Vírusok, férgek, rootkitek, a legújabb fenyegetések

ADAMKÓ PÉTER

Budapesti Műszaki és Gazdaságtudományi Egyetem, Informatikai Központ
ap331@hszk.bme.hu

Kulcsszavak: vírus, féreg, rootkit, algoritmus, detektálás, kernel

Körülbelül másfél éve kezdtek elterjedni a Windows-on működőképes rootkitek, megkeserítve ezzel a felhasználók és biztonsági szakemberek életét egyaránt. Fény derült arra, milyen nehéz az új típusú kártevők ellen védekezni. Ami nem volt világos, hogy lehetséges-e ezen új technológiát ötvözni a régi kártevőkével. Cikkünkben a férgek és rootkitek legveszélyesebb tulajdonságait mutatjuk be, valamint ismertetjük a rendelkezésre álló detektálási technikákat, szoftvereket.

1. Helyzetünk

A számítógépes vírusok több, mint 20 éve keserítik meg napjainkat. Eközben folyamatosan átalakultak, változtak s érték el pillanatnyi fejlettségüket. Eleinte különböző adathordozó médiákon terjedtek, azonban később a különböző hálózatok összekapcsolásával sokkal gyorsabb terjedésre nyílt lehetőségük, végül pedig az Internet elterjedésével betörtek az emberek mindennapjaiba.

Ekkorra végeredményben el is tűntek a klasszikus értelemben vett vírusok, helyüket az úgynevezett programférgek vették át (legnagyobb különbség, hogy a férgek önálló programok, immár nincs szükségük más hordozóra a működéshez, terjedéshez). Az elmúlt 4 évben sorozatban lehettünk tanúi az egész világot behálózó féregjárványoknak, melyek mintha csak az utóbbi időben csillapodtak volna. Általánosan igaz, hogy minden egyes esetben valamilyen széles körben használt szoftver hibáját használták ki, gyakran ötvözve a felhasználók megtévesztésével szolgáló technikákkal.

Kimondottan érdekes téma, hogy mitől tudtak enynyire elterjedtté válni ezek a kártevők? Emberi hanyagság, lustaság, esetleg jó algoritmusok és kiemelkedő programozás eredménye? Mára már ott tartunk, hogy egy-egy féreg detektálása és a védelmi szoftverek frissítése közötti idő rohamosan csökken, akár pár óra is lehet. Egészében véve elmondhatjuk, hogy a különböző IT-biztonsági megoldásokat szállító cégek kidolgozták a tömeges és nagyméretű féregtámadások elleni védelmet. Azonban ezzel kapcsolatban még mindig felmerülnek kérdések:

Mi történik, ha a frissítés előtt a kártékony program már megfertőzte a sebezhető gépek többségét (a Slammernek kevesebb, mint 15 percre volt szüksége)?

Mi történik azokkal a programokkal, amelyek nem mutatnak tipikus vírusviselkedést (gyors terjedés, komoly hálózati forgalom, fájlműveletek stb.), vagy még nincsenek benne a leíró adatbázisban?

Mi történik azokkal a programokkal, melyeket rejtőzködő életmód folytatására terveztek?

Biztonsági szakértők szerint az elkövetkező támadások már nem világméretűek lesznek, hanem egy-egy szervezet, cég, intézmény IT infrastruktúrája ellen fognak irányulni. Kétségtelen, hogy a rejtőzködés és a gyors terjedés némileg ellentmondanak egymásnak, de az is igaz, hogy egy gyors terjedésű, rootkit technológiát használó programféreg akár eltávolíthatatlanná is tudja magát tenni, miközben sokkal nehezebben detektálhatóvá válik. Sok víruskereső program már most is küszködik a rootkitek detektálásával.

A dolog aktualitását mutatja, hogy az évek óta jelenlevő Bagle immár rootkit technológiát használ. Nem véletlen, hogy a 2005-ös évben a Kaspersky Labs adatai alapján a rootkitek száma 413%-os növekedést ért el.

2. Algoritmusok

Mi is tette annak idején annyira pusztítóvá ezeket a férgeket? Egy gyors féreg sikeressége azon múlik, milyen gyorsan találja meg új célpontjait, milyen gyorsan képes őket megfertőzni.

Az új célpontokat jellemzően valamilyen szkennelés révén találja meg. A szekvenciális szkennelés (Blaster) során egy véletlen címtől kezdve szekvenciálisan nézi végig az IP címeket az algoritmus. Ez rosszabb hatékonyságú, mint az egyszerű véletlenszerű szkennelés, hiszen sok redundancia van benne, azonban ha gyorsan talál egy sérülékeny gépekkel teli hálózatot, akkor azt végigpásztázza és végigfertőzi. Ha mindez a terjedés elején történik, akkor az meggyorsítja az algoritmust, esetenként még a véletlen szkennelés gyorsaságát is megközelítheti, de ez nagyban függ a szerencsétől.

Maga a fertőzési sebesség szintén fontos tényező. Például annak megállapítása, hogy nem sebezhető a rendszer, gyorsabb lehet, mint maga a fertőzés. A fertőzés gyorsasága nagyban függ a megvalósítás módszerétől. A TCP kapcsolat felépítése, például TFTP esetén (a féreg törzse TFTP-vel töltődik fel a fertőzés

után) sokkal lassabb, mint mondjuk egy UDP csomagon keresztül terjedő féreg esetén. A Slammert az tette olyan gyorssá, hogy nem kellett állapotokat tárolnia, visszajelzésekre várnia, csak elküldték egy rendszer felé, és véget is ért a támadás. Ellenben az általános működésű férgek először fertőznek, majd a féreg testét, és a többi funkciót tartalmazó részt is feltöltik a megtámadott hosztra.

További fontos dolog, hogy mennyi sérülékeny gép van elérhető állapotban, egy nagyobb populációban ugyanis nagyobb a találati esély is. Legvégül pedig a szülő és gyerek férgek feloszthatják egymás között a szkennelési tartományt, így párhuzamosítva a műveletet. Így a célpontkeresés bináris fában történő keresésé válik, ami gyorsabb, mint a lineáris keresés.

Javít a gyorsaságon az is, ha a féreg fel tudja ismerni a már megfertőzött rendszereket, s ezeket a szkennelés során kihagyja illetve, ha egyes címtartományokról tudja, hogy azok nem tartalmaznak sérülékeny gépeket.

Ezek alapján a következőt mondhatjuk el egy fejlett féreg viselkedéséről: gyors, állapotmentes célfelismerő algoritmust használ, elkerüli az üres, vagy már megfertőzött címtartományokat, párhuzamos szálakat használ, gyorsan fertőz és kisméretű [4].

Ilyen algoritmusokat már 2001-ben kidolgoztak, s igazán szerencsések voltunk, hogy csak 2003-ban jelent meg olyan programféreg, mely használta ezeket. A Warhol és a Flash algoritmusok mindegyike lehetőséget ad arra, hogy az Interneten fellelhető sérülékeny gépeket kevesebb, mint 15 perc alatt végigfertőzzék.

Azonban, ha jobban belegondolunk, a jelenlegi sávszélességi állapotok mellett csak az elsőnek van létjogosultsága. A legtöbb féreg problematikája a továbbterjedés irányítása volt, melyre különböző módszereket használtak: címlisták szerzése a fertőzött gépről, hálózati szkennelés, random címek generálása, előre beállított szerverekről történő címfrissítés, talált email címekre való továbbküldés. Viszont egy kimerítő keresés meglehetősen nagy hálózati forgalmat generál...

Minden féreg a kezdeti terjedési szakaszában állítható meg a legkönnyebben, amikor még nem érte el azt a fertőzési számot, mely után a továbbterjedés robbanásszerűen megugrik. Természetesen ehhez az kell, hogy korai szakaszban fogjunk be egy példányt, s alkossunk rá felismerési mintát (ez általában valamilyen hash érték létrehozása). Ez órák kérdése lehet, de alapvetően a fent említett algoritmusok esetén nincsen elég idő erre. Éppen ebben rejlik nagyfokú veszélyességük.

Az algoritmusok egy előszkenneléssel kapott címlista generálásával (DNS lekérdezések, Google, web-robotok stb.) meghatározzák a kezdeti célpontokat, s egy időben támadva őket, gyorsan túllépnek a kezdeti szakaszon, létrehozva egy 10000-es kezdeti populációt. A következő lépés a további gépek megfertőzése, s ez az a pont, ahol a két algoritmus elkülönül egymástól. Az egyik megpróbálja „gazdaságosan” végigszkennelni a hálózatot, az egész IP címteret felosztva a féregpéldányok között. Minden fertőzés után a szülő és a gyerek között felezi a keresési teret. Ha egy már megfertőzött gépet talál, úgy egy új random, vagy szekvenciában rögzített pontról kezdi újra a szkennelést, hiszen tudja, a már fertőzött példány felelős a tér következő részének átkutatásáért, és már előrébb is tart ebben.

Ez egyes kutatók szerint gyorsabb is lehet, mint a hagyományos random szkennelés, de ha nem is gyorsabb, mindenképpen kevesebb üzenetet generál. A Flash algoritmus teljesen kihagy mindenféle szkennelést, mivel már az előszkennelés során feltérképezte az összes sebezhető hosztot (az egész Interneten), s terjedése során ezt a listát menedzseli, s osztja szét a különböző féregpéldányoknak. Bár ez még gyorsabb lehet, mint a Warhol technika, azonban ez a lista már kellően nagyméretű ahhoz, hogy sávszélességi problémákat vessen fel. [4], [5]

Milyen védelmi lehetőségeink vannak? A fent említett terjedési algoritmusok leginkább jellemző tulajdonsága a hálózati forgalomban létrehozott anomáliák. Ezek detektálhatóak különböző hálózatfigyelő szoftverekkel, NIDS-ekkel. További lehetőség a vállalati hálózat határán különböző szűrések végzése. Általában itt helyezkedik el a különböző vírusvédelmi szoftverek egy része, melyek ismert minták után szűrik az átmenő forgalmat. A publikus szervereken is szinte kötelező érvényű a szűrés, tipikus példa erre a levelezőszervereken felállított védelem.

Különösen fontos ez, hiszen mára már a férgek két legszignifikánsabb fajtája a levelező (mass-mailer), és a hálózati (network) férgek típusa. Az ismert vírusok keresésén kívül a gyakran ismétlődő minták, csatolt fájlok is felkelthetik figyelmünket (van olyan szoftver, mely ezekből úgynevezett szürkelistát hoz létre, s amíg nem tisztázódik vírusmentessége, nem továbbítja), sőt általában a csatolt fájlok kiterjesztései is, vagy gyakran ismétlődő nevei.

Véleményem szerint ez mindössze addig megoldás, amíg egy jobb polimorf motorral rendelkező programféreg meg nem jelenik. További segítség lehet, ha csak olyan levelező szerverekről fogadunk el leveleket, melyek címe feloldható. Legvégül pedig fontosak a munkaállomáson telepített vírusvédelmi szoftvereket is.

3. Rootkitek

A rootkitek már régóta jelen vannak életünkben, a Windows-t használók számára azonban csak az elmúlt egy-két évben váltak ismerté. Szeretnénk leszögezni, hogy teljes mértékben egyetértünk Greg Hoglund-dal, aki szerint ez mindössze egy olyan eszköz, mely képes arra, hogy elrejtsen folyamatokat, adatokat egy rendszerben.

Sok felügyeleti eszköz, védelmi program és biztonságos dokumentumkezelést biztosító szoftver is alkalmazza ezeket a technikákat (tűzfalak, CSA, ISeeSec).

Most a vírusokhoz kapcsolódóan mégis a támadási lehetőségeik kapcsán tárgyalom a rootkitek működését.

Az x86-os architektúra négy privilégiumszintet határoz meg (Ring0-3), ezek közül az operációs rendszerek általában kettőt használnak, a felhasználói és a kernel szintet. A privilégiumok meghatározzák, hogy a programok milyen műveleteket hajthatnak végre, hiszen a kernelt meg kell védeni a felhasználói programoktól, viszont bizonyos szolgáltatásokat biztosítani kell számukra. Míg felhasználói módban egyes memóriaterületek védettek, addig kernel módban az egész memóriához, s a processzor összes utasításához hozzáférhetünk. Ennek eredménye, hogy kernel módban jól lehet nyomon követni, elfogni, s módosítani a rendszerben lévő üzeneteket, állapotokat.

A felhasználói módú támadó rootkitekre jellemző, hogy többnyire önálló alkalmazásként futnak, vagy pedig egy már létezőt cserélnek le.

Ezen rootkitek és a védelmi szoftverek között állandó harc dúl, hogy melyik tudja átvenni az operációs rendszer fölötti irányítást, s ennek kimenetét az határozza meg, hogy melyik tud mélyebb szinten a rendszerbe nyúlni (hiszen az operációs rendszer különböző szintjei egymáson keresztül érik el az adott szolgáltatásokat). Például az operációs rendszer teljes körű irányítással rendelkezik egy-egy alkalmazás memórianézetéről, hiszen a fizikai és virtuális memória közötti leképzés absztrakcióját ő maga végzi el.

Általánosságban két típusú rootkitről beszélhetünk, perzisztens és memória alapúakról. Legnagyobb különbség, hogy az előbbi túléli a hoszt újraindulását. Ehhez két dolog szükséges: valahol tárolniuk kell a kódjukat a rendszerben (többnyire a merevlemezek egyikén), valamint valahol be kell tölteniük magukat a rendszerbe (például beépülve az indítási szekvenciába). Ezeket a változtatásokat valahogy rejteniük is kell. A memória alapú rootkitek csak a memóriában léteznek, s újraindításkor törlődnek. Bár ez gyengeségnek hathat, azonban a szervereket viszonylag ritkán indítják újra, ennek fejében lényegesen nehezebb detektálni őket, s jóval kevesebb nyomot is hagynak. Ettől függetlenül el kell rejteniük a memóriában található végrehajtható kódjukat, valamint el kell rejteniük a memóriabeli módosításokat is (ezek nélkül akár a legegyszerűbb mintaillesztéses memóriaszkennelés is elárulhatja őket) [2].

A rootkit elrejtésére egy polimorf technika is jó lenne, azonban ez még mindig jól láthatóan otthagyná a memóriában végzett változtatásokat, így a rendszerkomponensek változásai egy integritásellenőrzés során kimutathatóak lennének. Ennek következtében a rendszer rootkitről látott képét kell módosítani.

3.1. Rootkit technológiák

A legelterjedtebb technikák közé tartozik az úgynevezett kampók (hook) használata, valamint a kernel objektumainak (DKOM) manipulálása. Az első során az operációs rendszer normális végrehajtási útvonalát módosítja a rootkit, és így módosítani tudja azokat az információkat, melyeket egy rendszerhívás visszatérésekor ad.

3.1.1. Felhasználói módban használható módosítási lehetőségek

Import Address Table

A Windowst úgy tervezték, hogy hardvertől független legyen, valamint kompatibilis legyen más környezetekkel (pl. POSIX). Fontos volt, hogy a fejlesztőknek ne kelljen mindig újraírniuk kódjaikat egy-egy rendszerfrissítésnél, ezért különböző alrendszereket használ, melyeket dll-ként implementál. Ezeken az interfészeken keresztül érhetőek el a kernel szolgáltatásai.

A különböző alkalmazások nem közvetlenül hívják a Windows rendszerszolgáltatásokat, hanem az alrendszereken (OS/2, POSIX, Win32) keresztül. Ezek a könyvtárak exportálják azon interfészeket, amelyekkel a programok kapcsolódnak hozzájuk. A leggyakrabban használt a Win32, mely a Kernel32, a User32, a Gdi32.dll és az Advapi dll-ekből áll. Az Ntdll.dll olyan speciális rendszertámogatási könyvtár, amelyet az alrendszer dll-jei használnak. Mikor egy bináris fájl betöltésre kerül, a töltő automatikusan átnézi a fájl egy részét, az Import Address Table-t (IAT), amelyben megtalálható a dll-k listája, valamint azok a függvények, amelyeket belőlük használni fog a program. Ezután a töltő a függvények címeit a memóriába teszi. Általában a Kernel32 és az Ntdll függvényei találhatóak meg itt, de más speciális függvények is jelen lehetnek (például a Ws2_32.dll-ben található socketkezelő függvények). A kernel eszközmeghajtói szintén importálnak függvényeket a dll-ekből (pl. Ntoskrnl.exe, Hal.dll). A bináris fájlok IAT-jaiban található címek módosításával egy program magára irányíthatja a végrehajtást, s befolyásolhatja az eredeti függvény futását: egy program egy könyvtár listázását végzi, és néhány műveletet hajt végre rajtuk.

Tételezzük fel, hogy felhasználói módban fut, és Win32-es alkalmazás. Ekkor a Kernel32, a User32 és a Gui32 dll-eket fogja használni. A könyvtár listázásához a FindFirstFile és amennyiben az sikeres, a FindNextFile API függvényeket fogja meghívni. Amikor az alkalmazás meghívja a függvényt, akkor az import táblából kikeresi a címet, s a megfelelő függvényre ugrik a Kernel32.dll-ben. A rootkit az IAT címet átírva a saját függvényére irányíthatja a hívást, és tulajdonképpen bármilyen utasítást végrehajthat: meghívhatja az eredeti függvényt, s annak visszatérési adatait szűrheti stb. Fontos tudnunk, hogy amikor átírja ezt a címet, a rootkit átlépi a folyamat virtuális címtartományát, s ezzel detektálhatóvá válik.

Inline kampó

Az inline kampó létrehozása során az ellenőrizni kívánt függvény elején kell módosítani néhány bájtnyi kódot, általában egy feltétel nélküli ugrást, mellyel a rootkit kódjára ugrik a processz, majd onnan vissza. A leg-

több függvény ugyanazzal a kódrészlettel kezdődik, majd ezt követik a függvény tényleges utasításai.

A rootkitnek a függvény első 5 bájtját (1 bájt az ugrási utasítás, 4 a cím) kell átírni egy ugrási utasításra, s az ugrási címre. Ezt a rootkitnek érdemes lehet elmentenie, mivel a Service Pack2 előtti időkben az azonos kódrészlet mindössze 3 bájt volt, s mivel az ugrás + cím 5 bájtnyi helyet foglal, tudnia kellett, hogy milyen utasítást írt át (hogy megőrizze a függvény eredeti funkcionalitását).

A Service Pack2 utáni időben 5 bájtra növekedett ennek a kódnak a hossza, legális haszna, hogy lehetővé teszi a „hot patching"-et, vagyis, hogy újraindítás nélkül lehessen módosítani a függvényeket. Viszont így probléma nélkül lehet módosítani rossz indítékkal is őket.

Dll injektálás

Dll injektálás révén betölthető a rootkit kódja egy másik folyamat memóriaterületére. Az NT/2000/XP/2003 Windows családban a HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLL kulcsból olvassa ki a rendszer, hogy az egyes alkalmazások milyen dll-eket töltenek be indításukkor. Ezt módosítva tetszőleges dll tölthető az alkalmazással együtt a memóriaterületre.

Lehetőség van eseményvezérelten hozzáférni más folyamatok üzeneteihez. A SetWindowsHookEx API függvényt használva más folyamatok eseményeihez férhetünk hozzá, s tetszőleges feldolgozási függvényt hívhatunk meg adott esemény bekövetkeztekor.

Legvégül pedig lehetőség van távoli szálat létrehozni a CreateRemoteThread API függvény segítségével.

3.1.2. Kernelszintű módosítási lehetőségek

Interrupt Descriptor Table

Az interrupt vektor tábla (Interrupt Descriptor Table) kampók használata még a DOS-os időkbe nyúlik vissza. Ez a tábla határozza meg, hogy az egyes események, megszakítások (például billentyűlenyomás) után milyen feldolgozó egységnek adja át a vezérlést (billentyűlenyomás, memóriahiba stb.) Fontos azonban tudni, hogy nem tér vissza, s a vezérlést sem kapja vissza, így szűrésre nem alkalmas, azonban blokkolhatja egyes szoftvereknek (pl. IDS) címzett megszakítási kéréseket.

IRP kampó

Az összes eszközmeghajtó programban jelenlévő függvénytábla szintén alkalmas kampók létrehozására, mivel ebben találhatóak meg azok a mutatók, melyek az egyes feldolgozó függvények helyeit mutatják meg. A függvények különféle I/O kérés csomagokat (I/O Request Packets) dolgoznak fel, mint például olvasás, írás, lekérdezés. Egy TCP/IP lekérdezést változtatva például elrejthetjük a nyitott portok listáját. Az IDT-nél tapasztaltaknak megfelelően ez sem hívja meg az eredeti függvényt, s nem ad módot a szűrésre.

System Service Descriptor Table

A rootkitek ahelyett, hogy alrendszereken keresztül kapcsolódnak a kernelhez, átírhatják közvetlenül annak a táblának (System Service Descriptor Table) a függvénycímeit, mely az aktuális függvényimplementációk címeit tartalmazza az adott operációs rendszerben. Ezek a címek az Ntoskrnl.exe-ben található Nt*** függvények címeit tartalmazzák.

Ezzel lényegében nem egy programra, hanem az egész rendszerre kiterjedő kampó valósítható meg. Az előző IAT példát véve, hasonló funkcionalitás elérésére a rootkit átírhatná az NTQueryDirectoryFile címét, a hívást önmagára irányítva.

Runtime patching

A kampók használata meglehetősen régi technika, s mivel a hívási táblákat módosítják, viszonylag jól észrevehetőek. A programok futási időben a memóriában történő módosítása (runtime patching) lehet a következő módszer, amelyet alkalmazni fog egy magát elrejteni kívánó program. A memóriában található adatstruktúrák vezérlik az egyes programok futási logikáját, ennek módosításával a program működése is megváltoztatható. Fontos dolog azonban, hogy ekkor a memórialapok írási/olvasási jogosultságával is rendelkeznie kell a rootkitnek.

A „detour patching" során a rootkit az eltéríteni kívánt függvény belsejében helyez el egy magára vonatkozó ugrási utasítást, ezáltal az összes függvényre mutató táblára hatott, másrészt nem kell számon tartania az összes ilyen táblát.

Layer driver

A réteges meghajtók (layer driver) újabb helyként szolgálhatnak a rootkitek elrejtésére. Az eszközmeghajtók lehetőséget adnak egymás sorba láncolására, s az egymástól bejövő adatok alapján dolgoznak. Ez módot ad arra, hogy egy-egy meghajtó program frissítéséhez ne kelljen mindig újraírni az egészet. Minden meghajtó program az operációs rendszer fontos funkcióit valósítja meg: hálózati kommunikáció, fájlműveletek stb. Sok víruskereső is egy eszközmeghajtót telepít fel a fájlok megnyitásakor történő szkenneléséhez: az operációs rendszer által biztosított fájl meghajtók által szolgáltatott adatok (például egy fájl megnyitásakor) a víruskereső meghajtójához kerülnek, amely képes mintakeresést végezni rajtuk.

Értelemszerűen ezt a rootkitek is használhatják adatszűrésre, és ha a megfelelő meghajtók előtt helyezkednek el a sorban, akkor a róluk szóló adatokat módosítva adhatják tovább.

DKOM

A közvetlen kernel objektumok manipulálásának módszere (DKOM) arra épül, hogy a rootkit kihasználja, hogy az operációs rendszer ezeket az objektumokat használja nyomonkövetésre és auditálásra. Ha ezeket módosítja, az egész rendszer önmagáról alkotott képét változtatja meg.

Általános esetben a kernel objektumok (folyamatok, tokenek) az Object Manageren keresztül érhetőek el, mely az általános funkciókat, mint például létrehozás, törlés, védelmi szintek beállítása biztosítja számukra. A rootkitek pont ezt kerülik meg, átlépve az összes ellenőrzést az adott objektumon. Bár nehezen detektálható módszer, hátránya, hogy csak a memóriában tárolt objektumokon képes módosításokat végrehajtani, így például a folyamatok, vagy portok listáját tudja változtatni. Ellenben a fájlrendszert egyetlen objektum sem képviseli a kernelben, ezért fájlokat elrejteni nem tud. Ezen kívül képes eszközmeghajtókat elrejteni, valamint folyamatok privilégiumszintjét változtatni.

Például a hírhedt FU rootkit a folyamatok nyilvántartására használt objektumot módosítja: minden folyamat egy EPROCESS adatstruktúrában van tárolva, s ezek kétszeresen láncolt listában helyezkednek el. Ebben a listában PID, vagy név alapján keresve megkeresi a kívánt folyamatot, s kiláncolja a listából [1,2].

4. Milyen védekezési eszközeink vannak a rootkitek ellen?

Lényegében két módszer közül választhatunk, megpróbáljuk jelenlét, vagy viselkedés alapján kiszűrni őket. A jelenlétet többféle módon is kimutathatjuk:

Mintakeresés

A minta alapú keresés már régen a víruskeresők sajátosságai közé tartozik. A rendszerben található fájlok végigszkennelése önmagában korlátozott hatékonyságú (a rootkitek rejtőző tulajdonságai miatt), a rendszermemória szkennelése már sikeresebb lehet. A legtöbb nyilvános kernel rootkit tipikusan a nem lapozható memóriában foglal helyet, s ritkán tesz arra erőfeszítést, hogy kódját valamilyen formában elfedje. Így nyilvánvalóan a kernel memória átvizsgálása révén leleplezhetőek. A kulcsszó itt a nyilvánoson van, hiszen egy nem ismert kódra nincsen minta az adatbázisban. Másrészt a virtuális memória menedzselésére képes rootkitek (pl. Shadow Walker) ellen sem használható, hiszen az képes a memóriaolvasások vezérlésére.

További memória alapú módszer a különböző kampók keresése. Az általános módszer az lehet, hogy végignézzük a különböző rendszertáblákat (IVT, SSDT, IDT, IAT stb.), ahol is a különböző függvények egy adott címtartományba kell, hogy essenek. Például az SSDT összes függvénycímének a kernelben található ntoskrnl.exe-re kellene mutatnia.

Betöltési helyek ellenőrzése

A memóriába töltés lehetőségeit számba véve sok olyan módszert találhatunk, melyekkel a rootkit a memóriába töltheti magát. Ezen helyek megfigyelésével és a megfelelő függvényekre kampók létesítésével (ZwSetSystemInformation, ZxOpenProcess stb.) történhet. Viszont ezekből a pontokból meglehetősen sok van.

Ma már különböző víruskereső alkalmazások is használják ezt a módszert (Nod32), így mutatva ki a rootkitek jelenlétet. Természetesen ebben benne van a hamis pozitív találatok lehetősége, hiszen számos nem kártékony alkalmazás is használ kampókat.

Viselkedés vizsgálat

Ennek során annak kimutatása szükséges, hogy az operációs rendszer valamilyen hamis információt adott meg számunkra. Itt elsősorban az elrejtett fájlok és folyamatok kimutatására teszünk kísérletet. [2]

Keresztnézeti (Cross-view) detektálás

Egy viszonylag új és sok sikerrel kecsegtető viselkedésdetektálásra alkalmas technika. A módszer feltételezi, hogy az operációs rendszer már módosított és megbízhatatlan.

Ekkor ugyanazt a lekérdezést végzi el, csak többféle módon. Először a szoftver az API függvényeken keresztül (Zw*** függvények stb.) kérdezi le a kívánt adatot (fájl, processz lista, regisztrációs adatbáziskulcsok), majd ugyanezt az adatot kinyeri valamilyen algoritmus révén, mely nem függ az API hívásoktól. Bármilyen különbség a rootkit jelenlétéről árulkodik. Az API kampók, vagy a DKOM használatából adódó hamis adatokat az alacsony szintű lekérdezésekből származó adatok leplezik le.

Meg kell jegyeznünk azt is, hogy ez a fajta detektálási módszer sebezhető a létező rootkitek támadási módszereivel szemben, hiszen nagymértékben függ az implementációtól, főleg az alacsony szintű rendszerinformációk kinyerése esetén. Például az NTFS diszk elrendezési információinak kinyeréséhez először szüksége van egy handler-re az adott kötethez, majd el kell olvasnia a szektorokat. Ha az API függvényt használja a feladathoz, akkor arról feltételeznie kell, hogy nem megbízható. Tehát az implementáció sikeressége a diszkvezérlővel folytatott közvetlen kommunikáción alapszik [3].

Most pedig lássuk ezen elméletek néhány gyakorlati megvalósítását:

VICE

Ez egy olyan eszköz, mely egy eszközmeghajtó programot telepít fel, a felhasználói és kernel módú programok figyelésére. Ellenőrzi az SSDT-t, s megvizsgálja azokat a címeket, amelyek nem az ntoskrnl.exe-be mutatnak. Emellett hozzáadhatunk fájlokat a driver.ini-hez és ekkor ellenőrzi a meghajtóhoz tartozó IRP főbb funkciós tábláját. Ha valamelyik mutató nem a meghajtóra mutató címmel rendelkezik, akkor az IRP-re egy másik meghajtó, vagy valamilyen kernel kód kampót hozott létre.

Felhasználó módban ellenőrzi az alkalmazások dll-jeit IAT-beli kampók után kutatva. Az inline függvénymódosításokat a dll-ben és az SSDT-ben is detektálja. Ekkor megpróbálja visszakeresni, hogy melyik függvény hozta létre a kampót és hol található.

A rootkitek ki tudják kerülni a programot, mivel mindig ugyanazzal a folyamatnévvel fut, s ezt a nevet érzékelve a rootkit nem hozza létre a kampókat. Egy másik támadás a VICE eszközmeghajtója és a felhasználói módban futó modul közötti kommunikációt zavarja meg. Azonban a program legnagyobb hibája a nagyszámú hamis pozitív találat, amit visszaad (hiszen létezhetnek a rendszerben legitim kampók is: hot patching, dll forwarding) [6].

Patchfinder

Ez a szoftver a statikus analízissel szemben futási idejű végrehajtási útvonalak vizsgálatával keresi a rootkiteket. A program működési elve azon alapszik, hogy egy rootkit, ha módosítja a végrehajtási útvonalat (például: kampó beillesztésével szűri egy adott szolgáltatás visszatérési eredményeit), akkor hozzáad kódot. Meg lehet számolni, hogy az egyes rendszerszolgáltatások hány műveletből állnak. Egy kompromittált rendszerben mindenképpen több műveletből kell állnia, mint egy tiszta rendszerben. A program az x86-os processzor „single step” funkcióját használja a műveletek számolására.

Sajnos a Windows operációs rendszerben több végrehajtási útvonal is lehet, amely nem determinisztikus viselkedést eredményez (vagyis egymás után megismételt számolás más-más eredményt adhat különböző rendszer feltételeknek köszönhetően). A program egy hisztogramot hoz létre, s az első csúcsértékkel számol, mely rootkit jelenlétében érezhetően elmozdul. Mindenesetre ez a módszer is képes hamis pozitív találatokat adni, valamint a programot támadó rootkitekkel szemben is sebezhető.

Rootkit Revealer

Perzisztens rootkitek ellen dolgozták ki, ezért a rendszerleíró adatbázis és a fájlrendszer lekérdezésével dolgozik. Az API függvényektől kapott adatokat a nyers fájlrendszer struktúrával és a regisztrációs adatbázist alkotó puszta fájlokkal veti össze.

Mint a legtöbb detektálási eszköz, a Rootkit Revealer is sebezhető azon rootkitekkel szemben, melyek blokkolják vagy eltérítik a hozzáférést a diszkrendszerhez vagy az adatbázis fájlokhoz. Szintén produkál hamis pozitív találatokat, amennyiben fájlok vagy adatbázis kulcsok jönnek létre, módosulnak két lekérdezés között.

Klister

A DKOM-ot használó rootkitek ellen készült, melyek az FU-hoz hasonlóan rejtik el futó folyamataikat, vagyis a PsActiceProcessList-ből kiláncolják az elrejteni kívántakat. Ekkor elvileg a processz futásának is le kellene állnia, vagyis az operációs rendszer időzítőjétől nem szabadna több CPU időt kapnia, gyakorlatilag azonban az operációs rendszer több különálló időzítőt is fenntart, melyek segítségével kiosztja a CPU időket.

Ezt a redundanciát kihasználva a Klisters összehasonlítja a folyamatok listáját a különböző időzítőkben található listákkal. A különbség nyilvánvalóan rootkit jelenlétre utal. Bár ez a módszer egyelőre csak rejtett folyamatok felderítésére szolgál, a jövőben kiteljesíthető a redundáns kernel adatstruktúrák teljes vizsgálatára.

Icesword

Egyike a legjobb ingyenesen elérhető rootkit detektoroknak, SSDT kampókat, rejtett fájlokat, folyamatokat, regisztrációs adatbázis kulcsokat és könyvtárakat érzékel, valamint nyitott portokat és rejtett socket kommunikációt. Nagy előnye, hogy sok információt ad a felhasználóknak, megmutatva, melyik rendszer-hívásokon vannak kampók, milyen meghajtó programok rejtettek stb.

Gyengesége, hogy az alacsonyszintű processz információkat egy kernel adatstruktúrára (PspCidTable) alapozva nyeri ki, s ez sebezhetővé teszi egy olyan rootkittel szemben, mely erre az egy táblára hoz létre kampót.

Blacklight

Elrejtett fájlok, folyamatok felderítésére szakosodott. Hasonló képességei vannak, mint az előző programnak, sőt, az alacsonyszintű lekérdezési algoritmusa is hasonló.

Strider GhostBuster

A Rootkit Revealer-hez hasonlóan működik, magas szintű API lekérdezéseket alacsonyszintű adatstruktúrákkal (NTFS master file table, Windows registry hive) hasonlít össze. Különlegessége, hogy keresés során először egy futó rendszeren, majd a rendszer újraindítása során egy általa produkált tiszta rendszeren (WinPE által produkált live cd) fut le. Ekkor viszonylag könnyen kimutathatja a fájlrendszerben található különbségeket, ami a perzisztens rootkitek ellen működhet. Viszont egy szerverrendszer újraindítása nem mindig kézenfekvő.

Ez azonban arra is képessé teszi, hogy adatokat találjon utólagos analízishez egy kompromittált rendszerben.

System Virginity Verifier

Ez a program egyesíti magában a VICE rendszerintegrációs ellenőrzését, valamint heurisztikus keresést is végez a találatok között, hogy a hamis pozitív találatok arányát csökkentse. A memóriaintegritás ellenőrzése a fontos rendszerkönyvtárak és meghajtó programok a merevlemezen található kódjainak és a nekik megfelelő memóriaképek összehasonlítása révén történik.

Copilot

A Copilot egyedülálló a maga nemében, hiszen hardveres megoldást nyújt a rootkit detektálásra. Jelenleg egy PCI csatlakozású kártya, mely figyeli a rootkit aktivitást a rendszerben. A PCI kártya használatának célja, hogy egy olyan tiszta elemet építsen a rendszerbe, mely bármilyen szoftveres változtatás után is független

és lehetőleg változatlan marad. Ennek érdekében saját CPU-val rendelkezik és a fizikai memóriát DMA-n keresztül éri el.

A memóriában különböző rootkitre utaló nyomokat keres: SSDT kampókat, a kernel függvényeinek és adatstruktúráinak módosításait stb. A Copilot saját hálózati interfésszel is rendelkezik, hogy biztonságos csatornán kommunikálhasson a vezérlő komponensével. A hardver miatt nagyfokú biztonságot nyújt, azonban sokkal költségesebb, mint egy szoftveres megoldás, és persze frissítése is nehézkesebb [6].

RAIDE

Egy új kezdeményezés a Rootkit Revealer készítőitől, mely a detektáláson kívül az esetleges „gyógyításra" is vállalkozik. Kombinálja a legtöbb létező eszköz detektálási módszereit, igyekszik viszonylag sok fontos és részletes információt nyújtani a felhasználók számára. Képes a következők elvégzésére:

Utólagos bizonyítékok gyűjtése (processz dump készítése), különböző, már létező motorok technikáit használja: rejtett folyamatok (Blacklight), kampók detektálása (SVV, VICE) stb. A program a felhasználói és kernel komponensei közötti kommunikációra megosztott memóriát használ, mely sokkal védettebb, mint a többi szoftver kommunikációja, révén ez csak titkosított adatot tartalmaz, sokkal nehezebb visszafejteni, megzavarni. Véletlen folyamatneveket és eseményneveket használ tovább nehezítve az esetleges támadó rootkitek dolgát [7].

Tripwire

A Tripwire egyedi CRC hash-t használ minden egyes fájl megjelölésére. Egy ellenőrzésnél minden fájlra újragenerálja az értéket, majd összehasonlítja az adatbázisában szereplővel.

Azon a megfigyelésen alapszik a működése, hogy a rendszer fájljai nem változhatnak meg (kivéve frissítés), tehát ez betörésre utaló nyom. A Tripwire nagyon eredményes volt azokkal a kezdeti rootkitekkel szemben, melyek egyszerűen csak helyettesítették a rendszerfájlokat saját binárisaikkal. Mivel azóta a fájlrendszerből a memóriába mozogtak a rootkitek, s képesek fájlolvasási műveleteknél a valóstól eltérő képet mutatni, lényegében használhatatlanná tették ezt a fajta védelmet.

5. Konklúzió

Kétségtelen, hogy egyre újabb és újabb ötletek kerülnek napvilágra, s egyre mélyebben nyúlnak bele az operációs rendszerekbe. Ezt kombinálva a vírusterjedési módszerekkel igen veszélyes kártevőt kapunk. Nem elég, hogy gyorsan terjed, de fertőzés után még szinte érzékelhetetlen is lesz. Ráadásul egyáltalán nem biztos, hogy érzékelés után biztosan el tudjuk távolítani, hiszen kernel modulokat kivenni egy rendszerből nem mindig egyszerű megoldás.

Ezen felüll az egyes vírusvédelmi szoftvereknek a rendszer stabilitásával is törődniük kell mielőtt akcióba lépnek. Amiatt pedig, hogy nem bízhatnak meg a kapott adatokban, sokkal mélyebbre kell majd nyúlniuk a rendszerben, ez pedig egyre nehezebb lesz...

Terjedelmi okokból a Linux operációs rendszeren futó rootkiteket nem tárgyaltuk. A teljes cikk elérhető a http://viruslab.ik.bme.hu/ honlapon.

Irodalom

[1] Ed Skoudis, Lenny Zeltser:
Malware: Fighting Malicious Code,
Prentice Hall PTR, 2003.

[2] Greg Hoglund, Jamie Butler:
ROOTKITS, Subverting the Windows Kernel,
Addison Wesley Professional, 2005.

[3] Joanna Rutkowska:
Thoughts about Cross-View based Rootkit Detection,
http://invisiblethings.org/papers/
crossview_detection_thoughts.pdf, 2005.

[4] Stuart Staniford, Vern Paxson, Nicholas Weaver:
How to 0wn the Internet in Your Spare Time,
11th USENIX Security Symposium, 2002.

[5] Tom Vogt:
Simulating and Optimalising Worm Propagation
Algorithms,
http://www.megasecurity.org/papers/
WormPropagation.pdf, 2003.

[6] James Butler, Sherri Sparks:
Windows rootkits of 2005,
http://www.securityfocus.com/infocus/1854, 2006.

[7] James Butler, Peter Silberman:
RAIDE: Rootkit Analysis Identification Elimination,
BlackHat Europe 2006.