Óbudai Egyetem

Doktori (PhD) értekezés

tézisfüzete

Memória korrupciós szoftverhibák modern, kód újrafelhasználáson alapuló kiaknázási módszereinek vizsgálata

Dr. Erdődi László

Témavezető: Dr. habil Tick József

Alkalmazott Informatikai Doktori Iskola

Budapest, 2015. szeptember 25.

1. A kutatás előzményei

A számítógépes hálózatok elleni támadások mindennapos eseménnyé váltak napjainkban. A támadók egyre kifinomultabb módszereket és eszközöket használnak, amelyek ellen egyre nehezebb és költségesebb védekezni.

Egy informatikai támadás nem minden esetben illegális tevékenység. Az etikus hacker [1] egy olyan informatikai szakértő, amely a támadás összes formáját ismeri és gyakorolja és a rossz-szándékú hackerekhez nagyon hasonló módszertannal dolgozik. Az öncélú és felelőtlen hackerekkel szemben, az etikus hacker nagyon alaposan és gondosan jár el a támadás során, minden cselekedetét dokumentálja, emellett a támadás megkezdése előtt írásos szerződést köt a célpont hivatalos képviselőjével. Egy számítógépes rendszer etikus hackeléssel történő vizsgálata az esetek jelentős részében olyan hibákra képes rávilágítani, amelyek csupán védekező szempontból történő vizsgálatokkal nem derültek volna ki.

Az informatikai támadásokkal kapcsolatos kutatások haszna napjainkban már megkérdőjelezhetetlen. Ezek a kutatások az etikus hackerek működését nagymértékben elősegítik, ezáltal közvetve hozzájárulnak ahhoz is, hogy az informatikai rendszerek biztonságosabbak legyenek. Emellett az informatikai rendszerek elleni támadások kutatási eredményeinek publikálásával a gyártók, üzemeltetők és felhasználók egyaránt biztonságosabb, a támadásnak jobban ellenálló rendszereket tudnak építeni.

Jelen kutatás az informatikai támadások egyik legveszélyesebb formájával a memória korrupción alapuló támadások legújabb fajtájával foglalkozik. A memória korrupciós hibák általában rendkívül veszélyesek, mivel egy ilyen jellegű hibával kedvezőtlen esetben akár tetszőleges kártékony kód futtatására is kényszeríthető a sérülékeny programot futtató operációs rendszer. A memória korrupció legkedvezőbb esetben is a hibás szoftver leállásához vezet (szolgáltatás-megtagadás). Az utóbbi években számos memóriakorrupciós hiba látott napvilágot. Ezek közül a legnagyobb hatásúak közé tartozik többek között a *Heartbleed* névre keresztelt *openssl* sérülékenység (CVE-2014-0160) [2] vagy az *lzo* tömörítőben több mint 20 évig jelen lévő (CVE-2014-4608) [3] sérülékenység is. Az előbbi a webszerverek mintegy 30%-át érintette, az utóbbi hatása is óriási, mivel az *lzo* tömörítést számtalan helyen használják.

A memória korrupció kiaknázása során a támadónak először kiaknázási módszert kell választani, amellyel eléri a célját. A hiba fajtája gyakran meghatározza a kiaknázás módját. A szoftvert futtató környezet védekezése is befolyásolhatja az alkalmazható támadási

technikákat olyan módon, hogy bizonyos támadás típusokat eleve kizár a védelem. Napjaink egyik legmodernebb szoftverhiba kiaknázási módja a Return Oriented Programming (ROP) [4], amellyel az operációs rendszer által biztosított Data Execution Prevention (DEP) [5] védelem hatékonyan kikerülhető. A gyakorlatban még kevésbé elterjedt, de elméletben sokat vizsgált kiaknázási módszer a Jump Oriented Programming (JOP) [6] [7], amely a ROP egy továbbfejlesztett, általánosított változata. A kutatás során a ROP és a JOP módszerek különböző megvalósítási módjait vizsgáltam figyelembe véve ezek korlátait.

2. Célkitűzések

A kutatás elsődleges célja a jelenleg rendelkezésre álló memória korrupción alapuló szoftverhiba kiaknázási módszerek fejlesztése. A memória korrupció okozta problémát a szoftvergyártók a védekezési módszerek és eszközök folyamatosan fejlesztésével próbálják kiküszöbölni. A szoftverhibák kiaknázására épülő informatikai támadásokkal szembeni kockázatot jelentősen lehet csökkenteni a szoftverek biztonsági tesztjeivel, a fordítóprogramokba beépített védelmi módszerek fejlesztésével és az operációs rendszerek által nyújtott védelmek fejlesztésével egyaránt.

A jelenlegi általános tapasztalat azonban az, hogy a szoftvereket készítő emberek gyakran hibáznak, a szoftverhibák kiszűrésére alkalmazott eszközök mind a forráskód ellenőrzésnél mind pedig a lefordított állomány ellenőrzésénél nem nyújtanak tökéletes védelmet és az operációs rendszereknél alkalmazott kifejezetten a memória korrupciós hibakiaknázások meggátolását célzó módszerek is megkerülhetőek. Gyakorlati szempontból elmondható, hogy általában minden védekező módszerre létezik azt megkerülő támadás és minden támadásra készíthető megfelelő védekezés. Mindezek miatt a memória korrupciós szoftverhiba kiaknázások módszerei folyamatosan fejlődnek és sajnos gyakran a gyártók csak utólag reagálnak egy-egy hibakiaknázási módszerre.

A kutatás során fő célomul tűztem ki, hogy kifejlesszek, és ezáltal felhívjam a figyelmet olyan új típusú memória korrupciós hiba-kihasználási lehetőségekre, amelyek biztonsági problémát jelenthetnek a jövőben. Ezen eredmények publikálásával lehetőség nyílik a védekezési oldalon időben megtenni a szükséges lépéseket.

A Return Oriented Programming és Jump Oriented Programming típusú hibakiaknázási módszerek jelentősége hatalmas, mivel ezen módszerekkel önálló támadó kód megírása nélkül is képes a támadó kártékony kódot futtatni a hibás szoftver nevében olyan módon, hogy a virtuális memóriában megtalálható kódrészletekből rakja össze a támadást.

Mivel mindkét kiaknázási módszer bizonyítottan Turing teljes, így elvben tetszőleges támadó kód előállítható ezekkel. Gyakorlati megvalósítás szempontjából azonban a hibakiaknázás során végrehajtandó tetszőleges kód futtatása több akadályba is ütközhet. Az egyik legnagyobb korlátot a rendelkezésre álló kódrészletek (úgynevezett gadgetok) jelentik. Hiába van lehetőség tetszőleges kód végrehajtására elméletben, ha nincs hozzá megfelelő kódrészlet az aktuális folyamat virtuális memóriájában. Szintén problémát jelenthet a memóriakorrupció helyén rendelkezésre álló hely nagysága. A ROP és JOP támadásoknál alkalmazott ténylegesen hasznos kód (payload), lényegesen hosszabb egy hagyományos adatszegmensen futó támadó kódhoz képest.

A fenti ROP-hoz és JOP-hoz köthető problémákat figyelembe véve az alábbi célkitűzéseket alkottam a kutatás során:

Mivel a rendelkezésre álló kódrészlet gadgetok erősen befolyásolják a ROP és a JOP támadások kódjait, ezért fontos, a gadgetok megkeresésére szolgáló algoritmusok fejlesztése. A JOP kiaknázásnál alkalmazott gadgetok közül a legnagyobb jelentőségű az úgynevezett dispatcher gadget, amely a teljes JOP kiaknázás lefutását vezérli. A jelenleg elérhető dispatcher gadget kereső algoritmusok használhatóak [10], bár túl kevés szempontot vesznek figyelembe, így csak csekély számú dispatcher gadgetnek megfelelő kódrészlet megtalálására alkalmasak. Célom volt, egy a rendelkezésre álló algoritmusoknál pontosabb és több szempontot figyelembe vevő algoritmus megalkotása, továbbá az elérhető dispatcher gadget jelöltek több szempontot figyelembe vevő osztályozása.

A ROP és JOP kiaknázási módszereknél korlátot jelenthet a rendelkezésre álló hely nagysága is. A hagyományos adatszegmensen történő kódvégrehajtáson alapuló hibakiaknázások során a rendelkezésre álló hely jelentette problémát úgynevezett egg-hunting [8] megoldásokkal és heap spray [9] payload elhelyezési technikával szokták megkerülni. A kód újrafelhasználáson alapuló technikák (ROP, JOP) esetében azonban ilyen módszereket eddig még nem vizsgáltak. Ezek alapján célom volt a ROP és JOP módszernek a heap spray technikával történő együttes alkalmazhatóságának vizsgálata és amennyiben ez a kombináció lehetséges, úgy ezen lehetséges új hibakiaknázás típus tulajdonságainak elemzése is. További célom volt a ROP módszer és az egg-hunting technika együttes alkalmazásának vizsgálata és

amennyiben ez a kombináció lehetséges, úgy ezen új hibakiaknázás típus tulajdonságainak elemzése is.

3. Vizsgálati módszerek

3.1 Dispatcher gadget keresési algoritmus fejlesztése

A dispatcher gadgetek keresésére vonatkozó kutatásaim során abból a hipotézisből indultam ki, hogy egy a virtuális memóriában megtalálható tetszőleges hosszúságú kódrészlet is működhet dispatcher gadgetnek megfelelően, ha a támadó kód összeállításához használt funkcionális gadgetek figyelembe veszik a dispatcher gadget belsejében található utasításokat és a kódrészlet első és utolsó utasítása megvalósítja az indexváltoztatás és indirekt ugrás kombinációt. Ezen állítás igazolásához egy olyan algoritmus kifejlesztése szükséges, amely azonosítja a virtuális memóriában megtalálható lehetséges dispatcher gadget kódrészleteket és feltételeket rendel a funkcionális gadgetek helyes működéséhez.

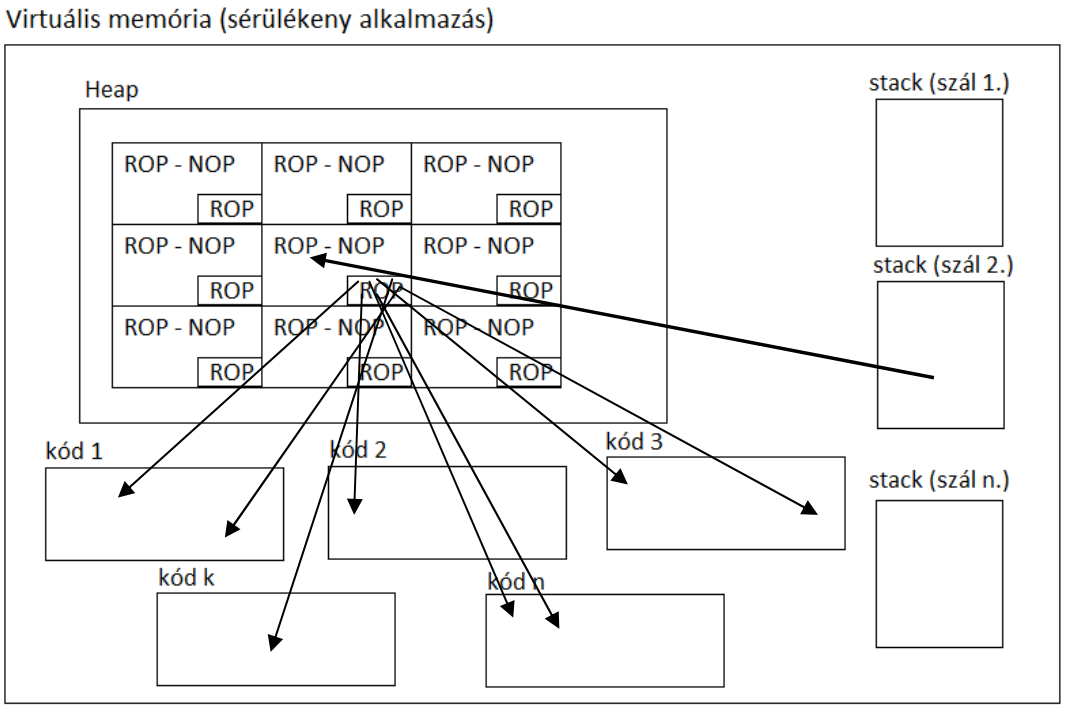
Az algoritmus első lépéseként a dispatcher gadget utolsó utasítását keresem, majd visszafelé lépkedve folyamatosan frissítem a funkcionális gadgetek helyes működéséhez tartozó feltételeket egészen addig, amíg a dispatcher gadget egy lehetséges első utasítását meg nem találom vagy a dispatcher gadget helyes működését kizáró utasítást nem kapok. Ez utóbbi esetben a dispatcher gadget jelöltet elveti az algoritmus.

A dispatcher gadgetek elemzésénél további fontos kérdés, hogy az egyes kódrészletek milyen típusú memóriakorrupcióhoz használhatóak. Ennek megállapításához a dispatcher tábla memóriában történő elhelyezkedését kell figyelembe venni.

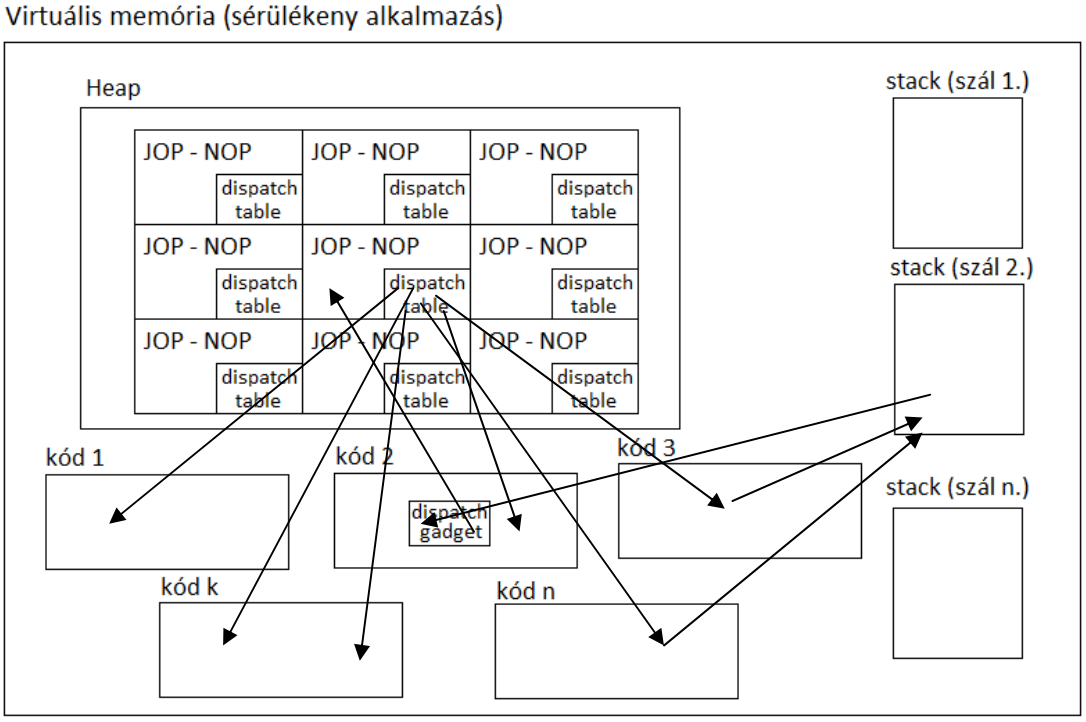
3.2. Kód újrafelhasználáson alapuló támadó kódok együttes használata a heap spray payload elhelyezéssel

A kifejlesztett új memóriakorrupciós kiaknázási technikám abból a hipotézisből indul ki, hogy a ROP és a JOP által biztosított tényleges támadó kód írása nélküli részekből összefűzött kódvégrehajtás kombinálható a heap spray payload elhelyezési technikával és ez esetben a két technika előnyös tulajdonságai összeadódhatnak.

Ennek bizonyításához egy olyan részletes kiaknázási technika leírás és elemzés szükséges mind a ROP és heap spray (1. ábra) mind pedig a JOP és heap spray (2.ábra) kombinációjához, amely magában foglalja az új kiaknázási technika részleteit.



1. ábra A Return Oriented Programming és a heap spray hibakihasználás kombinációja



2. ábra A Jump Oriented Programming és a heap spray hibakihasználás kombinációja

3.3. ROP módszer és egg-hunting együttes használata

Az egg-hunterekre kidolgozott új memóriakorrupciós hiba kiaknázási technikám abból a feltételezésből indul ki, hogy az egg-hunting típusú kiaknázásoknál alkalmazott payload keresési technika megvalósítható a DEP védelem mellett is olyan módon, hogy a payload keresést a ROP módszer segítségével valósítom meg.

Ennek igazolásához egy olyan elemzés szükséges, amely megvizsgálja a DEP megkerülési lehetőségek és az egg-hunting együttes használatát, valamint értékeli azokat használhatóság szempontjából. A DEP védelem megkerülése elméletben több megoldással is lehetséges.

3.3.1. A DEP kikapcsolása

Hagyományos szoftverhiba kiaknázások során a DEP védelem kikapcsolását legtöbbször valamely a virtuális memóriában már meglévő API metódushívással valósítanak meg. Ilyen pl. Windows operációs rendszerek esetén a *VirtualProtect* Windows API metódushívás. Az egg-hunter esetén a DEP kikapcsolását az egg-hunter helyén és a tényleges payload helyén is meg kell tenni, amelyhez egy a ROP működésének megfelelő kódlefutást kell biztosítani.

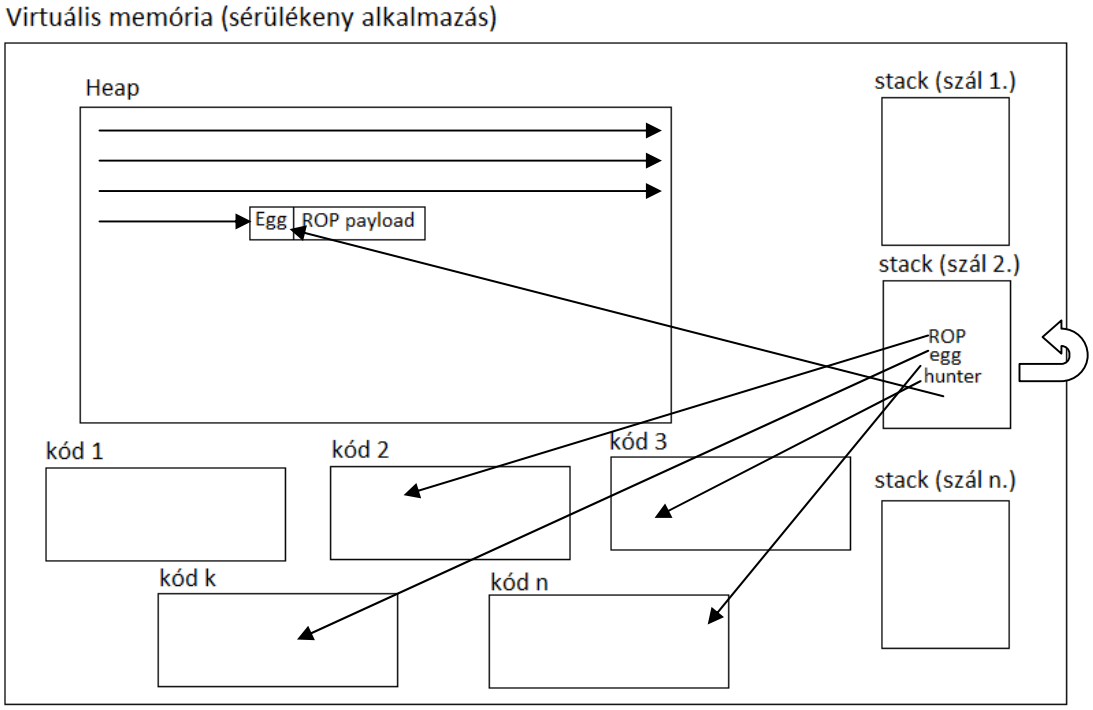
3.3.2. Egg-hunter másolás

A DEP megkerülésének egy másik lehetséges módja, az egg-hunter bemásolása és futtatása egy DEP mentes helyen. Ez a módszer csak akkor alkalmazható, amennyiben található a virtuális memóriában olyan szegmens, amely mentesül a DEP védelem alól (írás és végrehajtás is megengedett).

3.3.3. Tisztán ROP egg-hunter

Mivel a ROP Turing teljes, ezért elvben elképzelhető az egg-hunter típusú payload keresés tisztán ROP technikával történő megvalósítása (3. ábra). Ebben az esetben a ROP program lineáris kereséssel végignézi a memóriát, amelyhez egy ROP ciklusra van szükség.

A lehetséges megoldások gyakorlati szempontból történő vizsgálata választ ad arra a kérdésre, hogy mennyire őrzik meg a DEP védelem megkerülése miatt módosított egg-hunter megoldások az egg-hunter kódok alapvető jellemzőjét: a kód rövidségét.



3. ábra A Return Oriented Programming és az egg-hunter hibakihasználás kombinációja

3.4. Új megoldások helyességének igazolása

A kifejlesztett új megoldások helyes működését a szoftverhibák meglétének igazolásánál már megszokott úgynevezett "proof of concept" típusú exploitok megalkotásával lehet szemléltetni. Ezen exploitok segítségével megállapíthatóak a kifejlesztett megoldások előnyös és hátrányos tulajdonságai, kombinált módszerek esetén az egyesült kedvező jellemzők és a nem kívánt mellékhatások is.

4. Új tudományos eredmények

1. Tézis

A Jump Oriented Programming memóriakorrupciós hibakiaknázási technika dispatcher gadget-jának keresésére szolgáló meglévő algoritmusok számos lényeges kérdésben egyszerűsítéseket tesznek [10]. A dispatcher gadget helyes működését nem feltétlenül befolyásolja annak hossza, viszont kiválasztásánál figyelembe kell venni a memóriakorrupció típusát is. A funkcionális gadgetek kiválasztásánál figyelembe kell a dispatcher gadget közbenső utasításait is.

1. a, Új algoritmust dolgoztam ki a Jump Oriented Programming memória korrupciós szoftverhiba kiaknázási módszerhez tartozó dispatcher gadgetek keresésére. A kidolgozott módszert nem befolyásolja a vizsgált kódrészlet hossza és képes figyelembe venni a gadget belsejében található közbenső utasításokat. Erre a feladatra jelenleg még nincs pontos algoritmus az irodalomban. Meghatároztam a dispatcher gadget megfelelő működésének feltételeit. Igazoltam, hogy számos jelenlegi windowsos és linuxos operációs rendszer tartalmaz jól működő dispatcher gadgetnek alkalmas kódrészleteket.
1. b, Új eljárást dolgoztam ki a dispatcher gadgetek használhatóság szerinti osztályozására, amely figyelembe veszi a szoftverhiba típusát, a verem használhatóságát valamint az architektúra típusát is. Egy egyszerű mintatámadással szemléltettem a dispatcher gadgetek helyes működését.
[11] [12], (dolgozat 34.-54. oldal)

2. Tézis

A Return Oriented Programming és a Jump Oriented Programming memóriakorrupció hibakiaknázási technikák egyik lehetséges problémája a nem megfelelő nagyságú rendelkezésre álló hely a tényleges memóriakorrupció helyén. A heap spray payload elhelyezési technika esetén a payload nagysága lényegesen nagyobb lehet, így a két módszer kombinációja előnyös jellemzőkkel rendelkezhet.

2. a, Megvizsgáltam a Return Oriented Programming (ROP) memória korrupciós kiaknázási technika és a heap spray payload elhelyezési technika kombinálhatóságát, és a két módszer együttes használatával egy hatékonyan alkalmazható új támadási technikát dolgoztam ki. A kidolgozott technika azzal jellemezhető, hogy a payloadot a memóriakorrupció kiaknázása előtt helyezi el a memóriában és képes az adatvégrehajtás elleni védelem megkerülésére. Meghatároztam az együttes használat feltételeit, előnyeit és hátrányait. "Proof of concept" jellegű támadó kóddal bizonyítottam a ROP és heap spray technika kombinációjának helyes működést.
2. b, Megvizsgáltam a Jump Oriented Programming (JOP) memória korrupciós kiaknázási technika és a heap spray payload elhelyezési technika kombinálhatóságát és a két módszer együttes használatával egy hatékonyan alkalmazható új támadási technikát dolgoztam ki. A kidolgozott technika azzal jellemezhető, hogy az előzetes

payload elhelyezés és az adatvégrehajtás elleni védelem megkerülésén túl a ROP ellen kidolgozott védelmek megkerülésére is alkalmas. Meghatároztam az együttes használat feltételeit, előnyeit és hátrányait. "Proof of concept" jellegű támadó kóddal bizonyítottam a JOP és heap spray technika kombinációjának helyes működést.
[13], (dolgozat 55.-77. oldal)

3. Tézis

Az egg-hunting payload keresési technika egyik legnagyobb problémája a Data Execution Prevention (DEP) védelem. Mivel a ROP módszer kikerüli a DEP védelmet ezért a kettő kombinációja egy hatékony hibakiaknázási megoldást eredményezhet.

3. Megvizsgáltam a Return Oriented Programming (ROP) memória korrupciós kiaknázási technika és az egg hunting payload keresési technika kombinálhatóságát és a két módszer együttes használatával több, a memórialapok futtatásvédelmének megkerülésére alkalmas módszert dolgoztam ki. A kidolgozott módszerek képesek a payload megkeresésére és futtatására az adatvégrehajtás elleni védelem működése esetén is.
3. a, Kidolgoztam az egg-hunterek ROP technikával történő futtatható memóriaterületre történő másolásának és végrehajtásának néhány lehetséges megoldását, amelyek megkerülik az operációs rendszer DEP védelmét. Definiáltam az együttes használat feltételeit, előnyeit és hátrányait.
3. b, Kidolgoztam a tisztán ROP technikán alapuló egg-hunter kódvégrehajtás egy lehetséges megoldását, amely megkerüli az operációs rendszer DEP védelmét. Definiáltam az együttes használat feltételeit, előnyeit és hátrányait. "Proof of concept" jellegű támadó kóddal bizonyítottam a helyes működést.
[14] [15], (dolgozat 78.-97. oldal)

5. Az eredmények hasznosítási lehetősége

A kifejlesztett algoritmus és új hiba kihasználási technikák közvetlenül a támadó oldalt segítik. Ugyanakkor nem szabad elfelejteni, hogy az ellenőrzött és etikus körülmények között végzett támadások minden esetben hozzájárulnak az informatikai rendszerek biztonságnöveléséhez.

A megvalósított technikák és módszerek határozottan azzal a céllal készültek, hogy felhívják a figyelmet a szoftverhiba kiaknázások elleni védekezések gyenge pontjaira. Emellett lehetőséget adnak a védekezési módszerek fejlesztőinek mind az operációs rendszer, mind a fordító, mind pedig a szoftverek oldaláról nézve, hogy időben felkészüljenek az ehhez hasonló típusú hibakiaknázási módokra.

A különböző támadástechnikák publikálása azért is előnyös, mert ezekből más kutatók újabb és újabb ötleteket nyerhetnek ezáltal még kifinomultabb figyelemfelhívó szoftverhiba kiaknázási módszereket alkothatnak, amelyek közvetett módon tovább növelhetik szoftvereink biztonságát.

6. Irodalmi hivatkozások listája

[1] EC Council - Certified Ethical Hacker exam, http://www.eccouncil.org/Certification/certified-ethical-hacker, 2012

[2] OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160) https://www.uscert.gov/ncas/alerts/TA14-098A, 2014

[3] R. Lazarus, The 20 years old bug that went to Mars, http://blog.securitymouse.com/2014/06/raising-lazarus-20-year-old-bug-that.html, 2014

[4] E. Buchanan, R. Roemer, S. Savage, Return-Oriented Programming: Exploits Without Code Injection - http://cseweb.ucsd.edu/~hovav/talks/blackhat08.html, 2008

[5] Microsoft Support: A detailed description of the Data Execution Prevention on Windows XP SP2 - https://support.microsoft.com/en-us/kb/875352, 2006

[6] T. Bletsch, X. Jiang, and V. W. Freeh: Jump-oriented programming: A new class of code-reuse attack, ASIACCS '11, Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ACM New York, NY, USA 2011, pp. 30-40.

[7] S. Checkoway, L. Davi, A. Dmitrienko, A. Sadeghi, H. Shacham, M. Winandy, Return-oriented programming without returns, Proceedings of the 17th ACM conference on Computer and communications security, pp.559-572, 2010

[8] A. Ansari (HackSys Team): Egg-hunter - https://www.exploit-db.com/docs/18482.pdf, 2012

[9] corelanc0d3r (Corelean Team): Heap spraying demystified: https://www.corelan.be/index.php/2011/12/31/exploitwriting-tutorial-part-11-heap-spraying-demystified/#0x0c0c0c0c, 2011

[10] P. Chen, X. Xing, B. Mao, L. Xie, X. Shen, X. Jin: Automatic construction of jump-oriented programming shellcode (on the x86) Proceeding ASIACCS '11 Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, 2011, pp. 20-29

7. Tudományos közlemények

[11] L. Erdődi, Finding dispatcher gadgets for jump oriented programming code reuse attacks, SACI 2013 – 8th International Symposium on Applied Computational Intelligence and Informatics, Timisoara, Romania (IEEE), 2013, pp. 321-325.

[12] L. Erdődi, Comparison of different control gadgets for jump oriented programming, Scientific Bulletin of the Politechnica University of Timisoara, Transactions on Automatic Control and Computer Science, 2013, pp. 157-162.

[13] L. Erdodi, Applying Return Oriented and Jump Oriented Programming Exploitation Techniques with Heap Spraying, Acta Polytechnica Hungarica (közlésre elfogadva)

[14] L. Erdődi, Z. L. Nemeth: When Every Byte Counts – Writing Minimal Length Shellcodes - 13th IEEE International Symposium on Intelligent Systems and Informatics, Subotica, Serbia, 2015 (közlésre elfogadva)

[15] L. Erdődi, Conditional Gadgets for Return Oriented Programming, Conference: 5th IEEE International Symposium on Logistics and Industrial Informatics (LINDI 2013), 2013, pp.

[16] Erdodi L**,** File compression with LZO algorithm using NVIDIA CUDA architecture, LINDI 2012 – 4th IEEE International Symposium on Logistics and Industrial Informatics, Smolenice, Slovakia, 2012.09.05-2012.09.07. (IEEE)**,** pp. 251-254.

[17] Erdődi L, Memória alapú támadások kiaknázási lehetőségei, ISCD2013 Conference, 2-3. September 2013, Balatonöszöd, www.nbf.hu/anyagok/prezentaciok/Erdodi Laszlo_Obuda Uni.ppt, 2013.