å ena sidan ett skydd eftersom det kräver att kompetenta personer utreder exakt hur bristerna fungerar, men å andra sidan är det en svaghet eftersom samma noggranna utredning måste till för att hitta sårbarheterna över huvud taget. Hade systemen varit enklare konstruerade och bättre dokumenterade hade de som säkrar systemen haft lättare att hitta sårbarheterna.

Säkerhetsbrister i Outlook och Outlook Express

Säkerhetsbristen i ActiveX-kontrollen scriptlet.typelib som används i den här versionen av Triops utgör bara en av flera brister som är möjliga att angripa via e-post. Vilka brister som går att utnyttja i varje enskilt fall beror i första hand på vilket e-postprogram mottagaren använder, men också på hjälpprogram som e-postprogrammet använder i sin tur. Till exempel använder Outlook och Outlook Express webbläsaren Internet Explorer för att visa e-post i HTML-format, och därför går det potentiellt att utnyttja brister i Internet Explorer genom att skicka angreppskoden i HTML-format via e-post.

När Microsofts e-postprogram först kom ut rättades de flesta säkerhetsbrister som gick att utnyttja via dessa genom uppdateringar som åtgärdade varje enskild brist i sig. Därefter kom uppdateringar till Outlook 98 och 2000 för att förebygga hela grupper av brister. I de nyare versionerna Outlook 2002 och Outlook Express 6 är dessa åtgärder införda redan från början. Numera visas alltid email i zonen ”Ej tillförlitliga platser” där Active Scripting och ActiveX som standard är avstängda. Dessutom finns en spärr som gör att det inte går att starta exekverbara bilagor direkt i e-postprogrammet. Om det skulle dyka upp nya brister av liknande slag som många av de gamla så skulle dessa inte gå att utnyttja på grund av de nya generella spärrarna.

Men även om säkerhetsnivån höjts betydligt i Outlook och Outlook Express så vore det naivt att tro att det inte kommer dyka upp några nya allvarliga säkerhetsbrister. Detta eftersom i princip all programvara som inte är trivialt enkel är behäftad med diverse fel. Däremot är det inte helt osannolikt att det framöver kommer upptäckas färre allvarliga brister än i tidigare versioner så länge ingen upptäckt nya klasser av brister som berör de bägge programmen.

Generella slutsatser om val av e-postprogram

Förutom att öka säkerheten i ett e-postprogram genom att hålla det uppdaterat och väl konfigurerat så föreslås ofta andra sätt eller kombinationer av sätt att höja säkerhetsnivån. Exempelvis genom att välja ett program som

- inte har så hög marknadsandel, eller
- har haft ett lågt eller obefintligt antal kända säkerhetshål tidigare, eller
- är enkelt till konstruktionen.

Att välja ett program som inte har så hög marknadsandel kan ha fördelar i vissa lägen. Fientlig kod som är skriven för att spridas via e-post genom att utnyttja en eller flera säkerhetsbrister är normalt begränsad till ett visst e-postprogram. Det finns minst två huvudsakliga anledningar till detta. Den ena är att en specifik säkerhetsbrist oftast bara finns i ett specifikt e-postprogram. Den andra är att om ett visst e-postprogram har huvuddelen av marknaden så uppnår upphovsmannen (eller upphovsmännen) till den fientliga koden normalt sina syften utan att behöva lägga in stöd för flera olika e-postprogram. De mest uppmärksammade fallen av spridning av fientlig kod är sådana som har global omfattning och där upphovsmannen sannolikt uppnått sina syften även om inte genomslaget varit hundraprocentigt. Om man vill undvika att bli offer för den typen av fientlig kod så kan det löna sig att använda ett e-postprogram med låg marknadsandel. Däremot utgör detta inget skydd mot angripare som är ute efter ett specifikt mål och dessutom är kapabla att hitta nya säkerhetsbrister själva.

För Triops skulle detta sätt att välja mailklienter inte göra någon skillnad eftersom målet för Triops bara är att ta sig in i ett relativt begränsat antal system.

Att välja ett program som har haft ett lågt eller obefintligt antal kända säkerhetshål tidigare kan också ha sina fördelar. Tyvärr är det ofta så att förklaringen till den bra statistiken ligger mer i att ingen letat på allvar efter säkerhetshål i programmet snarare än att det skulle vara säkrare än andra program av samma typ. Detta är extra sannolikt om programmet har en låg marknadsandel. Anledningen till detta är dels direkt - färre användare ger lägre sannolikhet att någon hittar ett fel. Dels är den också indirekt, eftersom det inte ger lika stor uppmärksamhet för buggjägare att hitta säkerhetsbrister i en mindre känd programvara. Om det är få som har letat efter säkerhetshål kan det alltså tvärtom innebära att det finns säkerhetsbrister kvar som är enkla att hitta. Det gör att en någorlunda sofistikerad angripare får större möjligheter att hitta en brist att utnyttja som ingen annan vet om, och som därför inte rättas till.

För Triops skulle det vara en fördel om målet använder programvara som inte blivit noggrant genomsökt efter säkerhetsbrister eftersom det då är lättare att finna en brist som är aktuell i målsystemen.

Att välja ett program som är enkelt till konstruktionen, och samtidigt väl utvecklat, gör att mängden säkerhetsbrister minskar radikalt. Säkerhetsmässigt finns inga nackdelar med små enkla program så länge de inte är för små för att kunna utföra det de måste i säkerhetshänseende.
För Triops skulle det självklart vara en stor nackdel om målet använder program som är enkla till konstruktionen och dessutom väl utvecklade.

4. Verkansdel

4.1 Översikt

I den här versionen av Triops används en verkansdel som utnyttjar en säkerhetsbrist i form av en buffer overflow. Programmet kallas inject.exe. Detta är en direkt portning till Windows av attackprogrammet inject avsett för Linux. Verkansdelen reserverar en minnesarea på startdatorn, fyller denna denna med data som motsvarar en buffert hos ett av måldatorns program och öppnar en förbindelse via nätverket till måldatorn. Vår prototyp angriper enbart en dator den känner IP-numret till. Detta av säkerhetsskäl. En operativ Triops skulle troligen traversera nätverket för att leta upp lämpliga mål. När verkansdelen har kontaktat måldatorn skickar den över data från minnesarean och skriver över en buffert i ett specifikt program på måldatorn. En medveten buffer overflow skapas genom att den överförda datamängden är något större än bufferten på måldatorn kan ta emot. Detta ersätter delar av detta program med nya programinstruktioner. De instruktioner prototypen sänder in startar bara upp ett program från måldatorns hårddisk. En operativ triops skulle utföra någon form av attack specifik för det system som måldatorn tillhörde.

4.2 Teknisk beskrivning av Verkansdelen

Buffer overflow

När man programmerar behöver man ofta använda sig av ett datautrymme för att lagra till exempel indata eller utdata. Man kan i programspråken C och C++ allokera detta utrymme explicit med en speciell funktion, eller så sker det automatiskt när ett underprogram behöver en egen liten dataarea. Denna temporära area kallas för buffert och innehåller ofta textsträngar. Problemet är att programspråket C inte kontrollerar om man skriver data utanför buffertens gränser; denna kontroll måste läggas in av programmeraren och på något sätt säkerställa att ingen data hamnar utanför gränserna. Tyvärr är många programmerare slarviga och skriver inte program som utför denna kontroll.

Om data skrivs utanför gränserna kan detta få många olika konsekvenser. Det som händer är att indata skriver över de data som tidigare fanns i utrymmet ”bredvid” bufferten. Exakt vilka konsekvenser detta får beror på vad som skrivs över och vilken data som hamnar utanför gränsen. Detta blir ett potentiellt säkerhetshål eftersom känsliga data kan skrivas över. Bland annat kan återhoppsadressen skrivas över för den funktion som programmet just nu exekverar.

Det finns minst två typer av “buffer overflow”. Den ena kan uppkomma när programmet allokerar mer minne med funktionen “malloc”. Då används minne på “HEAP:en”, som är en dataarea som ligger direkt efter programkoden. Den andra typen av “overflow” uppkommer efter ett funktionsanrop där minnesutrymme allokeras på stacken, som är en dataarea som används för temporär lagring av data. En utförligare beskrivning av “buffer overflow” finns i [Wilander] och [McGraw].

Ett känt exempel på en “buffer overflow”-attack är den mask som Robert Morris var skyldig till 1988 [Spafford, “The Internet Worm Incident”]. Den utnyttjade en overflow i programmet “fingerd”, som är ett program med vilket man kan från en dator kan kontrollera om en person är inloggad på en annan dator.

Om det finns ett säkerhetshål, t ex i form av en “buffer overflow”, i en dator så kan detta hål utnyttjas för ett datorintrång. Intrånget består i att en kort bit programkod skickas in, *injiceras,* genom detta säkerhetshål och fås att exekvera och utföra lämplig verkan. Med injicering menas det steg där koden skickas från angriparens dator till måldatorn, och in i bufferten. Programkoden skickas med hjälp av ett speciellt injektionsvertyg.

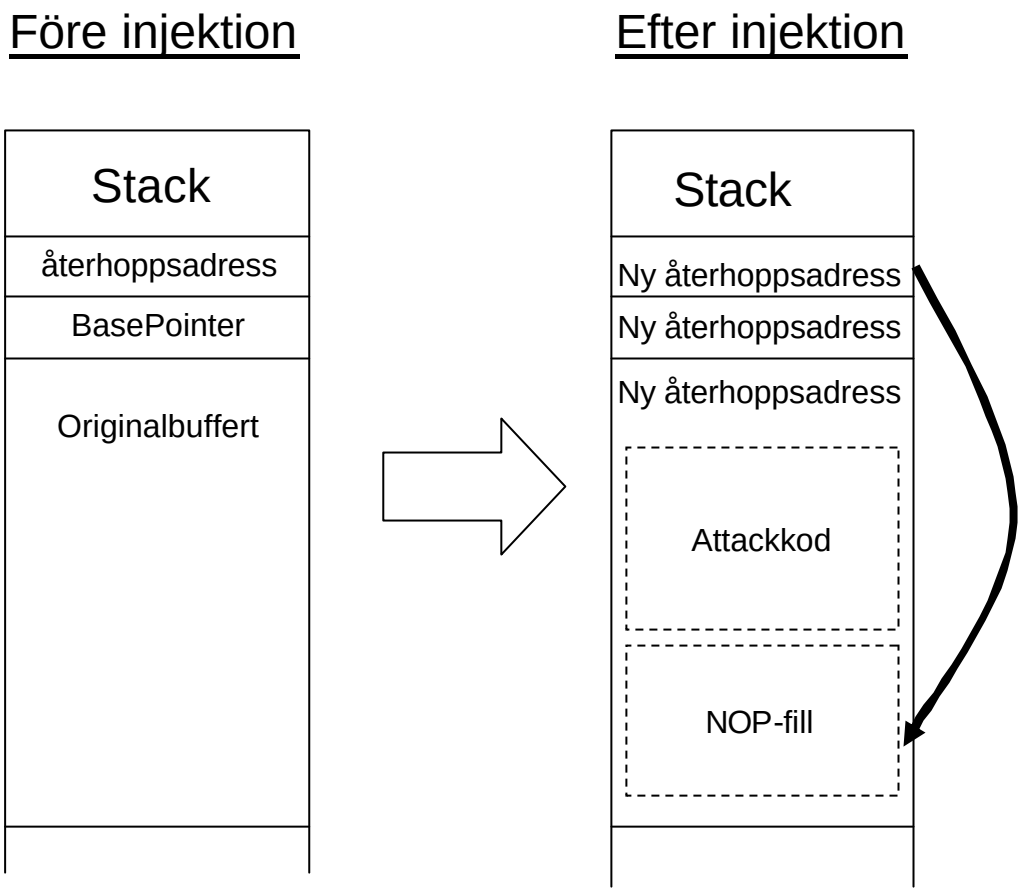
Det finns även andra säkerhetshål som kan utnyttjas på motsvarande sätt. Ett annat vanligt sätt är “format string attack”. Kortfattat innebär detta att man utnyttjar fel som en programmerare gjort när hon använde programspråkets C:s printf-funktion. [Wilander].

Injektionsverktyget

Injektorn är ett program skrivet i C. Programmet öppnar en förbindelse till måldatorn och injicerar attackkoden, bestående av ägget (se nedan) och verkansdelen. Injektorn behöver fyra inparametrar: namnet eller adressen till måldatorn, portnummer där serverprogrammet lyssnar, storlek på bufferten i servern och återhoppsadressen för den buffer overflow som utförs. Dessutom behöver givetvis injektorn även attackkoden som skall injiceras. Ett datorprogram som utnyttjar en viss sårbarhet, plus adresser och attackkod, kallas med ett mycket vanligt slanguttryck för “exploit”. Denna term har blivit vanlig även på svenska och kommer troligen att få status som ett låneord.

Tekniskt sett utför injektorn följande steg:

1. Reserverar en minnesarea, något större än bufferten i måldatorn, som skall innehålla attackkoden.
2. Minnesarean fylls med den nya återhoppsadressen och attackkoden. Slutet av minnesarean är den del som kommer att skriva över den egentliga återhoppsadressen i måldatorns buffert.
3. Den första delen av minnesarean fylls med NOP-instruktioner. Det är till denna area återhoppsadressen kommer att peka. Anledningen till att fylla en del av måldatorns buffert med NOP är att man kan inte vara säker på var bufferten hamnar i minnet och får gissa på återhoppsadressen. Ett tillräckligt antal NOP ger lite utrymme för felmarginal.
4. Attackkoden, ägget och verkansdelen, läggs någonstans i mitten av minnesarean, efter NOP-arean. Attackkoden finns för närvarande inkompilerad i injektorn.
5. En TCP-förbindelse öppnas till måldatorn på den angivna porten.
6. Hela minnesarean skickas genom förbindelsen till måldatorns öppna port och skriver över bufferten och den gamla återhoppsadressen i måldatorn.
7. Injektorn väntar på eventuellt svar från måldatorn, men avslutar efter några sekunder om inget svar inkommit.



Problem vid injicering

När attackkoden injiceras i måldatorn kan det uppkomma en del problem. Främst är dessa beroende på hur programmen på måldatorn exekveras och var de placeras i minnet. Detta kan variera med olika versioner av operativsystemet, och främst version på kärnan.

Storlek på programkoden

Den möjliga storleken på attackkoden är direkt beroende av storleken på bufferten i måldatorn. Ofta brukar maximalstorleken på attackkoden även begränsas av att bufferten innehåller annan text eller data. I allmänhet brukar programmerare allokera buffertar med storlek kring tusen bytes när de skriver rutiner för nätverkskommunikation. Det kan dock variera mellan några hundra och några tusen bytes beroende på tillämpning.

Detta gör att man bör skriva en attackkod som är mindre än cirka 250 bytes. Det enda man kan göra på det utrymmet är att avkoda maskinkoden och göra några systemanrop. Vill man göra mer avancerade saker måste attackkoden själv ladda hem mer kod. Mer om detta i nästa kapitel.

Placering i buffert

Ett problem är var i bufferten attackkoden bör placeras. Under programmering av attackkoden vill man gärna att mängden NOP-instruktioner ska vara så liten som möjligt, eftersom det underlättar felsökning med en “debugger”. Däremot kan man i skarpt läge fylla upp en större del av bufferten med NOP-instruktioner så att koden hamnar i slutet av bufferten, följt av cirka 20 återhoppsadresser.

Återhoppsadressen

Angreppet sker när programmet i servern exekverar inuti en funktion. Denna har på ett eller annat sätt blivit anropad från en annan plats i programmet och lägger därför en returadress

på stacken. Returadressen är den minnesadress som programmet skall återvända till när funktionen avslutar,Denna adress kallas även återhoppsadress. Den buffert som injiceras ligger också på stacken eftersom funktionen använder bufferten. Slutet på bufferten ligger minnesmässigt nära återhoppsadressen och vid en attack skrivs några bytes över stacken så att återhoppsadressen ändras. När funktionen då avslutar hoppar den in i attackkoden istället.

Hur skyddar man sig mot den här typen av angrepp?

Det finns ett antal metoder att skydda sig mot buffer overflows. Några exempel: proaktiv analys av programkod, aktivt skydd i form av filtrering av inkommande data från nätverket, modifiering av operativsystemet kallat “rebasing”.

- Programanalys görs innan man driftsätter servern. Det innebär att man analyserar programkoden med ett särskilt verktyg som pekar ut känsliga funktioner eller potentiella säkerhetsbrister. Till exempel varnar den för använding av `sprintf` på buffertar.
- Ett mailfilter, en brandvägg, intrångsdetekteringssystem eller ett antivirusprogram, kan kontrollera inkommande data från nätet. Om datana innehåller möjliga virus eller signaturer för attackkod, kan dessa datapaket filtreras bort. Det finns även speciella skydd som bevakar stacken och signalerar om den används på fel sätt.
- Ett annat sätt att skydda sig är att placera stacken på andra adresser än de normala, vilket gör att det blir svårt att gissa lämplig återhoppsadress. (Se tidigare stycke om återhoppsadressen). Det finns flera andra sätt att göra det svårare för attackkoden att exekvera. Till exempel kan man ändra placering av biblioteksrutiner, flytta HEAP-arean, eller ändra numreringen av systemanrop.

Preparerat säkerhetshål - badserv

För att kunna demonstrera vår prototyp behövs en server med ett säkerhetshål i form av en buffer overflow. Det finns ett antal kända buffer overflows i en del programvaror. Till exempel har imapd i Red Hat Linux 6.2 ett säkerhetshål, och sendmail har haft ett antal. Ett problem är att hitta källkoden till dessa gamla program eftersom källkoden ofta tas bort från publika ftp-servers. Ett annat problem är att tillgängligt buffertutrymme är ganska litet.

En annan sak man bör tänka på är att det kan vara en dålig idé att utveckla attackkod som skulle kunna fungera ute på Internet. Om vår attackkod skulle komma ut skulle den orsaka negativa konsekvenser.

Därför har vi utvecklat en egen server, kallad “badserv.c”, med ett förpreparerat säkerhetshål i form av en buffer overflow. Servern lyssnar på en särskild port och tar emot textsträngar. Dessa strängar läses in i en läsbuffer “rbuf” och skrivs sedan till skrivbuffern “wbuf” med funktionen `snprintf`. Läsbuffern har ett utrymme på 20000 tecken och skrivbuffern har 10000 tecken. En medveten “felskrivning” vid anrop av `snprintf` gör att 20000 tecken skrivs till skrivbuffern och gör det möjligt att utnyttja denna säkerhetsbrist för en buffer overflow. Eftersom funktionen `snprintf` vanligtvis räknas som säker vid programanalys, finns det risk för att denna brist inte upptäcks.

Status hos verkansdelen i dag

Verkansdelen är den del som gör det egentliga jobbet, och är en del av den attackkod som injiceras. Attackkoden består alltså av ett ägg och en verkansdel. Den senare kan innehålla avancerad kod för spridning och injektion i andra datorer, eller den kan vara ett enkelt anrop till något eller några systemfunktioner för att åstadkomma någon slags verkan i målet. De nuvarande fungerande prototypen innehåller enbart kod som startar annat valfritt program på måldatorn.

Designval

En tidigare version av Triops var byggd som shell- och Perl-scripts. Anledningen till att bygga den på detta sätt var att det blev kodningsmässigt mycket enklare. Perl är vanligtvis lämpligt för att bygga prototyper. Nackdelen med denna version är att den var ganska långsam och drog mycket datorkraft. Dessutom hade angreppet ett ganska stort fotavtryck, och var lätt att detektera. Fördelen var att den lätt kunde modifieras och utökas. Den kunde även sprida sig från dator till dator, men den utnyttjade inte något vanligt säkerhetshål som "buffer overflow". [Memo]

Nästa mål blev då att skriva en Triops som var snabbare och mindre, samt kunde utnyttja ett äkta säkerhetshål. Eftersom det vanligaste hålet är buffer overflow så valdes denna angreppsmetod. För att bli enklare att tillverka och modifiera var det nödvändigt att använda ett högnivåspråk som t ex C. Alternativet hade varit att programmera direkt i assembler, vilket kan vara ganska tidskrävande. Detta är dock det normala för denna typ av angrepp, men man kan med hjälp av gcc och speciella optimeringar generera tillräckligt bra kod.

En mellanversion av Triops hade med en interpretator i verkansdelen, men det visade sig att detta blev för otympligt. Denna skulle dock vara nödvändig om triopsen behöver göra mer avancerade operationer. Interpretatorn skulle kunna laddas ner dynamiskt istället.

Ägget

Den första sekvensen av instruktioner i attackkoden består av ägget. Detta har till funktion att initiera och avkoda resten av Triopsen eftersom den är xor-kodad för att kunna ta sig in i servern (se kapitel om xor-kodning).

Det första som händer i ägget är att programmet hoppar till slutet, och sedan gör ett anrop tillbaks till början, och därefter en pop av stacken för att få ut återhoppsadressen. Detta för att ta reda på var i minnet ägget hamnade. Efter detta läser den in två bytes med längden på verkansdelen och en byte med det värde koden har xor-kodats med. I en loop omkodas sedan verkansdelen, och slutligen hoppar programmet in i verkansdelen med en `ret`.

Ett problem med att ha ett likadant ägg vid alla angrepp är att det blir lätt att detektera av en IDS som ställts in för att leta efter det. Det har visserligen en låg profil men en signatur som är lätt att känna igen. En idé vore att variera äggets utseende vid varje angrepp, vilket kan göras i injektorn. Ett sådant polymorft ägg kan sättas ihop från ett antal kodbitar med varierande utseende som gör samma sak som nuvarande ägg.

Xor-kodning - problem med nolla

Slutet av en sträng i programspråket C markeras vanligtvis med ett NULL-tecken (karaktärsvärdet 0 eller "nolla"). Många av strängoperationerna avslutar när ett NULL-tecken påträffas, till exempel i funktionen "strcpy" som kopierar en sträng till en annan buffer. Därför är det en dålig idé att ha med NULL-tecken i attackkoden. Detta problem kan man undvika på två sätt. Antingen kodar man assemblerkoden så att den inte genererar några NULL-tecken i maskinkoden, eller så kodar man attackkoden. Ett sätt är att göra detta är genom att göra xor med en särskild konstant på alla bytes. Denna konstant väljs så att inga NULL-tecken uppkommer vid xor-kodningen. En motsvarande avkodning utförs innan koden exekveras.

Modularitet

Ett av de tidiga målen med Triops var att den skulle vara modulär. Detta hade två syften. Det skulle vara lätt att byta ut komponenter för att anpassa den till nya situationer och det

skulle göra strukturen mer lättbegriplig. För närvarande är Triops uppdelad i några moduler: injektorn, ägget, verkansdelen, samt de moduler som finns i leveranssystemet.

En poäng med modularitet och högnivåspråk i modulerna är att de blir lättare att anpassa till nya situationer. Säkerhetshål av typen buffer overflow dyker upp regelbundet, men ofta fixas hålen ganska snabbt så det gäller att snabbt kunna modifiera modulerna så att de är aktuella.

Hur tillverkas en Triops? Vilka delar behövs för att tillverka en triops?

Stora delar av Triops är skrivet i programspråket C. För operativsystemet Linux finns en kompilator ‘gcc’ som normalt sett genererar exekverbara filer. Dessa duger inte som attackkod eftersom de blir ganska stora och är beroende av biblioteksrutiner. Istället använder vi en annan metod (se nedan).

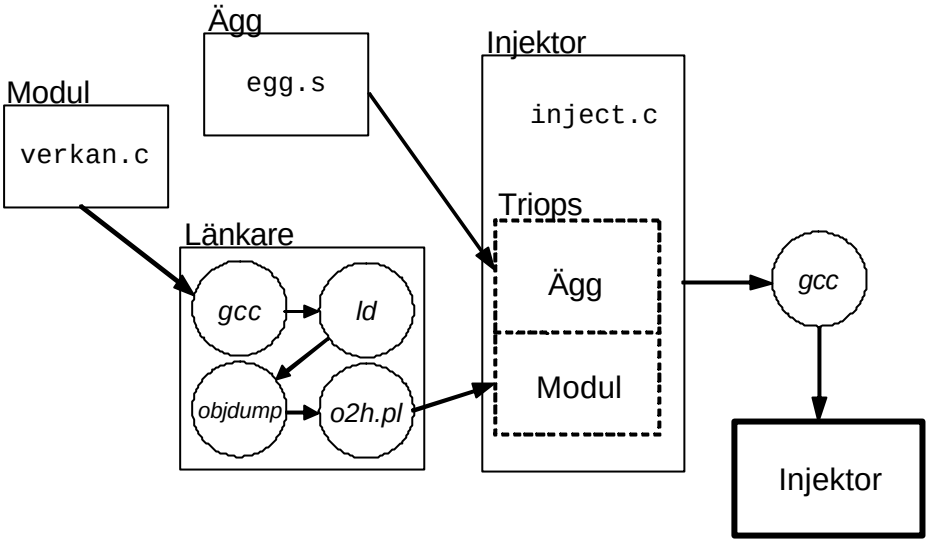
Som utgångspunkt har vi ett antal krav:

- Verkansdelen bör bli så liten som möjligt.
- Den ska vara oberoende av stora biblioteksrutiner.
- Koden skall vara relokerbar. Det innebär att koden ska vara möjlig att placeras och exekveras i vilket minnesområde som helst.

Om man vill ha en liten mängd kod bör man optimera koden med flaggan -O och ej heller statiskt länka in en mängd biblioteksrutiner. Om den ska vara helt oberoende de senare får man skriva egna rutiner för dessa. Om koden skall vara relokerbar skulle man kunna använda flaggan -fPIC till gcc, men det visade sig att denna inte alltid genererade bra kod. Istället får man skriva hela programmet som en enda stor nästlad funktion, med inre funktioner. Detta gör att alla hoppinstruktioner och rutinanrop blir relativa. Strängkonstanter blir normalt inte heller relokerbara utan istället måste man använda en något krånglig metod med assembler och labels. Inuti C-koden lägger kan man lägga in nedanstående rader som låter strängpekaren s peka på texten ”TRIOPS”.

```
asm(” call label1
  .ascii \”TRIOPS\”
  .byte 0
label1:
  pop %0
  ”:”=g” (s));
```

Själva tillverkningen av en injektor för Triops kan ses i följande figur:



I figuren ovan ligger verkansdelen i modulen ’verkan.c’ och ägget i ’egg.s’. Verkansdelen skickas till länkaren. Ägget är i det format som krävs för att direkt kunna läggas in i injektorn. Länkaren är ett samlingsnamn för ‘gcc’, ‘ld’, ‘objdump’ och ‘o2h.pl’. Dessa kompilerar en källkodsfil i C och översätter den till hexkod. ‘gcc’ är C-kompilatorn och ‘ld’ är objektkodslänkaren som här används för att generera användbar maskinkod. Programmet ‘objdump’ plockar ut maskinkoden ur objektkodsfilen och visar den som hexadecimala värden. Programmet ‘o2h.pl’ översätter hexkoden till ny C-kod som sedan kan läggas in i injektorn. Denna kompileras till slut av ’gcc’ och kan sedan exekveras.

Biblioteksrutiner

Triopsen kan inte använda sig av biblioteksrutiner i glibc eftersom den inte känner till på vilka adresser dessa ligger. Därför skrivs egna vilka blir ganska enkla. De använder sig ofta av systemanrop, vilka man kommer åt med assemblerinstruktionen “int 0x80h” i Linux. Ett exempel på en sådan rutin är “execve” som startar valfritt annat program på datorn:

```
void execve(const char *filename, char *const argv[], char
*const envp[])
{
  asm(“
  xor %%eax,%%eax
  mov $11, %%al
  mov 8(%%esp), %%ebx
  mov 12(%%esp), %%ecx
  mov 16(%%esp), %%edx
  int $0x80
  “:::"eax”);
}
```

5. Framtida utveckling

Vidareutveckling av injektionsdel, attackkod och verkansdel

Det finns flera sätt att vidareutveckla Triops. Det skulle vara möjligt att tillverka ett mer generellt injektionsverktyg där detta läser in parametrar, attackkod och angreppsinstruktionen från en fil. Dessutom skulle verktyget kunna pröva flera mål och olika buffertstorlekar eller återhoppsadresser. Den skulle eventuellt kunna förändra attackkoden från gång till gång så att den undkommer detektionssystem (polymorfism).

En mer avancerad injektor skulle kunna läsa in attackkod från fil och sedan skicka in denna. Den skulle även kunna utnyttja heap-overflows eller format string-attacker. Ännu bättre vore det om injektorn kunde bestämma typen på målet och därav tillverka en lämplig attackkod. Det har även nämnts ett mer avancerat ägg som kunde undgå detektion.

En vidareutvecklad verkansdel skulle kunna innehålla möjlighet för dynamisk nerladdning av ytterligare attackkod. Den initiala verkansdelen som injiceras i målet skulle då endast innehålla kod för att koppla upp en förbindelse tillbaka till föregående dator, kod för hämtning av ytterligare kod och en enkel symboltabell för att hålla ordning på modulerna. Sedan laddas en enkel interpretator och fler moduler ner och installeras i symboltabellen. Även en injektormodul och servermodul laddas ner, och med denna kan Triopsen sedan sprida sig vidare.

Alternativa leveransdelar

Även om vi valde att använda email som bärare i Triops så finns det andra bärare som skulle gå att använda istället. Ett intressant exempel är att utnyttja säkerhetsbrister i mediaspelare av olika slag. Just den typen av brister och hur dessa skulle kunna utnyttjas av maskar beskrevs till viss del i januari 2003 när en grupp som kallar sig Gobbles Security postade till säkerhetsmailinglistan Bugtraq om ett nyupptäckt hål i mp3-spelaren mpg123. [BugTraq 1] Gruppen utgav sig för att ha konstruerat en mask, Hydra, åt skivbranschens organisation RIAA för att den skulle sprida sig och samla information om alla som har piratkopierad musik i sina datorer. De flesta experter förstod att det bara var en bluff och efteråt erkände också Gobbles Security att så var fallet enligt en artikel i Wired News [Wired].

Det intressanta med Hydra är att själva konceptet skulle kunna fungera i praktiken. Gobbles Security upptäckte ett verkligt säkerhetshål i mp3-spelaren mpg123 som gör att en specialkonstruerad mp3-fil kan innehålla kod som kör igång exempelvis en mask i datorn av att man spelar upp mp3-filen. Om en angripare hittar liknande hål i de vanligaste spelarna (som gruppen bluffade om att ha gjort) så skulle denne kunna bryta sig in i en stor mängd datorer enbart genom att sprida specialkonstruerade mp3-filer. Även om många användare förstår riskerna med att köra igång opålitliga program så är det sannolikt mycket få som förstår riskerna med att spela upp t ex en ljudfil.

För att slippa använda en säkerhetsbrist i programvara skulle en angripare istället kunna föra in manipulerade CD-R eller DVD-skivor med programvara på. En annan väg vore att på något sätt manipulera programvara som laddas ner från Internet.

För att kunna skydda sig mot alla dessa metoder måste man alltså ha kontroll både på programvara som man tar in samt på ren data som skulle kunna vara specialkonstruerad för att utnyttja säkerhetshål i de program som används för att tolka dessa.

6. Slutsatser och erfarenheter

Vad är egentligen nytt i Triops? Vilka nya ideer finns? Vilka erfarenheter har vi fått?

- Det som är nytt är att attackkoden är skriven direkt i C vilket gör att den kan kompileras. Tidigare har exploits varit kodade i assembler. Dessutom är Triops oberoende av operativsystemets biblioteksrutiner. Nytt är också omkodning av bytes för att undvika NULL-tecken.
- Triops skulle förmodligen fungera bra om det finns ett lämpligt säkerhetshål (tex en Buffer Overflow) och det inte appliceras några säkerhetsuppdateringar särskilt ofta.
- Det tog längre tid än beräknat att koda triopsen. Speciellt när vi kodade programrutiner i assembler men även i C som ligger på gränsen för att anses tillhöra högnivåspråken. Detta är dock till viss del även beroende på att vi sysslar med forskning och därmed provar nya lösningar på nya problem.
- Säkerhetshål av typen buffer overflow är svåra att utnyttja. Vapnen måste hållas väldigt ”skarpa” och det finns inte färdiga verktyg för detta. Manuell programmering måste alltså ske kontinuerligt för att modifiera vapnen.
- Det är teoretiskt (och förmodligen även praktiskt) möjligt att tillverka ännu mer avancerade maskar än denna Triops. Det är dock lite som att famla i mörkret, det verkar vara ett outforskat område. Potentialen för dessa som vapensystem är beroende på vilka datorsystem framtida militära satsningar resulterar i.
- För närvarande finns det en enkel men fungerande version av Triops. Den klarar enbart av att injiceras i en måldator och på denna exekvera ett annat lokalt program på måldatorn.

8. Referenser

[Wilander] John Wilander, "Security Intrusions and Intrusion Prevention", master thesis Linköping University, April 15 2002, http://www.ida.liu.se/~johwi/msc_thesis/Masters_Thesis_John_Wilander.pdf

[McGraw] John Viega and Gary McGraw, "Building Secure Software", ISBN 0201-72152-X, September 2001

[Spafford] Eugene Spafford, "The Internet Worm Incident",{ESEC}`89 2nd European Software Engineering Conference, 1989, http://citeseer.nj.nec.com/spafford91internet.html

[Memo] David Lindahl, Mats Persson, Arne Vidström, Mikael Wedlin, "Projekt IT-vapen 2002", H386

[Bugtraq] http://www.securityfocus.com/archive/1/306476/2003-01-11/2003-01-17/0

[Wired] http://www.wired.com/news/infostructure/0,1377,57229,00.html