UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH

PRÍRODOVEDECKÁ FAKULTA

VIZUALIZÁCIA AKTÍVNYCH SPOJENÍ V HONEYNETE

2014 **Terézia MEZEŠOVÁ**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH

PRÍRODOVEDECKÁ FAKULTA

VIZUALIZÁCIA AKTÍVNYCH SPOJENÍ V HONEYNETE

BAKALÁRSKA PRÁCA

Študijný program:	informatika
Pracovisko (katedra/ústav):	Ústav informatiky
Vedúci bakalárskej práce:	RNDr. JUDr. Pavol Sokol
Konzultant bakalárskej práce:	PhDr. RNDr. Peter Pisarčík

Košice 2014

Terézia MEZEŠOVÁ

Zadanie záverečnej práce

Zadanie záverečnej práce (ďalej len „zadanie“) je dokument, ktorým vysoká škola stanoví študentovi študijné povinnosti v súvislosti s vypracovaním záverečnej práce. Zadanie spravidla obsahuje: typ záverečnej práce, názov záverečnej práce, meno, priezvisko a tituly študenta, meno, priezvisko a tituly školiteľa, v prípade externého školiteľa meno, priezvisko a tituly konzultanta, školiace pracovisko, meno, priezvisko a tituly vedúceho pracoviska, anotáciu záverečnej práce, jazyk, v ktorom sa práca vypracuje, dátum schválenia zadania.

Vyhlásenie

Vyhlasujem, že som túto bakalársku prácu vypracovala samostatne na základe vedomostí získaných štúdiom a s pomocou uvedenej literatúry

_____

Terézia Mézešová

Poďakovanie

Rada by som poďakovala vedúcemu bakalárskej práce RNDr. JUDr. Pavlovi Sokolovi a konzultantovi práce PhDr. RNDr. Petrovi Pisarčíkovi za trpezlivosť, cenné pripomienky a obetavosť počas tvorby mojej bakalárskej práce.

Abstrakt v štátnom jazyku

Honeynet je bezpečnostný nástroj používaný na odhaľovanie nových bezpečnostných hrozieb a sledovanie správania útočníkov, ktorý sa vyznačuje tým, že je oddelený od počítačovej siete, do ktorej sa pripájajú používatelia. Vďaka tejto vlastnosti vieme s istotou povedať, že akákoľvek aktivita v tomto systéme je podozrivá a môže ísť o niektorý zo známych útokov alebo o neznámy spôsob útoku. Na zjednodušenie sledovania stavu, v ktorom sa honeynet aktuálne nachádza sa v práci venujeme vytvoreniu vizualizácie aktívnych sieťových spojení, ktoré sú jednou z prvých indikácií začínajúceho útoku dostupnej prostredníctvom webového prehliadača. Systém sa skladá z viacerých virtuálnych honeynetov. Informácie z nich sa ukladajú do vzdialenej centrálnej databázy. Serverová časť aplikácie tieto informácie predspracuje tak, aby vizualizácia prebiehajúca v prehliadači bola rýchla a nezaťažovala klientske systémové prostriedky. Zobrazujú sa údaje o aktívnych sieťových spojeniach v jednotlivých virtuálnych honeynetoch a vyťaženie pamäte a procesora virtuálneho honeynetu. **Kľúčové slová**: bezpečnosť, honeypot, honeynet, vizualizácia.

Abstrakt v cudzom jazyku

Honeynet is a network security tool used for discovering new security threats and observing attacker's behaviour. Its main feature is detachment from the computer network to which regular users connect. Based on this feature we can say with certainty that any activity in the system in suspicious and can be one of the known attacks or unknown forms of attack. To simplify monitoring the current state of the honeynet we create a web based visualization of the active network connections in the network, which are one the first indications of a starting attack. System has more virtual honeynets. Information from those are sent to remote central database. Server side of the application preprocesses the information in a way that the visualization running in the web browser is fast and efficient towards client's system resources. Data about the currently active network connections in all virtual honeynets are shows, are well the memory and CPU usage of the virtual honeynet.

1 Obsah

2 Zoznam skratiek a značiek

IP	Internet Protocol
IPv4	Internet Protocol, verzia 4
IPv6	Internet Protocol, verzia 6
NAT	Network Address Translation
ARP	Address Resolution Protocol
DHCP	Dynamic Host Configuration Protocol
MAC	Media Access Control
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SVG	Scalable Vector Graphics
PHP	Hypertext Preprocessor
JSON	JavaScript Object Notation

3 Slovník termínov

Forenzná analýza je interpretácia rozsiahleho objemu dát za použitia vizualizačných a extrakčných techník

Malvér je škodlivý softvér

Firewall je softvér slúžiaci na oddelenie sietí s rozličnými prístupovými právami a kontrolu toku dát medzi týmito sieťami

Paket je blok prenášaných dát v počítačovej sieti

IP forwarding je proces rozhodovania sa kam smerovať prichádzajúci paket

Broadcast je metóda smerovania prichádzajúceho paketu všetkým zariadeniam zapojených do počítačovej siete

IP adresa je logický číselný identifikátor uzla v počítačovej sieti

MAC adresa je identifikačné číslo sieťového adaptéra

NAT protokol je spôsob úpravy sieťovej premávky prepisom zdrojovej a/alebo cieľovej IP adresy

ARP protokol je používa sa na zistenie MAC adresy zariadenia na základe IP adresy

DHCP server je počítač, ktorý si uchováva zoznam dostupných IP adries v rámci počítačovej siete

Parser je program, ktorý vykonáva analýzu postupnosti formálnych prvkov s cieľom určiť ich gramatickú štruktúru voči dopredu známej formálnej gramatike

4 Úvod

Každým dňom na nás na Internete číhajú rôzne hrozby. Vedomosti o priebehu útokov a spôsoboch ich odhaľovania sa stávajú stále užitočnejšími informáciami. Ochrana pred krádežou a zneužitím osobných údajov a dát je najdôležitejším aspektom, ktorý sa snažia zabezpečiť bezpečnostní analytici po celom svete. S vývojom nových prostriedkov na odhaľovanie útočníkov a spôsobov ochrany sa objavujú stále nové spôsoby obídenia týchto opatrení a preniknutí do systémov. Medzi klasické prístupy, ktoré organizácie používajú patria firewall a detekčné systémy. Tieto, hoci zabránia veľkému počtu dobre známych útokov a poskytujú dostatočnú mieru ochrany nedokážu detegovať nové hrozby.

Potrebujeme nástroje, pomocou ktorých budeme schopní odhaliť nové typy útokov, sledovať ich priebeh a analyzovať správanie útočníkov tak, aby o tom nevedeli. Na odhaľovanie nových hrozieb a sledovanie správania útočníkov je vhodným nástrojom honeypot. Je to bezpečnostný nástroj vytvorený špeciálne na tento účel. Akákoľvek útočníkova aktivita zanecháva po sebe stopy, ktoré môžeme sledovať. Problémom však je veľké množstvo dát, ktoré je pri útokoch vygenerovaných, ktoré musia bezpečnostní analytici spracovať. Aby s týmito dátami mohli pracovať efektívne, musia byť schopní sa v nich rýchlo zorientovať. Z tohto dôvodu vzniká veľké množstvo rôznych bezpečnostných nástrojov na uľahčenie práce s honeypotom a administráciu honeypotu.

Najužitočnejšie nástroje používajú na prezentáciu dát získaných z honeypotu vizualizáciu. Ľudský mozog vníma a spracúva vizuálne podnety okamžite. Správne vytvorená vizualizácia pomáha pochopiť súvislosti medzi dátami oveľa rýchlejšie ako ich klasická textová forma. Pri prezentácií informácií vizuálnou formou sa kladie dôraz aj na psychológiu vnímania vizuálnych podnetov človekom, keďže práve táto časť najviac ovplyvňuje to, či výsledná vizualizácia slúži svojmu účelu.

V prvej a druhej kapitole sa venujeme teoretickým poznatkom z oblasti honeypotov, honeynetov a virtuálnych honeynetov, ich výhodám a nevýhodám a spôsobom použitia ako bezpečnostných nástrojov. V tretej kapitole sa zaoberáme vizualizácií v oblasti sieťovej bezpečnosti, psychológií vnímania a základným princípov zobrazovania údajov. Tieto poznatky aplikujeme v záverečnej kapitole, v ktorej popisujeme návrh a implementáciu vlastného riešenia vizualizácie ako webovej aplikácie. Súčasťou kapitoly je tiež prehľad podobných riešení.

1 Honeypoty

Táto kapitola prestavuje úvod do problematiky honeypotov. V rámci tejto kapitoly sa zaoberáme definíciou honeypotu, vlastnosťami jednotlivých typov honeypotov a použitím honeypotu ako bezpečnostného nástroja. Rozoberáme možné umiestnenie honeypotu v rámci systému a počítačovej siete a v závere kapitoly sa venujeme výhodám a nevýhodám honeypotu.

1.1 Definícia honeypotu

Hoci sa slovo **honeypot** začalo používať iba nedávno, honeypoty sa v rámci ich podstaty v počítačových systémoch používajú viac ako dvadsať rokov [1], [16]. Honeypoty predstavujú model používaný na forenznú analýzu. Ide o prostredie, kde sa slabiny vytvoria schválne za účelom sledovať útoky a prieniky do systému [10].

Honeypot tiež môže byť definovaný ako počítačový nástroj, ktorého hodnota leží v tom, že bude napadnutý. Je to počítačový nástroj nasadený v počítačovej sieti za účelom získavania informácií a štúdia útokov naň. Tieto systémy sú schválne ponechané nezabezpečené tak, aby nalákali útočníkov a my sme boli schopní študovať ich techniky, zámery a použité nástroje [15].

Na druhej strane Lance Spitzner definuje honeypot ako informačný systémový nástroj, ktorého hodnota leží v neautorizovanom alebo nedovolenom použití [13]. Má na starosti získavanie informácií o útočníkoch, malvéri a iných hrozbách za účelom získania detailov ohľadom útočníkovho správania, praktikách a cieľoch.

1.2 Klasifikácia honeypotov

Honeypoty môžeme deliť podľa niekoľkých kritérií. V tejto práci sme vybrali 4 základné kritéria, medzi ktoré zaraďujeme spôsob interakcie, spôsob implementácie, využitie honeypotu a akú rolu honeypot zastáva.

1.2.1 Rozdelenie podľa spôsobov využitia

Podľa spôsobu využitia delíme honeypoty na produkčné a výskumné. **Produkčné honeypoty** sú používané na ochranu organizácií. Môžu značne znížiť riziko napadnutia,

tým, že odhaľujú nedostatky a slabiny systému a upozornia administrátorov siete na prípadný útok. Produkčný honeypot však neslúži ako ochrana proti útoku, samotný honeypot útočníkov nezadrží. Organizácia sa nesmie spoliehať iba na produkčný honeypot a sieť, dáta a systémy musí zabezpečiť aj klasickými bezpečnostnými prostriedkami ako sú firewall, prevenčné a detekčné systémy, antivírusové programy a autentifikačné mechanizmy.

Výskumné honeypoty sú používané na skúmanie hrozieb, ktorým organizácie čelia. Výskumné honeypoty sú rôzne nakonfigurované tak, aby útočníka prilákali. Následne zaznamenávajú celú jeho činnosť. Získavajú tak informácie hlavne o zatiaľ neznámych útokoch. Sú vynikajúce nástroje na zachytávanie automatizovaných útokov, ako sú červy. Tieto útočia na celú sieť a výskumné honeypoty ich vedia rýchlo zachytiť.

1.2.2 Rozdelenie podľa miery interakcie

Rozdelenie podľa miery interakcie je po spôsobe využitia najčastejšie spomínané rozdelenie honeypotov. Pod mierou interakcie sa rozumie rozsah udalostí a príkazov, na ktoré honeypot dokáže zareagovať. **Vysoko-interaktívne honeypoty** sú reálne systémy poskytujúce reálne služby, s ktorými sa dá interagovať. Útočník dokáže stroj skompromitovať a získať nad ním kontrolu. Takto môžeme získať o spôsobe útoku a jeho priebehu veľa cenných informácií (motív, spôsob útoku a nástroje, ktoré sú použité).

Je však potrebné myslieť na to, že nechceme aby mal útočník prístup ku súkromným dátam. Pri honeypotoch s vysokou mierou interakcie je vysoké riziko, že honeypot bude zneužitý na ďalší útok na iné služby na Internete, čo môže mať aj právne dôsledky [5]. Vysoko-interaktívny honeypot je konvenčný počítačový systém ako klasický počítač, smerovač alebo prepínač. V rámci siete nemá žiadnu špecifickú úlohu ani bežných používateľov, nemali by na ňom bežať žiadne nezvyčajné procesy a nemal by komunikovať po sieti okrem bežných systémových procesov bežiacich pod operačných systémom.

Akákoľvek aktivita na honeypote je teda podozrivá a môže viesť k detekcii útoku. Môžeme sledovať ako útočník hľadá ciele pre útok, aké techniky používa, aby získal informácie o systéme, ako postupuje pri kompromitácií systému. Dokážeme priamo sledovať jeho kroky, ako komunikuje s ostatnými ľuďmi, prípadne ako po sebe zahladzuje stopy. Príkladom vysoko-interaktívnych honeypotov je Sebek, Qebek a pod.

Vysoko-interaktívne honeypoty sú náročné na inštaláciu. Potrebujú viacero nástrojov na správu, monitorovanie činnosti a zabezpečenie. Preferuje sa virtuálny stroj pred fyzickým, keďže je menšie riziko, že bude skompromitovaný priamo samotný stroj, na ktorom fungujeme. Virtuálne honeypoty bližšie popisujeme v kapitole 2.

Nízko-interaktívne honeypoty, na rozdiel od vysoko-interaktívnych honeypotov, nepredstavujú reálny systém a služby iba emulujú. Väčšinou ide iba o jednu službu ako napríklad ftp alebo http server. Miera interakcie by mala byť dostatočná na to, aby emulovaná služba bola vierohodná na zmätenie útočníka alebo automatizovaných nástrojov. Výhodou nízko interaktívnych honeypotov je ich jednoduchosť a ľahká údržba.

Keďže nízko interaktívne honeypoty služby systému iba emulujú, poskytujú iba obmedzený prístup k operačnému systému honeypotu a teda nemôže byť úplne ovládnutý. Je nevhodný na detekciu ešte neznámych útokov. Používa sa skôr na detekciu už známych útokov a meranie ich výskytu. Nízko interaktívne honeypoty majú viaceré výhody - ľahko sa inštalujú a spravujú, nie sú náročné na zdroje a nemôžu byť kompletne ovládnuteľné. Príkladom tohto typu honeypotov je Dianea, Kippo a pod.

1.2.3 Rozdelenie podľa spôsobu implementácie

Fyzický honeypot je reálny stroj bežiaci na reálnom operačnom systéme a poskytuje reálne služby. Je pripojený v sieti a dostupný cez jednu IP adresu. Väčšinou sú úzko späté s vysoko interaktívnymi honeypotmi. Sú náročné na hardvér a údržbu.

Virtuálne honeypoty sú implementované na jednom fyzickom stroji ako virtuálny operačný systém. Sú efektívnejšie, vieme na jednom stroji prevádzkovať viacero honeypotov, dokážeme emulovať viacero IP adries [7]. Virtuálny honeypot je škálovateľný a jednoduchý na údržbu. Nevýhodou môže byť to, že na fungovanie musí byť honeypot prístupný z Internetu. Aby sme dostali hodnotné dáta, neodporúča sa umiestniť honeypot do siete využívajúcej NAT [5].

1.2.4 Rozdelenie podľa role honeypotu

Serverové honeypoty sú tradičné honeypoty. Vydávajú sa za sieťové prvky a pasívne čakajú na útok. Oproti tomu **klientske honeypoty** sú aktívne. Vydávajú sa za klienta určitej služby a pozorujú odchýlky od očakávaného správania. Môžu napríklad

emulovať webové prehliadače a odchytávať malware, ktorý sa šíri cez napadnuté stránky (napr. glastopf).

1.3 Umiestnenie honeypotu

Honeypot môžeme vzhľadom na firewall umiestniť na 3 miestach:

1. **externe**,
2. **interne**,
3. v tzv. **demilitarizovanej zóne** (DMZ) [7]

1.3.1 Externé umiestnenie

Pri externom umiestnení je honeypot umiestnený mimo počítačovej siete organizácie a je pripojený do Internetu bez použitia akéhokoľvek firewallu. Takto je priamo vystavený útokom a sondám. Toto umiestnenie sa používa, ak chceme zachytávať činnosť najzlomyseľnejších útočníkov. Externé umiestnenie je pomerne jednoduché na inštaláciu. Chýbajúci firewall však sťažuje zachytávanie dát z honeypotu, a teda je vhodné takto umiestnený honeypot neustále sledovať, respektíve zapnúť ho iba v prípade, že ho aktívne sledujeme. V iných prípadoch ho vypnúť. To však porušuje význam honeypotu.

Hoci umiestnenie honeypotu mimo firewallu znižuje riziko napadnutia produkčného systému, takisto však aj jeho schopnosť emulovať tento produkčný systém a generovať logy je značne limitovaná.

1.3.2 Interné umiestnenie

Interné umiestnenie honeypotu znamená, že sa nachádza v počítačovej siete organizácie za firewallom. Takéto umiestnenie je najlepšie na vytvorenie systému varujúceho pred útokmi. Dokáže zachytiť útok, ktorý prejde ochrannými prostriedkami. Tento systém zachytí aj hrozby z vnútra počítačovej siete organizácie.

Nevýhoda spočíva v tom, že ak je interne umiestnený honeypot skompromitovaný, je kontrola dát v lokálnej sieti náročná. Honeypot je umiestnený za firewallom, a teda je na sieťovom administrátorovi, či povolí smerovanie internetovej komunikácie cez honeypot, alebo či táto komunikácia bude smerovaná cez produkčné

prostriedky, prípadne, či budú presmerované iba niektoré porty. Ak napríklad v produkčnom systéme neprevádzkujeme webový server, môžeme všetky HTTP požiadavky presmerovať na honeypot. Týmto spôsobom sa dá výrazne spomaliť šírenie červov ďalej do produkčnej siete organizácie. Je však potrebné poriadne premyslieť, ktoré porty majú byť presmerované a kam.

1.3.3 Umiestnenie v demilitarizovanej zóne

Demilitarizovaná zóna (DMZ) je spôsob implementácie firewallu. Dá sa prirovnať ku zóne medzi dvoma vojnovými stranami. Predstavuje neutrálne územie, prístupné z obidvoch strán. Najdôležitejšie je, aby sa z jednej strany na druhú nedalo prejsť priamo, teda je teda zakázaný tzv. IP forwarding. Komunikáciu medzi dvoma stranami zabezpečujú proxy brány. Najjednoduchšie riešenie je použitie dvoch prepínačov, medzi ktorými táto demilitarizovaná zóna vznikne a tieto prepínače zastávajú úlohu proxy brán.

Umiestnenie honeypotu v demilitarizovanej zóne je vo väčšine prípadov najlepším riešením. Do nej sa pridá prepínač, ktorý zabezpečí vrstvu na kontrolu a zachytávanie dát. Honeynet a produkčné DMZ servery majú tú istú logickú podsieť, IP adresa môže byť verejná aj neverejná. Toto umiestnenie je lepší výhodnejšie riešenie ako interné umiestnenie honeypotov, keďže cez firewall neprejde všetka komunikácia. Takto môžeme v rámci DMZ emulovať servery, ktoré sú voľne dostupné. Tie zvyšujú bezpečnosť internej siete a produkčného prostredia. Všetky ostatné systémy v DMZ nepatriace k honeypotu musíme zabezpečiť a ochrániť, aby nemohli byť nepadnuté [7].

1.4 Výhody a nevýhody honeypotu

Tak ako iné bezpečnostné nástroje aj honeypot má isté výhody a nevýhody. Názory na používanie honeypotov sa medzi odborníkmi na bezpečnosť líšia. Tí, ktorí používanie honeypotu podporujú hovoria, že dobre zabezpečený honeypot, čo najviac sa približujúci reálnemu systému dokáže útočníka zachytiť. Ďalšou výhodou, ktorá sa spomína je použitie honeypotu na identifikáciu kompromitovaných používateľský účtov a možnosť analýzy techník, ktoré útočník používa.

Názory proti používaniu honeypotu najčastejšie argumentujú, že snahu a úsilie vynaložené na zabezpečenie honeypotu môžeme radšej obrátiť na zabezpečenie

produkčného systému. Argumentom je tiež, že v prípade nedostatočnej ochrany sa honeypot môže stať skôr nedostatkom ako výhodou produkčného systému.

1.4.1 Výhody honeypotu

Keďže honeypot sa v produkčnom systéme nepoužíva, väčšinu dát zachytených z honeypotu môžeme považovať za dáta zachytávajúce útok. Eliminuje tak falošne pozitívne ale aj falošne negatívne varovania o útokoch, teda komunikáciu vyhodnotenú ako útok, prípadne nerozpoznaný útok. Práve nízka hodnota šumu v dátach vedie ku rozpoznaniu útoku v jeho začiatkoch a pomerne rýchlej detekcii. Na útok vieme vďaka tomu rýchlo a primerane reagovať.

Každý pokus o komunikáciu s honeypotom je považovaný za potenciálnu hrozbu, čo znamená, že sme schopný zachytiť nielen známe spôsoby útokov, ale aj doteraz neznáme útoky. Honeypoty pravidelne objavujú nové spôsoby preniknutia do systému, zachytávajú sa všetky informácie spojené s útočníkom, od všetkej sieťovej komunikácie, zadaných príkazov po stiahnutý škodlivý malware.

Ak škodlivý malvér predsa len prenikne cez firewall alebo prevenčný a detekčný systém, šanca, že bude zachytený honeypotom, je vysoká. Honeypot teda môže predstavovať istý stupeň ochrany, nemali by sme sa však spoliehať iba naňho.

Medzi ďalšie výhody honeypotu patrí jednoduchosť, škálovateľnosť (pri virtuálnych honeypotoch), použiteľnosť ako systémov skorého varovania, schopnosť zachytenia nepoznaných nástrojov a taktík, nízke hardvérové požiadavky.

1.4.2 Riziká a nevýhody honeypotu

Prevádzkovanie honeypotu má aj isté riziká, medzi nich patria napríklad:

- sú bezcenné, ak ich nikto nenapadne
- dokážu zachytiť iba útoky vykonávané priamo na nich, ak je napadnutý iný systém v sieti, honeypot tento útok nezachytí
- môžu byť zneužité na vykonanie ďalších útokov voči iným systémom, čo môže mať aj právne dôsledky
- veľkým nedostatkom honeypotov je ich detekcia

- vysoká finančná a časová náročnosť na inštaláciu a správu, v prípade fyzických vysoko interaktívnych honeypotov

1.5 Detekcia honeypotu

O tzv. **fingerprintingu** hovoríme vtedy, keď útočník rozpozná honeypot na základe niektorých jeho vlastností alebo správania sa. Nesprávne nakonfigurovaný honeypot sa dokáže prezradiť sám. Fingerprinting je veľkou nevýhodou najmä pri výskumných honeypotoch, kde je snaha o zachytenie nových typov útokov. Ak by útočníci honeypot odhalili, mohli by dáta znehodnotiť či zneškodniť.

Ak neprídu na to, že zaútočili na honeypot, môžu ho použiť na uloženie nástrojov, ktoré plánujú využiť na vykonanie ďalších útokov z honeypotu. Honeypoty s nízkou mierou interakcie sa dajú rozpoznať po hĺbkovej analýze odpovedí zo siete a nájdení logických nedostatkov ako nezrovnalosti medzi operačným systémom a službami, ktoré honeypot emuluje. Virtuálne stroje síce poskytujú výkon, škálovateľnosť a flexibilitu, nie sú však transparentné a je veľmi jednoduché ich odhaliť [5].

Pri vysoko interaktívnych honeypotoch chceme znížiť riziko zneužitia na ďalší útok a odchádzajúca komunikácia je častokrát obmedzená na 15 TCP spojení denne. Takto reštriktívne opatrenia sú však najrýchlejšie odhaliteľné, čo je pre útočníka viac ako podozrivé.

Honeypoty si od svojho počiatku našli stúpencov i odporcov v radoch bezpečnostných analytikov. Argumenty pre a proti vždy vychádzajú z účelu, na ktorý je honeypot použitý. Myslíme si však, že ako taký, je honeypot užitočných bezpečnostným nástrojov, najmä na odhaľovanie nových hrozieb a má zmysel sa tejto oblasti informačnej bezpečnosti venovať.

2 Honeynety, virtuálne honeynety

V tejto kapitole sa venujeme definícií honeynetov, virtuálnych honeynetov. V rámci tejto kapitoly sa zaoberáme základným častiam honeynetov. Súčasne popisuje, čo je virtualizácie a aké poznáme prístupy virtualizácie.

2.1 Definícia honeynetu

Honeynety možno definovať ako vysoko-interaktívny honeypot. Bezpečnostní špecialisti definujú honeynet ako sieť honeypotov, či už nízko-interaktívnych alebo vysoko-interaktívnych. Koncept týchto honeynetov je veľmi jednoduchý a vyžaduje vybudovanie siete s viacerými štandardnými produkčnými systémami. Útočníci tak v rámci honeynetu môžu skúmať a interagovať s akýmkoľvek systémom. V tejto sieti nič nie je emulované, všetko sú reálne operačné systémy a služby.

Honeynety sú extrémne flexibilné, môžu zastávať akúkoľvek rolu honeypotu, keďže sú to autentické systémy, sú vynikajúcim nástrojom na zmätenie útočníka. Koncept honeynetu je rovnaký ako pri honeypote, namiesto jedného systému je ich viac. Sú náročné na vybudovanie, hlavne kvôli nárokom na architektúru.

2.2 Základné časti honeynetu

Pri budovaní honeynetu je nutné sa zaoberať najmä 3 dôležitými časti [7]:

- kontrolou dát,
- zber dát a
- uchovávanie dát.

2.2.1 Kontrola dát (data control)

Kontrola dát zmierňuje riziko. Kontroluje útočníkovu aktivitu a určuje, aká miera komunikácie je povolená v rámci siete honeynetu a aká do vonkajšej siete. Nastáva riziko, že ak je skompromitovaný jeden systém v honeynete, tento môže byť ďalej zneužitý na útok voči systémom mimo honeynetu. Práve tieto systémy musíme ochrániť a je na nás zabezpečiť, aby toho útočník nebol schopný. Najväčšou výzvou je automatizovaná kontrola dát, najlepšia je kombinácia automatizovanej a manuálnej kontroly (často

používaným nástrojom je Honeywall). Je dobré zabezpečiť aspoň dve vrstvy kontroly dát v prípade zlyhania niektorej z nich.

Kontrola dát by takisto mala byť z pohľadu útočníka nedetekovateľná a vzdialene dostupná pre administrátora. Ak útočník kontrolu dát odhalí môže dáta zmanipulovať, čím sa stanú bezcenné.

2.2.2 Zachytávanie dát (data capture)

Pre zachytenie aktivity útočníkov je **zachytávanie dát (data capture)** kľúčovou činnosťou honeynetu. Medzi požiadavky na nástroje, ktoré túto funkcionalitu zabezpečujú patria:

- zachytenie akejkoľvek sieťovej, systémovej, aplikačnej a používateľskej aktivity,
- prezeranie tejto aktivity administrátormi v reálnom čase vzdialene a
- neukladanie dát priamo na honeypote, ale na inom mieste v rámci honeynetu.

Jedným zo spôsobov ako zachytávať dáta je logovať systémovú prevádzku lokálne a na vzdialený logovací server. V Unixových systémoch umožníme vzdialené logovanie pridaním vzdialeného servera do konfiguračného súboru. Pre operačný systém Windows existujú aplikácie, ktoré umožnia logovať systémovú prevádzku na vzdialené miesto. Skutočnosť, že systémovú prevádzku logujeme aj na iné ako lokálne úložisko nemusíme nijako špeciálne maskovať. Ak útočník deteguje, že logovanie je uskutočňované aj na vzdialený server, prinajmenšom znemožní toto vzdialené logovanie. Napriek tomu, však budeme mať informácie o tom, ako útočník získal do systému prístup. V horšom prípade sa pokúsi logovací server kompromitovať. Logovací server je ale oveľa lepšie zabezpečený, útočník bude musieť použiť pokročilejšie techniky, ktoré môžeme odchytiť. Pre prípad kompromitácie logovacieho servera môžeme použiť detekčný systém, ktorý pasívne takisto zachytáva všetku sieťovú aktivitu.

2.2.3 Zber a uchovávanie dát (data collection)

O **zbere dát (data collection)** uvažujeme v prípade, ak má organizácia niekoľko honeynetov. Tento zber je vykonávaný centrálne z jedného miesta pre všetky systémy v

honeynete. Korelácia všetkých týchto dát pridáva hodnotu pre výskumné honeynety. Je vhodné mať jednotnú konvenciu pre pomenovávanie a mapovanie na uľahčenie analýzy. Prenos dát z honeynetu do centrálneho úložiska musí zabezpečiť integritu, autenticitu a diskrétnosť dát.

2.3 Virtualizácia

2.3.1 Definícia virtualizácie

Virtualizácia je spôsob rozdelenia prostriedkov počítača na viacero samostatných prostredí. Virtualizáciu môžeme uskutočniť viacerými spôsobmi: rozdelením hardvéru alebo softvéru, zdieľaním času, čiastočnou alebo úplnou simuláciou počítača, emuláciou a inými [12].

Virtualizácia umožňuje dynamické škálovanie systémových prostriedkov, jednoduchú administráciu a zdieľanie zdrojov, bez čoho by nebolo možné realizovať napr. cloud-computing.

2.3.2 Výhody virtualizácie

Virtuálne stroje môžeme použiť na zmiernenie záťaže napr. jedného servera. Na jednom fyzickom stroji vytvoríme niekoľko virtuálnych systémov, z ktorých každý dokáže samostatne zastávať úlohu daného servera. Šetria sa náklady na systémové prostriedky a administráciu infraštruktúry. Virtualizácia je vhodná v prípadoch, ak potrebujeme spustiť staršiu aplikáciu, ktorá nie je kompatibilná s novým typom hardvéru alebo novom verziou operačného systému.

Vo virtuálnom stroji môžeme spustiť nedôveryhodné aplikácie, bez obáv, že aplikácia poškodí systém. Dokážeme emulovať hardvér, alebo hardvérové prvky, ktoré nemáme dostupné. S pomocou virtualizácie vieme prevádzkovať súčasne niekoľko operačných systémov, prípadne rôzne verzie jedného operačného systému.

Virtualizácia izoluje prostredie, v ktorom program beží, čo vieme využiť na pozorovanie správania sa systému, ak sú v ňom chyby. Poskytovaná izolácia prostredia robí z virtualizácie vhodný nástroj na výskum a experimenty [12].

2.3.3 Spôsoby virtualizácie

Platformová virtualizácia umožňuje **serverovú** aj **desktopovú virtualizáciu**. Pod platformou sa myslia všetky hardvérové komponenty počítača, ako procesor, úložný priestor, ale aj sieťové rozhrania a zbernice, ako USB alebo sériové porty. Komponent, ktorý umožňuje zdieľať tieto fyzické prostriedky a implementuje pravidlá na ich zdieľanie medzi viacerými užívateľmi nazývame **hypervízor**. Hostované stroje sa nazývajú virtuálne stroje(skratka VM je tak konzistentná so skratkou z anglického názvu virtual machine). Názov je zoskupením spolu pre operačný systém a aplikácie, ktoré v ňom bežia

Hypervízor môže byť implementovaný dvoma spôsobmi – buď je nasadený priamo na hostiteľskom stroji a slúži ako platforma (v tomto prípade hovoríme o serverovej virtualizácií), alebo je to aplikácia, ktorá beží v rámci hostiteľského operačného systému. V tomto prípade hovoríme o desktopovej virtualizácii) [6].

Virtuálny stroj používa obal, ktorý špecifikuje a ohraničuje jeden konkrétny virtuálny stroj. Ten nie je nič iné ako súbor nejakého formátu. Virtuálny disk, ktorý virtuálny stroj používa je takisto súbor zaobalený v rámci tohto virtuálneho stroja. Jednoducho teda vieme virtuálny stroj presunúť na iný fyzický stroj, napríklad počas plánovanej údržby, čím sa zabezpečí takmer neprerušený chod virtuálneho stroja.

Desktopová virtualizácia je model virtualizácie, kedy fyzický stroj je virtualizovaný ako klient/server model cez sieť. Fyzický stroj existujúci na inom mieste a je cez sieť virtualizovaný na iný stroj. Je to stále viac a viac používaný model virtualizácie, ktorý našiel široké uplatnenie v cloud-computingu. Desktop existuje ako virtuálny stroj na serveri. Jediný spôsob ako s ním komunikovať je rozšíriť ho cez sieť ku klientovi. Toto rozšírenie sa deje cez protokol virtuálneho desktopového rozhrania. Protokol bol navrhnutý špeciálne pre potreby virtualizácie desktopu a jeho rozhrania vzdialeným užívateľom.

Keďže jednotlivé virtuálne stroje sú zoskupené na jednom fyzickom stroji, je pravdepodobné, že medzi sebou budú komunikovať. Z tohto dôvodu má v sebe hypervízor zabudované nástroje na optimalizovanie sieťovej prevádzky. Virtuálny stroj má k dispozícií virtuálnu sieťovú kartu, ktorá je pripojená k fyzickej sieťovej karte fyzického stroja alebo na virtuálne rozhranie vnútri hypervízora. Virtuálne sieťové karty je možné pripojiť na virtuálne prepínače na oddelenie prevádzky vnútri virtuálnej sieťovej

vrstvy u hypervízora. Túto techniku nazývame **sieťová virtualizácia** a využíva sa napríklad pri testovaní softvéru, kedy vytvorením virtuálnej siete dokážeme simulovať podmienky, do ktorých bude softvér nasadený. Testuje sa tak prepojenie aplikácií, servisov, závislostí a koncových užívateľov so systémom. Príkladom sieťovej virtualizácie je softvér Microsoft Virtual Server.

Techniky sieťovej virtualizácie logicky oddeľujú rôzne siete umiestnené na jednom fyzickom stroji. Sú užitočným nástrojom na zabezpečenie izolácie a zdieľania sieťových prostriedkov medzi rôznymi užívateľmi. Nové mechanizmy zjednodušujú preposielanie paketov a umožňujú flexibilnejšiu a manažovateľnejšiu centralizovanú správu [7]. Sieťová virtualizácia pridáva na komplexnosti prostredia. S každou novou virtuálnou sieťou, ktorú vytvoríme, pridávame na náročnosti administrácie, je náročnejšie priradiť virtuálne prostriedky späť ku fyzickým [2].

2.3.4 Typy virtualizácie

Pre platformovú virtualizáciu rozlišujeme 3 spôsoby virtualizácie:

- plná virtualizácia
- paravirtualizácia a
- virtualizácia na úrovni operačného systému

Plná virtualizácia poskytuje dostatočnú emuláciu platformy tak, že hosťujúci operačný systém a v ňom bežiace aplikácie dokážu fungovať bez úprav a nemajú vedomosť o tom, že sú virtualizované. Zariadenia, ktoré emulujeme musia dovoliť hosťujúcemu operačnému systému manipulovať s nimi na natívnej úrovni, ako sú napríklad registre.

Emulácia zariadenia musí vedieť emulovať aj špeciálne vlastnosti a požiadavky niektorých zariadení, čo nevytvára z pohľadu výkonu, práve najefektívnejší spôsob. Cena môže byť dosť vysoká, keďže hosťujúci operačný systém s emulovaním zariadením pracuje jemu vlastným spôsobom a hypervízor, ktorý emuláciu implementuje zároveň komunikuje s fyzickým zariadením, ktoré je častokrát úplne iného typu. Príkladom implementácie plnej virtualizácie je Hyper-V, Virtual Box a pod.

Paravirtualizácia rieši problém emulácie zariadení úplnej virtualizácie tým, že hosťujúci operačný systém si je vedomý toho, že je virtualizovaný. Hosťujúci operačný systém nepoužíva svoje ovládače, ale tie sú integrované spolu s ovládačmi hypervízora na efektívny prístup a zdieľanie fyzických zariadení. Príkladom implementácie paravirtualizácie je XEN, KVM.

Virtualizácia operačného systému je vysoko efektívny spôsob virtualizácie. Táto technika neemuluje hardvérovú platformu. Implementuje sa v rámci hostiteľského jadra, poskytuje prostriedky na izoláciu pamäte, I/O zariadení, sieťového rozhrania a procesora. Umožňuje spúšťať viac navzájom nezávislých a izolovaných virtuálnych strojov. Tie sa označujú ako **kontajnery**. Aplikácia zabezpečujúca virtualizáciu na úrovni operačného systému vytvorí virtualizačnú vrstvu, ktorá riadi beh týchto kontajnerov, zaisťuje abstraktné rozhranie na prístup k hostiteľskému systému a prideľovanie systémových prostriedkov. Medzi najznámejšie implementácie patrí OpenVZ, Linux V-Server alebo Linux container (LXC).

Takáto forma virtualizácie má najnižšiu réžiu z hľadiska straty výkonu hardvéru. Na rozdiel od hardvérovej virtualizácie je pri virtualizácií na úrovni operačného systému možný prenos virtuálnych kontajnerov na iný fyzický stroj. Niektoré virtualizované operačné systémy dokonca poskytujú mechanizmy na zdieľanie virtualizovaného súborového systému medzi jednotlivými kontajnermi. Tie sú pri zmene súboru schopné automaticky vytvoriť jeho kópiu pre daný virtuálny kontajner. Je to jednoduchšie na zálohovanie a jednotlivé kontajnery môžu pracovať so zmenenými kópiami súborov, vytvárať nové alebo sa dokonca vrátiť k predchádzajúcim snímkam celého systému, tzv. snapshot systému. Pri implementácií virtuálneho honeynetu používame v práci práve túto techniku virtualizácie.

2.4 Virtuálny honeynet

V predchádzajúcich kapitolách sme si zadefinovali pojem honeynetu a virtualizácie. V nasledujúcich podkapitolách sa budeme venovať spojeniu práve týchto technológií.

2.4.1 Definícia virtuálneho honeynetu

Virtuálny honeynet je typ honeynetu, ktorý umožňujúce spustiť celý honeynet na jednom fyzickom stroji. Tento systém je navrhnutý tak, aby vyzeral ako reálny funkčný systém. Na rozdiel od honeypotu, ktorý je vo väčšine prípadov fyzický stroj, virtuálny honeynet používa virtualizačné techniky na simuláciu siete. Výhodou virtuálneho honeynetu je výhodnejšie použitie systémových prostriedkov a jednoduchšia správa jednotlivých honeypotov.

Virtuálny honeynet používa virtualizačný softvér na vytvorenie nového, samostatného prostredia. Softvér, ktorý na virtualizáciu rozhodneme použiť by mal spĺňať nasledujúce kritériá [7]:

- podpora rôznych operačných systémov v rámci virtuálneho prostredia
- niekoľko spôsobov virtualizácie siete
- možnosť vytvorenia obrazu hosťovského operačného systému – vďaka tomu dokážeme v prípade útoku ľahko získať aktuálny stav systému a následne ho obnoviť.

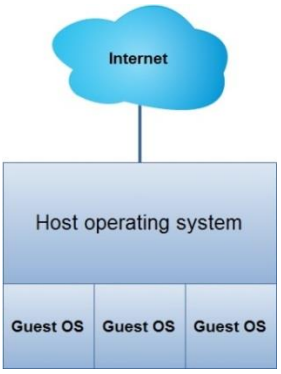
Na zachytávanie dát z virtuálneho honeynetu uplatňujeme rovnaké pravidlá ako pri klasickom honeynete. Požadujeme zachytenie akejkoľvek aktivity útočníka s tým, že dáta nemusia byť ukladané len na vzdialenom serveri, aby neboli zmazané, alebo zmenené aktivitou útočníka, ale je možné ich ukladať priamo na virtualizovanom hardvéri. Podmienkou je zabezpečenie ich nezmeniteľnosti a osobitného prístupu pre administrátora.

Najväčšia výhoda virtuálneho honeynetu postaveného na virtualizácii na úrovni operačného systému oproti klasickému je to, že, keďže virtuálny honeypot beží na hosťovskom operačnom systéme dokážeme monitorovať stav honeypotov bez prístupu cez sieť.

2.4.2 Typy virtuálnych honeynetov

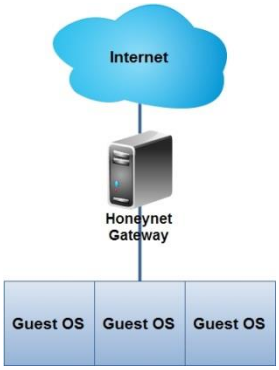
Virtuálne honeynety rozdeľujeme na samostatné a hybridné [14]. **Samostatný virtuálny honeynet** je kompletný honeynet virtuálne obsiahnutý na jednom samostatnom fyzickom systéme. Pod **kompletným honeynetom** rozumieme honeypoty, ktoré tvoria honeynet a bránu firewall na zachytávanie a kontrolu dát.

Medzi výhody samostatného honeynetu patrí: prenosnosť, modularita systému a šetrenie nákladov. Nevýhodami sú tzv. single point of failure – ak sa pokazí niektorá časť hardvéru, je mimo prevádzky celý systém. K nasadeniu tohto typu virtuálneho honeynetu je potrebný výkonný počítač s dostatočnou pamäťou a výkonom, keďže na jednom počítači emulujeme viacero operačných systémov. Bezpečnosť tohto typu virtuálneho honeynetu závisí od virtualizačného softvéru. Pri určitých typoch (napr. pri použití virtualizácie na úrovni operačného systému) máme k dispozícii limitovaný výber operačných systémov. Samostatný virtuálny honeynet je zobrazený na obrázku 1, na ktorom vidíme, že v rámci jedného hostiteľského operačného systému pripojeného do Internetu sa nachádza viacero virtuálnych strojov, predstavujúcich honeypoty v systéme.



Obr. 1 *Samostatný virtuálny honeynet*

Ďalším typom je **hybridný virtuálny honeynet**, ktorý je kombináciou klasického honeynetu a virtualizačného softvéru. V tomto prípade je brána firewallu oddelená od honeypotov, ktoré bežia virtuálne na jednom počítači, tak ako je znázornené na obrázku 2. Oddelenie systémov na zachytávanie a kontrolu dát zmenšuje riziko kompromitácie.



Obr. 2 *Hybridný virtuálny honeynet*

Výhodami hybridného typu virtuálneho honeynetu je väčšia bezpečnosť ako pri samostatnom type a flexibilita. Keďže systémy na zachytávanie dát sú oddelené útočník sa k nim nedostane len s veľmi malou pravdepodobnosťou. Možnosti použitého softvéru na zachytávanie a kontrolu dát sú väčšie, keďže je oddelený systém môže mať diametrálne odlišnú hardvérovú konfiguráciu.

2.4.3 Sieťové rozhrania virtuálneho honeynetu

V práci sa venujeme virtuálnemu honeynetu, ktorý používa virtualizáciu na úrovni operačného systému. Z tohto dôvodu sa v tejto podkapitole venujeme dvom prístupom k sieťovej virtualizácii, a to:

- virtuálna sieť (virtual network device)
- virtuálne sieťové rozhranie (virtual ethernet device)

Virtuálna sieť (virtual network device - venet) je primárne sieťové zariadenie pre virtuálny stroj, ktoré vyzerá ako priame spojenie medzi hosťujúcim a hostiteľským systémom. Vzniká automaticky pri štarte virtuálneho stroja. Pakety prepína na základe hlavičky IP protokolu tak, že ich presúva z virtuálneho stroja so zdrojovou IP adresou do virtuálneho stroja s cieľovou IP adresou. Zariadenie venet nepodporuje ARP protokol (protokol na získanie MAC adresy zariadenia na základe jeho IP adresy), teda vo vnútri virtuálneho stroja nefunguje napr. DHCP server a nie je možné posielať broadcastové správy. Ďalej nepodporuje premostenie a IPv6 protokol.

Alternatívnym zariadením je virtuálne sieťové rozhranie (**virtual ethernet device** - veth) Používa sa vnútri virtuálneho stroja. Tomuto zariadeniu môže byť pridelená MAC adresa a ako také teda môže byť použité vo viacerých konfiguráciách. Po premostení so sieťovým rozhraním fyzického stroja sa virtuálny stroj môže správať ako nezávislý hostiteľský systém.

Zariadenie veth povoľuje broadcastové správy vnútri virtuálneho stroja, čo znamená, že pri jeho použití bude vnútri fungovať DHCP server. Keďže je premostený priamo na hostiteľské sieťové rozhranie musíme k nemu z hľadiska bezpečnosti pristupovať ako k takémuto zariadeniu. Zariadenia veth môžeme premostiť navzájom alebo s inými zariadeniami. Na rozdiel od zariadenia venet podporuje IPv6 protokol. IPv6 adresa sa vygeneruje automaticky z MAC adresy. Z hľadiska výkonu a rýchlosti je lepšie

zariadenie venet, ktoré je tiež bezpečnejšie ako veth, jeho nevýhodou však je, že nepodporuje premostenie.

2.5 Proc súborový systém

V Linuxových operačných systémoch sú všetky informácie o stave hardvérových prvkov dostupných cez **Proc súborový systém**. Nejedná sa však o skutočný súborový systém.- Súbory, ktoré sa v tomto systémovom adresári nachádzajú, nie sú nikde fyzicky uložené na disku, ale dostupné iba na vyžiadanie. Ide teda o virtuálny súborový systém, čím je zachovaná filozofia linuxových operačných systémov (všetko je súbor). Informácie sú dostupné každému užívateľovi nevynímajúc aj softvér ako obyčajný textový súbor bez potreby použitia špeciálnej knižnice.

Na najvyššej úrovni obsahuje Proc súborový systém adresáre pomenované podľa aktuálne bežiacich procesov (názov je zhodný s identifikátorom procesu) a súbory poskytujúce informácie o stave operačného systému. Najzaujímavejšie z hľadiska pravidelnej kontroly administrátorom operačného systému sú zobrazené v tabuľke nižšie.

Virtuálny súbor alebo adresár	Popis
PID (adresár)	poskytuje informácie o procese s identifikátorom procesu *PID*. vlastník a skupina priečinka je zhodný s vlastníkom a skupinou procesu. medzi užitočné súbory v tomto priečinku patria: *cmdline* príkaz procesu *cwd* symbolická linka k priečinku, v ktorom je proces vykonávaný *environ* premenné prostredia

Virtuálny súbor alebo adresár	Popis
	exe symbolická linka ku príkazom *fd/n* desktriptory súborov *maps* mapa pamäte a súborov knižníc *root* symbolická linka do koreňového priečinka *status* využitie pamäte a stav procesu
bus (adresár)	informácie o systémových zberniciach
cmdline	zoznam parametrov použitých na štart systému
cpuinfo	informácie o procesore
crypto	informácie o nainštalovaných kryptografických šifrách
devices	zoznam mien a čísel aktuálne nakonfigurovaných zariadení
driver (adresár)	informácie o ovládačoch používaných jadrom os
fs (adresár)	informácie o pripojených súborových systémoch
meminfo	informácie o použití pamäte
modules	aktuálne natiahnuté moduly v jadre os

Virtuálny súbor alebo adresár	Popis
net	informácie o sieťových protokoloch a štatistike siete
stat	zaznamenáva informácie o systéme od kedy bol spustený vrátane: *cpu* celkový čas procesora v užívateľskom móde, systémovom móde, nečinnom, čakaním na vstupno/výstupné operácie atď.
sys	poskytuje informácie o systéme a umožňuje modifikovať vlastnosti jadra os

Tab. 1 Užitočné súbory a priečinky v adresári */proc*

3 Vizualizácia v oblasti sieťovej bezpečnosti

Jedným z najužitočnejších nástrojov sieťovej bezpečnosti sú vizualizácie. Tak ako sa grafy používajú na zobrazenie rôznych ekonomických údajov, či výsledkov experimentov, rovnako dobre slúžia na lepšie pochopenie množstva dát vzťahujúcich sa k aktivite v sieti. Ľudia najlepšie vnímajú zrakové informácie, keďže zrak je vyvinutejší ako ostatné zmysly. Ako hovorí slovné spojenie „Lepšie vidieť ako stokrát počuť," je zraková informácia v mozgu spracovávaná oveľa rýchlejšie než si to stihneme uvedomiť.

Cieľom vizualizácie v sieťovej bezpečnosti je nie vytváranie nových informácií, ale prezentovanie dostupných údajov ľahko pochopiteľným spôsobom. Použitím nesprávnych zobrazovacích techník dokážeme pozorovateľa nielen zmiasť, ale vyvodiť v ňom nesprávne pochopenie toho, čo je zobrazované. Na vytvorenie účelného nástroja musíme najprv pochopiť základné princípy zobrazovania údajov.

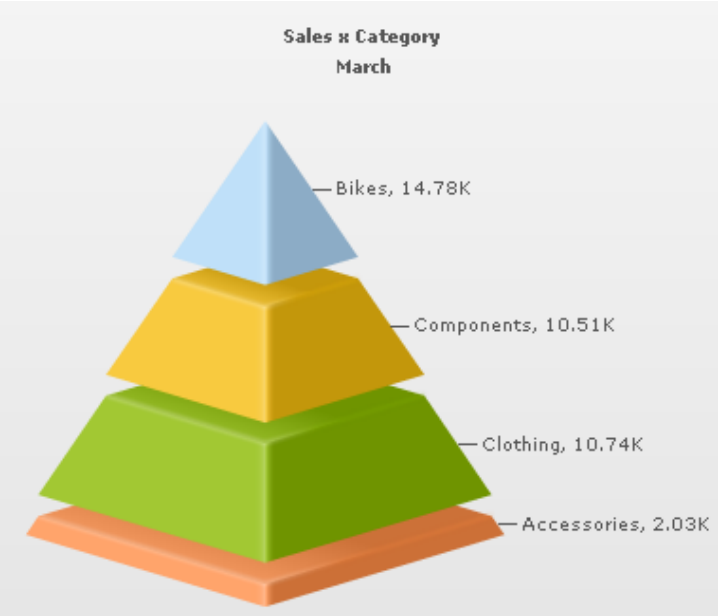
3.1 Princípy zobrazovania údajov

Najprv sa pozrime na to, čo sa stane, keď na zobrazenie dostupných dát použijeme nesprávnu techniku. Zoberme si príklad vizualizácie predaja tovaru v športovom obchode za mesiac marec:

Bicykle	Súčiastky	Oblečenie	Doplnky
14 780	10 510	10 740	2 030

Tab. 2 Tržby v obchode podľa druhu tovaru

Z tabuľky 2 vidíme, že tovar je rozdelený na 4 skupiny a ku každej máme dostupnú informáciu o výške tržby. Na jednoduché porovnanie, ktorého tovaru sa v obchode predá viac stačí aj jednoduchá tabuľka. Pozrime sa však na zobrazenie týchto tržieb na obrázku 3.



Obr. 3 *Vizualizácia tržieb z tab. 2*

Autor použil pyramídové zobrazenie, s tým, že bicykle, ako najpredávanejší artikel sú na vrchole. Prvý dojem z obrázka tomu však nezodpovedá. Proporcionálne najväčšiu časť tvorí oblečenie. Ak sa pozrieme na obrázok z diaľky, respektíve, ak vynecháme v obrázku informáciu o výške tržby, zdá sa, že najväčšiu tržbu malo oblečenie. Čo však, ako vidíme z tabuľky, nie je pravda.

Vizualizácia sa v iných vedných disciplínach ako chémia a geografia požíva na reprezentáciu molekúl a geografických štruktúr, ktoré sú založené na meraniach. Vizualizácia teda pracuje s konkrétnymi hodnotami dát. Dáta získané v rámci sieťovej bezpečnosti sú abstraktné a nekorešpondujú fyzikálnym vlastnostiam. Avšak obyčajné pretransformovanie textových logovacích súborov do obrázka nestačí. Vytvorená vizualizácia musí viac zviditeľniť dôležité informácie pred tými menej dôležitými.

Správne vytvorená vizualizácia môže podporiť spracovanie informácií už na úrovni prijímania podnetov v mozgu. Pri vizualizácií informácií je dôležitá rýchlosť spracovania podnetov, keďže nie všetky stimuly človek zachytáva rovnako rýchlo. Atribúty obrazu, ktoré mozog rozpozná ešte pred tým než na nich vedome upriami pozornosť sa členia do štyroch skupín [3]:

- **tvar** – dĺžka, hrúbka, súbežnosť čiar, geometrických útvarov,

- **farba** – odtieň, jas
- **pohyb** – smer pohybu, blikanie
- **poloha** – pozícia v priestore, hĺbka

Je potrebné dať pozor na to, aby sa použila kombinácia malého počtu týchto vlastností na zdôraznenie najdôležitejších informácií. Opak spôsobí šum a naruší prirodzenú hierarchiu. Dôležité informácie vizualizácie teda vieme zvýrazniť použitím týchto vlastností a docieliť tým to, že užívateľ si ich všime rýchlo a bez námahy.

3.2 Vnímanie kompozície

Spracovanie vizuálnych podnetov tiež podlieha istým zákonitostiam, na ktoré je potrebné brať ohľad pri vytváraní vizualizácie. Poznatky o vnímaní vizuálnych podnetov priniesla **Gestalt psychológia**, ktorá vychádza z Aristotelovho tvrdenia „celok je viac ako suma častí“. Ide o vzťah medzi kompozíciou a prvkami, ktoré ju tvoria. Ich výpovedná hodnota sa môže zmeniť v závislosti od interakcií, ktoré vzniknú po začlenení do kompozície.

Človek pracuje s obrazovými podnetmi ako s celkom, najskôr vníma kompozíciu a prvky, ktoré ju vytvárajú prichádzajú na rad až neskôr, po tom, čo na nich vedome zacieli pozornosť. Vnímanie celku je rýchle, podlieha okamžitému spracovaniu naopak vnímanie detailov vyžaduje viac času, pretože pozorovateľ informácie aktívne vyhľadáva. Základné informácie by mali byť dostupné od začiatku, mali by byť súčasťou celkovej kompozície a doplňujúce a vysvetľujúce informácie dostupné po upriamení pozorovateľovej pozornosti, prípadne až na vyžiadanie. Takáto hierarchizácia informácií poskytuje pozorovateľovi rozhodnúť sa do akej hĺbky sa chce informáciami zaoberať.

Kritériá na základe ktorých mozog zoskupuje vnímané prvky boli experimentálne skúmané na začiatku dvadsiateho storočia, dnes sú známe ako **Gestalt princípy:** [17]

- **princíp podobnosti:** objekty s podobnými vlastnosťami vníma pozorovateľ ako skupinu objektov s podobným významom
- **princíp blízkosti:** objekty umiestnené blízko pri sebe vníma pozorovateľ akoby patrili k sebe a pripisuje im podobný význam
- **princíp uzavretosti:** mozog organizuje objekty do celkov s tým, že pridáva chýbajúce časti tak, aby vytvárali celok

- **princíp kontinuity:** mozog vníma ľahšie spojité, hladké prechody. Objekty zoskupuje tak, aby vytvárali spojitý vzor.
- **princíp symetrie:** prvky, ktoré sú navzájom symetrické vníma ľudské oko akoby patrili k sebe nezávisle od toho aké veľké je medzi nimi prázdne miesto.

Gestalt princípy využijeme pri vytvorení vizuálnej hierarchie a vedení ľudského oka cez obsah. Najjednoduchší spôsob ako zdôrazniť vzťahy medzi jednotlivými prvkami je začleniť ich blízko seba a oddeliť prázdnym priestorom od iných prvkov. Zoskupenie možno robiť aj s použitím iných atribútov, ktoré podliehajú okamžitému spracovaniu, napríklad farbou alebo tvarom. Podobnosť zdôrazňuje vzťahy medzi prvkami a ovplyvňuje poradie čítania informácií. Kontrastný vzhľad pomôže oddeliť navzájom rôzne významové celky. Na zjednodušenie vizualizácie je možné použiť princíp uzavretosti, keďže mozog si niektoré prvky dokáže doplniť sám. Takéto prvky je možné odstrániť a zredukovať tak nadbytočné informácie a hustotu vizuálneho šumu.

3.3 Organizácia informácií

Pri organizovaní a členení vizualizovaných dát je dobré vychádzať z konceptu pomenovaného z anglickej skratky **LATCH**. Táto skratka je súhrnom anglických názvov označujúcich päť princípov, na základe ktorých možno organizovať údaje [17]:

- **miesto (location)** – v prípade, že chceme zdôrazniť fyzické prepojenie medzi údajmi je užitočné geografické členenie,
- **abeceda** – lexikografické poradie v prípade, že pracujeme s veľkým objemom dát prevažne textového charakteru,
- **čas (time)** – chronologické zoradenie údajov v prípade, že chceme zdôrazniť ich časovú súslednosť,
- **kategória (category)** – zoskupovanie informácií na zdôraznenie spoločných vlastností a
- **hierarchia** – rozdelenie podľa hodnosti alebo dôležitosti významu, prípadne poradia.

Vizualizácia dát, ktorá okrem základných údajov obsahuje aj doplňujúce informácie, však údaje nezobrazuje lineárne. Informácie sú primárne rozmiestnené v dvojrozmernom priestore, niekedy ďalšiu dimenziu poskytuje interaktivita s

používateľom. Vďaka nej je možné údaje odčleniť do niekoľkých informačných rovín. Vďaka takémuto vrstveniu tak nie je užívateľ ihneď zahltený množstvom informácií.

Prvým krokom pri tvorbe grafického zobrazenia dát je rozloženie informácií do **dvojrozmerného priestoru**. Členenie je dané charakteristikou informácií, je však vhodné vychádzať z niekoľkých základných pravidiel. Najčastejšie využívaným je systém mriežky. Základnou myšlienkou je zvislé a vodorovné rozdelenie plochy na menšie časti. Ďalším krokom je organizovanie informácií do **vizuálnej hierarchie**, ktorá pomôže vytvoriť v nelineárnom obsahu logickú následnosť. Jej cieľom je nasmerovať človeka pri skúmaní informácií a naviesť ho od najdôležitejších údajov k tým menej dôležitým. V tejto časti je možno vychádzať z Gestalt princípov. Vytvorenie efektívnej hierarchie je kľúčové, pretože vedie užívateľa procesom získavania informácií a pomáha mu zorientovať sa v obsahu.

4 Vizualizácia sieťových spojení

Cieľom tejto kapitoly je priblíženie pojmov sieťové spojenie, priblíženie rozdielu medzi dvoma najpoužívanejšími protokolmi na prenos informácií medzi jednotlivými stanicami na internete a popísanie riešenia vizualizácie takýchto sieťových spojení. V krátkosti sú popísané podobné riešenia vizualizácie spojení virtuálnych honeynetov. V závere sa venujeme implementačným detailom webovej aplikácie na vizualizáciu sieťových spojení.

4.1 Sieťové spojenia

Komunikácia na Internete prebieha cez sieťové rozhranie nazývané **soket**. Prostredníctvom soketov prijímajú a posielajú procesy správy z transportnej vrstvy, ktorá sa stará o zabezpečenie spojenia medzi dvoma procesmi na rôznych počítačoch. Transportná vrstva prijme od procesu správu a informáciu o tom, kam má túto správu poslať. Obmedzená veľkosť paketu častokrát neumožňuje odoslanie celej správy naraz, transportná vrstva teda túto správu rozdelí na menšie časti, nazývané **segmenty**. Segment predá sieťovej vrstve, ktorá sa postará o jeho odoslanie do cieľovej stanice. Na internete sa štandardne používajú dva transportné protokoly: **UDP** (User Datagram Protocol) a **TCP** (Transmission control protocol). Každý z nich má iné špecifické vlastnosti a používa sa pri rôznych druhoch dátového prenosu alebo dátovej komunikácie.

Základnou vlastnosťou **User Datagram Protocol-u** (UDP) je neudržiavanie stáleho spojenia počas komunikácie. Cieľový soket, kam má byť segment odoslaný je v prípade protokolu UDP jednoznačne určený dvojicou cieľová IP adresa a cieľový port. Protokol UDP je charakterizovaný týmito vlastnosťami:

- **nezabezpečuje spoľahlivosť prenosu dát** – správy odovzdáva v takom poradí, v akom prichádzajú, segmenty sa môžu počas prenosu stratiť, proces využívajúci tento protokol musí byť schopný sa z možnej straty zotaviť
- **nevytvára spojenie** – protokol neinicializuje ani nezatvára žiadne spojenia, dáta sú odosielané bez čakania na potvrdenie prijatia segmentu
- **nekontroluje tok dát ani zahltenie siete** – umožňuje odosielať tak rýchlo ako to sieťové pripojenie umožňuje

Najpoužívanejším protokolom je **Transmission Control Protocol** (TCP). Od UDP protokolu sa líši tým, že vyžaduje potvrdzovaný prenos segmentov. Návrh protokolu je podriadený tomuto princípu, keďže nižšia, sieťová, vrstva potvrdzovaný prenos nepodporuje. V prípade TCP protokolu je spojenie jednoznačne určené štvoricou zdrojová IP adresa a zdrojový port a cieľová IP adresa a cieľový port. Protokol TCP má nasledujúce vlastnosti:

- **zabezpečuje spoľahlivý prenos dát** – správy sú doručené bez straty poradí, v akom boli odosielané,
- **počas prenosu dát udržiava spojenie** – je teda potrebné uchovávať informáciu o tom, v akej fáze sa aktuálne spojenie nachádza,
- **poskytuje možnosť kontroly toku dát a zahltenia siete** – v hlavičke segmentu je možné dynamicky meniť veľkosť dát, ktoré budú odoslané v nasledujúcom segmente, v prípade, že prijímajúca stanica nestíha dáta spracovávať.

Na získanie informácií o aktuálnom stave soketov na sieťovom rozhraní slúži v operačnom systéme príkaz **netstat**. Ten poskytuje informácie o tom, aké služby počúvajú a na akých portoch, aký proces je zviazaný, s ktorým portom a aké spojenia sú vytvorené na ktorých soketoch. Okrem toho je v prípade TCP protokolu, ktorého základným rysom je udržiavanie spojenia, dostupná informácia o tom, v akej fáze sa toto spojenie práve nachádza. Poznáme nasledujúce fázy TCP spojenia:

1. **LISTEN** – server čaká na pripojenie
2. nadviazanie spojenia:
    a. **SYN_SEND** – aktívne otvorenie spojenia klientom
    b. **SYN_RECEIVED** – server prijal požiadavku na pripojenie
3. prenos dát
    a. **ESTABLISHED** – spojenie je otvorené a prebieha dátová komunikácia
4. ukončenie spojenia
    a. **FIN_WAIT_1** – klient aktívne požiada o ukončenie spojenia
    b. **TIMED_WAIT** – klient sa dostane do tohto stavu po stave FIN_WAIT_1

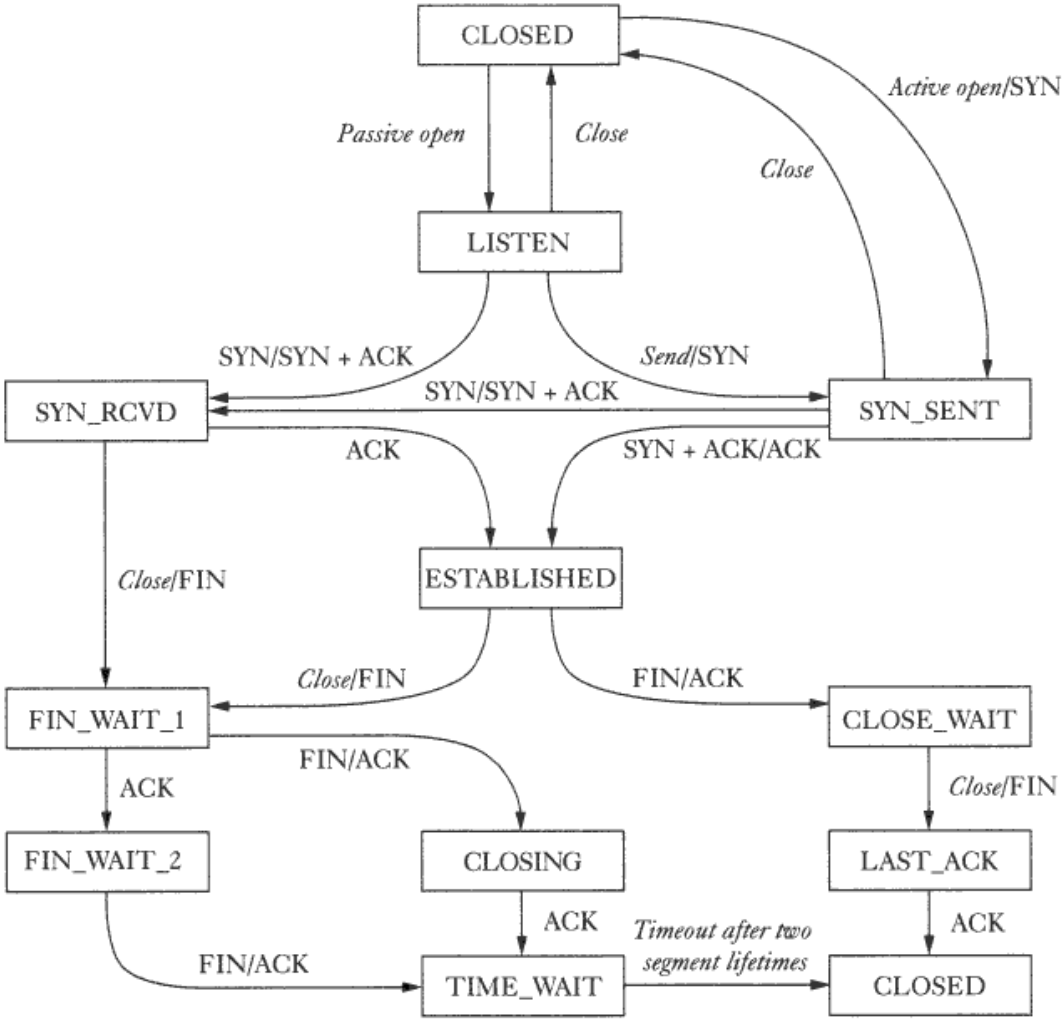
c. **CLOSE_WAIT** – server prijal požiadavku na uzatvorenie spojenia

d. **FIN_WAIT_2** – klient prijal potvrdenie o uzatvorení spojenia od klienta

e. **LAST_ACK** – server sa dostane do tohto stavu po stave CLOSE_WAIT

f. **CLOSED** – server prijal potvrdenie o uzatvorení od klienta a spojenie je ukončené

Jednotlivé prechody medzi týmito stavmi TCP spojenia sú zobrazené na obrázku 4 nižšie.



Obr. 4 *Stavový diagram TCP spojenia*

4.2 Podobné riešenia

V tejto časti stručne popíšeme niektoré, už existujúce riešenie zobrazovania údajov z honeypotov a honeynetov.

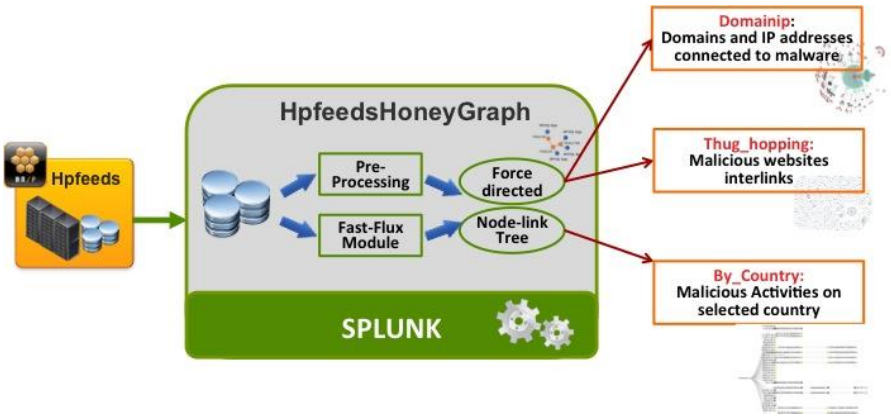
4.2.1 HoneyMap

HoneyMap [11] je projekt organizácie The Honeynet Project, ktorý zobrazuje útoky zachytené honeypotmi v reálnom čase. Sú zobrazené dáta z honeypotov pripojených k tejto vizualizácii. HoneyMap potom vyhľadá geografickú polohu korešpondujúcej IP adresy. V reálnom čase teda môžeme sledovať, z akých miest na svete sú aktuálne vykonávané útoky.

Obr. 5 *HoneyMap* – obrázok prevzaný z http://www.honeynet.org/node/960

4.2.2 HpfeedsHoneyGraph

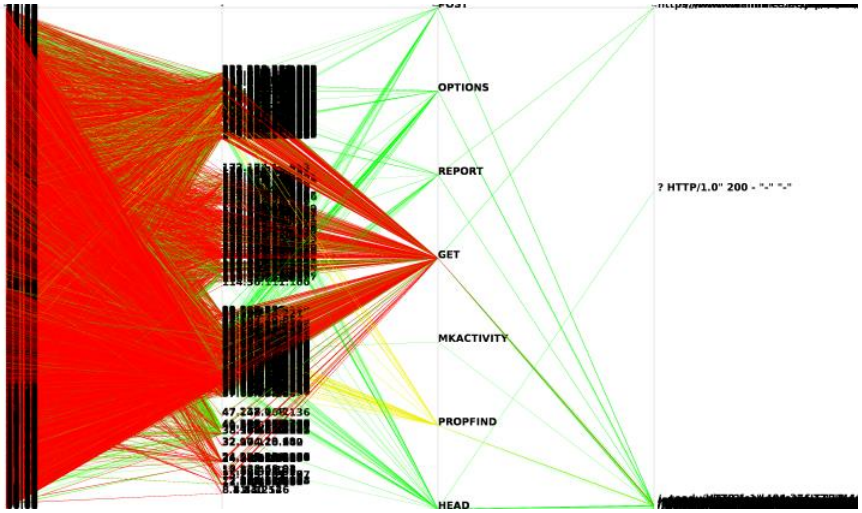
Projekt HpfeedsHoneygraph [18] taktiež spadá pod organizáciu The Honeynet Project. Okrem získavania dát z honeypotov organizácie ich však aj indexuje. Útoky zobrazuje ako stromový graf. Vyhľadáva krajinu, v ktorej sa IP adresa nachádza DNS informácie, ak sú dostupné. Je dostupných viacero možností použitia, vyhľadanie útokov na základe krajiny, zobrazenie, ktoré IP adresy so spojené s ktorými útokmi, apod.



Obr. 6 *HpfeedHoneyGraph* – obrázok prevzaný z http://www.honeynet.org/node/957

4.2.3 PicViz

PicViz [19] pomáha pochopiť, čo sa deje v sieti zobrazovaním udalostí vo viacerých rozmeroch s pomocou paralelných priamok. Dokáže spracovať milióny udalostí a pomáha odhaliť anomálie v sieti a na základe toho priamo spracovať nové pokyny pre detekčné systémy. Na obrázku 7 je znázornená vizualizácia prístupového logovacieho súboru z istého webového servera. Na prvej priamke (zľava) je zaznačený čas, pričom 00:00 je dole a 23:59 hore. Druhá priamka predstavuje rozsah IP adries, ktoré na server pristupovali – od IP adresy 0.0.0.0 dole po adresu 255.255.255.255 hore. Tretia priamka predstavuje typ HTTP požiadavky a štvrtá priamo HTTP požiadavku. Priečne čiary medzi týmito požiadavkami sú jednotlivé záznamy v logovacom súbore. Často sa vyskytujúce záznamy sú vykreslené červenou farbou.



Obr. 7 *PicViz* – obrázok prevzaný z http://secviz.org/content/picviz-graphing-apache-logs

4.3 Implementačné a testovacie prostredie

4.3.1 Testovacie prostredie

V rámci testovania je **klientska časť aplikácie** nasadená na testovacom serveri umiestnenom v rámci počítačovej siete ŠDaJ UPJŠ v Košiciach. Použité technológie vyžadujú moderný webový prehliadač s podporou jazykov **HTML5** a **CSS3**. Samotná vizualizácia vyžaduje podporu SVG a povolený JavaScript v prehliadači. Ako také je testovanie vykonávané v prehliadači Google Chrome verzie 34. **Serverová časť** je tvorená PHP skriptami, ktoré komunikujú s MySQL databázou obsahujúcu testovacie dáta.

4.3.2 Použité technológie

Na vizualizáciu je použitá knižnica jazyka JavaScript **D3.js** podporujúca deklaratívny prístup k manipulácií s elementami **dokumentovo objektového modelu** (z angl. Document Object Model – DOM). D3 umožňuje k tomuto modelu pripojiť ľubovoľné dáta a ďalej s nimi pracovať. D3 je rýchla a flexibilná knižnica podporujúca veľké dátové množiny a dynamické správanie pre interakciu a animáciu, čo vizualizácií dodáva moderný vzhľad.

Základné rozhranie je napísané použitím moderných štandardov jazykov HTML a CSS na docielenie moderného vzhľadu. Animácie a ostatné z najnovších vlastností týchto štandardov sú však používané minimálne, vizualizácia je dostupná aj za použitia starších verzií webových prehliadačov, za predpokladu, že podporujú SVG formát a JavaScript. Práve tieto sú použité na samotné vykresľovanie.

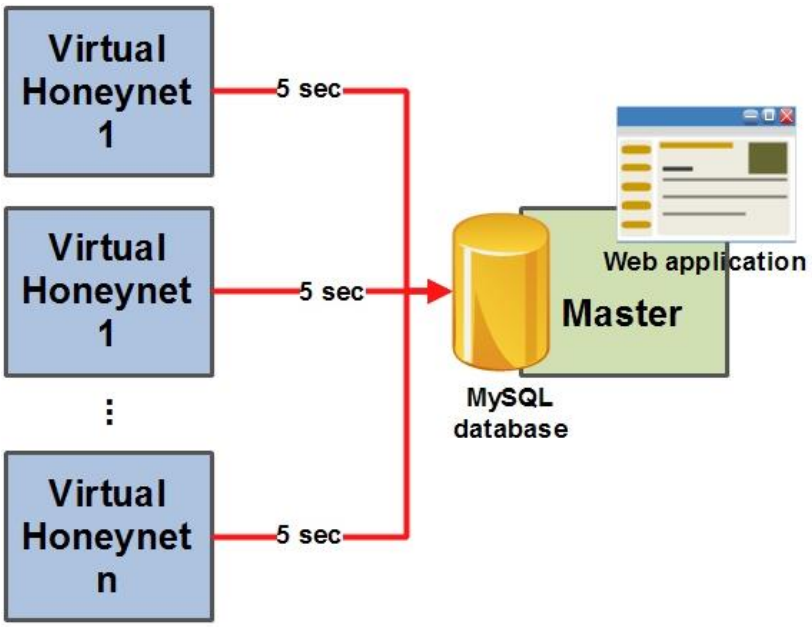
4.4 Návrh systému

Vizualizáciu implementujeme ako **klient-server webovú aplikáciu**. Samotná vizualizácia prebieha na klientskej strane. V rámci administrácie honeynetu uvažujme nasledujúci spôsob použitia aplikácie:

- zobrazenie aktuálne prebiehajúcej sieťovej aktivity
- zobrazenie informácií o vyťažení prostriedkov v honeynete

- zobrazenie údajov z viacerých virtuálnych honeynetov naraz

Serverovú časť systému tvorí **logovací server** virtuálneho honeynetu, ktorého sieťovú aktivitu skúmame. Systém sa skladá z distribuovaného virtuálneho honeynetu a hlavného servera, na obrázku označeného ako master. Virtuálne honeynety posielajú v pravidelných intervaloch všetky dostupné logy do databázy nachádzajúcej sa na hlavnom serveri. Tento interval musí byť dostatočne malý na to, aby bola zachovaná požiadavka vizualizácie v reálnom čase, ale tiež primerane veľký tak, aby aplikácia stíhala nové údaje spracovať a prekresliť vizualizáciu a aby užívateľ stihol prípadnú výraznú zmenu zaregistrovať a spracovať.



Obr. 8 *Schéma prepojenia serverovej časti aplikácie s virtuálnym honeynetom*

4.4.1 Vizualizácia v rámci návrhu

V kapitole 3 sme sa venovali všeobecným poznatkom vizualizácie informácií. V tejto časti popíšeme ich využitie pri návrhu vizualizácie sieťových spojení virtuálneho honeynetu.

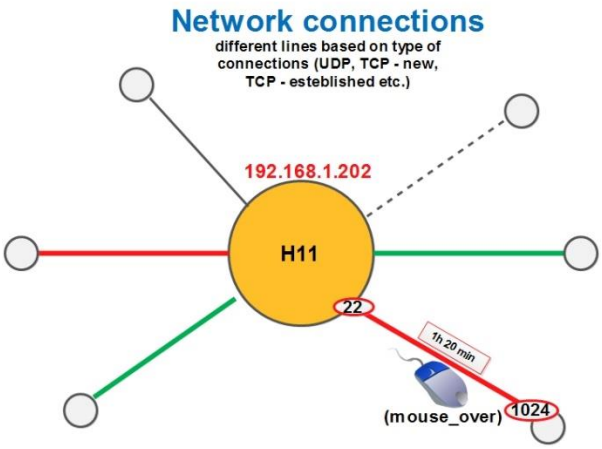
Centrálnou časťou je zobrazenie sieťovej aktivity vo virtuálnom honeynete, a teda sieťových spojení medzi jednotlivými honeypotmi, ktoré sú v honeynete nasadené a IP adresami, ktoré sa na tieto honeypoty pripájali. Prirodzene sa na toto zobrazenie ponúka grafová štruktúra, kde vrcholy grafu predstavujú koncové body sieťovej komunikácie a spojenia medzi týmito bodmi vytvárajú hrany grafu.

Na rýchle a efektívne odlíšenie vrcholov predstavujúcich honeypoty a útočníkov zobrazíme vrchol predstavujúci honeypot výrazne väčší. Ďalej bude odlíšený aj farebne, farebné odlíšenie však nestačí, keďže toto bude závislé od iného typu dát a v prípade ich nedostupnosti chceme toto základné rozdelenie vrcholov zachovať.

Hlavnou kompozíciou obrazu je zobrazenie informácií pre každý virtuálny honeynet. Tento obraz tiež tvorí najvrchnejšiu časť hierarchie informácií. V rámci vizualizácie spojení pre jednotlivé virtuálne honeynety sa potom postupuje ďalej na do hĺbky, v druhej vrstve zobrazením spojení a v tretej vrstve, textových informácií.

Informácie dostupné o spojení chceme, aby boli viditeľné a rozoznateľné na prvý pohľad. Pripomeňme si najprv, ktoré informácie sú pre nás relevantné:

- **zdrojová IP adresa**, keďže v prípade honeypotu sa jedná o útočníka,
- **cieľový port**, podľa čoho sa dá určiť služba, na ktorú bolo útočené,
- **protokol,** ktorý bol použitý na komunikáciu a
- v prípade, že sa jedná o protokol TCP chceme zobraziť aj **stav**, v ktorom sa spojenie aktuálne nachádza.

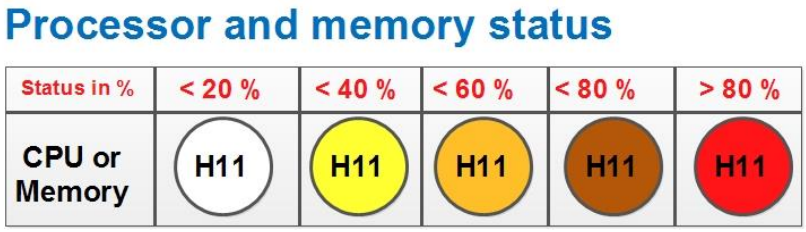


Obr. 9 *Návrh zobrazenia sieťových spojení na konkrétny virtuálny honeypot*

Všetky tieto informácie sú textového charakteru, ktoré stačí na obrazovku jednoducho vypísať ku danej hrane predstavujúce dané spojenie. Informácie o stave spojenia však vieme premietnuť do hrúbky a farby čiary nasledovne (obr. 9):

- **vytvárajúce spojenia** – zakreslené tenkou plnou čiarou sivej farby v prvých fázach vytvorenia spojenia a zelenej farby vo fáze kedy je spojenie pripravené a čaká na dátovú tok
- **spojenia s prebiehajúcim dátovým tokom** – zakreslené najhrubšou plnou čiarou červenej farby
- **zatvárajúce spojenia** – zakreslené tenkou prerušovanou čiarou sivej farby na indikáciu, že spojenie zaniká
- **spojenia, ktorých stav nie je dostupný** sú zakreslené čiernou čiarou nie veľmi tenkou tak, aby boli dostatočne viditeľné

Ďalším typom údajov, ktoré môžeme zobraziť sú okrem aktívnych spojení **informácie o vyťažení systému**. Sem patrí **vyťaženie pamäte a procesora** virtuálnych honeypotov a fyzického stroja, celkový **počet sieťových spojení** a **aktívne prihlásení užívatelia**. Znova sú to textové informácie, kde presné údaje sprístupníme po nabehnutí myšou na vrchol predstavujúci honeypot, ktorý užívateľa zaujíma. Percentuálne vyťaženie pamäte a procesora honeypotu vieme tiež zobraziť odstupňovaním odtieňov farby, ktorou vykreslíme vrchol, od najjemnejšej pri malom vyťažení do 20% a postupne zvyšovať sýtosť až po úplne sýty pri vysokom vyťažení nad 80% ako je zobrazené na obr. 10.



Obr. 10 *Návrh **zvyšovania sýtosti farby uzla***

4.4.2 Výhody a nevýhody systému

Od podobných existujúcich riešení, ktoré sme popísali v časti 4.1 sa naša implementácia vizualizácie líši v niekoľkých veciach. Berie do úvahy jednu z hlavných výhod virtuálneho honeynetu a tou je škálovateľnosť systému. Virtuálne honeynety je možné pridávať a odoberať na čo naša aplikácia reaguje automaticky, bez nutnosti meniť akékoľvek nastavenia. Sú zobrazované dáta z výskumného honeynetu, ktorý sa nachádza na ŠDaJ UPJŠ v Košiciach.

Vyhľadávanie geografickej polohy jednotlivých IP adries a zobrazovanie ich polohy nie je v našom prípade prioritou. Medzi hlavné použitie našej vizualizácie je okamžité zobrazenie, na ktorý virtuálny honeypot sa útočí. V prípade, že by pozadím vizualizácie bola mapa sveta, tak za použitia takej mierky, aby bola celá mapa viditeľná na obrazovke by náš systém bol zobrazený iba jediným bodom v strede Európy, ktorý neposkytuje dostatočne veľkú oblasť na viditeľné odlíšenie virtuálnych honeynetov.

Oproti riešeniu PicViz je **nevýhodou nášho systému**, že neposkytuje priame prepojenie so detekčnými systémami a bránou firewall. To poskytuje možnosť priamo doplniť týmto systémom pravidlá, ktorými sa majú riadiť.

4.5 Implementácia systému

Nasledujúca časť práce popisuje implementačné detaily vizualizácie aktívnych spojení virtuálneho honeynetu a podrobnejšie sa venuje jednotlivým súčastiam webovej aplikácie.

4.5.1 Implementácia parsera

Informácie, ktoré virtuálne honeynety posielajú na hlavný server v pravidelných intervaloch sú do databázy zapisované vo formáte, ktorý je zhodný s výpisom jednotlivých príkazov operačného systému, ktoré tieto informácie získavajú. Do databázy sú zapisované nasledujúce informácie: identifikátor honeynetu, identifikátor honeypotu, typ dát, o ktoré sa jedná, dáta a časová pečiatka. V aplikácií rozlišujeme nasledujúce typy dát a k nim tieto atribúty:

- **informácie o pamäti** virtuálneho honeypotu: veľkosť použitej, voľnej, zdieľanej pamäte, cache pamäte, celková veľkosť swapovacej pamäte a veľkosť zdieľanej swapovacej pamäte
- **vyťaženie disku**: celkové a použité miesto na disku
- **prihlásení užívatelia** a miesto odkiaľ sú prihlásení
- **počet sieťových spojení**: počet spojení používajúcich TCP a UDP protokol
- **vyťaženie procesora**: počet procesov, vyťaženie procesora

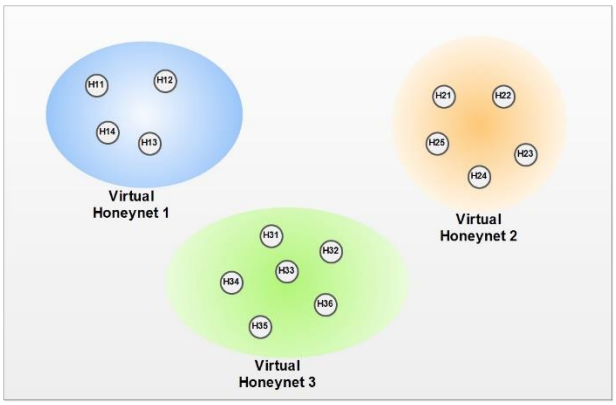
- **aktívne sieťové spojenia**: protokol, zdrojová a cieľová IP adresa a port, stav spojenia, identifikátor procesu, s ktorým je spojenie asociované
- **veľkosť prenesených dát**

Všetky tieto informácie sú z databázy získavané v **nespracovanej textovej forme** (raw formát). Pri získaní požiadavky od klienta vytiahne serverová časť aplikácie príslušné informácie z databázy. Všetky údaje podliehajú predspracovaniu a klientovi sú posielané ako **pole objektov JSON**.

Na komunikáciu s databázou a implementáciu parsera je použitý skriptovací jazyk **PHP**. Keďže sa výpis výstupu príkazu netstat, ktorý získava informácie o aktívnych spojeniach líši od príkazu na získanie informácií o stave honeypotu, má parser dva módy použitia. Jednotlivé informácie o stave honeypotu sú medzerou oddelené čísla. Parser ich teda za sebou v poradí prečíta a do JSON objektu, do ktorého sú zapisované, pridá ich príslušný atribút. V prípade vyžiadania objektu s aktívnymi sieťovými spojeniami získa parser surovú textovú informáciu, ktorú musí najprv rozdeliť po riadkoch a potom už v riadku číta jednotlivé medzerou oddelené atribúty.

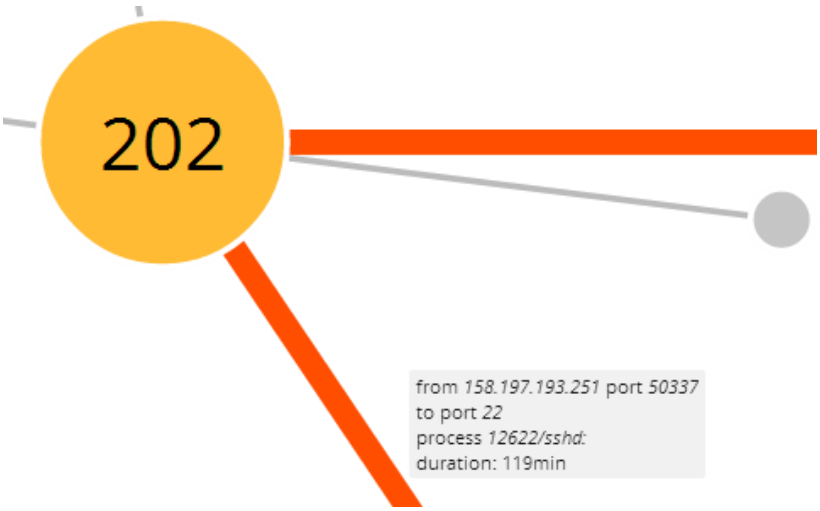
4.5.2 Implementácia zobrazovacej časti

Samotná vizualizácia aktívnych spojení prebieha priamo vo webovom prehliadači prostredníctvom skriptu v jazyku **JavaScript**. Pri prvom načítaní webovej stránky sa najprv získajú informácie o počte virtuálnych honeynetov a honeypotov v nich, podľa ktorého sa určí, aká veľká bude mriežka stránky. V prípade, že je virtuálny honeypot jeden, je na obrazovke zobrazený celý priamo. Ak je ich viac, sú zobrazované do mriežky, schematicky je to znázornené na obrázku 11.



Obr. 11 *Návrh zobrazenia viacerých virtuálnych honeynetov súčasne*

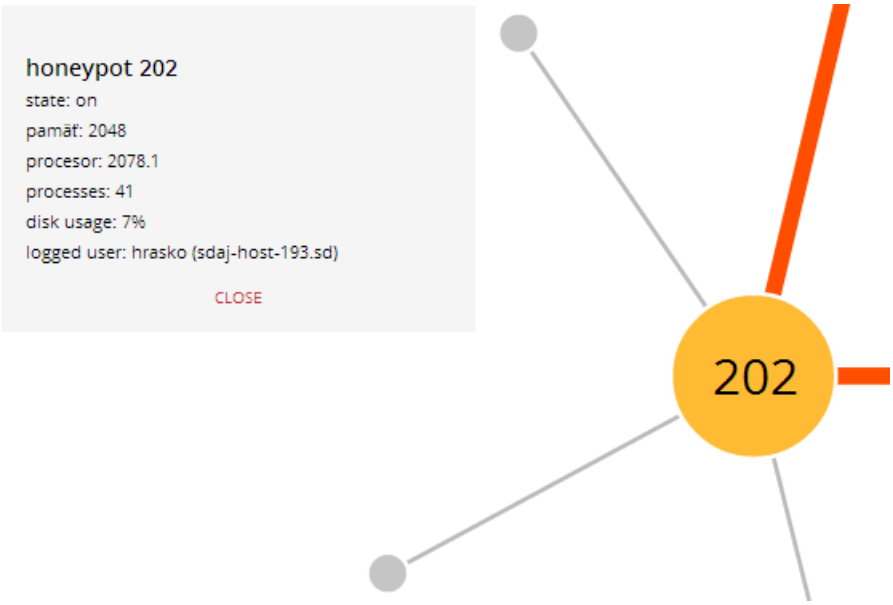
Pre zobrazenie sieťových spojení si klient vyžiada príslušné informácie z databázy na základe času, keďže ide o vizualizáciu v reálnom čase. Dáta získané v JSON formáte sú potom postupne prejdené a každé spojenie je vykreslené podľa príslušných pravidiel. Textové informácie nie je z hľadiska prehľadnosti dobré vypísať hneď. Radšej sme vybrali možnosť ich zobrazenia až na vyžiadanie užívateľa. Užívateľ dostane na začiatku základný prehľad o tom, v akom stave sa virtuálne honeynety nachádzajú a všetky konkrétne informácie získa po nabehnutí myšou na jednotlivé objekty – ide o IP adresy a porty, s ktorými je spojenie asociované, konkrétne hodnoty vyťaženia pamäte a procesora, atď.



Obr. 12 *Zobrazenie informácií o spojení*

Implementácie zobrazovania nezabúda ani na implementáciu kontroly dát. V tomto prípade je rozhodnutie prenechané na užívateľa a v prípade, že chce virtuálny honeypot vypnúť, má možnosť tak urobiť pravým klikom myši na daný vrchol.

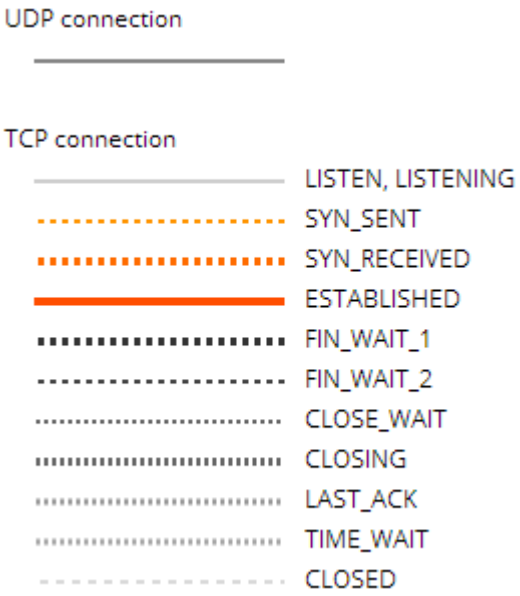
4.5.3 Implementácia zobrazenia dodatočných informácií



Obr. 13 *Zobrazenie informácií o stave po kliknutí na vrchol honeypotu*

Vrcholy, ktoré predstavujú honeypoty sú farebne vykreslené a odtieň farby je odstupňovaný podľa toho, aké veľké je vyťaženie pamäte a procesora virtuálneho honeypotu. Konkrétna hodnota vyťaženia je pravidelne získavaná spolu s informáciami o aktívnych spojeniach. Na výpočet percenta zaťaženia, podľa ktorého sú určené hranice odstupňovania odtieňa farby vrcholu je ešte potrebný údaj o celkovej veľkosti pamäte a procesora virtuálneho honeypotu. Rovnaké údaje ako pre virtuálne honeypoty sú dostupné aj pre hypervízor, ktorý virtualizáciu riadi. Tieto informácie sa nemenia, klientovi ich teda stačí vyžiadať raz na začiatku. Ako potom vidíme na obrázku farba vrcholu sa postupne mení v odtieňoch oranžovej farby od úplne svetlej ba až bielej v prípade, že vyťaženie je menšie ako 20% cez tmavooranžovú, hnedú a červenú, ak je vyťaženie väčšie ako 80%. Takéto vyťaženie je kritické a vieme jednoznačne povedať, že v danom čase dochádza k útoku na honeypot. To sa síce tiež prejaví aj vysokým počtom aktívnych spojení, napriek tomu, je však dobré zahrnúť do vizualizácie viacero markerov, ktoré indikujú rovnakú udalosť.

Okrem farebného odlíšenia vrcholu honeypotu je možné farebne rozlišovať aj uzly predstavujúce útočníkov. Kritéria, podľa ktorých je možné útočiace uzly okamžite označovať môžu byť napríklad konkrétne programy, ktoré spojenie vlastnia, alebo počet otvorených spojení. Takáto analýza je už však mimo rozsahu našej práce.



Obr. 14 *Legenda pre stavy TCP spojenia*

Všetky dostupné informácie o aktivite v honeynete, ktoré sú zbierané a spôsoby akými sú zobrazované sa nachádzajú v tabuľke 3.

Získavaný údaj	Zdroj	Jednotka	Vizualizácia
aktívne spojenia	netstat	-	hrany grafu
typ spojenia	netstat	-	farba čiary
zdrojová IP adresa a port	netstat	IPv4	text
cieľová IP adresa a port	netstat	IPv4	text
proces	netstat	PID	text
dĺžka spojenia	ss	min	číselná hodnota
veľkosť prenesených dát	nethogs	kB	číselná hodnota
útočníci		-	kruh
honeypoty		-	kruh
vyťaženie pamäte	free	MB	číselná hodnota

Získavaný údaj	Zdroj	Jednotka	Vizualizácia
vyťaženie procesora	ps	%	číselná hodnota
vyťaženie disku	df	%	číselná hodnota
počet procesov	ps	číslo	číselná hodnota
prihlásení užívatelia	last	-	text

Tab. 3 Dostupné informácie a spôsob ich zobrazenia

Príkaz **netstat** slúži na získanie informácií o aktuálnych prichádzajúcich a odchádzajúcich sieťových spojeniach, zobrazuje smerovacie tabuľky, počet sieťových rozhraní a poskytuje štatistiky podľa jednotlivých sieťových protokolov. V prípade použitia bez akýkoľvek prepínačov je výstupom tabuľka obsahujúca 6 stĺpcov: typ protokolu, veľkosť dát na prijatie príp. odoslanie, zdrojovú IP adresu, cieľovú IP adresu a stav spojenia. Identifikátor procesu, ktorý je vlastníkom daného spojenia získame v prípade zavolania príkazu *netstat –o*.

Ps[1] je príkaz zobrazenie aktuálne bežiacich procesov. Veľkosť voľnej a využitej pamäte vypíše v operačnom systéme Linux príkaz **free**[2]. **Df**[3] (*disk free*) poskytuje informácie o použitom a dostupnom diskovom priestore pripojených súborových systémov. Na zistenie posledných prihlásených užívateľov a miesta odkiaľ sú prihlásení slúži príkaz **last**[4]. Štatistiky soketov podobné príkazu *netstat* poskytuje aj príkaz **ss**[5]. V rámci operačného systému Linux je v poslednom čase tento príkaz preferovaný, pokrýva širšie spektrum informácií, lepší prehľad o aktuálne otvorených portoch a procesoch, ktoré sú s portom asociované. Na zistenie veľkosti prenesených dát podľa jednotlivých programov slúži v OS Linux príkaz **nethogs**[6].

Tieto príkazy získavajú všetky tieto informácie z Proc súborového systému. Zvolili sme získavanie údajov cez tieto príkazy pred ich priamym získaním z Proc

[1] http://linux.about.com/od/commands/l/blcmdl1_ps.htm

[2] http://www.linfo.org/free.html

[3] http://www.linfo.org/df.html

[4] http://linux.about.com/library/cmd/blcmdl1_last.htm

[5] http://www.cyberciti.biz/tips/linux-investigate-sockets-network-connections.html

[6] http://www.cyberciti.biz/faq/linux-find-out-what-process-is-using-bandwidth

súborového systému pretože sú súčasťou OS a nachádzajú sa v repozitároch. Všetky tieto nástroje sú distribuované pod Open Source licenciou.

V práci priamo s nástrojom na odosielanie dát nepracujeme, len využívame týmto spôsobom získané dáta.

4.6 Bezpečnostné aspekty

Keďže sa jedná o webovú aplikáciu chceme samozrejme zabezpečiť aby integrita dát, ktoré sú zobrazované ostala nepoškodená a nedošlo k obfuskácií kódu. Jedným zo spôsobov je implementácia kryptografickej kontroly integrity dát pomocou hashovacích funkcií HMAC, kde hashovacia funkcia je počítaná pomocou tajného kódu. Problémom však je to, že tento tajný kód by musel byť buď priamo zabudovaný v aplikácií alebo jej nejakým spôsobom byť doručený.

Riešením ako kontrolu integrity dát obísť je použitie protokolu HTTPS. O zaistenie integrity dát sa postará priamo protokol a na nás ostáva iba kontrola proti tzv. „man in the middle“ útoku, kedy útočník odchytí spojenie medzi klientom a serverom, ktorí komunikujú cez HTTPS protokol a s klientom ďalej komunikuje iba cez protokol HTTP.

Možnosťou je aj používať aplikáciu iba na lokálnom počítači administrátora a nepristupovať k nej cez Internet. Úplné odpojenie od Internetu však nie je možné kvôli prístupu na vzdialenú databázu, kde sa údaje z honeynetu ukladajú. Pri tomto spôsobe však sa však môžu vyskytnúť problémy s povoleniami JavaScript-u vo webovom prehliadači. Ide o tzv. Access-Control-Allow-Origin, kedy vzdialený server nepovolí vykonať požiadavku naň pokiaľ táto požiadavka nepochádza z rovnakej domény. Vo väčšine prípadoch je to však možné vyriešiť pridaním hlavičky Access-Control-Allow-Origin do HTTP odpovede zo servera.

5 Záver

Hodnota honeypotu ako bezpečnostného nástroja stále rastie. Témou výskumu sa stáva najmä virtuálny honeypot a distribuovaný honeynet pre jeho mnohé výhody ako napr. možnosť spojenia viacerých honeynetov do jedného väčšieho systému, čím sa získa viac informácií. V práci sme venovali problematike virtuálneho honeynetu a typom informácií, ktoré sa z neho dajú získať. Za základné informácie, na ktoré sa musí administrátor honeynetu sústrediť, považujeme sieťovú aktivitu prebiehajúcu v honeynete, ktorá indikuje, či už začínajúci alebo prebiehajúci útok. Popisujeme konkrétne typy týchto informácií a ich jednotlivý význam pre administrátora.V úvode práce sme sa venovali teoretickým poznatkom z oblasti honeypotov a honeynetov a ich významu ako bezpečnostného nástroja. Dôležitými výhodami virtuálnych honeynetov je ich škálovateľnosť a finančná nenáročnosť. Distribuovaný honeynet zase prináša možnosť prepájať siete honeynetov po celom svete a získavať tak komplexné informácie.

Najdôležitejšou časťou našej práce bolo vytvorenie webovej aplikácie na zobrazovanie spojení v rámci honeynetu. Pri jej implementácií bolo najväčším problémom vyriešiť efektívne, rýchle a prehľadné zobrazenie veľkého množstva spojení v jednom časovom útoku. Viacero na prvý pohľad efektívnych riešení bolo zamietnutých, keďže sme kládli veľký dôraz na to, aby vizualizácia získané dáta neskresľovala.

Pri vizualizácií sme použili metódu vytvorenia hierarchickej štruktúry informácií tak, aby bol minimalizovaný vizuálny šum. Využili sme kombináciu základných tvarov a farieb na rýchle pochopenie dát bez nutnosti vedomého sústredenia sa na nich. Výslednú aplikáciu sme konzultovali so skúsenými sieťovými administrátormi na overenie našich teoretických záverov.

Asi najzaujímavejšou časťou z pohľadu užívateľa je možnosť núdzového vypnutia honeypotu priamo z aplikácie. Túto vlastnosť odporúčame používať minimálne a s veľkým uvážením. Ako možnosť hlbšieho spracovania našej práce vidíme práve v možnosti kontroly dát priamo z aplikácie a rozšírenie práve tejto možnosti vypínania a zapínania honeynetu. Ponúka sa aj prípadné rozšírenie aplikácie o obsiahnutie štatistických údajov, pomocou ktorých by bolo možné porovnávať jednotlivé časové úseky.

6 Zoznam použitej literatúry

[1] BELLOVIN, S. 1992. There Be Dragons. Proc. of the Third Security Symposium, 1992.

[2] BIGELOW, S.J. 2008. *Network virtualization explained.* [online]. Dostupné na internete: http://searchitchannel.techtarget.com/feature/Network-virtualization-explained

[3] FLIGG, K. – MAX, G. 2012. *Network Security Visualization*. [online]. Dostupné na internete: http://www.cs.arizona.edu/~collberg/Teaching/466-566/2012/Resources/presentations/2012/reports.pdf

[4] HAO F. et al. *Secure Cloud Computing with a Virtualized Network Infrastructure*. [online]. Dostupné na internete: http://static.usenix.org/events/hotcloud10/tech/full_papers/Hao.pdf

[5] HOLZ, T. – PROVOS, N. 2007. Virtual honeypots: from botnet tracking to intrusion detection. Addison-Wesley Professional, 2007.

[6] JONES, T.M. 2010. *Virtualization.* [online]. Dostupné na internete: http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm

[7] JOSHI, R.C. – SARDANA A. 2011. Honeypots: A New Paradigm to Information Security. Science Publishers, 2011.

[8] KIRK, A. 2012. Data Visualization: A Successful Design Process. Packt, 2012.

[9] POTFAJ, M. 2011. Korelácia a vizualizácia IP tokov v počítačovej sieti. Brno: Masarykova univerzita, 2011.

[10] POWELL, D. – STROUD, R. 2003. Conceptual Model and Architecture of Maftia. Maftia Project, 2003.

[11] SCHÖSSER, M. 2012. *HoneyMap – Visualizing Worldwide Attacks in Real-Time.* [online]. Dostupné na internete: http://www.honeynet.org/node/960

[12] SINGH, A. 2004. *An Introduction to Virtualization.* [online]. Dostupné na internete: http://www.kernelthread.com/publications/virtualization

[13] SPITZNER, L. 2002. Honeypots: Tracking Hackers. Addison-Wesley Professional, 2002.

[14] SPITZNER, L. *Know Your Enemy: Defining Virtual Honeynets*. The Honeynet Project. [online]. Dostupné na internete: https://www.honeynet.org/papers/virtual

[15] SPITZNER, L. 2004. The Honeynet Project: Trapping the Hackers. IEEE Security & Privacy, 2004. s. 15-23.

[16] STOLL, C. 1988. Talking the Wiley Hacker. Communications of the ACM, 1988.

[17] TOKÁROVÁ L. 2010. Princípy vizualizácie informácií. Brno: Masarykova univerzita, 2010.

[18] YUCHIN CHENG, J. 2012. *HpfeedsHoneyGraph – Automated Attack Graph Construction for Hpfeeds Logs.* [online]. Dostupné na internete: http://www.honeynet.org/node/957

[19] *PicViz*. [online] Dostupné na internete: http://sourceforge.net/projects/picviz/

7 Prílohy

Príloha A: CD médium – bakalárska práca v elektronickej podobe, prílohy v elektronickej podobe, program implementovaný v práci.

Príloha B: Skripty na parsovanie

Príloha C: Skripty na zobrazenie

Príloha B: Skripty na parsovanie

Časť kódu na získanie údajov o aktívnych spojeniach z databázy:

```
<?php
$connection = mysqli_connect($hostname,$username,$password,$dbname,$dbport) or die('Could not connect: ' . mysql_error());
if (!($stmt = $connection->prepare("SELECT * FROM hn_data WHERE CTID!='HV' AND PARAM='NETSTAT' AND hn_id=? AND date BETWEEN ? AND ?"))) {
        echo "Prepare failed: (" . $connection->errno . ") " . $connection->error;
}
$stmt->bind_result($data_id, $hnid, $ctid, $param, $data, $date, $opt);
while ($row = $stmt->fetch()) {
   $parts = explode("\n", $data);
   $count = count($parts) - 1;
   for ($i=2;$i<$count;$i++){
                $removed = preg_replace('!\s+!', ' ', $parts[$i]);
                $parts2 = explode(" ", $removed);
                $divider_idx1 = strripos($parts2[3],":");
                $target = substr($parts2[3],0,$divider_idx1);
                $trgport = substr($parts2[3],$divider_idx1+1,strlen($parts2[3]));
                $divider_idx2 = strripos($parts2[4],":");
                $source = substr($parts2[4],0,$divider_idx2);
                $srcport = substr($parts2[4],$divider_idx2+1,strlen($parts2[4]));
                if($parts2[0]==="udp"){
                        $js = array('hnid' => $hnid, 'hpid' => $ctid, 'date' => $date, 'protocol' => $parts2[0], 'target' => $target, 'trgport' => $trgport, 'source' => $parts2[4], 'srcport' => $srcport, 'state' => "no_state", 'pid' => $parts2[5]);
                }
                else{
                        $js = array('hnid' => $hnid, 'hpid' => $ctid, 'date' => $date, 'protocol' => $parts2[0], 'target' => $target, 'trgport' => $trgport, 'source' => $parts2[4], 'srcport' => $srcport, 'state' => $parts2[5], 'pid' => $parts2[6]);
                }
                echo json_encode($js);
   }
}
mysqli_close($connection);
?>
```

Časť kódu na získanie údajov o stave honeypotu z databázy:

```
<?php
$connection = mysqli_connect($hostname,$username,$password,$dbname,$dbport) or die('Could not connect: ' . mysql_error());
if (!($stmt = $connection->prepare("SELECT * FROM hn_data WHERE param!='NETSTAT' AND hn_id=? AND CTID=? AND date BETWEEN ? AND ?"))) {
        echo "Prepare failed: (" . $connection->errno . ") " . $connection->error;
}
$stmt->bind_result($data_id, $hn_id, $ctid, $param, $data, $date, $opt);
$js = array('hnid' => $hnid);
while ($row = $stmt->fetch()) {
        $js['hpid'] = $ctid;
        $parts = explode(" ", substr($data,0,strlen($data) - 1));
        switch($param){
                case "MEMORY":
                        $js['mem_used'] = $parts[0];
                        $js['mem_free'] = $parts[1];
                        $js['mem_shared'] = $parts[2];
                        $js['mem_cache'] = $parts[3];
                        $js['mem_swap_total'] = $parts[4];
                        $js['mem_swap_shared'] = $parts[5];
                        break;
                case "DISK":
                        $js['disc_avail'] = $parts[0];
                        $js['disc_used'] = $parts[1];
                        break;
                case "LOGIN":
                        $js['user'] = $parts[2];
                        $js['log_from'] = $parts[3];
                        break;
                case "CONNECTION":
                        $js['conn_tcp'] = $parts[0];
                        $js['conn_udp'] = $parts[1];
                        break;
                case "PROCESSES":
                        $js['num_proc'] = $parts[0];
                        $js['cpu'] = $parts[1];
                        break;
        }
        $js['date'] = $date;
}
echo json_encode($js, JSON_NUMERIC_CHECK);
mysqli_close($connection);
?>
```

Príloha C: Skripty na zobrazenie

Časti kódu na zobrazenie údajov získaných v JSON formáte:

Rovnomerné rozloženie honeypotov na plochu:

```
honeypots.forEach(function(d) {
        d.fixed = true;
        var n = honeypots.length;
        if(n == 1){
                d.x = d.px = width/2;
                d.y = d.py = height/2;
        } else {
            var result = $.grep(honeypotsObj, function(e) { return e.hnid==d.hnid && e.hpid == 
d.hpid; });
            var angle = (2 * result[0].k * Math.PI) / n;
                d.x = d.px = width/2 + 200 * Math.cos(angle);
                d.y = d.py = height/2 + 150 * Math.sin(angle);
        }
});
```

Zakreslenie honeypotov a určenie farby vrchola:

```
node.append("circle").attr("cx", function(d) { return d.x; }).attr("cy", function(d) { return d.y;
}).attr("r", function(d){
        if(isHoneypotNode(d)){ return 2*radius; }
        else { return radius/2; }
}).attr("style", colorNode);

function colorNode(d){
        if(isHoneypotNode(d)){
         if(state.state == "off"){
          return "fill: #fff; stroke: #aaa;";
         }else if(state.mem_used>=0.8 * totalObj.mem_used || state.cpu>=0.8 * totalObj.cpu){
          return "fill: #CC0000;";
         }else if(state.mem_used>=0.6 * totalObj.mem_used || state.cpu>=0.6 * totalObj.cpu){
          return "fill: #FF8800;";
         }else if(state.mem_used>=0.4 * totalObj.mem_used || state.cpu>=0.4 * totalObj.cpu){
          return "fill: #FFBB33;";
         }else if(state.mem_used>=0.2 * totalObj.mem_used || state.cpu>=0.2 * totalObj.cpu){
          return "fill: #669900;";
         }else{ return "fill: #99CC00;"; }}
}
```