

TDK Dolgozat

Horváth Botond

Dunaújvárosi Főiskola

Mérnökinformatika szak

Számítógéprendszerek és Irányítástechnika Tanszék

Készítette: Horváth Botond (2009-es évfolyam)

Konzulens: Dr. Leitold Ferenc főiskolai tanár, Hadarics Kálmán mérnöktanár

TDK cím: Operációs rendszerek sebezhetőségeinek vizsgálata automatizált környezetben

Operációs rendszerek sebezhetőségeinek vizsgálata automatizált környezetben



Tartalom

Kivonat	4
Bevezetés.....	5
Exploitok	5
Az exploitok osztályozása	6
Proof of Concept (röviden PoC).....	6
Local exploit.....	7
Metasploit Framework	7
Meterpreter.....	7
Msfpayload	8
A rendszer felépítése és működése.....	8
A vizsgálati környezet strukturális felépítése és tesztelési módszere	9
A rendszer kialakítása	10
A kliens kialakítása	11
A vizsgálati környezet automatikus működésének folyamatábrája.....	13
Az eredmény és annak értékelése.....	14
Összefoglalás.....	16
Irodalomjegyzék	17

Kivonat

Az utóbbi években a kártékony kódok terjesztése drasztikusan megváltozott és a feketepiacokat ellepték a biztonsági hibákat kihasználó exploitok, melyeket később kártékony kódok terjesztésére vagy épp zombi hálózatok kialakítására használnak. Manapság a feketepiacon egy Adobe Flash sebezhetőséget kihasználó exploit-ért akár több ezer dollárt is elkérnek, mely annak fényében nagyon is aggasztó, hogy a felhasználók több mint 98%-nál megtalálható. Ebből kifolyólag a kutatók számára létfontosságúvá vált e biztonsági hibák mielőbbi feltárása, az exploitok vizsgálata és készítése, hogy az Antivirus termékek működése mind pontosabbá váljék, és ami a legfontosabb, az adott biztonsági hiba mihamarabbi megszüntetése. Előadásomban rávilágítok a biztonsági hibák kihasználásának vizsgálatába, az exploitok készítésének menetébe és bemutatok egy olyan automatizált keretrendszert mely segítségével e kódok tesztelése drasztikusan lecsökkenthető.

Bevezetés

Régen a kártevők floppykon és cd-ken terjedtek, manapság ez teljes mértékben megváltozott és első számú hordozóként az internetet tekinthetjük. A támadás irányulhat egy program ellen, vagy ami a statisztikákat tekintve a könnyebb a felhasználó ellen. A szoftverek biztonsági hibáinak feltárása ez által nagyon fontossá vált. További veszélyt a böngészők és az azokhoz szorosan fűződő futtatókörnyezetek jelentik azáltal, hogy ezek egy egységes környezetet alkotnak és működésük (tehát programozásuk) a különböző platformokon ugyanúgy történik. Tehát a biztonsági kutatók számára a biztonsági hibák detektálása mindnél fontosabbá vált. A kutatókat a naponta több 100 sebezhetőség vizsgálata új megoldásokra sarkalja. Erre kívánok megoldást nyújtani a rendszeremmel, mellyel a biztonsági hibák tesztelése teljesen automatikussá válik és így a tesztelési idő drasztikusan lerövidül. A dolgozatomban kitérek a sebezhetőségek vizsgálatára, majd bemutatom a teljes rendszert. Kiemelném, hogy mivel az exploit írásáról nincs jó minőségű, hiteles egységes dokumentáció, így minden a szakdolgozatban leírt tudás illetve technika a saját tapasztalom alapján írtam le.

Exploitok

Sokakban téves elképzelés él az exploit-okat illetően. Lényegében az exploit egy program, amely a kódban található biztonsági hibákat (sérülékenységet) használja ki. Az exploit tehát egy olyan kód vagy bináris program, amelynek felhasználásával a rosszindulatú támadó kihasználhatja az exploit céljául szolgáló programozási hibával rendelkező programok sebezhetőségét, elsősorban web böngészőket és az azokhoz szorosan fűződő futtatókörnyezeteket. Léteznek olyan weboldalak, melyek az ingyenes letöltés lehetőségével vonzzák magukhoz a látogatót, ám letöltés helyett csak a weboldalba elhelyezett exploit települ számítógépére. Sajnos nagyon sok internetes oldal kódja feltörhető, és ebből kifolyólag biztonságos oldalakba is elhelyezhető kártékony kód, ezt a helyzetet bonyolítja, hogy sokszor nem feltétlenül kell, magának a weboldalnak a kódjába belenyúlunk gondoljunk csak az XSS alapú támadásokra[5]. Az exploitok egy részét nagy tudással rendelkező biztonsági szakemberek készítik a rendszerek védelmi képességének, biztonsági szintjének tesztelésére, felmérésére, míg másik részét szintén nagy tudással rendelkező, rossz szándékú programozók, akik nem tévesztendő össze az exploitokat letöltő, lefordító, és azokat rossz célra használó, de emellett csekély (vagy, ahogy általában lenni szokott

semmilyen) programozói tudással rendelkező script kiddie-kel. Az exploitokat funkció, felhasználás szerint csoportosítjuk. Fontos megemlíteni, hogy az exploitok írásához elengedhetetlen az x86 assembly ismerete, továbbá egy minimális windows shellcode és valamilyen scriptnyelv (a legtöbb exploitot ruby nyelven írtak).

Az exploitok osztályozása

Proof of Concept (röviden PoC)

Az exploitok speciális fajtáját képviselik a szakdolgozat témáját is adó úgynevezett proof of concept exploitok, azaz olyan programok, amelyek egy bizonyos hiba kihasználásának lehetőségét bizonyítják. A proof of concept exploitok nagy általánosságban nem ártó céllal készülnek[2]. Egyik nagy előnyük, hogy gyorsan elő lehet állítani őket, ez természetesen azért fontos, hogy mindnél hamarabb be lehessen foltozni az adott biztonsági hiányosságot az adott szoftverben. Mind mindennek, ennek is megvan a hátránya, konkrétan a hordozhatóság.

Rengeteg publikus exploit tölthető le az internetről, de pont egy részénél kiderül, hogy nem minden esetben működik. Ennek oka a hordozhatóságban, a hibás kódban vagy egyszerűen abban rejlik, hogy az exploit nem képes valamilyen védelmi mechanizmust megkerülni. Ekkor kénytelenek vagyunk magunk módosítani az exploitot, hogy az képes legyen lefuttatni a mi környezetünkben is. A leggyakoribb hibát a lent látható kódrészlet demonstrálja. A kódrészlet egy Internet Explorer 9 esetén fellépő veremtúlsordulásos sebezhetőséget használ ki. A piros színnel kiemelt rész az svchost kezdőcíme a memóriában. A problémát az okozza, hogy a készítő csak Windows 7 RTM verzióval tesztelte, ami esetében működik, de ugyanezen kód Windows 7 SP1 verzió esetén BSOD(Blue Screen of Death)-hoz vezet, amit nem dokumentált, így a tesztelés kifejezetten nehézé válik.

```
js_pivot = <<-JS
var heap_obj = new heapLib.ie(0x20000);
var code = unescape("#{js_code}");
var nops = unescape("#{js_nops}");

while (nops.length < 0x80000) nops += nops;
var offset = nops.substring(0, #{my_target['Offset']});
```

1. ábra: Egy bedrótozott kezdőcím érték az exploit kódjában.

Az exploit hiányosságát persze nem nehéz pótolni, elegendő csak egy feltételes ágba beilleszteni a többi verziójú Windows visszatérési címértékeket.

Local exploit

A lokális, vagyis az áldozat gépen közvetlenül futtatott exploit hozzáférést biztosít a sérülékeny rendszerhez és általában privilégium emelést hajt végre annak érdekében, hogy rendszergazdai jogokat szerezzen[8].

Metasploit Framework

A keretrendszert még 2003-ban adta ki HD Moore amit 2007-ben a Rapid7 megvásárolt. Azóta két verziója létezik, egy ingyenes és egy fizetős verzió, ami lényegében csak az állandó szupportban nyújt többet ingyenes társánál[10]. A Metasploit Framework-öt számítógépes hibák tesztelésére használják a biztonsági kutatók. Nagyon fontos, hogy a keretrendszer által használt exploitok működése szabadon paraméterezhető és beállítható, hogy a sikeres támadás esetén milyen parancs vagy parancsok hajtódjanak végre[9]. A következőekben bemutatom azokat a komponenseket, melyeket az általam kifejlesztett automatizált rendszerben is aktívan használlok.

Meterpreter

A Meterpreter a Metasploit Framework egyik összetevője, egy fejlett payload, egy "memória-rezidens host exploitáló modul". Egyszerűen bővíthető a hozzá fejlesztett bővítményekkel. A Meterpreter egyik fő sajátossága, hogy mind maga, mind a hozzá írt kiterjesztések kizárólag a megtámadott host memóriájában hajtódnak végre, egyáltalán nem érintve a merevlemez (kivéve, ha nincs más lehetőség), így a jelenléte rejtve maradhat a megszokott Antivirus megoldások és lemez alapú analizáló, nyomkereső technikák előtt. Mivel a detektálás hagyományos eszközökkel bonyolult, szinte lehetetlen a támadást detektálni és szinte képtelenség tudni, hogy a Meterpreter-rel megtámadott gépen mit csinál a támadó. Az alábbi ábrán a payload szerkezeti megvalósítását láthatjuk.

Memória (Stack)		
Payload eleje	EIP	
Eredetileg allokált puffer	Payload része	Ugrási cím

2. ábra: Payload szerkezeti felépítése

Msfpayload

A Metasploit Framework tartalmaz néhány kiegészítőt a támadás egyedivé tételére, hogy az Antivirus termékek nem ismerhessék fel a kártékony kódot. Az egyik ilyen eszköz az Msfpayload, mellyel képesek vagyunk új vagy a jelenlegi payload-ból szinte teljesen újat csinálni. Különlegességét az is jelzi, hogy bár az msfconsole-ból is elérhető az eszköz, de a paramétereinek és tudásának csak töredéke használható[11].

```
msfpayload windows/shell_bind_tcp EXITFUNC=seh LPORT=1234 C
/*
* windows/shell_bind_tcp - 341 bytes
* http://www.metasploit.com
* VERBOSE=false, LPORT=1234, RHOST=, EXITFUNC=seh,
* InitialAutoRunScript=, AutoRunScript=
*/
unsigned char buff[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\x31\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
```

3. ábra: shell_bind_tcp kódja

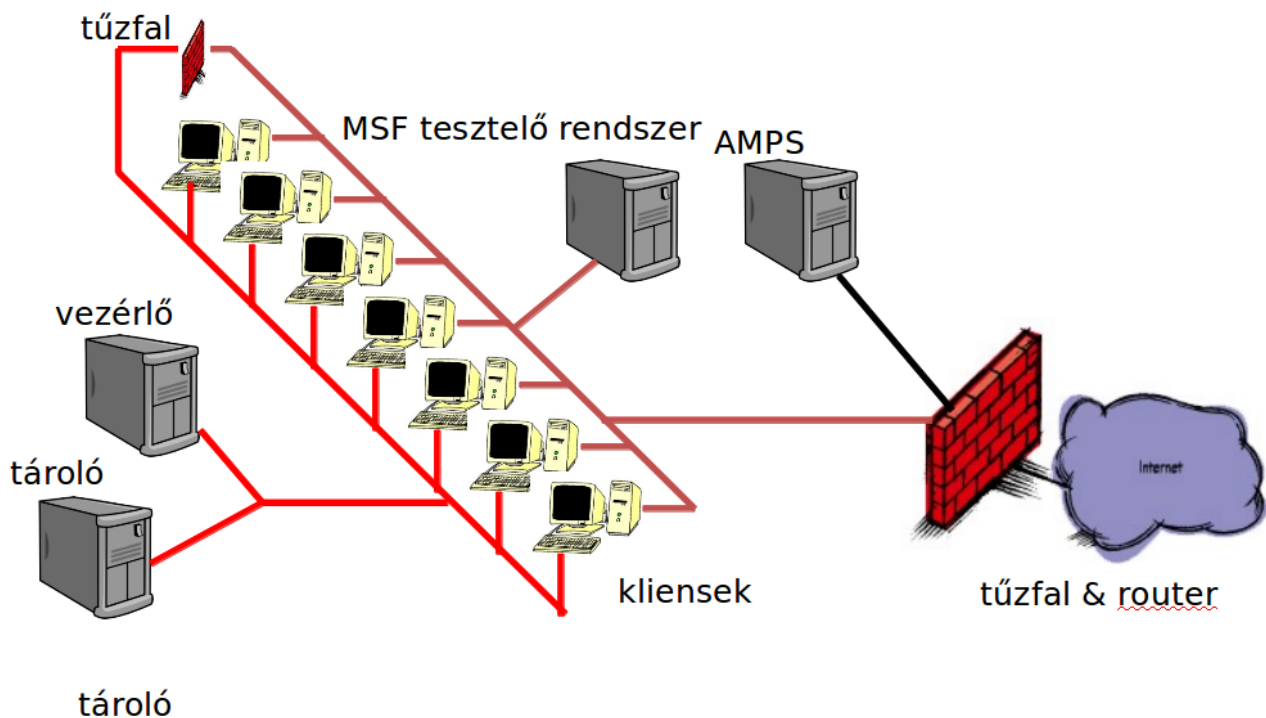
Itt a shell_bind_tcp payload-nak az egyik variációját láthatjuk. A konfigurálása nagyon egyszerű, de paraméterek listája szinte kimeríthetetlen. A payloadok készítésekor assembly tudásra van szükség, ezért nem nagyon találni helyes leírást a teljes funkcionalitásról.

A rendszer felépítése és működése

A bevezetőben tárgyalt probléma, hogy a nagy számban publikált és felfedezett sebezhetőségek nagy száma miatt a tesztelés emberi erővel szinte lehetetlen. Ezért az alapkoncepció egy olyan automatizált rendszer kifejlesztése volt, amelynek segítségével a megjelent exploitok tesztelése emberi beavatkozás nélkül működjön és az eredmények könnyen értelmezhetőek legyenek. A cél pedig az egyes rendszerek védetségének vizsgálata, végpont védelmekkel és anélkül. Ezáltal mérhetővé válnak az egyes védelmi megoldások.

Továbbá nagyon fontos volt, hogy a különböző Exploit Framework-ökkel kompatibilis legyen, hogy későbbiekben minden probléma nélkül be lehessen építeni az automatizált környezetbe.

A vizsgálati környezet strukturális felépítése és tesztelési metódusa

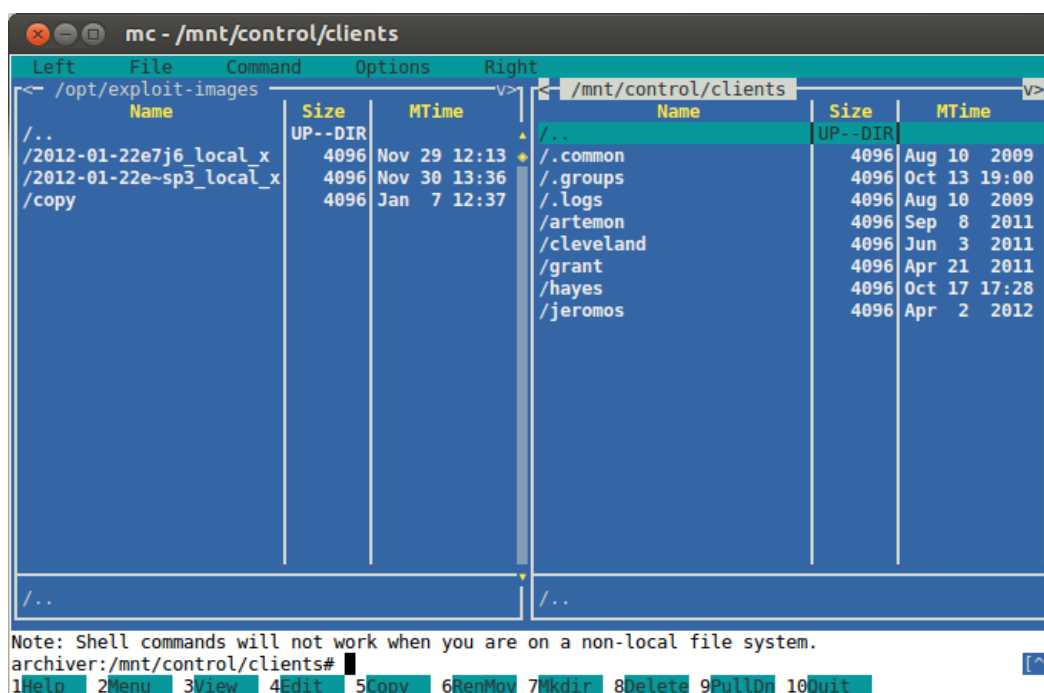


4. ábra: A rendszer sematikus hálózati felépítése

A fent látható ábrán az automatizált környezetbe kapcsolt gépeket láthatjuk. A rendszerhez minimális működéséhez szükséges egy szerver és legalább egy kliens gép. Az MSF tesztelő rendszeren található Metasploit Framework és egy automatizálást végző keretrendszer. Az egyes feladatok kiosztását és végrehajtásának figyelését a Manager szerver látja el[6]. A szerver gép két hálózati csatlakozóval rendelkezik (eth0 és eth1), melyekből az eth0 csatlakozik a külvilág fele, az eth1 csatlakozik a kliens gépekhez. Fontos tudni, hogy míg az eth0 lehet dhcp áttal konfigurálva addig az eth1 mindig fix ip címmel kell, hogy rendelkezzen[7]. Az MSF tesztelő rendszer részletes felkonfigurálását és a kezelését a 4.2 fejezet tárgyalja. Tesztelés közben a kliensek nem érhetik el a külvilágot, viszont frissítés esetén manuálisan engedélyezhető, hogy a vírusirtók frissíthessék a definíciós fájlokat.

A rendszer kialakítása

A szerver elsősorban az egyes feladatokat osztja szét a kliens gépek közt. A feladatok írják le a klienseknek, hogy melyik verziójú operációs rendszert kell elindítaniuk és utasításokat adnak, hogy melyik exploitot kell elindítani. A szerver gépen egy Debian Linux disztribúció található egy NFS hálózati megosztással. Ebben a hálózati megosztásban minden gépről megtalálható egy saját könyvtárstruktúra, amiben az egyes kliensek megtalálják az éppen aktuális feladatot[3].



5. ábra: A Kliensek könyvtárstruktúrája

A fenti kép jobb oldalán látszanak az kliensek könyvtárai.

Az egyes parancsfájlok elkészítése

Az egyes feladatok leírják, hogy a kliensnek melyik virtuális gépet kell elindítani és, hogy melyik exploitot kell elindítani[1]. A lent látható képen a parancsfájl tartalma látható, amiben látható a feladattípusa, ami jelen esetben egy virtuális gép indítása.

```
JOB_ID="519"
JOB_TYPE="VMTEST"
TEST_TYPE="exploit-oa"
AV_NAME="exploit"
DISPLAY_NAME="Exploit test"
SW_VERSION="test.version"
DB_VERSION="test.version"
PRE_RUN="pre.exploit_local_x.pl"
POST_RUN="post.exploit.pl"
DOC_ROOT="/mnt/virii/avtest"
WINTASK_MOUNT="--bind /mnt/rack/avtest"
SAMPLES_MOUNT="--bind /mnt/rack/test/exploit"
SAMPLES_SRC="ex520"
NUM_OF_RUNS="1"
IMAGE_MOUNT="--bind /mnt/store/vmware_images"
RES="/mnt/arack/avtest/results_h"
DATABASE="no"
IMAGE_SRC="2012-01-22e7j6_local_x"
A_TIMEOUT="20m"
```

6. ábra: A job fájl tartalma jól áttekinthető

Továbbá látható, hogy a későbbi fejezetekben megtalálható megnevezés szerint az ex520-as exploit futatása van előkészítve az IMAGE_SRC="" paraméterben megadott virtuális gépen. Ami még nagyon fontos, hogy a teszt lefutásakor keletkező eredmények könyvtárát is itt lehet megadni ami RES="" paraméterben van megadva.

A kliens kialakítása

Első körben meg kell vizsgálni, hogy a tesztelni kívánt exploit milyen sebezhetőséget, milyen hibát próbál kihasználni. Ezt legkönnyebben az exploitban lehet megtalálni, ami általában egy CVE hibajegy. A CVE a Common Vulnerabilities and Exposures rövidítése, ezen az oldalon egy egységes csoportosítási rendszerrel katalogizálva vannak az egyes sebezhetőségek. A sebezhetőség leírásában megtalálható, hogy ki publikálta az exploitot és ami nagyon fontos, hogy milyen operációs rendszeren és környezetben képes lefutni.

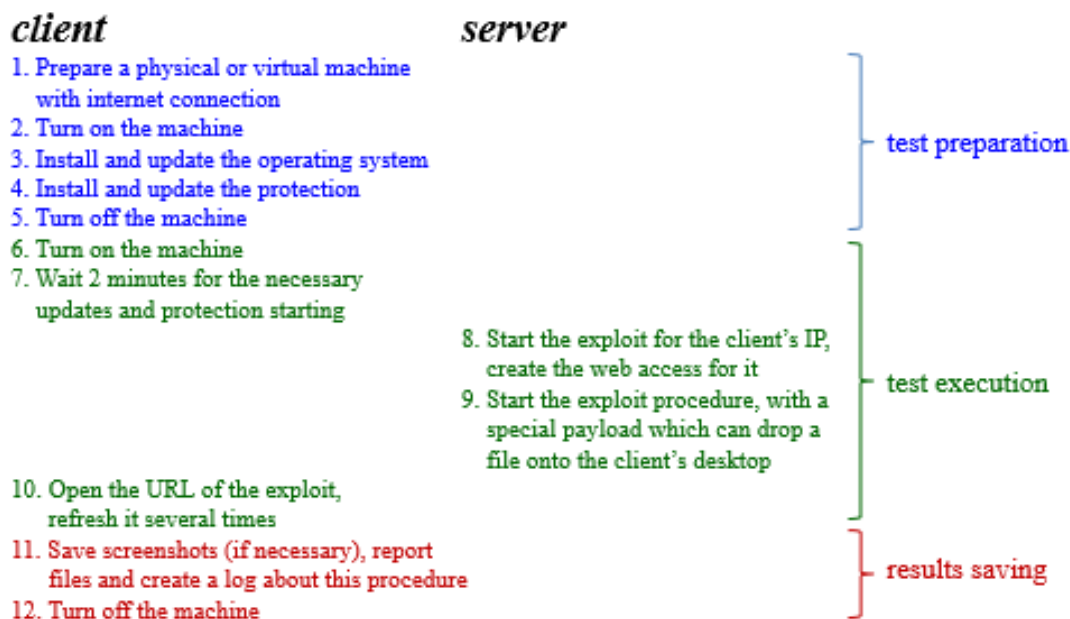
Vulnerability Summary for CVE-2012-5678	
Original release date:	12/12/2012
Last revised:	12/12/2012
Source:	US-CERT/NIST
Overview	
Adobe Flash Player before 10.3.183.48 and 11.x before 11.5.502.135 on Linux, before 11.1.111.29 on Android 2.x and 3.x, and before 11.1.111.29 on Windows and before 3.5.0.890 on Mac OS X allow attackers to execute arbitrary code or crash the application.	
Impact	
<u>CVSS Severity (version 2.0):</u>	
CVSS v2 Base Score: 10.0 (HIGH) (AV:N/AC:L/Au:N/C:C/I:C/A:C) (leg	
Impact Subscore: 10.0	
Exploitability Subscore: 10.0	
<u>CVSS Version 2 Metrics:</u>	
Access Vector: Network exploitable	
Access Complexity: Low	
Authentication: Not required to exploit	
Impact Type: Allows unauthorized disclosure of information; Allows un	

7. ábra: A CERT leírása a sebezhetőségről

Miután a sebezhető 7. ábra: A CERT leírása a sebezhetőségről e kell állítanunk a z automatikus futtatáshoz. A kliens gép keretrendszere percenként keresi az új feladatot a fent említett NFS megosztásban [4]. A kliensek két hálózati interfésszel rendelkeznek egy a manager géppel való kommunikációt teszi, lehetővé a másik egy közvetlen kapcsolatot biztosít a MSF tesztelő rendszerrel.

A vizsgálati környezet automatikus működésének folyamatábrája

A lenti képen egy tesztelés folyamata látható különböző színekkel jelölve az egyes folyamat állomásokat.



8. kép: A rendszer működésének folyamata

Kék színnel jelöltem azt a szakaszt, amit a tesztelőnek manuálisan kell elvégeznie, vagyis a fizikai vagy virtuális gép kialakítását. Ez egy fontos állomás, hiszen csak akkor kaphatunk értékelhető eredményt, ha a teszteknek egy megfelelő környezetet alakítunk ki. Ez alatt értendő a megfelelő szoftveres környezet és a végpont védelem frissítése. A zölddel jelölt munkafolyamat teljesen automatikusan hajtódik végre, vagyis a keretrendszer automatikusan elindítja a virtuális gépet és ott egy wintask script segítségével az MSF-hez bekopog. Ehhez egy linuxos alkalmazást használ a knock-ot. Ennek a működési elve, hogy egy bizonyos kopogási értékre bizonyos folyamat indul el az MSF rendszerben. Itt fontos megemlíteni, hogy minden exploit kapott egy knock értéket. Amikor az exploit lefut, megpróbál egy fájlt létrehozni a kliensen, így jelezve, hogy betörési kísérlet sikerrel járt. A pirossal jelölt rész során kerül lementésre opcionálisan egy képernyőkép, illetve a sikeres vagy sikertelen támadásról egy report fájl. Majd a virtuális gép kikapcsolásra kerül.

Az eredmény és annak értékelése

Teszt céljából a keretrendszeren lefutattam 14 darab exploitot két virtuális image-en. Ez a két image csak a végpont védelemben különbözött. Az első oszlopban találhatóak az exploit azonosítói, amiket a rendszerben használunk. Második oszlopban a detektált kártevő azonosítója. Az utolsó kettő oszlopban a sikeres majd sikertelen futások száma látható. Minden tesztet többször is elvégeztünk, hiszen mint azt a korábbi fejezetekben kifejtettem, nem minden esetben működnek az exploitok.

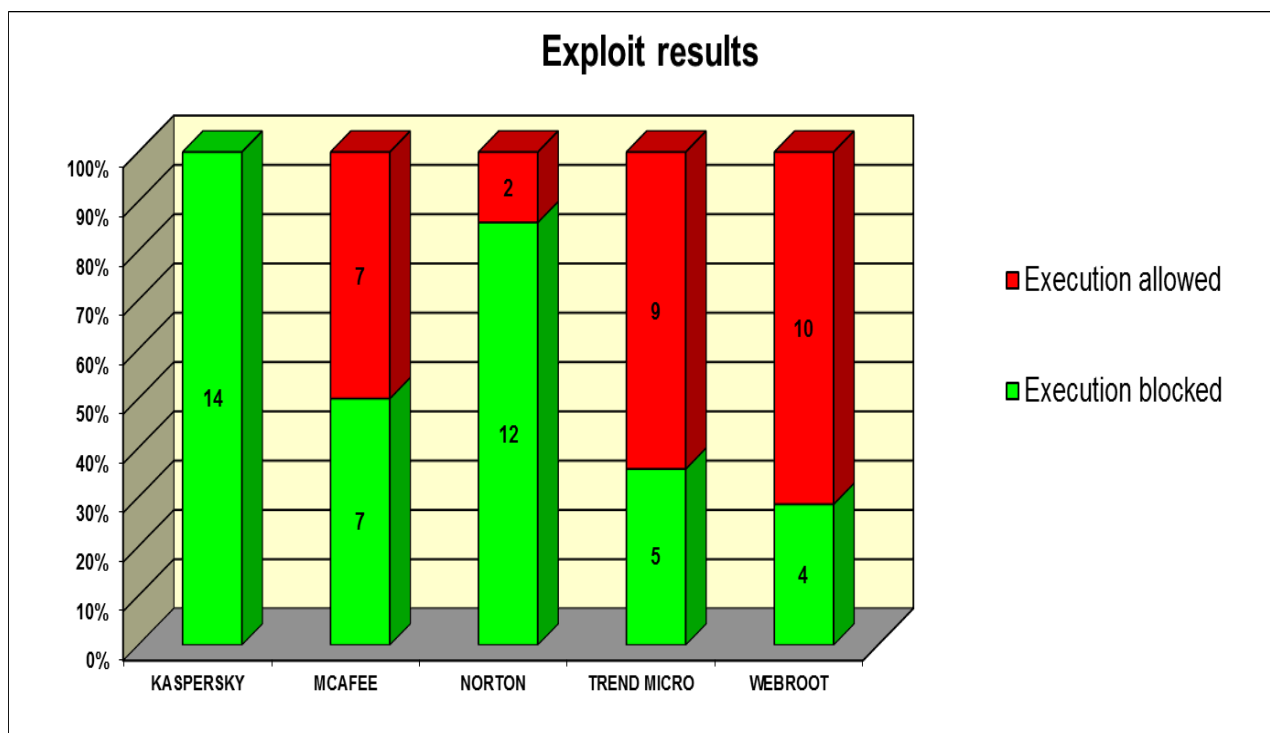
Exploit	KASPERSKY		
	Detection	Execution blocked	Execution allowed
exploit200	Exploit.Java.Agent.f	4	0
exploit202	Exploit.Java.CVE-2011-3544.ac	4	0
exploit203	Exploit.Java.CVE-2010-0094.e	4	0
exploit206	Exploit.Java.Agent.eq	4	0
exploit209	HEUR:Trojan-Downloader.Script.Generic	4	0
exploit211	HEUR:Trojan.Win32.Generic	4	0
exploit213	HEUR:Exploit.Script.Generic	4	0
exploit215	HEUR:Exploit.Script.Generic	4	0
exploit217	Exploit.JS.CVE-2006-1359.m	4	0
exploit219	HEUR:Exploit.Script.Generic	4	0
exploit220	HEUR:Exploit.Script.Generic	4	0
exploit223	HEUR:Exploit.Script.Generic	4	0
exploit225	HEUR:Exploit.Script.Generic	4	0
exploit226	HEUR:Exploit.Script.Generic	4	0
TOTAL	14	14	0
	100,00%	100,00%	0,00%

9. ábra: A Kaspersky antivírus teszteredményei

Exploit	MCAFEE		
	Detection	Execution blocked	Execution allowed
exploit200		4	0
exploit202	Generic Exploit.j	0	4
exploit203	Generic Exploit.j	0	4
exploit206	Generic Exploit.j	0	4
exploit209	Swrort.d, Swrort.f	0	4
exploit211	Swrort.d, Swrort.f	0	4
exploit213		0	4
exploit215		0	4
exploit217		4	0
exploit219	Exploit-CVE2006-3730	4	0
exploit220		4	0
exploit223		4	0
exploit225		4	0
exploit226		4	0
TOTAL	6	7	7
	43,00%	50,00%	50,00%

10. ábra: Pirossal a sikeres támadást

Jól látható, hogy a második végpontvédelem ugyan felismerte betörési kísérletet de az exploit képes volt végrehajtódni így nem kielégítő a védelem.



11. ábra: Összesített oszlopdiagram az eredményekről

Az alábbi ábrán egy összesítés tekinthető meg az tesztlésekről, amiket elvégeztem. Ebből jól kitűnik, hogy szép számmal előfordul olyan eset, amikor a végpontvédelmek nem nyújtanak kielégítő teljesítményt. Ezért nagyon fontos a keretrendszerem segítségével automatikusan képesek vagyunk tesztelni az egyes végpontvédelmek minőségét, hogy azok később hatékonyabban képesek legyenek elhárítani a támadást.

Összefoglalás

A programozási hibák kiszűrésére körültekintő tesztlésre van szükség. Amíg azonban a funkcionális és terheléses teszteknek kifinomult módszertana van, a biztonsági hibák felderítésére csak az elmúlt években kezdtek eljárásokat kidolgozni. Jelen TDK e vizsgálati módszerek automatizálásával és lélektanával foglalkozik. Első körben megvizsgáltam a jelenleg használt legnépszerűbb ilyen keretrendszert a Metasploit Framework-öt. Megvizsgáltam hogyan lehetne felgyorsítani, megkönnyíteni egy végpont védelem tesztlését. Majd elkészítettem egy keretrendszert mely automatikusan képes tesztekot lefuttatni és kihasználni a Metasploit Framework minden funkcióját.

Ez a rendszer képes megkönnyíteni a biztonság informatikus dolgát és könnyebbé teszi a kártevők elemzését, a védelem fejlesztését, illetve a védelmi rendszerek vizsgálatát.

Irodalomjegyzék

- [1]Dr. Leitold Ferenc, Horváth Botond – Dobozba zárt internet, Hacktivity 2011, Budapest
- [2]Ször Péter – A vírusvédelem művészete SZAK KIADÓ KFT. 2010
- [3]Pere László - GNU/Linux rendszerek üzemeltetése II. 2005
- [4]Andrew S. Tanenbaum - Számítógép-hálózatok (új, bővített kiadás - Panem) 2003
- [5]Peter Norton – A hálózat biztonság alapjairól Kiskapu 2000
- [6]Leitold Ferenc: Automatikus vírusanalizáló rendszer, Proceeding of the HISEC'96 Conference, Budapest, pp. 112-119, 1996
- [7]Leitold Ferenc: Automatic Virus Analyser System, Proceeding of the 5th International Virus Bulletin Conference, Boston, USA, pp. 99-107, 1995
- [8]<http://www.buhera.blog.hu>
- [9]<http://www.hup.hu>
- [10]<http://msdn.microsoft.com/en-us/ms348103>
- [11]<http://www.securitytube.net/>