FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Klasifikace komunikace SSH protokolu
Student:	Bc. Radek Smejkal
Vedoucí:	Ing. Karel Hynek
Studijní program:	Informatika
Obor / specializace:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Seznamte se s problematikou monitorování síťového provozu na úrovni paketů a tzv. síťových toků (IP Flows) a nastudujte si specifikaci SSH protokolu.
Nasbírejte vzorky vybraných typů SSH provozu (např. přihlašování heslem/klíčem, kopírování souborů, vzdálený terminál) v podobě zachycených paketů a rozšířených síťových toků a tím vytvořte anotovanou datovou sadu síťového provozu. Proveďte analýzu zachyceného provozu, zaměřte se na charakteristické vlastnosti (např. délky určitých paketů, časování), které je možné využít pro jejich identifikaci.
Navrhněte algoritmus pro rozpoznávání typů provozu přenášeného skrz šifrovaný SSH kanál na základě charakteristik chování komunikace.
Dle návrhu vytvořte softwarový prototyp, který je schopen zpracovávat provoz z reálné sítě.
Navržené řešení otestuje a vyhodnoťte přesnost klasifikace a výkonové parametry prototypu (např. potřebné zdroje a rychlost zpracování dat).

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 20. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Klasifikace komunikace SSH protokolu

Bc. Radek Smejkal

Katedra informační bezpečnosti
Vedoucí práce: Ing. Karel Hynek

6. května 2021

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce panu Ing. Karlu Hynkovi za jeho cenné rady, připomínky, nápady a v neposlední řadě za čas a ochotu konzultovat v průběhu vzniku celé diplomové práce. Dále bych rád poděkoval panu Bc. Petru Medonosovi za veškeré připomínky k této práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 6. května 2021 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2021 Radek Smejkal. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Smejkal, Radek. *Klasifikace komunikace SSH protokolu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá problematikou šifrovaného provozu SSH protokolu z pohledu bezpečnostního monitoringu sítě a jeho následnou klasifikací se zaměřením na část ověřující identitu uživatele.

Práce analyzuje zachycenou komunikaci i samotný SSH protokol a odhaluje specifické rysy různých situací. Následně implementuje výstupy analýzy do softwarového prototypu. Detekční algoritmus je navržený s ohledem na vysokou propustnost tak, aby byl vhodný i pro vysokorychlostní sítě, kde není možné z výkonových důvodů provádět inspekci jednotlivých paketů.

Pro vyhodnocení úspěšnosti se používá zachycený provoz reálné sítě a ve většině situacích prototyp dosahuje velmi přesných výsledků. Závěrem práce jsou diskutována možná opatření ke zvýšení přesnosti detektoru u neobvyklých situací nebo u zatím méně rozšířených parametrů spojení.

Klíčová slova SSH, analýza šifrovaného provozu, SSH autentizace, klasifikace síťového provozu, IDS, NEMEA

Abstract

This work focuses on the issue of SSH protocol encrypted traffic in terms of network security monitoring and its subsequent classification with emphasis on the authentication phase.

The aim of this work is to perform an SSH protocol analysis using intercepted communication and the protocol definition itself to reveal specific features of various situations. The outputs of the analysis are then implemented into a software prototype. The detection algorithm is designed with regard to high throughput so that it is also suitable for high-speed networks, where due to performance reasons is not possible to inspect all individual packets.

Captured traffic from a real network is used to evaluate the accuracy of the detector and in most situations the prototype achieves very accurate results. At the end of work, possible measures to increase the accuracy in unusual situations or in the less common connection parameters are discussed.

Keywords SSH, encrypted traffic analysis, SSH authentication, network traffic classification, IDS, NEMEA

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Úvod

Nové technologie, pokročilá mobilní či chytrá zařízení a především jejich propojení skrze internet nám přináší velké možnosti, ale s tím přirozeně přichází také zvyšující se riziko jejich zneužití. Počítačová bezpečnost, nejen na internetu, se stává čím dál více diskutovaným tématem a měla by být patřičně zohledněna ve všech nových i starších technologiích.

Počítače propojují čím dál více lidí, organizací, firem i oblastí důležité infrastruktury, z nichž každý potřebuje pro své prostředí zajistit adekvátní úroveň zabezpečení přizpůsobenou jeho potřebám. Jednou z používaných defenzivních technik je monitorování síťového provozu specializovanými nástroji, které upozorňují na potenciálně rizikové až škodlivé komunikace.

Tato práce pojednává o šifrovaném protokolu SSH, který se používá např. k přihlášení na vzdálené servery, automatizované správě systémů, přenosům dat apod. V oblasti informačních technologií je velice rozšířený a díky své popularitě i možnostem se stává častým cílem útoků [1].

Získání neoprávněného přístupu k SSH účtu může mít až fatální následky včetně úniku dat, narušení běžících služeb nebo i úplnou kompromitaci serveru popř. dalších napojených systémů. Proto je nezbytné automaticky detekovat a zamezovat různým druhům útoků např. využívajících hádání hesel, ale také preventivně validovat, zda použité nastavení odpovídá bezpečnostním politikám organizace. Jedním z přínosů práce jsou právě implementované detekční mechanismy odhalující neúspěšné pokusy o přihlášení, které ve zvýšeném počtu implikují útoky hádání hesel.

Práce se skládá ze dvou částí, teoretické a praktické. Cílem teoretické části je seznámení čtenáře s detailním fungováním SSH protokolu se zaměřením na fázi ověření identity. Dále práce seznamuje s problematikou monitorovaní síťového provozu včetně používaných technik, jejich výhod, nevýhod a limitací reálného nasazení.

Praktická část analyzuje komunikaci SSH protokolu pomocí síťových toků a navrhuje signatury k jeho klasifikaci s ohledem na náročnost využití ve velkých a vysokorychlostních sítích. Součástí je také následné vytvoření prototypu detektoru a ověření jeho přesnosti na testovací datové sadě zachyceného provozu reálné sítě. Závěrem práce diskutuje naměřené výsledky a možnosti dalšího rozvoje.

Kapitola 1

SSH protokol

Secure Shell protokol se zkratkou SSH je obecně rozšířený způsob nejen pro vzdálené přihlášení. Protokol navazuje šifrovanou komunikaci, zajišťuje důvěrnost a integritu dat pomocí bezpečnostních algoritmů a v neposlední řadě také ověřuje idetitu jak serveru, tak i uživatele.

Celosvětová organizace IANA [2] rezervuje SSH protokolu port 22 a dle studie [1] zabývající se škodlivým provozem proti nastraženému honeypotu ukazuje, že spolu s jeho předchůdcem, protokolem telnet na portu 23, patří mezi dva nejčastější cíle. Dokonce tvořily 90 % zachycených útoků. To svědčí o nutnosti zajištění adekvátní úrovně bezpečnosti, a proto některé nasazení mění výchozí hodnotu portu, což komplikuje odhalení služby. Analýza dat z reálné sítě [3] uvádí nejčastější alternativní porty 22222 a 2222.

1.1 Historie a použití

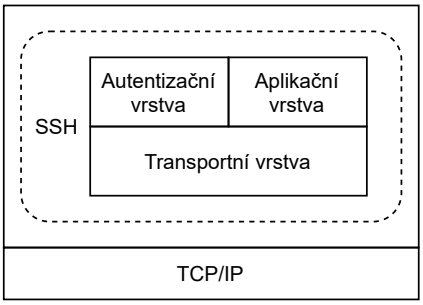
První verze SSH protokolu, dnes známá pod označením SSH-1, vznikla v roce 1995 v reakci na četné síťové odposlechy hesel jako bezpečná šifrovaná alternativa k historicky nezabezpečeným protokolům pro vzdálené přihlášení a datové přenosy jako telnet, rlogin, rcp a ftp. V současnosti se používá ve standardizované verzi SSH-2, která byla upravena tak, aby odpovídala i dnešním nárokům především v oblasti bezpečnosti, a kvůli některým odlišnostem nejsou verze vzájemně kompatibilní [4, 5].

SSH se nejčastěji využívá pro vzdálené přihlášení popř. spouštění příkazů, správu síťových prvků a datové přenosy. V neposlední řadě umožňuje také vytvoření tzv. SSH tunelů mezi dvěma koncovými TCP porty, které mají řadu užitečných využití. Jedním z nich je zabezpečení komunikace např. se starší aplikací, která neumožňuje šifrování. Kromě výhody šifrování spojení lze využít i poskytovanou autentizaci např. ke zpřístupnění služeb, které nemohou být veřejně dostupné z internetu. Veškeré použití SSH protokolu je možné realizovat jak interaktivní, tak plně automatizovanou cestou [5].

Koncept tunelů se do jisté míry podobá VPN připojení. Ovšem ve firemním prostředí může propojení vzdálených TCP portů skrze šifrovaný SSH tunel představovat také určité riziko zneužití resp. možnost obcházení bezpečnostních ochran jako je firewall [6].

1.2 Struktura SSH protokolu

Protokol je založen na architektuře typu klient-server nad síťovým protokolem TCP transportní vrstvy ISO/OSI modelu. Skládá se ze třech navazujících komponent resp. vrstev, které oddělují výměnu šifrovacích klíčů, autentizaci uživatele a samotný obsah spojení. Díky konceptu a specifikaci poslední vrstvy je možné vytvářet další protokoly využívající bezpečné šifrované spojení, které SSH poskytuje [7]. Strukturu vrstev ukazuje obrázek 1.1.



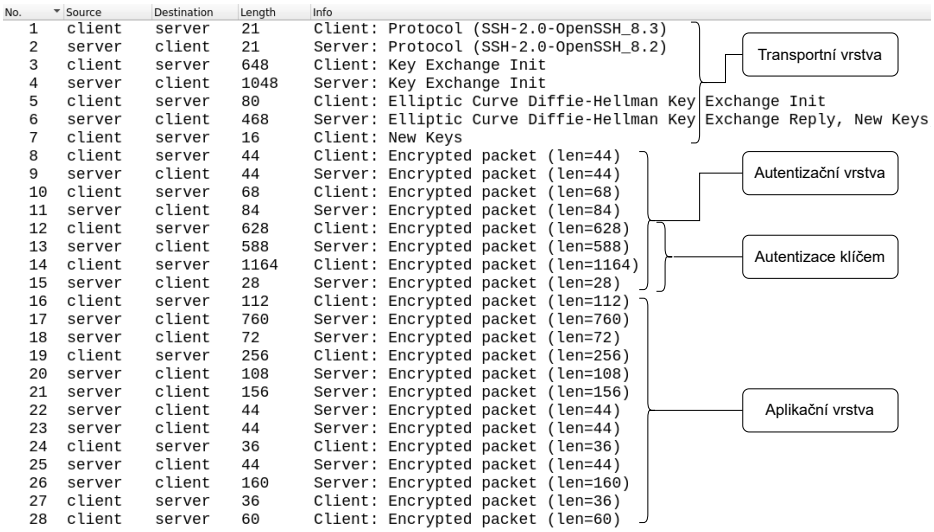
Obrázek 1.1: Struktura SSH protokolu [7]

Transportní vrstva ustanovuje parametry pro navázání zabezpečené komunikace. K tomu využívá šifrování a algoritmy typu MAC (Message Authentication Code) pro zajištění důvěrnosti a integrity zasílaných dat. Rovněž umožňuje klientům ověřit identitu serveru pomocí digitálního podpisu během ustanovování šifrovacích klíčů.

Autentizační vrstva navazuje na předchozí transportní vrstvu a jejím jediným cílem je autentizovat uživatele vůči serveru. Standard definuje více možných způsobů autentizace. Výběr použité autentizační metody závisí na volbě klienta, ale zároveň server nemusí podporovat všechny autentizační metody a je potřeba vzájemná shoda.

Aplikační vrstva staví také na transportní vrstvě, leží tedy na srovnatelné úrovni jako autentizační vrstva, nicméně v praxi z pravidla následuje až po úspěšném ověření identity uživatele. Hlavním přínosem této vrstvy je abstrakce spojení do logických kanálů. Díky tomu lze definovat další aplikace nad šifrovaným a autentizovaným spojením. Vrstva zajišťuje také multiplexování vytvořených kanálů do jediného spojení, takže se kanály neovlivňují navzájem a jsou na sobě nezávislé.

Pro účely práce používáme tyto názvy vždy ve vztahu k protokolu SSH. Jako prevenci možné záměny názvů s vrstvami modelu ISO/OSI budeme situace vztažené k síťové úrovni více konkretizovat. Nejčastěji deklarací TCP protokolu, tedy jednoho z protokolů transportní vrstvy ISO/OSI modelu. Následující obrázek zobrazuje příklad zachycené komunikace SSH protokolu pomocí nástroje Wireshark [8] s rozdělením na jednotlivé vrstvy.



Obrázek 1.2: Ukázka zachyceného SSH spojení

1.3 Transportní vrstva

Transportní vrstva SSH spojení staví na síťovém protokolu TCP a zajišťuje navázání zabezpečené komunikace i přes nedůvěryhodnou síť a její úspěšné dokončení je nutným základem pro pokračování SSH protokolu. Parametry bezpečného spojení se neodvozují od identifikačních údajů uživatele či serveru, proto ani po případné kompromitaci nelze v budoucnu dešifrovat dříve zachycená spojení bez originálních šifrovacích klíčů. Tato důležitá kryptografická vlastnost se nazývá dopředná bezpečnost (forward secrecy) a poskytuje užitečnou ochranu minulých spojení při kompromitaci serveru či uživatele [9].

První zprávou obou stran musí být dle standardu [10] identifikační řetězec verze protokolu a knihovny použité implementace. Aktuální verze SSH protokolu 2.0 není zpětně kompatibilní s předchozí, a proto ji musí obě strany kontrolovat, aby předešly možným komplikacím. V dnešní době by se ovšem starší verze neměla nikde vyskytovat, proto lze tento začátek považovat spíše za symbolický pozdrav bez dalšího účelu.

1.3.1 Formát zpráv

SSH protokol používá jednotný formát zpráv napříč celým spojením mimo první zprávy s verzí protokolu. Jednotlivé části zajišťují bezpečnostní koncepty a umožňují bezproblémové dešifrování. Znalost jejich možných velikostí a způsobu vytvoření se nám bude hodit v analýze při rozpoznávání některých druhů zpráv. Strukturu zobrazuje obrázek 1.3.

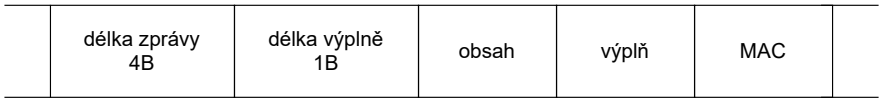
Délka zprávy – První 4 B každého paketu tvoří celková délka, která ovšem nezahrnuje MAC ani sebe sama. Za normální situace se šifruje celá zpráva včetně této hodnoty. Výjimku tvoří situace s MAC algoritmem v módu ETM (Encrypt Than MAC), kde zůstává délka zprávy po celou dobu spojení nešifrovaná.

Délka výplně – Pro korektní dešifrování je potřeba oddělit náhodnou výplň od obsahu zprávy, proto je informace o její délce nezbytná. Tato část zabírá 1 B a musí být šifrovaná, jinak by došlo k eliminaci významu náhodné výplně.

Obsah zprávy – Užitečný obsah zprávy je jediné místo, které prochází kompresí, pokud se na ní strany dohodly.

Výplň – Náhodná výplň tvoří důležitý prvek zprávy a komplikuje určení přesné velikosti obsahu zprávy. Standard požaduje alespoň 4 B výplně. Navíc stanoví, že délka paketu bez části MAC je násobkem osmi resp. násobkem šifrovacího bloku šifry (záleží co je větší). Nejvyšší možná délka výplně je omezena na 255 B v souladu s maximálním rozsahem hodnot pole délka výplně.

MAC – Poslední část MAC zajišťuje integritu a autenticitu zprávy. Transportní vrstva zatím nemá dohodnutý MAC algoritmus, proto v ní tato část chybí.



Obrázek 1.3: Formát zprávy SSH protokolu [10]

1.3.2 Ustanovení šifrovacích klíčů

Proces ustanovení parametrů zabezpečeného spojení začíná obousměrnou výměnou podporovaných algoritmů zprávou SSH_MSG_KEXINIT. Protokol podporuje použití odlišných šifrovacích/MAC/kompresních algoritmů pro každý směr komunikace odděleně, proto zpráva obsahuje několik těchto seznamů.

Jednotlivé seznamy jsou seřazeny dle preferencí. Za zmínku stojí náhodné „cookie“ o velikostech 16 B, které se dále používají při vytváření unikátního identifikátoru spojení, a tím komplikují odhalení jak identifikátoru spojení, tak ustanovených klíčů.

Použité algoritmy se vybírají metodou první podporované shody. Obě strany komunikace procházejí seznamy algoritmů protistrany a jakmile narazí na společně podporovaný algoritmus, tak ho použijí. Pokud ovšem nenaleznou žádnou shodu v některém ze seznamů, nelze ve spojení pokračovat a nezbývá nic jiného než ho ukončit.

Jinak se pokračuje zvoleným algoritmem výměny klíče pro symetrické šifrování. Algoritmus musí obsahovat kryptografické ověření identity serveru. Bez této vlastnosti by šlo jednoduše vytvořit falešný SSH server a tím realizovat útok typu „Man In The Middle“. Standard transportní vrstvy SSH protokolu [10] rozšiřuje všeobecně známý Diffie-Hellmanův algoritmus právě o ověření identity serveru pomocí digitálního podpisu viz následující sekce 1.3.3. Obecně lze použít libovolný algoritmus výměny klíčů, očekává se však výstup jako dvojce (K, H), kde H je unikátní identifikátor aktuální relace a K je ustanovený sdílený klíč, ze kterého se dále odvodí šifrovací klíče.

1.3.3 Diffie-Hellmanův algoritmus s ověřením identity serveru

Rozšíření o ověření identity serveru zachovává počet zpráv i matematický aparát Diffie-Hellmanova algoritmu, a tím není dotčena náročnost řešení problému diskrétního logaritmu. Ověření zakládá na podpisu identifikačního řetězce relace serverovým klíčem. Tento unikátní identifikátor relace zahrnuje také výše zmíněné náhodné „cookie“ a ustanovený sdílený klíč, což zajišťuje aktuálnost podpisu.

Protokolem očekávaný identifikátor relace se tvoří s použitím předchozích informací tj. identifikačního řetězce použité verze protokolu (první 2 pakety) a zpráv SSH_MSG_KEXINIT, které mimo jiné obsahují právě náhodnou „cookie“ na obou stranách, což zajišťuje unikátnost identifikátoru i pro totožné preference vyměňovaných bezpečnostních algoritmů. Celý proces probíhá ve třech krocích resp. zprávách.

1. Klient vygeneruje soukromý náhodný klíč x splňující $1 < x < q$, kde q je řád dohodnuté podgrupy $GF(p)$ s prvočíslem p. Spočítá $e \equiv g^x \bmod p$, kde g je generátor a následně posílá e serveru jako svůj veřejný klíč.

2. Server také vygeneruje soukromý náhodný klíč y splňující stejné podmínky jako x a spočítá $f \equiv g^y \bmod p$. Poté ustanoví výsledný sdílený klíč K a vytvoří hash H pomocí následujících vztahů:

$$\mathbf{K} = \mathbf{e^y} \bmod p \equiv (g^x)^y \bmod p \equiv g^{xy} \bmod p \equiv g^{yx} \bmod p \equiv (g^y)^x \bmod p$$

$$\mathbf{H} = hash(V_C \,||\, V_S \,||\, I_C \,||\, I_S \,||\, K_S \,||\, e \,||\, f \,||\, K)$$

V_C	...	identifikační řetězec klienta (z 1. paketu)
V_S	...	identifikační řetězec serveru (z 2. paketu)
I_C	...	klientská zpráva SSH_MSG_KEXINIT **s náhodnou cookie**
I_S	...	serverová zpráva SSH_MSG_KEXINIT **s náhodnou cookie**
K_S	...	**veřejný klíč serveru** (např. RSA)
e	...	Diffie-Hellman veřejný klíč klienta
f	...	Diffie-Hellman veřejný klíč serveru
K	...	sdílený klíč

Nakonec server podepíše hash H svým soukromým klíčem ke K_S tj. $s = sign_{K_S}(H)$, čímž umožní kontrolu identity serveru a vše odešle ve formátu $(K_S \,||\, f \,||\, s)$.

3. Klient porovná veřejný klíč serveru K_S se svým záznamem z dřívějších spojení. Následně z přijatého f vypočítá sdílený klíč K výše zmíněným vztahem a z předchozích informací také odvodí hash H. Nakonec pomocí veřejného klíče K_S ověří podpis s [10].

1.4 Autentizační vrstva

Autentizační vrstva navazuje na předchozí transportní vrstvu a jejím účelem je poskytnout ověření identity uživatele vícero nabízenými způsoby dle konfigurace serveru. Pro zajištění určité flexibility volí autentizační metodu vždy klient. Server ovšem nemusí všechny podporovat, čímž nastavuje požadovanou úroveň ověření identity. Před samotným pokusem o ověřením klient iniciuje autentizační vrstvu jako službu pomocí zprávy SSH_MSG_SERVICE_REQUEST s obsahem „ssh-userauth“. Stejným principem funguje i pozdější navázání aplikační vrstvy spojení hodnotou „ssh-connection“.

Autentizace probíhá zprávou SSH_MSG_USERAUTH_REQUEST, která obsahuje všechny potřebné informace jak o zvolené metodě, tak o uživateli vč. jména. Ke konci se nachází specifické údaje zvolené autentizační metody. Standard [11] definuje 3 základní ověřovací metody, které rozebereme zvlášť. Klient může využít i tzv. „none“ autentizaci. Ta slouží ke zjištění podporovaných autentizačních metod a neměla by vést k úspěšnému ověření.

1.4.1 Autentizace heslem

Nejjednodušší způsob ověření identity klienta, nejen u SSH protokolu, je pomocí sdíleného tajemství resp. hesla. V případě SSH se heslo posílá v normalizovaném formátu ISO-10646 UTF-8 [12] jako prevence multiplatformních problémů. Na síti je samozřejmě chráněno šifrováním předchozí transportní vrstvy. Server heslo ověřuje svým způsobem např. porovnáním s vlastní či vzdálenou databází. V případě zamítnutí může klient ve stejné relaci ověření opakovat či zvolit jiný způsob autentizace.

Díky své jednoduchosti, plošnému rozšíření, možnosti opakovaně hádat a předvídatelnosti se hesla stávají častým cílem slovníkových útoků. Studie [13] z preventivních důvodů doporučuje několik opatření ke snížení rizika kompromitace, mezi které patří např. omezení dostupnosti služby nezbytné skupině uživatelů nebo i náhrada ověření heslem za jinou autentizační metodu.

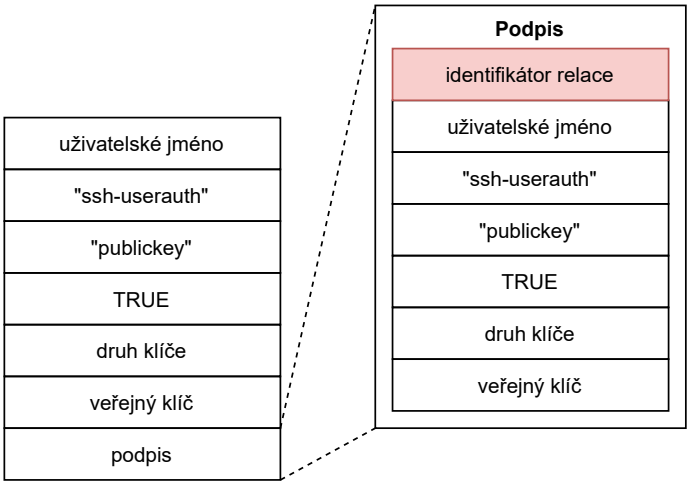
1.4.2 Autentizace veřejným klíčem

Vyšší odolnost proti útokům poskytuje způsob ověření klíčem. Místo ověření pomocí sdíleného tajemství využívá principy asymetrické kryptografie, tedy vytvoření digitálního podpisu a jeho ověření. Aby autentizace úspěšně proběhla, musí být veřejná část klíče předem uložena na serveru. V cloudovém či firemním prostředí lze proces distribuce veřejných klíčů centralizovat a automatizovat. Příslušný soukromý klíč je nutné chránit, protože jeho odhalení může vést až ke kompromitaci důležitých systémů. Proto se z bezpečnostních důvodů klíče ukládají šifrovaně, aby ani v případě ztráty či zcizení zařízení nedošlo k odhalení soukromého klíče.

Kvůli úsporám výpočetní náročnosti digitálního podpisu protokol umožňuje zjistit „platnost“ veřejného klíče, což odpovídá kontrole zda daný veřejný klíč pro daného uživatele existuje a je možné ho využít pro autentizaci. Dle standardu [11] server takový autentizační požadavek buď rovnou zamítá zprávou SSH_MSG_USERAUTH_FAILURE, nebo naopak potvrzuje existenci veřejného klíče zprávou SSH_MSG_USERAUTH_PK_OK. Součástí potvrzující zprávy je taktéž přijatý veřejný klíč. Přestože není toto (před)ověření vyžadované a nemá žádný vliv na výsledek autentizace, hojně se využívá.

K provedení autentizace klient zasílá stejnou zprávu jako v předchozím případě doplněnou o podpis použitím příslušného soukromého klíče. Podpis se tvoří z celé zprávy SSH_MSG_USERAUTH_REQUEST (samozřejmě bez jejího podpisu) spolu s předcházejícím identifikátorem relace, což zajišťuje rozpoznání aktuálnosti vytvářeného podpisu. Zahrnované údaje autentizačního požadavku a podpisu ukazuje obrázek 1.4.

Po přijetí této zprávy server opět kontroluje zda má uživatel nastavený zaslaný veřejný klíč, i když kontrolu provedl v předchozím nepovinném kroku, a v neposlední řadě ověří platnost digitálního podpisu. Je-li ověření úspěšné, server autentizaci potvrzuje zprávou SSH_MSG_USERAUTH_SUCCESS.



Obrázek 1.4: Struktura zprávy autentizace klíčem [11]

V opačném případě musí klient ověření opakovat, nebo spojení ukončit. Procesu ověření klíčem se budeme dále věnovat v kapitole 3, kde se zaměříme na jednotlivé situace např. důvod používání (před)ověření veřejného klíče.

1.4.3 Autentizace serverovým klíčem

Poslední standardem [11] definovaný způsob je navržený k ověření identity klientského počítače místo samotného uživatele. Funguje velmi podobně jako předchozí autentizace veřejným klíčem, akorát místo klíče uživatele používá klíč serveru, který bude ovšem společný pro všechny uživatele systému.

Z pohledu bezpečnosti přináší tento přístup určitá rizika např. v případě kompromitace. Nicméně nevyžaduje žádnou uživatelskou akci, čímž může najít své uplatnění v automatizaci. Pro reálné použití je potřeba celý proces velmi důkladně nastavit vč. synchronizace uživatelských jmen, ale i tak zakládá na důvěře mezi systémy resp. jejich administrátory.

Způsob autentizace serverovým klíčem nebudeme dále analyzovat, protože se principiálně neliší od předcházejícího ověření veřejným klíčem a jeho praktické využití bude směřovat spíše do uzavřených sítí, ze kterých nemáme podkladová data o provozu.

1.4.4 Ostatní přípustné zprávy

Standard [10, 11] pamatuje také na velké množství různých situací a scénářů, jež mohou nastat. Proto předem definuje speciální zprávy, které z pravidla nemají vliv na chod protokolu a dokonce mohou být ignorovány, ale pro dopředné zajištění vzájemné kompatibility by jim měli implementace rozumět.

SSH_MSG_DEBUG zasílá ladící informace a příjemce je oprávněn ji zcela ignorovat.

SSH_MSG_USERAUTH_BANNER slouží k informování klienta zpravidla před zahájením autentizace o možných právních ujednáních a důsledcích narušení systému.

SSH_MSG_IGNORE je navržena pro zamezení analýzy provozu a její obsah i čas zaslání může být libovolný.

Práce se zaměřuje především na autentizační část protokolu a zpráva SSH_MSG_IGNORE představuje možné riziko ovlivnění detekcí. Nicméně během vytváření datové sady ani během analýzy jsme nenarazili na použití některé ze zpráv tohoto druhu, a proto se jimi dále nezabýváme.

1.5 Aplikační vrstva

Poslední vrstva zavádí tzv. logické kanály, které slouží jako jistá abstrakce nad šifrovaným spojením. Jejich účelem je umožnit komunikaci skrze vytvořené šifrované spojení více datovým proudům zároveň, aniž by se navzájem ovlivňovaly. Každý kanál se eviduje pod unikátním číslem, čímž se řeší jejich multiplexing do jediného šifrovaného SSH spojení. Otevření kanálu se provádí zprávou SSH_MSG_CHANNEL_OPEN a mohou ho iniciovat obě strany, avšak v některých situacích (např. vzdálené spuštění programu) bývá serverem iniciované vytvoření kanálu potlačeno. Standard [14] zároveň definuje 3 typy kanálů a popisuje jejich specifické možnosti.

Session obecně slouží pro spuštění vzdáleného programu. V rámci tohoto kanálu lze např. vytvořit vzdálený shell nebo spustit systémový příkaz. Standard pamatuje také na oddělení chybového výstupu, proměnných prostředí a zasílání návratových kódu či signálů.

X11 se používá pro účely zobrazení okna vzdálené grafické aplikace pomocí X window systému na straně klienta.

TCP/IP port forwarding vytváří propojení lokálního a vzdáleného TCP portu skrze šifrované SSH spojení. Tento koncept je známý jako již zmíněný SSH tunel.

Kapitola 2

Síťový monitoring

S rostoucím počtem technologií a jejich vzájemnou integrací skrze internet se přirozeně objevují nové zranitelnosti a podvody, před kterými je potřeba se bránit. Jednu z používaných vrstev obrany představují IDS nástroje (Intrusion Detection System) analyzující síťový provoz, ve kterém různými způsoby hledají stopy potenciálně škodlivé komunikace. Provozování takových nástrojů rozhodně není jednoduchý, ani levný úkol a jejich nasazení je nutné přizpůsobit danému prostředí. Přesto vzhledem k množství a riziku současných i budoucích hrozeb se stal síťový bezpečnostní monitoring pro mnoho organizací nezbytnou součástí ochrany jejich digitálních aktiv a do budoucna můžeme předpokládat pouze růst v tomto směru.

IDS systémy můžeme rozdělit podle principu prováděné detekce na několik druhů. Není neobvyklé kombinovat detekční principy za účelem zpřesnění detekce nebo pokrytí širšího spektra rozpoznávaných hrozeb.

Detekce signatur funguje na principu hledání známých vzorců škodlivého provozu podle předem vytvořené databáze signatur. Databáze se musí s přicházejícími novými hrozbami aktualizovat, jinak je nedokáže rozpoznat. Zástupcem jsou např. nástroje Suricata [15] a Snort [16].

Detekce anomálií dokáže upozornit na neobvyklé situace v provozu, které mohou naznačovat nové či zatím neznámé hrozby.

Behaviorální detekce dává do kontextu zachycená spojení monitorovaných zařízení, což umožňuje definovat typické rysy narušení, které se neprojeví v detekci signatur jednotlivých spojení.

Dále dělíme IDS systémy na dvě základní kategorie podle analyzovaného rozsahu dat, které si detailněji popíšeme v následujících sekcích. Oba přístupy ovšem vyžadují zachycení síťového provozu, který lze realizovat pomocí síťové sondy, tedy dedikovaného zařízení pro pasivní odposlech provozu a to i na optické lince.

2.1 Analýza paketů

Klasickým přístupem k síťovému monitoringu je zachycení kompletního provozu vč. obsahu jednotlivých paketů a jeho následná analýza. Díky přesné kopii komunikace lze provést prakticky libovolné druhy detekcí. Častým a poměrně spolehlivým přístupem jsou již výše zmíněné detekce signatur. Komplikaci představují stále se rozšiřující šifrované protokoly, jejichž obsah paketová analýza nedokáže plně využít.

Nevýhoda paketového přístupu je bezesporu vysoká náročnost na výpočetní zdroje i diskový prostor. Experimentální studie nástrojů Suricata a Snort [17, 18, 19, 20, 21] porovnávají náročnost na výpočetní zdroje a přesnost detekce pro různé rychlosti sítě. Při vyšších rychlostech oba systémy vykazovali zvýšené množství nezachycených paketů, což se odrazilo do úspěšnosti detekcí a míry falešné pozitivity.

S využitím specializovaných řešení hardwarové akcelerace je dosažitelné i jejich produkční nasazení na některých rychlejších sítích. Řešení společnosti Napatech [22] uvádí dosažitelnou rychlost pro nástroj Suricata až 40 Gbps bez ztráty paketů. Ale nasazení do současných vysokorychlostních sítí o rychlostech v řádu stovek Gbps by bylo velmi komplikované.

2.1.1 Nástroj Suricata

Suricata je multiplatformní opensource nástroj na odhalování síťových hrozeb pomocí analýzy paketů. Využívá princip hledání signatur pomocí databáze pravidel, které lze jednoduše rozšiřovat. Nechybí ani široká podpora protokolů síťové aplikační vrstvy jako HTTP, DNS, SSH, apod. Pomocí skriptovacího jazyka Lua lze také rozšiřovat detekční funkce pro situace, kde nelze aplikovat rozpoznání signatury. Samozřejmostí je i integrace s nástroji SIEM (Security Information and Event Management) či obdobným řešením pro centrální správu logů a událostí [15, 23].

2.1.2 Nástroj Zeek

Zeek funguje oproti klasickému paketovému přístupu na principu událostí, čímž dokáže redukovat množství analyzovaných dat, ale stále zachovává pohled do vyšších síťových vrstev. Konceptem připomíná nástroj analýzy síťových toků a dokumentace [24] dokonce uvádí i možné nasazení na rychlejších sítích.

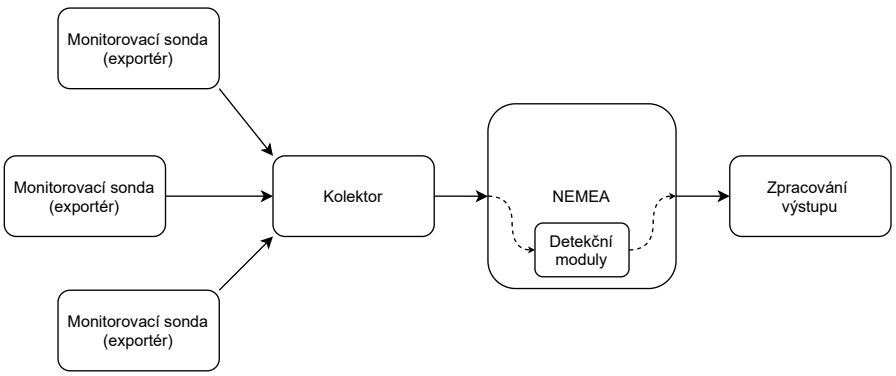
Výstupem jsou především logy s širokým spektrem extrahovaných informací užitečné např. pro SIEM systémy. Kromě získávání dat Zeek dokáže detekovat i slovníkové útoky nebo zranitelné verze softwaru na síti. Obsahuje vlastní skriptovací jazyk pro vytváření detekčních modulů napojených na generovanou frontu událostí. Skrze plně programovatelné moduly lze do analýzy zapojovat i externí zdroje např. k automatizované validaci extrahovaného SSL/TLS certifikátu.

2.2 Analýza síťových toků

Zdroj [25], který podrobně vysvětluje důvody, principy, architekturu nasazení, formáty i analýzu síťových toků, uvádí následující informace. Síťové toky vznikly jako reakce na výkonové nároky paketové analýzy a jejich monitoring by tak měl být aplikovatelný i na velkých sítích. Toky agregují zachytávanou komunikaci do statistických záznamů o jednotlivých spojeních na základě společných atributů (ip adresy, mac adresy, protokol a porty transportní vrstvy, čas...). Záznam obsahuje společné atributy spojení, které slouží také pro jednoznačnou identifikaci, a statistické informace jako např. celkový počet zaznamenaných paketů nebo objem přenesených dat. Množství exportovaných informací lze obohatit použitím rozšiřujících pluginů exportéru dle možností vybrané implementace např. ipfixprobe [26].

Díky agregaci celého spojení do krátkého záznamu je tento přístup mnohem úspornější na zdroje, což ho předurčuje k použití na sítích s vysokou propustností dat. Přirozenou nevýhodou síťových toků je absence obsahu jednotlivých paketů, čímž nesou i výrazně méně informací o potenciálně škodlivém provozu a jeho detekce nemusí být tak spolehlivá jako ve srovnání s paketovou analýzou.

O zachytávání provozu se starají síťové sondy, na kterých běží software pro exportování toků. Sondu může představovat i router s příslušným softwarem, ale většinou myslíme dedikované zařízení pro pasivní odposlech sítě. Vygenerované toky následně směřují na komponentu zvanou kolektor, kde se uchovávají a předávají detekčním systémům. Tento oddělený koncept zachytávání provozu a jeho centrální uložení umožňuje současné zapojení více sond např. pro sledování perimetru velké sítě. Schéma 2.1 ukazuje běžnou infrastrukturu monitorování sítě pomocí síťových toků s napojením na IDS systém NEMEA, který představujeme v následující sekci.



Obrázek 2.1: Infrastruktura monitorování sítě systémem NEMEA [27]

2.2.1 NEMEA Systém

NEMEA (Network Measurements Analysis) [28, 29] je modulární systém určený pro analýzu síťových toků, který poskytuje komunikační rozhraní detekčním modulům jak pro vstupy, tak výstupy. Moduly jsou nezávislé stavební bloky prováděných detekcí, ale mohou také navazovat na sebe. Pozici NEMEA systému v návaznosti na zachycení provozu a analýzu v reálném čase znázorňuje předchozí schéma 2.1.

Z pohledu operačního systému jsou moduly spouštěny jako oddělené procesy, které implementují definované komunikační rozhraní v jazyze C s pomocí připravené knihovny libtrap [30]. K dispozici je také nadstavba pro python vhodná pro snadný vývoj nových prototypů.

Kapitola 3

Analýza SSH provozu

Při analýze protokolu jsme postupovali dle specifikace RFC4251 [7], tedy postupně od transportní vrstvy výše. Protože cílem práce je vytvořit prototyp detektoru určeného pro velké sítě, zdrojem dat analýzy jsou především záznamy síťových toků místo kompletně zachyceného provozu. V některých situacích bylo pro odhalení charakteristik a chování protokolu výhodné použít nástroj Wireshark při interaktivním zkoumání komunikace spolu s ladícími výstupy SSH klienta. Důležitým zdrojem byly také standardy RFC4251 [7], RFC4252 [11], RFC4253 [10], RFC4254 [14] týkající se SSH protokolu, které samy o sobě zavádí rysy vhodné pro detekci.

3.1 Datová sada

Jako podklad pro analýzu SSH provozu jsme vytvořili menší datovou sadu se zaměřením na variabilitu autentizační vrstvy SSH protokolu. Datová sada obsahuje 175 zachycených SSH spojení, kde jsou zastoupeny vybrané kombinace použitých bezpečnostních algoritmů a způsobů ověření. Nerealizujeme všechny možné kombinace z důvodu velké vzájemné podobnosti a počtu kombinovaných parametrů. Účelem sady je pouze analýza chování protokolu. K vyhodnocení přesnosti slouží testovací datová sada reálného provozu velké sítě viz sekce 5.1. V sadě jsou zastoupeny následují kombinace parametrů spojení, druhů provozu i autentizační metod:

Výsledky autentizace

- opakované zadávání hesla
- opakované zkoušení více veřejný klíčů
- neúspěšné a nedokončené autentizace
- kombinace veřejný klíčů a hesel
- úspěšná autentizace v různém pokusu

Autentizační metody

- ověření heslem
- ověření klíčem (RSA 2048 b, RSA 4096 b, ECDSA 256 b, ECDSA 521 b, ED25519 256 b)

Časování autentizace

- automatizované přihlášení
- manuální přihlášení uživatele

Druhy SSH provozu

- interaktivní terminály
- krátké vzdálené příkazy
- přenosy dat (scp, rsync, git clone, git pull, git push)

Algoritmy výměny klíče

- curve25519-sha256
- diffie-hellman-group14-sha1
- ecdh-sha2-nistp256

Algoritmy ověření serveru

- ssh-rsa
- ssh-ed25519
- ecdsa-sha2-nistp256

Šifrovací algoritmy

- chacha20-poly1305@openssh.com
- aes256-ctr
- aes128-cbc

MAC algoritmy

- hmac-sha1
- umac-128-etm@openssh.com
- hmac-sha2-512-etm@openssh.com

Komprese

- bez komprese
- zlib komprese

3.2 Analýza transportní vrstvy

Transportní vrstva zajišťuje výměnu podporovaných bezpečnostních algoritmů a navázání šifrovaného spojení. Preference a pořadí algoritmů nejsou pro síťový monitoring až tak zajímavé, přesto může znalost zvolených algoritmů hrát svoji roli např. u detekcí pomocí thresholdu. Dlouhé seznamy převyšující maximální hodnotu MTU sítě se rozdělují do více paketů, a proto se nemůže spolehnout na jejich fixní počet. Všechny zasílané pakety SSH protokolu jsou čitelné až do začátku autentizační fáze, ale díky absenci obsahů jednotlivých paketů a možnosti konfigurace obou stran nelze přesně určit zvolené algoritmy pouze s využitím délek paketů. Z tohoto pohledu mají výhodu nástroje založené na paketové analýze, které mohou jednoduše určit zvolené algoritmy a s jejich znalostí zpřesnit detekce v pozdějších fázích celého SSH spojení.

Transportní vrstva končí ustanovením šifrovacích klíčů a zasláním potvrzovací zprávy SSH_MSG_NEWKEYS. Tato zpráva je poslední zprávou transportní vrstvy, tedy i poslední nešifrovanou zprávou a také poslední zprávou bez použití dohodnutého MAC algoritmu. Dle standardu má zpráva obsahovat pouze její identifikační kód (tj. 21 pro SSH_MSG_NEWKEYS) o velikosti 1 B. Spolu s minimální strukturou SSH zprávy, kterou jsme rozebírali v sekci 1.3.1 a na obrázku 1.3, tvoří nejmenší možný paket o délce $4 + 1 + 1 + 4 = 10$ B (délka zprávy + délka výplně + kód zprávy + min. výplň), která se výplní dorovnává na násobek 8 (tj. 16 B) resp. násobek šifrovacího bloku.

V provozu bychom měli snadno detekovat konec transportní vrstvy protokolu pouze pomocí pozice tohoto paketu s délkou 16 B. Bohužel není 100% vyloučeno, že obě strany např. v rámci optimalizace nespojí zprávu SSH_MSG_NEWKEYS s další či předchozí zprávou, což může výrazně narušit tuto detekci a bude potřeba statistické ověření na velké datové sadě z reálné sítě. Testovací datová sada ze sekce 5.1 obsahuje tento paket v 98,9 % záznamů.

3.3 Analýza SSH autentizace

Autentizační fáze pobíhá již v plně šifrovaném spojení navíc s použitím MAC. Informace o délce zprávy a délce výplně jsou rovněž šifrované. Variabilita výplně výrazně komplikuje přesné určení délky původní zprávy. Z předchozí sekce známe místo, kde končí transportní vrstva resp. začíná autentizace. Ze standardu víme, že první 2 pakety autentizační vrstvy obsahují zprávy SSH_MSG_SERVICE_REQUEST a SSH_MSG_SERVICE_ACCEPT. Samotná autentizace následuje až poté. Protokol podporuje také rezervovanou autentizační metodu „none“, která by měla sloužit ke zjištění podporovaných způsobů ověření ze strany serveru a dle experimentální pokusů ji SSH klienti (např. OpenSSH [31], PuTTY [32]) hojně využívají.

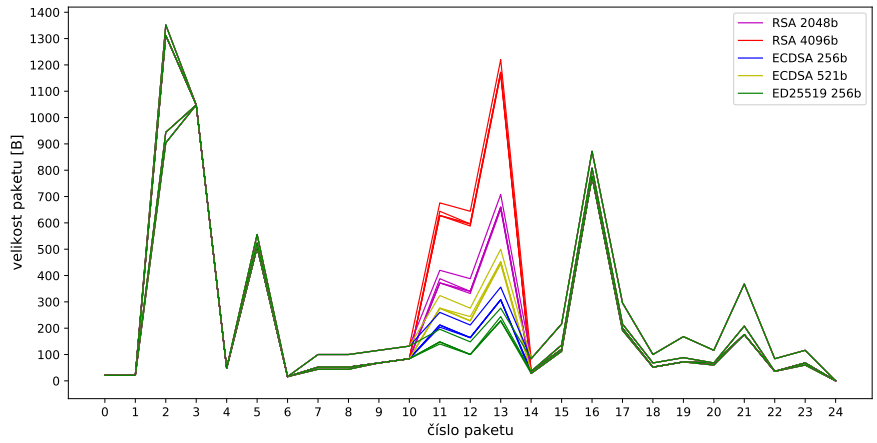
3.3.1 Autentiace veřejným klíčem

Ověření pomocí dvojce veřejného a soukromého klíče bývá výrazně specifické pokud klient využije možnost dobrovolného (před)ověření veřejného klíče na serveru. V takových případech SSH klient zasílá autentizační požadavek pouze s jeho veřejným klíčem a vynechává podpis, bez kterého nelze autentizaci úspěšně dokončit.

Server zvaliduje, zda má daný uživatel přiřazený přijatý veřejný klíč a buď rovnou klíč zamítá, nebo odpovídá zprávou SSH_MSG_USERAUTH_PK_OK, jejíž součástí je přijatý veřejný klíč. V dalším kroku nastává skutečné ověření, které z pohledu serveru nemá žádnou souvislost s předchozím pokusem a je potřeba k němu i takto přistupovat. Klient tedy opakovaně zasílá předchozí požadavek, tentokrát již s podpisem vytvořeným pomocí soukromého klíče.

Zda klient využije výše zmíněné (před)ověření veřejného klíče záleží pouze na něm a nemá to na chování protokolu žádný funkční vliv. Z pozorování chování SSH klientů a jejich ladících výpisů docházíme k závěru, že klient jednoduše využije tototo (před)ověření veřejného klíče právě tehdy, pokud má veřejnou část klíče předem k dispozici resp. nevyužije (před)ověření, pokud je např. celý klíč vč. veřejné části na disku uložen v šifrované podobě. Obecně tedy záleží na použitém formátu uložení klíče. Některé formáty pro uložení klíčů (např. PuTTY .ppk) umožňují číst veřejnou část klíče, i když je samotný soukromý klíč šifrovaný. Z chování klienta OpenSSH vyplývá, že pokud k přečtení veřejného klíče potřebuje dešifrovat i soukromý klíč, pak přeskočí krok (před)ověření a rovnou se pokusí autentizovat vytvořením podpisu.

Graf 3.1 ukazuje délky zasílaných paketů v průběhu vybraných spojení lišících se především variantou použitého klíče pro několik různých kombinací zvolených bezpečnostních algoritmů. V části autentizace (pakety 11–14, číslované od 0) vidíme typický vzorec úspěšného ověření klíčem s využitím (před)ověření veřejného klíče.



Obrázek 3.1: Autentizace pro vybrané druhy klíče a šifrovací/MAC algoritmy

Délky klíčů přirozeně musí ovlivnit velikosti paketů, nicméně relativní podobnost autentizační části spojení se zdá být téměř dokonale škálovaná délkou klíče s minimální realativní odchylkou. Tuto sekvenci spolu s potvrzovacím paketem se pokusíme detekovat.

Bez výše zmíněné fáze nepovinného ověření veřejného klíče je autentizace klíčem záležitostí pouze dvou paketů. Z pohledu velikostí paketů se pak tváří podobně jako autentizace pomocí hesla s tím rozdílem, že odchozí paket bude místo hesla obsahovat celý veřejný klíč klienta, a tím by měl převyšovat délku běžného hesla. Tento rozdíl se ovšem stírá při použití krátkých eliptických klíčů.

3.3.2 Autentizace heslem

V této sekci se zaměříme především na autentizace pomocí hesla. Ta probíhá standardně ve dvou paketech a je potvrzena zprávou SSH_MSG_USERAUTH_SUCCESS, která obsahuje opět pouze identifikační kód stejně jako v případě ustanovení šifrovacích klíčů zpráva SSH_MSG_NEWKEYS. Podstatným rozdílem je ovšem samotné šifrování a především algoritmus MAC, který délky paketů zvyšuje. Z důvodu variability těchto parametrů není bez jejich znalosti možné přesně určit velikost očekávaného paketu potvrzující úspěšnou autentizaci. Přesto by měl být nejmenší od zahájení šifrování. Tabulka 3.1 uvádí velikosti SSH_MSG_USERAUTH_SUCCESS zpráv pro vybrané šifry a MAC algoritmy rozepsané na jednotlivé části SSH paketu.

šifra **MAC** **velikost bloku**	aes256-ctr hmac-sha1 16 B	chacha20-poly1305 implicitní proudová šifra	aes256-ctr umac-128-etm 16 B	aes128-cbc hmac-sha2-512-etm 16 B
délka zprávy	4 B	4 B	4 B	4 B
délka výplně	1 B	1 B	1 B	1 B
kód zprávy	1 B	1 B	1 B	1 B
obsah zprávy	0	0	0	0
k vyplnění	6 B	2 B	2 B	2 B
výplň	10 B	6 B	14 B	14 B
délka MAC	20 B	16 B	16 B	64 B
celkem	36 B	**28 B**	36 B	84 B

Tabulka 3.1: Velikosti zpráv potvrzujících úspěšnou autentizaci

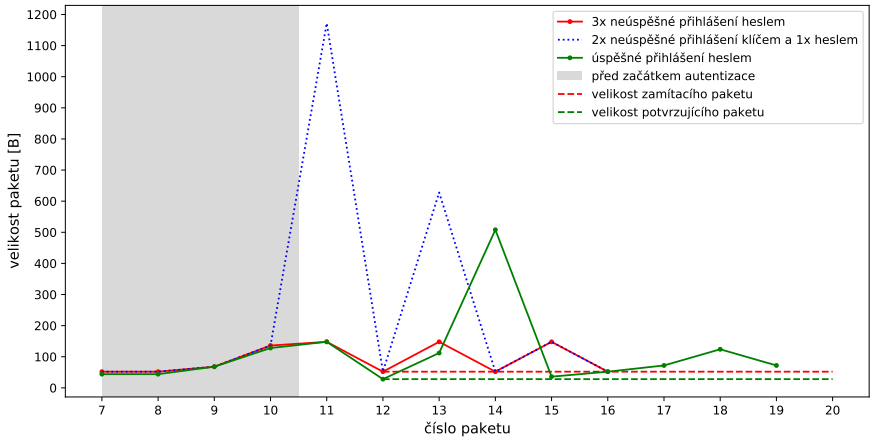
Velikosti polí délka zprávy, délka výplně a kód zprávy jsou jasně určené standardem [10]. Minimální délka výplně je 4 B s tím, že součet musí být násobek velikosti šifrovacího bloku nebo 8 B (podle toho co je větší). Obsahová část zprávy se v SSH_MSG_USERAUTH_SUCCESS nenachází vůbec, tedy má velikost 0. Délku pole MAC jednoznačně stanovuje použitý MAC algoritmus. Menší zmatení mohou působit MAC algoritmy v režimu ETM (Encrypt Than Mac), které nešifrují pole délky zprávy a MAC počítají až ze zašifrovaných dat. Díky tomu lze MAC ověřit před samotným započetím dešifrování a tím předcházet útokům postraními kanály (např. Oracle padding attack). Podobná situace nastává u SSH varianty proudové šifry *chacha20-poly1305*, která sice pole délky zprávy šifruje, ale v separátním šifrovacím proudu [33], čímž se z hlediska velikostí chová stejně jako v ETM režimu.

Vidíme, že celkovou velikost výrazně ovlivňuje právě použitý MAC algoritmus, který z dat síťového toků neznáme. Zde mě zaujala hodnota 28 B u šifry *chacha20-poly1305*, která je navíc nejmenší možná. Protože všechny ostatní zprávy po zahájení šifrování by měly být větší než zpráva potvrzující úspěšné přihlášení, měla by tato velikost paketu signalizovat úspěšné přihlášení.

3.3.3 Neúspěšná autentizace

Při neúspěšném pokusu o autentizaci libovolnou metodou server odpovídá jednotně zprávou SSH_MSG_USERAUTH_FAILURE, která se na rozdíl od potvrzení úspěšné autentizace liší tím, že obsahuje navíc seznam metod, kterými může klient dále pokračovat v autentizaci. Oproti zprávě SSH_MSG_USERAUTH_SUCCESS obsahující pouze 1B identifikační kód zprávy by měl neúspěšný pokus vytvářet větší odpovědi. Tento rozdíl ve velikostech se pokusíme detekovat. Jak jsme ukázali v předchozí sekci, do velikostí paketů se kromě délky samotného obsahu promítají i další vlivy jako velikost bloku šifry, MAC algoritmus, komprese. Tyto faktory mohou pakety ovlivnit až o desítky bajtů, což dokonce převyšuje detekovaný rozdíl.

Následující graf 3.2 zachycuje rozdíly úspěšného (zelená) a opakovaně neúspěšného (červená) pokusu o přihlášení heslem. Modrá znázorňuje neúspěšné ověření heslem se dvěma předchozími ověřeními veřejného klíče. První pokusy o přihlášení se nachází v paketu č. 11 a jeho odpověď v paketu 12. Horizontálně jsou zvýrazněny velikosti odpovědi úspěšného resp. neúspěšného ověření. Pro tuto konkrétní kombinaci je jejich rozdíl 24 B (52 B − 28 B), což vzhledem k vysoké variabilitě délek paketů není pro přesnou detekci tak znatelné.



Obrázek 3.2: Opakovaně neúspěšné pokusy o autentizaci

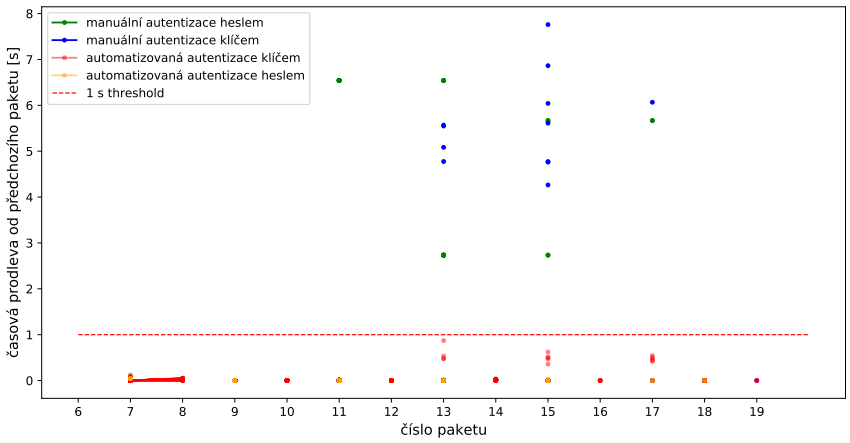
Povšimněme si, že opakované zamítání vrací v rámci stejného spojení stále konstantní odpovědi. Naopak u úspěšného spojení očekáváme větší variabilitu následujících odpovědí serveru. V kombinaci s pravidelným střídáním směrů komunikace se pokusíme rozpoznat tento vzorec.

3.4 Analýza časových rozestupů

Cílem této části je odhalení časových charakteristik SSH spojení především v části autentizace. Výrazné časové prodlevy mezi pakety můžeme rozdělit podle jejich příčiny na technické a uživatelské. Mezi technické zařadíme veškeré implementační důsledky, zpoždění sítě apod. Uživatelské prodlevy naopak vznikají z pochopitelných důvodů a představují zpoždění na straně uživatele oproti rychlosti sítě. Informace zda přihlášení provádí člověk nebo automat může být nápomocná např. při detekci slovníkových útoků, kde se předpokládá využití automatizovaných nástrojů. Časové prodlevy jsme analyzovali pro kombinace metod přihlášení (heslo, klíč, neúspěch) a časové průběhy přihlášení (automatizované, manuální) odděleně v každém směru.

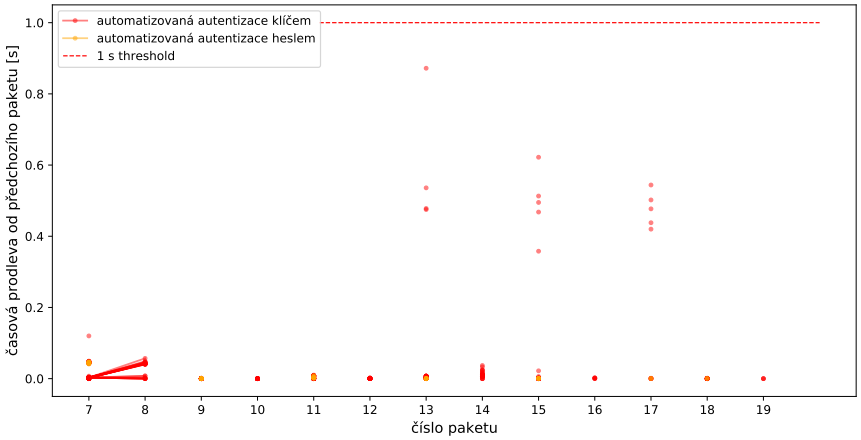
3.4.1 Směr provozu od klienta na server

V odchozím směru komunikace od klienta na server očekáváme výrazné uživatelské časové prodlevy v situacích, kde se čeká na vstup uživatele např. přihlášení heslem. Stejné prodlevy pozorujeme také u ověření klíčem v případech, kdy je privátní klíč uložen v šifrované podobě a uživatel ho v průběhu autentizace krátkodobě dešifruje. Zde velmi záleží na konkrétním spuštění SSH klienta, kterému buď lze předat všechny informace předem a spojení nebude vykazovat výrazné časové rozdíly mezi pakety, nebo se SSH klient bude uživatele v průběhu spojení doptávat na potřebné informace. Detekce automatizované a manuální autentizace by měla být realizovatelná pouze na základě existence výrazného časového skoku pomocí thresholdu.



Obrázek 3.3: Časové prodlevy autentizace v odchozím směru

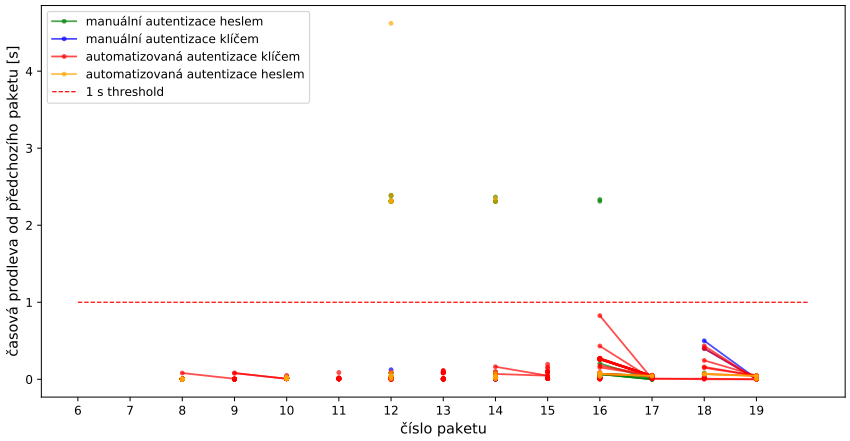
Bližší analýza automatizované autentizace odhalila také drobnější časové prodlevy u některých automatizovaných přihlášení pomocí klíče v době výpočtu digitálního podpisu v rozsahu hodnot 0,358–0,872 s. Hodnoty nepřesahují 1 s jako v případech manuální autentizace, a tím nebudou mít vliv na rozpoznání prodlevy zadávaného hesla pomocí 1s thresholdu.



Obrázek 3.4: Časové prodlevy automatizované autentizace v odchozím směru

3.4.2 Směr provozu ze serveru ke klientovi

Mezi příchozími pakety ze strany serveru očekáváme časové prodlevy v rámci běžné odezvy sítě bez větších výkyvů. Během analýzy se projevily více než 2s odezvy, které jsme blíže identifikovaly jako zprávy SSH_MSG_USERAUTH_FAILURE při autentizaci heslem. U správných hesel nebo u autentizace klíčem (i neúspěšné) tyto prodlevy nenastávají z čehož vyplývá, že se jedná o záměrné chování serveru, které se může lišit napříč implementacemi či konfiguracemi. Z tohoto důvodu můžeme informaci využít pouze ke zpřesnění detekce opakovaně zadávaného či neúspěšného hesla, ale není vhodné se na ni plně spoléhat.



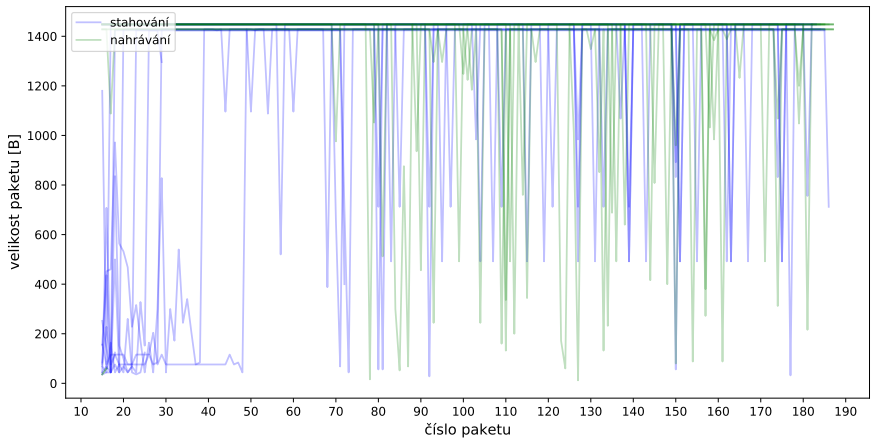
Obrázek 3.5: Časové prodlevy autentizace v příchozím směru

3.5 Analýza druhu obsahu SSH spojení

Následující sekce si klade za cíl analyzovat různé případy použití SSH protokolu a vybrat jejich typické charakteristiky pro následnou detekci. Předpokladem určení druhu spojení je úspěšně navázané spojení vč. autentizace. U výběru charakteristik se pokusíme zohlednit také přibližné výkonové nároky a množství potřebných informací k provedení detekce v produkčním prostředí. Například nebude reálné zachytávat délky všech paketů spojení, ale využít statistické hodnoty o jejich počtu už ano.

3.5.1 Analýza datových přenosů

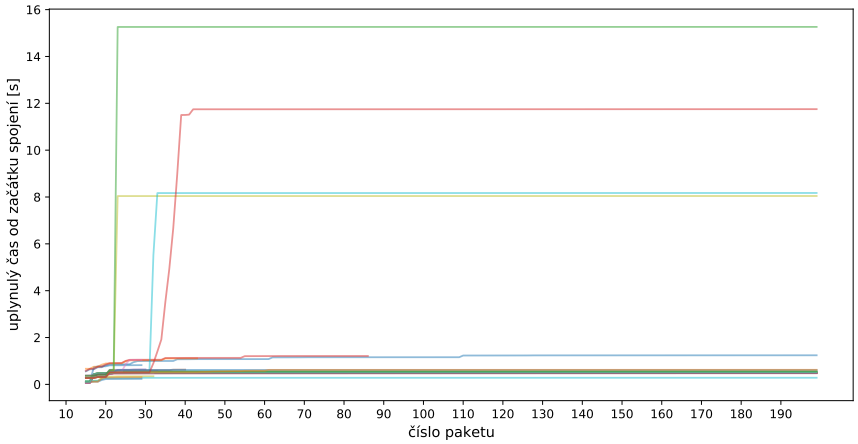
Přirozeným předpokladem přenosu souborů je zvýšený objem dat právě v jednom směru. Bohužel detekce založená pouze na celkovém objemu dat může být značně zavádějící např. u dlouhotrvajících spojení s velkým množstvím malých paketů. Podívejme se do grafu 3.6 na přesné chování velikostí paketů ve směru přenosu dat pro nástroje využívající SSH protokol (scp, rsync, git). Až na drobné výkyvy vidíme dlouhé posloupnosti paketů o konstantní velikosti dosahující hodnoty MTU sítě. Tyto krátké výkyvy mají v naprosté většině délku jednoho paketu a identifikovali jsme je jako přenos posledního fragmentu souboru především u nástroje scp.



Obrázek 3.6: Průběh velikostí zasílaných paketů při datových přenosech

Detekce opakujících se sekvencí paketů konstantní velikosti by byla možná, nicméně by kladla poměrně vysoké nároky na výpočetní zdroje a na množství zachytávaných informací potřebných k analýze. Jako alternativní charakteristiku jsme použili statistický plugin PHISTS nástroje ipfixprobe, jehož výstupem je histogramové rozdělení paketů podle jejich délek a časů do předem určených intervalů.

Z pohledu časových rozestupů během přenosu dat neočekáváme žádné výrazné výkyvy, jakmile je přenos zahájen. Tento fakt dokládá i graf 3.7. Ovšem pozice zahájení přenosu se může drobně lišit.

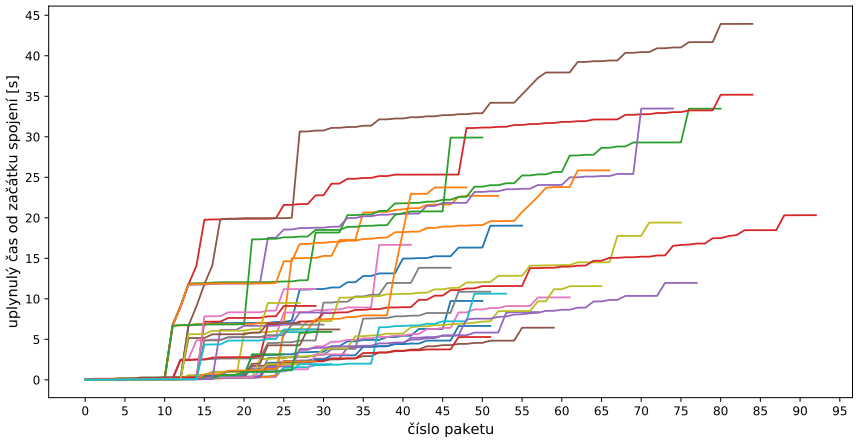


Obrázek 3.7: Časový průběh datových přenosů

3.5.2 Vzdálený terminál

Vzdálený terminál je bezesporu jeden z druhů provozu, které by se měly vyznačovat svou interaktivitou. Z fungování SSH klientů víme, že každý zadaný znak na klávesnici se SSH protokolem přenáší jako samostatná zpráva a teprve až odpověď serveru se uživateli zobrazí na obrazovce. Vzhledem k dnešním rychlostem sítí by se tak psaní na klávesnici mělo projevit v časových prodlevách mezi jednotlivými pakety ve směru od klienta. Tuto charakteristiku nebudou naopak sdílet jednorázově spuštěné příkazy bez udržovaného terminálu, ani datové přenosy. Protože zprávy vzdáleného terminálu nejčastěji přenáší pouze jeden zadaný znak, velikosti paketů by se v rámci SSH protokolu měly pohybovat mezi nejmenšími s přihlédnutím k ustanovenému MAC algoritmu.

V opačném směru se budou míchat dva druhy příchozích paketů. První druh bude povětšinou obsahovat stisknutý znak k následnému vypsaní do příkazové řádky navázaného terminálu, a proto budou jejich velikosti paketů stejné nebo velice podobné velikostem paketů ve směru od klienta. Navíc by neměly vykazovat jakékoli nadprůměrné zpoždění. Dalším druhem budou samotné výstupy zadávaných příkazů, které se mohou lišit jak velikostí, tak časem. Graf 3.8 ilustruje časový vývoj komunikace vzdálených terminálů vzhledem k prvnímu paketu daného spojení. Na první pohled vidíme markantní rozdíl oproti dříve zkoumaným datovým přenosům. Bohužel kvůli nutné interaktivitě při vytváření datové sady je tato kategorie výrazně omezena pouze na krátká sezení s jednotkami zadaných příkazů a v rozsahu maximálně několika desítek vteřin.



Obrázek 3.8: Časový průběh spojení vzdáleného terminálu

Nutnost informací o velkém počtu prvních paketů opět směřuje na využití histogramu z PHISTS pluginu. Bohužel v době tvorby testovací datové sady plugin PHISTS nerozlišoval volbu *includezeros* a do histogramů automaticky zahrnoval všechny pakety. Zahrnuté potvrzovací TCP ACK pakety tím výrazně ovlivnily rozložení histogramu, a proto tuto detekci nebylo možné realizovat a přenecháváme ji možné budoucí realizaci. V souvislosti s tím jsme také identifikovali drobnou chybu v implementaci PHISTS pluginu, jejímž vlivem docházelo k posunutí indexu histogramových intervalů o 1 vpravo s výjimkou prvního intervalu. Poslední interval tak obsahoval součet dvou posledních intervalů, zatímco druhý interval nabýval vždy hodnoty 0. Na základě hlášení byla chyba bez odkladu opravena spolu s přidáním volby *includezeros*.

Kapitola 4

Návrh a implementace prototypu

Celá práce je koncipovaná s ohledem pro využití na sítích o vysokých rychlostech, kde z výkonových důvodů není reálné ukládat kompletní zachytávaný provoz pro následnou paketovou analýzu. Prototyp byl vytvořen v jazyce python jako detekční modul pro systém NEMEA představený v sekci 2.2.1. Pro exportování síťových toků jsme využili nástroj ipfixprobe [26], který dokáže vytvářet a exportovat toky jak ze souboru ve formátu PCAP, tak ze síťového rozhraní v reálném čase. Získané toky se následně předávají na komunikační rozhraní systému NEMEA, kde se buď přeposílají navazujícím modulům nebo se uloží do souboru pro pozdější analýzy či uskladnění.

Předpokládaným vstupem prototypu jsou kromě základních informací síťových toků také rozšířené toky o následující pluginy nástroje ipfixprobe, které je obohacují o podstatné informace.

Základní tok obsahuje identifikační informace spojení, především IP a MAC adresy, počáteční a koncové časy, použité porty transportní vrstvy, ale také celkové počty paketů a množství přenesených dat odděleně v obou směrech. Z hlediska hlubší analýzy SSH spojení ovšem neobsahuje potřebné informace.

IDPContent přidává k základnímu toku obsah prvního zachyceného paketu z obou směrů. Tuto informaci použijeme pouze na spolehlivé odlišení SSH provozu od ostatní komunikace. V produkčním nasazení je možné tuto část nahradit např. předřazeným klasifikátorem, který řídí spouštění detektorů na základě rozpoznaného druhu provozu.

PSTATS obohacuje toky o velikosti, směry, časy a TCP příznaky prvních N paketů celého spojení. Protože většina nalezených charakteristik SSH protokolu se opírá právě o tato data, jedná se o klíčový plugin pro navrhovaný prototyp.

Pro analýzu i vyhodnocení prototypu jsme použili PSTATS plugin bez možnosti *includezeros*, čímž dochází k vynechání paketů s nulovou délkou obsahu TCP segmentu (tj. paketů bez obsahu síťové aplikační vrstvy ISO/OSI modelu jako TCP SYN/ACK/FIN). Exportované velikosti navíc reprezentují pouze obsahovou část TCP segmentu, tedy jsou očištěné od hlaviček předchozích síťových vrstev a přesně odpovídají velikostem zpráv SSH protokolu.

Pro účely analýzy jsme zvýšili počet exportovaných prvních paketů z výchozí hodnoty $N = 30$ na $N = 200$ paketů (dohromady v obou směrech). Spolu s absencí volby *includezeros* jsme získali podstatnou část spojení, ale v reálném nasazení nepředpokládáme takové množství dostupných dat.

PHISTS předpočítává histogramové rozdělení velikostí jednotlivých paketů a mezipaketových časových mezer odděleně v obou směrech. Rozdělení je škálované v logaritmickém měřítku do osmi definovaných intervalů viz následující tabulka.

č. intervalu	velikost paketu	délka časové prodlevy
1	< 16 B	< 16 ms
2	16–31 B	16–31 ms
3	32–63 B	32–63 ms
4	64–127 B	64–127 ms
5	128–255 B	128–255 ms
6	256–511 B	256–511 ms
7	512–1 023 B	512–1 023 ms
8	$> 1\,023$ B	$> 1\,023$ ms

Tabulka 4.1: Rozdělení histogramových intervalů pluginu PHISTS [26]

Tento plugin využijeme v detekci datových přenosů jako efektivní způsob pohledu na poměr velikostí paketů napříč celým spojením. Bohužel v době implementace a zachytávání datové sady nebyla odlišena volba *includezeros*, takže histogramy zahrnují např. i potvrzovací TCP ACK pakety, s čímž musíme v návrhu počítat.

Zařazením prototyp spadá mezi analyzátory síťových toků a detekce zakládá na signaturách vycházejících z předchozí analýzy a SSH standardu. V četných situacích se využívají nastavitelné thresholdy, které posouvají hladinu detekce rozpoznávané charakteristiky. Aktuálně jsou jejich hodnoty odhadnuté experimentálním způsobem z datové sady, ale předpokládáme i vhodnější způsoby nastavení, kterým se ovšem tato práce dále nevěnuje. Pro nastínění „chytřejšího“ konceptu uvádíme myšlenku individuálního určení threshold parametrů např. s využitím strojového učení, které by odhadovalo použité šifrovací/MAC algoritmy, což by umožnilo lépe předpovídat velikosti některých typů SSH zpráv, a tím zpřesnit úspěšnost detekce popř. potlačit falešnou pozitivitu.

Prototyp se skládá z detekčních funkcí a řídící logiky, která zajišťuje jejich volání v logické návaznosti a omezuje detekce odporujících si situací (např. detekce datového přenosu po rozpoznání neúspěšné autentizace). Testování principu detekčních funkcí probíhalo průběžně na vytvořené datové sadě s manuálním ověřením výstupů. Procesu vyhodnocení úspěšnosti detekcí se věnuje kapitola 5 včetně vytvoření testovací datové sady ze zachyceného provozu reálné sítě.

Následující sekce popisují jednotlivé detekční funkce. Celkový pohled na prototyp a řídící logiku uvádíme až ke konci kapitoly.

4.1 Detekce způsobu autentizace

Použitý způsob autentizace nám dává určitý vhled do provozu sledované sítě. Z obecného pohledu tušíme, že různé situace využívající SSH protokol preferují některý ze způsobů ověření identity a tím dokonce mohou částečně napovídat druh SSH komunikace, a proto nám tato informace přijde užitečná. Korelací mezi použitou autentizační metodou a druhem SSH provozu se tato práce již nezabývá, nicméně bych zde nastínil několik konkrétních aplikací SSH protokolu s typickým způsobem autentizace.

Git repozitáře využívají SSH jako jeden ze způsobů komunikace a typicky se klient autentizuje pomocí klíče, tedy by nemělo docházet k použití hesel.

Specializované proprietární síťové prvky nemusí vždy podporovat jiné ověřovací metody než heslo, ať už z důvodu chybějící implementace nebo nedostatku paměti pro uložení veřejných klíčů.

Firemní prostředí, kde se SSH využívá např. ke správě serverů, předpokládá řízené přidělování přístupů a často bude zavedena centralizovaná distribuce veřejných klíčů a ověření heslem může být dokonce vnímáno jako bezpečnostní riziko.

4.1.1 Rozpoznání klíče s (před)ověřením

Prerekvizitou úspěšné autentizace klíčem je správné pořadí směrů paketů. Funkce sekvenčně prochází informace o paketech z pluginu PSTATS omezené na odhadnutý rozsah autentizační fáze a hledá v nich podsekvence délky 4, které začínají u klienta a naplňují požadované střídání směrů. Požadavek střídání směrů plní funkci prvotního filtru, který vybírá relevantní části spojení pro samotnou detekci. Při testování se podmínka ukázala jako užitečná nejen po výkonové stránce, ale i pro situace s neznámým či anomálním průběhem. V takových případech dokonce snižuje falešně pozitivní detekce.

Vzhledem k variabilním délkám klíčů zde není požadavek na jejich minimální délku. Místo toho detekce vychází z principu nutného rozdílu velikostí paketů podle jejich obsahu daného standardem. Přesto funkce využívá dva následující thresholdy:

AUTH_SUCCESS_PCKT s výchozí hodnotou 50 B představuje maximální akceptovanou velikost paketu potvrzující úspěšnou autentizaci. Jedná se o pomyslnou hranici mezi zprávami USERAUTH_SUCCESS a USERAUTH_FAILURE. Právě tato hodnota bude výrazně ovlivněna použitým MAC algoritmem a pro některé z nich nemusí výchozí hodnota dostačovat, což může vést k chybné detekci. Na druhou stranu její nadměrné zvýšení vede k falešné pozitivitě vlivem rozpoznání neúspěšného ověření jako úspěšného.

AUTH_KEY_COEF určuje hodnotu maximálního předpokládaného poklesu velikosti zprávy USERAUTH_PK_OK, která potvrzuje existenci veřejného klíče na serveru, oproti bezprostředně předcházející zprávě s žádostí o toto (před)ověření. Koeficient je určovaný procentuálně s výchozí hodnotou 0,65 (tj. 65 %).

Následující tabulka ukazuje aplikované podmínky pro jednotlivé pakety autentizační sekvence s příkladem reálných hodnot z datové sady.

č. paketu	předpokládaný obsah	podmínka	příklad
p1	USERAUTH_REQUEST	—	628 B
p2	USERAUTH_PK_OK	$< p1$ $> p1 \cdot AUTH\_KEY\_COEF$	588 B
p3	USERAUTH_REQUEST	$> 2 \cdot p1 \cdot AUTH\_KEY\_COEF$	1 164 B
p4	USERAUTH_SUCCESS	$< AUTH\_SUCCESS\_PCKT$	28 B

Tabulka 4.2: Aplikované podmínky pro rozpoznání autentizace klíčem

Pro ilustraci uvádíme ukázku implementace celé této detekční funkce s podrobnými komentáři přímo v kódu. Ostatní detekční funkce zakládají většinou na stejném principu rozpoznání signatury ve vybrané části spojení a jsou tedy implementačně podobné. Z tohoto důvodu se zaměřujeme spíše na aplikované podmínky detekce než ukázky celých funkcí, které jsou dostupné v příloze B s jejich podrobným komentářem.

```
def detect_key(data, vals):
    """
    Function detects key authentication with public key precheck based on directions and packet
    lenghts pattern.

    data: input flows variable
    vals: precalculated typical values
    """

    #skip 3 packets from authentication phase beginning: packet 16B, 2 SSH_MSG_SERVICE packets
    #repeat until the threshold or flow lenght, successful finding will also terminate this loop
    for i~in range(vals["auth_start"] + 3, \
        min(len(data['pstats'].PPI_PKT_DIRECTIONS) - AUTH_KEY_DIR_PATTERN_LEN, AUTH_END_THRESHOLD)):

        #find directions pattern match required for the key authentication (->, <-, ->, <-)
        #and if so, verify detection packet size conditions
        if data['pstats'].PPI_PKT_DIRECTIONS[i:i+AUTH_KEY_DIR_PATTERN_LEN] == AUTH_KEY_DIR_PATTERN and \
           data['pstats'].PPI_PKT_LENGTHS[i+1] > data['pstats'].PPI_PKT_LENGTHS[i] * AUTH_KEY_COEF and \
           data['pstats'].PPI_PKT_LENGTHS[i+1] < data['pstats'].PPI_PKT_LENGTHS[i] and \
           data['pstats'].PPI_PKT_LENGTHS[i+2] > 2 * data['pstats'].PPI_PKT_LENGTHS[i] * AUTH_KEY_COEF and \
           data['pstats'].PPI_PKT_LENGTHS[i+3] < AUTH_SUCCESS_PCKT:

            return ResultAuthMethod.key

    return ResultAuthMethod.unknown
```

Ukázka kódu 4.1: Implementace detekční funkce pro rozpoznání ověření klíčem

4.1.2 Rozpoznání hesla a klíče bez (před)ověření

Jedinou charakteristikou úspěšného ověření pomocí hesla, o které z analýzy víme, je potvrzovací zpráva malé velikosti bezprostředně za zaslaným heslem. Stejné chování vykazuje také autentizace klíčem bez (před)ověření veřejného klíče. Jediným rozdílem těchto dvou situací je velikost paketu s heslem či klíčem. Protože tato signatura není příliš výrazná, její detekci použijeme až po ostatních detekčních funkcích, abychom předešli falešně pozitivním detekcím. Spouštění detekcí řídí logika prototypu.

Implementace samotné detekce se velmi podobá předchozí detekci klíče. Funkce kontroluje směry paketů, omezuje se na autentizační část spojení a validuje podmínky kladené na velikosti obou paketů. Rozhodnutí mezi heslem a klíčem určuje nastavitelný threshold AUTH_KEY_MIN s výchozí hodnotou 256 B, což by odpovídalo

přibližně heslu o délce 130 znaků při šifrování algoritmem *chacha20-poly1305*. Threshold AUTH_PASS_MIN jako minimální požadovaná délka autentizačního požadavku pomocí hesla slouží také pro redukci falešně pozitivních detekcí na anomálně zachycených spojeních, přičemž jeho výchozí hodnotu jsme nastavili na 80 B, což převyšuje minimální získanou hodnotu u pokusu ověření heslem nulové délky při šifrování algoritmem *chacha20-poly1305*, který je navíc poměrně úsporný.

č. paketu	**předpokládaný obsah**	**podmínka**	**příklad**
p1 - klíč	USERAUTH_REQUEST	$> AUTH\_KEY\_MIN$	660 B
p1 - heslo		$> AUTH\_PASS\_MIN$	148 B
p2	USERAUTH_SUCCESS	$< AUTH\_SUCCESS\_PCKT$	36 B

Tabulka 4.3: Aplikované podmínky pro rozpoznání autentizace heslem

4.2 Detekce úspěšnosti autentizace

Neúspěšně autentizované SSH spojení definujeme jako takové spojení, ve kterém nedošlo k úspěšnému ověření v žádném pokusu. Na první pohled tato definice jistě nijak nepřekvapí, ale musíme zdůraznit, že zahrnuje i další situace krom odmítnutých hesel či klíčů. V první řadě můžeme určit natolik krátká spojení, u kterých nemohlo dojít k žádné autentizaci pouze na základě minimální potřebné délky pro autentizaci. V reálném provozu by sem spadaly podrobnější scany síťových služeb, které jsou více než běžné. Ale také některé možné situace tranzientního provozu (tj. provoz kde zdroj ani cíl neleží v monitorované síti a ta slouží jen jako prostředník), kde došlo k zachycení pouze začátku spojení. Později samozřejmě mohlo dojít k úspěšné autentizaci, ovšem bez zachycených dat jsou taková spojení mimo rozpoznávací hladinu detektoru.

Z druhého podhledu také jednoznačně odlišíme některé úspěšné autentizace podle přítomnosti paketu o velikosti 28 B ve fázi po ustanovení šifrovacích klíčů. Tento předpoklad vychází z minimální možné velikosti paketu a je legitimní, protože všechny ostatní zprávy (kromě SSH_MSG_USERAUTH_SUCCESS) dosahují větší velikosti než 28 B. Při analýze jsme identifikovali pouze šifru *chacha20-poly1305*, která používá potvrzení této velikosti. Bohužel pro ostatní krypto/MAC systémy je komplikované určit správnou hodnotu velikosti potvrzovací zprávy bez znalosti použitého systému a její nadhodnocení by mohlo vést k nesprávným detekcím. Z tohoto důvodu ponecháváme situace ostatních šifer dalšímu výzkumu popř. upřesnění pomocí určení použitého krypto/MAC systému.

Podobným principem jako u krátkých spojení můžeme s vysokou pravděpodobností předpokládat dlouhá spojení za úspěšně autentizovaná. Tuto detekci implementujeme pomocí thresholdu AUTH_PROB_OK_MIN s výchozí hodnotou 30 paketů, kde delší spojení již považujeme za autentizovaná. Počet paketů zahrnuje pouze SSH zprávy, a tím by neměl započítávat např. TCP ACK pakety. Určená výchozí hodnota vychází z minimální a běžně pozorované pozice začátku prvního autentizačního pokusu mezi 8–13 pakety. Pro falešně pozitivní detekci by znamenalo, že uživatel vyzkoušel 8–11 ověřovacích pokusů ve stejném SSH spojení.

4.2.1 Rozpoznání opakovaných pokusů o přihlášení

Analýza neúspěšné autentizace odhalila konstantní velikosti odpovědí serveru při opakovaných pokusech o autentizaci. S přihlédnutím k možným anomáliím prototyp implementuje detekci tohoto vzorce pomocí nejvyšší resp. nejnižší odchylky od průměrné velikosti odpovědi serveru v autentizační části spojení, kterou ukončuje paket o velikosti $\leq$ AUTH_SUCCESS_PCKT nebo samotný konec spojení. Aktuální hodnota akceptované odchylky pro pozitivní detekci (tj. rozpoznány opakované pokusy o autentizaci) je nastavena relativně na 20 % průměrné velikosti odpovědi serveru ve zmíněné části.

Předpokládaným důsledkem neúspěšné autentizace je brzké ukončení spojení. Z tohoto důvodu funkce kontroluje zbývající délku spojení za rozpoznanou sekvencí opakovaných pokusů. Maximální počet akceptovaných paketů za rozpoznanou sekvencí pro detekci neúspěšného ověření určuje threshold AUTH_FAIL_POST_MAX s výchozí hodnotou 3 pakety a plní funkci pružného konce této detekce, kde strany mohou legitimně ukončit spojení zasláním ukončující zprávy SSH_MSG_DISCONNECT, aniž by to ovlivnilo detekci.

V případech úspěšně rozpoznaného opakovaní autentizačních pokusů a nenaplnění požadavku malé zbývající délky musíme pragmaticky hodnotit spojení jako autentizované. V ostatních situací, které nenaplňují dostatečnou zbývající délku a zároveň převyšují rozpoznávanou odchylku opakovaní autentizačních pokusů, hodnotíme spojení z hlediska úspěšnosti autentizace jako neznámé.

4.3 Detekce automatizované autentizace

Automatizace přihlášení sama o sobě není ukazatelem jakéhokoli bezpečnostního problému. V případě ověření pomocí klíče lze správný *key management* jen doporučit. Na druhou stranu případný útok hrubou silou si lze jen těžko představit pouze s manuálním zadáváním hesel bez jakékoli automatizace. A proto je tato časová charakteristika pro nás zajímavá a může přispět např. k rozpoznání útoku nebo i k validaci bezpečnostních politik monitorované sítě.

Implementace detekce manuálního přihlášení funguje jednoduše na principu nalezení významné časové prodlevy ve směru od klienta na server. Kontrola směru paketů je zde nezbytná, neboť server může uměle zpožďovat pakety, jak ukazuje analýza 3.4.2 u neúspěšných pokusů o přihlášení heslem, což by vedlo k výrazné chybovosti detekce. Threshold detekce významné časové prodlevy (HUMAN_AUTH_MIN_DELAY) jsme určili na základě analýzy na 1 s, což považujeme za dostatečně odlišující pro rozdíly mezi odezvou sítě a zadáním byť jen krátkého hesla.

4.4 Detekce datových přenosů

Implementace detekce přenosů dat používá výše zmíněný plugin PHISTS resp. jeho histogramy velikostí paketů. Analýza ukazuje majoritní podíl paketů dosahujících téměř MTU sítě, které spadají do posledního intervalu o velikostech nad 1023 B. Protože jsou objemy přenesených dat velmi variabilní a s tím i počty paketů, musíme hranici detekce nastavit relativně k celkovému počtu paketů. Z histogramu jednoduše spočítáme podíl posledního intervalu vůči celkovému počtu paketů pro oba směry odděleně. Experimentálně jsme určily threshold TRANSFER_TRESHOLD na 70 %, což považujeme za vhodné i vzhledem k tomu, že histogramy obsahují také začátek spojení s typicky menšími pakety. Díky oddělení směrů paketů automaticky získáváme informaci o směru přenosu dat (nahrávání, stahování), která může být rovněž užitečná.

I když použitá PHISTS data zahrnují všechny pakety vč. TCP ACK, detekci datových přenosů to neovlivní, protože tyto pakety budou generované příjímající stranou a funkce rozpoznává přenos podle strany odesílající.

```
def detect_traffic_type(data):
    """
    Function detects significant data traffic in effective way based on phists plugin

    data: input flows variable
    """

    #get the most numerous phists bin and calculate it's ratio to the total packets count
    #total packets count is calculated from phists, because of histogram uint16 limit,
    # if flow information would be used, detection could be manupulated by really long connections
    src_size_major = data['phists'].S_PHISTS_SIZES.index(max(data['phists'].S_PHISTS_SIZES))
    src_size_perc = data['phists'].S_PHISTS_SIZES[src_size_major]/sum(data['phists'].S_PHISTS_SIZES)
    dst_size_major = data['phists'].D_PHISTS_SIZES.index(max(data['phists'].D_PHISTS_SIZES))
    dst_size_perc = data['phists'].D_PHISTS_SIZES[dst_size_major]/sum(data['phists'].D_PHISTS_SIZES)
    if src_size_perc > TRANSFER_TRESHOLD and src_size_major > 6:
        return ResultTrafficType.upload

    elif dst_size_perc > TRANSFER_TRESHOLD and dst_size_major > 6:
        return ResultTrafficType.download

    else:
        pass

    return ResultTrafficType.other
```

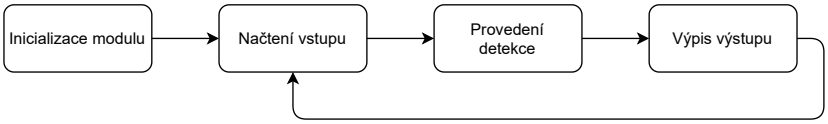
Ukázka kódu 4.2: Implementace rozpoznání datových přenosů

4.5 Struktura prototypu

Prvním úkolem NEMEA modulu je inicializace komunikačního rozhraní tj. specifikace počtu i formátu očekávaných vstupů a poskytovaných výstupů. Prototyp očekává tři vstupy z pluginů IDPContent, PSTATS a PHISTS. Pro účely této práce prototyp vypisuje výsledky pouze na standardní výstup, místo předání komunikačnímu rozhraní NEMEA. Modul řídí jednoduchá nekonečná smyčka viz obrázek 4.1, která opakuje načtení vstupu, provedení detekce a zpracování výstupu.

```
import pytrap
import sys

#Set module input/outputs parameters
trap = pytrap.TrapCtx()
trap.init(sys.argv, 3, 0)

#Prepare variable data for incomming flows
data = {}

#Set required inputs field list
ctxs = {"idpcontent": "ipaddr DST_IP,ipaddr SRC_IP, ...zkráceno...",
        "pstats": "ipaddr DST_IP,ipaddr SRC_IP, ...zkráceno...",
        "phists": "ipaddr DST_IP,ipaddr SRC_IP, ...zkráceno..."}

#Define required input fields
for i, (k, v) in enumerate(ctxs.items()):
    trap.setRequiredFmt(i, pytrap.FMT_UNIREC, v)
    data[k] = pytrap.UnirecTemplate(v)
```

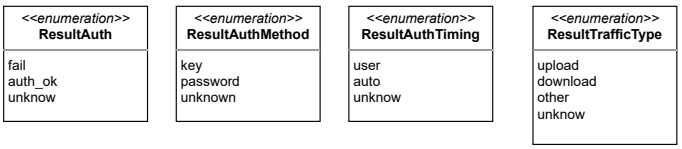
Ukázka kódu 4.3: Definice vstupů a výstupů komunikačního rozhraní modulu



Obrázek 4.1: Hlavní smyčka detektoru

4.5.1 Detekční procedura

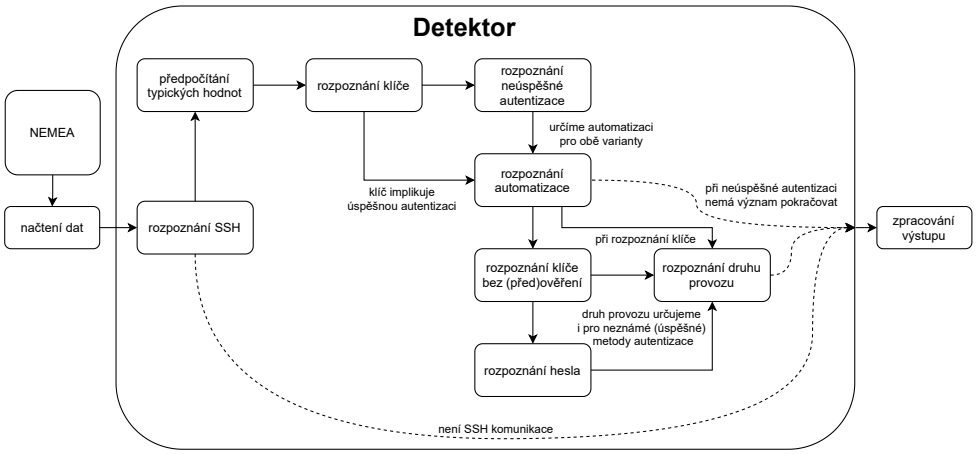
Před zahájením samotných detekcí ověřujeme zda se jedná o validní SSH spojení (tj. zachycené od jeho začátku). SSH spojení spolehlivě rozpoznáme podle obsahu prvního paketu, který nutně začíná řetězcem „SSH“. Pro jiná spojení nemá smysl implementované detekce provádět. Dále předpočítáváme typické hodnoty (např. pozici paketu velikosti 16 B), které se uplatňují ve více detekcích. K uchování výsledků jednotlivých detekčních funkcí definujeme čtyři výčtové typy (Enum třídy) odděleně pro rozpoznávané charakteristiky: úspěšnost autentizace, autentizační metoda, časování autentizace a druh SSH provozu.



Obrázek 4.2: Výčtové třídy pro uchování výsledků

Detekční procedura, funkce *do_detection()*, volá v logických návaznostech výše popsané detekční funkce a uchovává jejich výsledky. Návaznost prováděných detekčních funkcí vystihuje schéma 4.3 a je vytvořena za účelem omezení nerealizovatelných situací či nemožných kombinací rozpoznávaných charakteristik spojení jako např:

- určování druhu provozu při neúspěšné autentizaci
- rozpoznávání ověření heslem po úspěšné detekci klíče



Obrázek 4.3: Provázání detekčních funkcí modulu

Kapitola 5

Testování a vyhodnocení

Základní funkčnost navržených detekčních funkcí jsme testovali v průběhu celého vývoje na datové sadě z analytické části práce. Tvorbu testovací datové sady z reálné sítě pro vyhodnocení úspěšnosti detektoru popisuje následující sekce spolu s jejím filtrováním a přibližnou anotací. Závěrem kapitoly uvádíme zjištěné nedostatky a prostor pro navazující práci.

5.1 Testovací datová sada

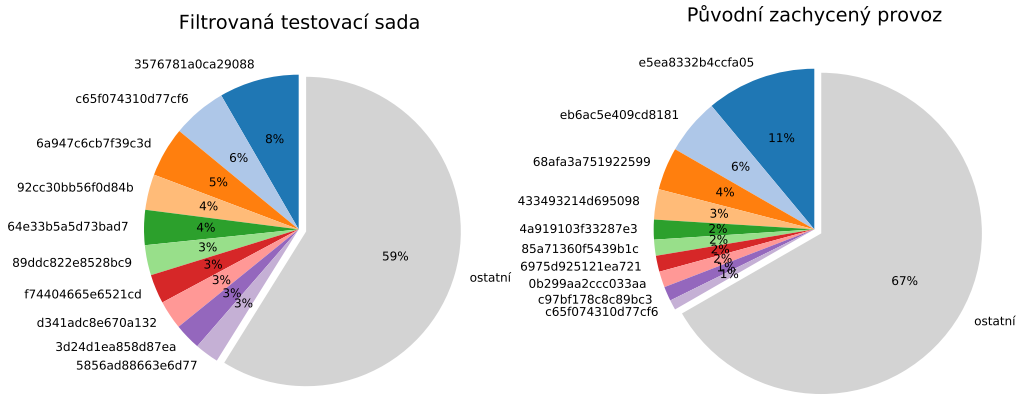
Podkladem pro vytvoření testovací datové sady je zachycený anonymizovaný provoz na perimetru sítě CESNET2, který poskytl vedoucí práce. Jediným použitým filtrem při sběru dat bylo omezení na TCP port 22, výchozí port SSH protokolu. Protože zachycená data obsahují kompletní provoz sítě, předpokládáme výskyty anomálií, částečně zachycených spojení, tranzientního provozu a v neposlední řadě také množství scanů síťových služeb či slovníkových útoků. Pro zachycená data máme k dispozici síťové toky rozšířené o pluginy IDPContent, PSTATS a PHISTS popsané v sekci 4. Konfigurace exportéru i pluginů byla stejná jako u datové sady pro analýzu.

Protože nemá význam testovat implementaci prototypu proti jiným druhům provozu než SSH (např. TCP SYN scanům), odfiltrujeme tyto záznamy a ponecháme pouze obousměrně validní SSH provoz, čímž vytvoříme testovací datovou sadu. Validním SSH provozem myslíme takový, který splňuje následující podmínky:

- První dvě zprávy začínají řetězcem „SSH“ (IDPContent)
- Spojení obsahuje alespoň 6 paketů v každém směru (základní tok)
- Spojení přeneslo alespoň 60 B dat v každém směru (základní tok)
- Zaznamenali jsme alespoň 11 SSH zpráv (paketů pluginu PSTATS)
- První paket je ve směru klient→server

Z původních 8 138 474 zachycených záznamů obsahuje testovací sada celkem 440 521 validních SSH spojení (přibližně 5,41 %). Naprostá většina odfiltrovaných spojení byla zapříčiněna podmínkou na obousměrnost provozu, což odpovídá právě scanu otevřených portů bez odpovědi. Jak zmiňuje sekce 3.2, bylo potřeba statisticky ověřit četnost výskytu paketu o velikosti 16 B na provozu reálné sítě, který značí konec transportní vrstvy SSH protokolu. Testovací sada obsahuje tento paket v 98,9 % záznamů, proto můžeme jeho pozici využít pro zpřesnění detekce téměř ve všech případech.

Z pohledu unikátnosti adres evidujeme v testovací sadě 1 283 unikátních zdrojových IP adres. Graf 5.1 ukazuje procentuální zastoupení 10 nejčastějších zdrojů zaznamenaných toků oproti zbytku provozu. Adresy představuje unikátní anonymizační řetězec.



Obrázek 5.1: Procentuální zastoupení nejčastějších zdrojových adres

Bohužel potřebná velikost a původ datové sady implikuje chybějící anotaci jednotlivých záznamů, která je ovšem nezbytná pro vyhodnocení přesnosti detektoru. Proto jsme museli anotaci přibližně určit za použití veřejných zdrojů a clustrovacích algoritmů. Průběh přibližné anotace testovací sady popisuje následující sekce.

5.2 Anotace testovací sady

Anotování jakéhokoli šifrovaného provozu bez znalosti jeho původního obsahu nelze provést se 100% přesností už jen z principu šifrování. Proto chceme zdůraznit nejistý charakter provedené anotace a samozřejmě připouštíme také existenci chybně určených záznamů. Důvěryhodnost anotace resp. následného vyhodnocení zakládá na použití odlišných principů a charakteristik oproti testovanému prototypu.

5.2.1 Využití veřejných databází

Pro prvotní separaci jsme využili externí zdroje reputací zdrojových IP adres. Vzhledem k anonymizaci testovací sady se reputace adres prováděla automatizovaně těsně po záchytu dat na infrastruktuře sítě CESNET. Hlavním zdrojem byla veřejná databáze AbuseIPDB [34], která funguje na komunitním principu uživatelských hlášení rizikových či agresivně vystupujících IP adres. Spolehlivost těchto dat se odráží v ukazateli abuseConfidenceScore, jehož podkladem je především čas vzniku události a počet unikátních uživatelů hlásících posuzovanou IP adresu s ohledem na rozumně krátké časové okno.

Druhým zdrojem byla experimentální aplikace NERD [35] od sdružení CESNET. Jejím podkladem jsou naopak informace z detekčních systémů, honeypotů nebo IDS nástrojů. Pro naše účely využijeme štítky „Scanner“ a „LoginAttempts“.

Účelem srovnání testovací sady s reputací zdrojových adres bude v co nejvyšší míře označit scany, útoky na hesla a podobné druhy provozu. Už z povahy původu veřejných dat neočekáváme 100% odlišení tohoto provozu. Mnoho systémů bohužel žádným způsobem neeviduje agresivně vystupující adresy, a proto žádný externí zdroj tohoto typu dokonce nemůže být úplný.

Ponecháním záznamů bez negativních referencí zdroje v obou externích databázích dostáváme provoz k další anotaci o počtu 249 493 záznamů (přibližně 56 % testovací sady), což svědčí o přítomnosti velkého množství „neautentizovaných“ aktivit. Záznamy s negativními referencemi dále neanotujeme a ve vyhodnocení k nim budeme přistupovat jako k separátní kategorii.

5.2.2 Deduplikace a agregace záznamů

Clustrovací algoritmy obecně počítají vzájemné charakteristiky mezi jednotlivými vstupy a na jejich základě je sdružují do podobných skupin tzv. clustrů. Volba počítaných charakteristik záleží na použitém algoritmu a jeho parametrech, nicméně obecně spadají mezi výpočetně náročné algoritmy. Z tohoto důvodu bylo potřeba testovací sadu zmenšit, avšak bez ztráty relevantních informací.

V první řadě se nabízí prosté odstranění duplicitních záznamů. Protože se jedná o zachycený provoz reálné sítě a záznamy obsahují i časové značky, tak výskyt dvou úplně shodných záznamů je prakticky vyloučený. Pokud se ale zamyslíme nad účelem, kterým je příprava pro vyhodnocení detektoru, většina informací o spojení nebude hrát ve vyhodnocení vůbec žádnou roli. Pro ilustraci bych tento fakt demonstroval na několika údajích zachycených síťových toků, které neovlivní průběh SSH protokolu, aniž by se to projevilo do ostatních extrahovaných informací (především do velikostí zasílaných paketů). Z experimentů a analýzy také víme, že spojení se stejnými parametry zachovávají velikosti zasílaných zpráv.

- čas začátku resp. konce spojení
- IP a MAC adresy
- použité TCP porty
- ...

Určující informací jsou pro nás pole s velikostmi jednotlivých zpráv (označení PSTATS pluginu: PPI_PKT_LENGTHS), které plně charakterizují průběh daného SSH spojení. Deduplikací záznamů podle údaje PPI_PKT_LENGTHS jsme dosáhly redukce testovací sady o 90,25 %, tedy na 24 320 unikátních záznamů z původních 249 493 záznamů testovací sady bez negativních referencí.

Protože analyzujeme SSH spojení až od autentizační části a předcházející výměna šifrovacích klíčů neuručuje ověřovací metodou ani druh spojení, můžeme záznamy dále deduplikovat podle již použité informace PPI_PKT_LENGTHS s vynecháním transportní vrstvy SSH protokolu. Algoritmus MAC plošně ovlivňuje velikosti všech zpráv od autentizační vrstvy, proto vynecháním transportní vrstvy SSH protokolu nemůže docházet např. k záměně autentizovaného a neautentizované spojení s různými MAC algoritmy, bez promítnutí do velikostí paketů v autentizační části spojení.

Začátek autentizace jednoznačně poznáme pomocí předcházející paketu velikosti 16 B. U záznamů bez tohoto specifického paketu neznáme konec transportní vrstvy

SSH protokolu a deduplikační krok na ně neaplikujeme. Výsledkem druhé deduplikace je 8 820 unikátních záznamů od autentizační části a 1 193 záznamů s neznámým začátkem autentizační části.

Poslední aplikovanou úpravou před nasazením clustrovacích algoritmů je agregace paketů ve stejném směru pod součet jejich velikostí. Zatímco deduplikace cílí především na autentizační část spojení a sdružuje např. opakované pokusy hádání hesel, agregace se bude více dotýkat aplikační vrstvy SSH spojení, kde se méně střídají směry komunikace jako např. u datových přenosů.

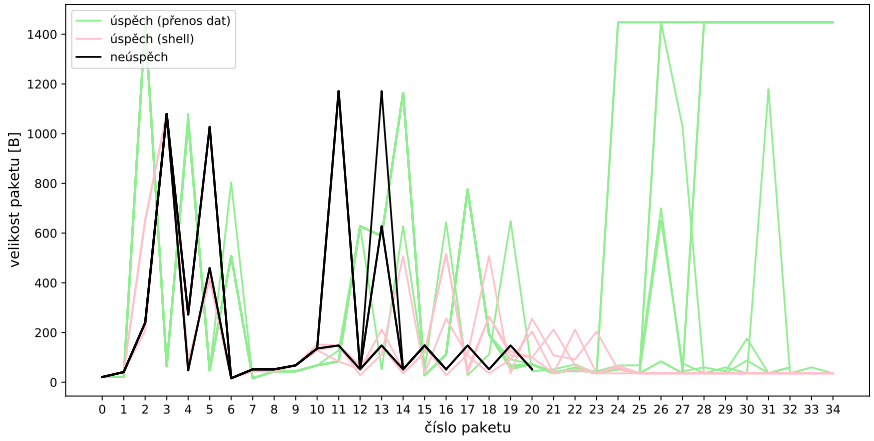
5.2.3 Clustrování a anotace

Primárním cílem celého třídění a clustrování testovací sady je přibližná anotace úspěšnosti autentizace SSH spojení na základě podobnosti velikostí paketů se známými případy. Ideálním výstupem by bylo každému záznamu přiřadit štítek typu úspěch/neúspěch. Do podrobnější anotace (např. autentizačních metod nebo druhu provozu) se nechceme příliš pouštět, protože bychom k jejímu určení museli využít konkrétní rysy provozu, které prototyp používá také. Tím by bylo následné porovnání přesnosti detektoru velmi zavádějící až předpojaté. Přesto v procesu anotace využíváme některé typické prvky SSH provozu odhalené analýzou, ale spíše pro separaci záznamů do disjunktní skupin, které anotujeme zvlášť. Každá vyčleněná skupina poté prochází manuálním posouzením podobnosti velikostí paketů v průběhu spojení k některé ze vzorových situací. Kvůli časové náročnosti by manuální posuzování nebylo možné bez předchozí deduplikace.

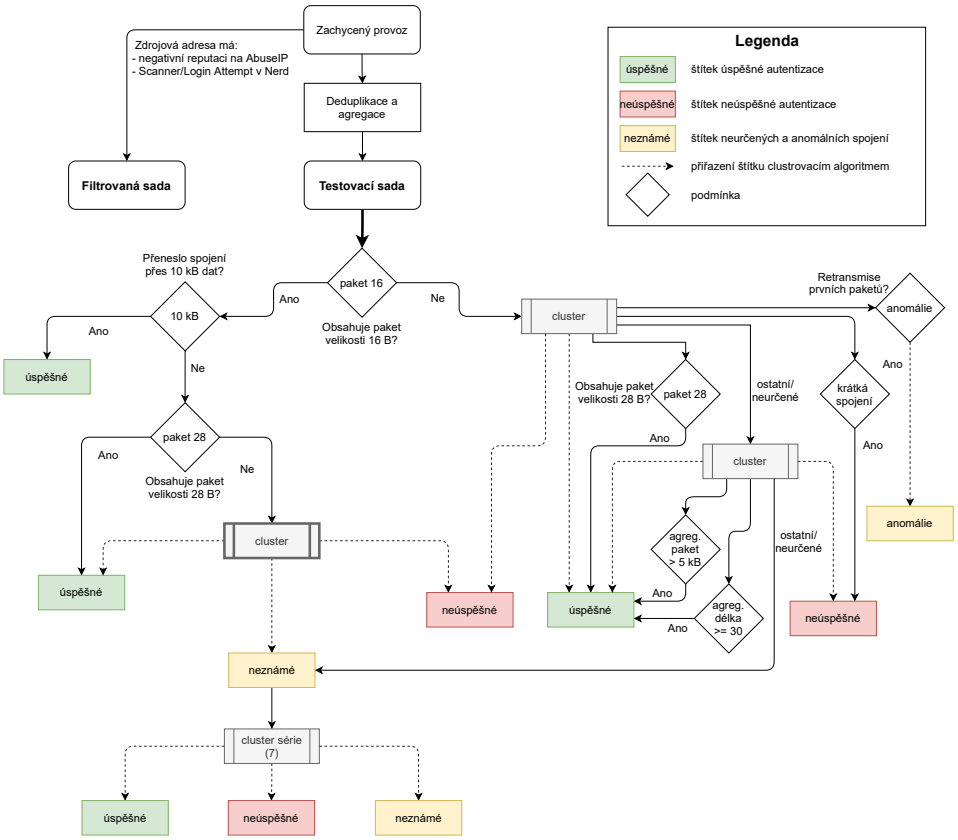
Pro účely nalezení průběhem podobných spojení jsme využili clustrovací algoritmus OPTICS z knihovny sklearn [36], který při experimentech dosahoval nejlepších výsledků. Ostatní zkoušené algoritmy (např. korelační) byly pro malý počet záznamů velmi přesné, nicméně nad celou deduplikovanou sadou buď nedokázaly záznamy sdružit s žádnou další třídou, čímž vznikaly separátní třídy o jediném záznamu, nebo sdružovaly očividně nesouvisející spojení.

Zmíněný algoritmus OPTICS vytváří třídy na principu hustoty grafu a každé použití separovalo vstupní data do několika tříd a jedné speciální s nezařazenými záznamy. Určeným třídám jsme následně na základě vizuální inspekce průběhu spojení a porovnání se známými průběhy různých situací z části analýzy přiřadily jeden ze štítků (úspěch/neúspěch/neznámý/anomálie). Ukázku odlišností modelových situací vidíme na grafu 5.2.

Míra neurčených spojení v jedné iteraci OPTICS algoritmu se pohybovala přibližně mezi 50–66 % vstupních dat. Přidělování štítků tedy probíhalo na etapy opakovaným spouštěním clustrovacího algoritmu na zatím neurčené množiny záznamů. Kvůli velikosti sady a časové náročnosti vizuální inspekce jsme se v některých situacích museli uchýlit k využití specifických charakteristik SSH spojení (např. přítomnost paketu velikosti 28 B signalizuje úspěšnou autentizaci). Použité charakteristiky buď vychází (jako zmíněný 28B paket) z technických omezení určených standardem [10], a proto jejich použití považujeme za legitimní i přes souběžné uplatnění v detektoru, nebo zakládají na špatně napodobitelných rysech, jejichž pravděpodobnost falešné pozitivity považujeme za zanedbatelnou. Celý proces deduplikace a anotace testovací sady znázorňuje následující schéma 5.3.



Obrázek 5.2: Ukázka modelových situací pro anotaci testovací sady



Obrázek 5.3: Deduplikace a anotace testovací datové sady

5.3 Zhodnocení přesnosti detekcí

Přesnost jakéhokoli detekčního mechanismu je klíčový parametr pro jeho produkční nasazení. Provedené zhodnocení zakládá na přibližné anotaci neznámého SSH provozu, a proto předpokládáme existenci chyb při nesprávné anotaci, které mohou ovlivňovat míru úspěšnosti detektoru oběma směry. Na základě dostupných dat bohužel nemůžeme ani odhadnout pravděpodobnost chyby. Anotovanou testovací sadu bereme jako referenční a kvůli možným chybám je třeba na výsledky nahlížet jako na aproximaci skutečné úspěšnosti prototypu.

Anotace	**Detekce**	**Popis**
správná	správná	reálná úspěšnost
správná	chybná	reálná chybovost
chybná	správná	chyba 1. druhu (snižuje úspěšnost)
chybná	chybná	chyba 2. druhu (zvyšuje úspěšnost)

Tabulka 5.1: Druhy chyb při vyhodnocení přibližnou anotací

5.3.1 Přesnost detekce úspěšnosti autentizace

Vyhodnocení úspěšnosti autentizace provádíme zvlášť pro anotovanou testovací sadu a odfiltrovanou sadu negativních reputací zdrojových adres. Úspěšnost definujeme jako poměr shodných klasifikací anotované sady a prototypu ku celkovému počtu záznamů daného anotačního štítku. Vzhledem k jasnému početnímu nepoměru anotovaných tříd neurčujeme přesnost jako jediné číslo, protože by se v něm odrazilo množství snadno rozpoznatelných scanů. Což by v tomto případě ukazovalo spíše na složení druhů provozu než na úspěšnost prototypu. Místo toho uvádíme hodnoty ve formě tabulky 5.2. Prototypem nerozhodnuté situace uvádíme samostatně v souladu s úplným pokrytím všech kombinací štítků, ale chápeme je stejně jako neúspěšné rozpoznání.

Neurčená část testovací sady povětšinou obsahuje tranzientní provoz či síťové anomálie (např. TCP retransmise), ale bohužel ji nemáme jak vyhodnotit. Nicméně téma chybovosti detekce u anomálního provozu je zcela na místě, avšak pozice prototypu v celém procesu monitorovaní provozu není nejvhodnější na rozpoznávání těchto situací. Nabízí se např. filtrování opakovaně zasílaných TCP paketů na základě jejich pořadového čísla už na úrovni exportéru.

Obdobným způsobem postupujeme i u sady zdrojových adres s negativní reputací s tím rozdílem, že ji klasifikujeme jako scan či slovníkový útok, a tím předpokládáme neúspěšnou autentizaci. Po přezkoumání především chybně určených záznamů této sady docházíme k závěru, že některá obsažená spojení vykazují jasné známky komunikace, kterou nelze přiřadit anomálnímu provozu a jedná se tedy o chybu 1. druhu. Reputace adres se tak neosvědčuje pro všechny jednotlivé případy (např. mohlo dojít k přidělení již negativně evidované adresy), ale v plošném statistickém měřítku stále poskytuje poměrně spolehlivou informaci.

		Anotovaná sada		
autentizace		ano	ne	neznámý
počet záznamů		14 157 (5,67 %)	234 075 (93,8 %)	1 261 (0,5 %)
Prototyp	ano	**97,47** %	0,16 %	neurčeno
	ne	2,44 %	**99,78** %	
	neznámý	0,078 %	0,05 %	

Tabulka 5.2: Procentuální úspěšnost prototypu na testovací datové sadě

		Sada s negativní reputací
autentizace		ne
počet záznamů		191 028
Prototyp	ano	3,95 %
	ne	**95,49** %
	neznámý	0,54 %

Tabulka 5.3: Procentuální úspěšnost prototypu na sadě s negativní reputací

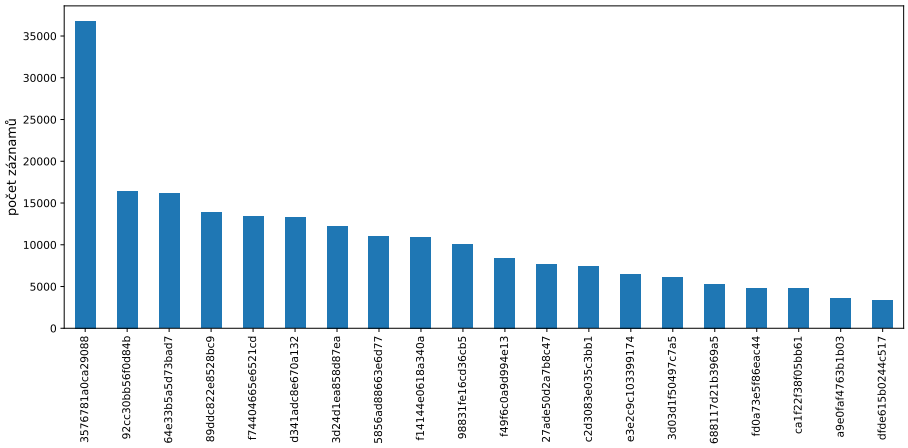
5.3.2 Přesnost detekce autentizačních metod a automatizace

Kvůli chybějící podrobné anotaci testovací sady bohužel nemůžeme vyhodnotit úspěšnost rozpoznání autentizační metody přímo. Pro představu uvedeme pouze získané výsledky a srovnáme s očekáváním vzhledem k povaze akademické sítě, na jejímž perimetru zachytávání probíhalo. Tabulka 5.4 ukazuje poměr detekovaných ověřovacích metod pro úspěšná ověření a také procentuální zastoupení jejich automatizace v rámci dané ověřovací metody. Dle výsledku prototypu v testovací sadě převládá jak ověření pomocí klíče, tak automatizace procesu autentizace, což odpovídá akademické sféře. Vyšší míra manuálního ověřování pomocí hesel odpovídá také.

	klíč	**heslo**	**neznámý**
	80,29 %	16,76 %	2,93 %
manuální	2,26 %	14,67 %	4,80 %
automatizované	97,73 %	85,32 %	95,19 %

Tabulka 5.4: Vyhodnocení autentizačních metod a automatizace

Pro neúspěšná ověření dosahuje míra automatizace 98,66 %, což svědčí o četných pokusech hádání hesel. V souvislosti s tím jsme z testovací datové sady, která neobsahuje zdroje s negativní reputací, získali nejčastější zdrojové adresy opakovaně neúspěšných pokusů o autentizace. Nejčastějších zdroje spolu s počtem jejich záznamů v testovací sadě ukazuje graf 5.4. Dohromady těchto 20 adres tvoří 85 % záznamů testovací anotované sady.



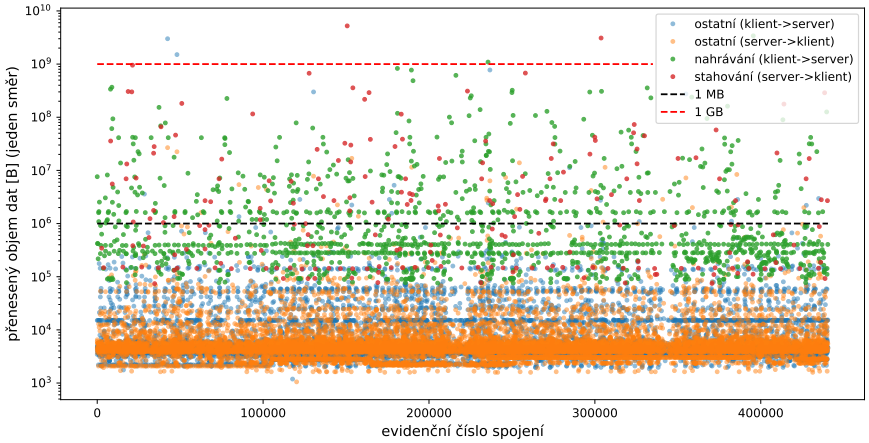
Obrázek 5.4: Nejčastější zdrojové adresy selhávající při autentizaci

5.3.3 Přesnost detekce datových přenosů

Vyhodnocení rozpoznání datových přenosů vychází z grafu 5.5, který v logaritmickém měřítku zachycuje objem přenesených dat pod evidenčním číslem spojení na ose x, které pomáhá pouze přehlednosti grafu a dokonce nevykresluje ani takové množství záznamů. Graf porovnává detekované nahrávání a stahování dat oproti neurčeným, ale autentizovaným druhům provozu a poměrně přehledně ukazuje separaci záznamů dle detekce. Rozpoznané datové přenosy se nevyskytují v záznamech s malým objemem přenesených dat (spodní část grafu) a naopak ostatní provoz přenáší data jen výjimečně.

Některé z těchto výjimečných situací jsme identifikovali jako kombinovaná spojení jak přenosu dat, tak i jiné komunikace. Spolu s velkým množstvím paketů došlo k naplnění maximální hodnoty datového typu některých histogramových intervalů a následným přidáváním pouze do ostatních intervalů vlivem kombinované komunikace docházelo v vyvažování histogramu, a tím k potlačení rozpoznávané charakteristiky.

S chybějící anotací také předpokládáme existenci kratších datových přenosů, které prototyp nezachytil. Z grafu jsme vypozorovali rozpoznávací hladinu detekce okolo 100 kB (10^5 B) s minimálními hodnotami 75 742 B u nahrávání a 74 110 u stahování. Kratší přenosy dat přirozeně nepotřebují takové množství paketů a kvůli zahrnutí začátku spojení v histogramu PHISTS nedojde k naplnění potřebného thresholdu. Možným řešením pro kratší přenosy by bylo odfiltrování začátku spojení s využitím délek jednotlivých paketů z pluginu PSTATS.



Obrázek 5.5: Vyhodnocení datových přenosů podle objemu přenesených dat

5.4 Výkonové požadavky

Primárním účelem prototypu je ověření funkčnosti navržených detekcí. Výkonové nároky uvádíme jako informační základ pro možnost budoucí optimalizace a také navrhujeme několik úprav vedoucích ke zrychlení prototypu. Měření probíhalo na serveru s procesorem Intel Xeon E5-2609 v3 1,90 GHz, avšak prototyp dokáže využít právě jedno jádro. Průměrné naměřené hodnoty shrnuje tabulka 5.5 níže jak pro kompletně zachycený provoz na TCP portu 22, tak jen pro filtrovanou sadu, která obsahuje obousměrně validní SSH provoz bez TCP SYN scanů.

sada	**počet toků**	**čas běhu**	**toků / s**
zachycený provoz	8 138 474	78,795 s	103 286,41
testovací sada	440 521	38,826 s	11 346,03

Tabulka 5.5: Měřený výkon prototypu

Výsledná hodnota průtoku záznamů se odvíjí také od obsahu resp. délky analyzovaných toků, které jsou předurčeny složením provozu na síti. Pro reálné nasazení by bylo vhodné zoptimalizovat především nejčastěji zastoupenou kategorii provozu monitorované sítě. Z technické stránky uvádíme možné optimalizační úpravy v následujícím seznamu.

Redukce počtu exportovaných paketů pluginem PSTATS na jen využívanou hodnotu. Pro analýzu jsme použili zachytávání prvních 200 paketů, nicméně navrhované detekční funkce nepoužívají více než 30–40 paketů.

Specifikace výstupního formátu je nezbytnou součástí pro napojení na IDS, ale omezení nepotřebných výstupních údajů snižuje čas zpracování. Např. redukce výstupu pouze na výsledky detekce (tj. odstranění údajů základního toku) zvýšila u testovací sady počet zpracovaných toků za vteřinu přibližně o 15 %.

Paralelizace detektoru. Detekční procedura sice pouští jednotlivé detekce v daném pořadí, ale nevyžaduje zapisovatelnou sdílenou paměť, což naznačuje možnost zpracování více záznamů současně za použití horizontálního škálování na úrovni hlavní smyčky detektoru.

Výměna programovacího jazyka. Díky dostupnosti širokého množství knihoven je Python vhodný pro datové analýzy a tvorbu prototypů. Ale z jeho podstaty skriptovacího jazyka nebude dosahovat výkonnosti kompilovaných programovací jazyků optimalizovaných na výkon jako např. jazyk C.

5.5 Limitace prototypu a plánovaný budoucí rozvoj

Největším omezením prototypu jsou bezesporu hodnoty používaných thresholdů, které určují rozpoznávanou signaturu na základě velikosti paketu bez přihlédnutí k použitému MAC algoritmu o němž víme, že dokáže velikosti paketů výrazně ovlivnit a důsledkem toho dochází k chybným detekcím. Řešením by bylo určení MAC algoritmu např. pomocí nového pluginu exportéru síťových toků nebo statistickým odhadem podle velikostí zpráv od začátku autentizační části.

Autentizace pomocí klíče bez (před)ověření se technicky velmi podobá autentizaci heslem, a proto mohou být krátké klíče detektorem zaměňovány za hesla. Jediným viditelným odlišujícím parametrem je velikost paketu s heslem resp. s klíčem. Opět se nabízí výše zmíněné určení MAC algoritmu a vytvoření modelu velikostí paketů pro různé druhy klíčů, který by přesněji rozlišoval situace oproti statickému thresholdu. Domněnkou k ověření je také možná korelace mezi použitými druhy klíčů při autentizaci uživatele a při předcházejícím ověření identity serveru, které není šifrované.

Vyhodnocení detekce datových přenosů v sekci 5.3.3 stanovuje senzitivitu této detekce okolo 100 kB přenesených dat a zároveň navrhuje zohlednit krátká spojení resp. přeskočit v PHISTS histogramu autentizační úvod. Další z možností je doplnění detekce o další typické rysy přenosů dat jako jsou časové rozestupy paketů nebo i obyčejný objem přenesených dat.

Implementované detekce povětšinou vychází z velikostí prvních N paketů a zachycený provoz prototyp pokládá za věrohodnou kopii aplikační síťové vrstvy bez jakýchkoli anomálií či retransmisí. V některých situacích jsou taková spojení chybně detekovaná kvůli narušení očekávané struktury celého spojení. Navrhované opatření na úrovni exportéru diskutujeme v sekci 5.3.1.

Závěr

Cílem práce bylo analyzovat šifrovanou komunikaci SSH protokolu v různých situacích a na základě analýzy navrhnout detekční algoritmus pro rozpoznání typu SSH provozu. Teoretická část detailně seznamuje s principy protokolu a spolu s vytvořenou datovou sadou slouží jako podklad pro následnou analýzu.

Praktická část analyzuje průběh chování SSH protokolu se zaměřením na autentizaci a navrhuje řadu charakteristických rysů pro rozpoznání analyzovaných typů provozu. Nejčastěji využívá velikosti prvních paketů, ale také z časové rozestupy mezi nimi. Navržené detekce fungují převážně na principu specifických signatur, které rozpoznávají třeba ověření klíčem a heslem.

Kromě způsobu autentizace, její úspěšnosti a automatizace dokáže implementovaný prototyp detekovat také opakované autentizační pokusy nebo datové přenosy. Prototyp byl navržen na principu síťových toků, čímž je vhodný i pro vysokorychlostní sítě s potřebou vysoké propustnosti dat.

Pro vyhodnocení úspěšnosti detektoru bylo potřeba vytvořit referenční testovací datovou sadu ze zachyceného reálného provozu. K její přibližné anotaci byly použity clustrovací algoritmy, které shlukují zachycená spojení na základě jejich vzájemné podobnosti. Při vyhodnocení dosahoval prototyp velmi dobré přesnosti. Mimo jiné se podařilo detekovat velké množství opakovaných neúspěšných pokusů o přihlášení ze zdrojů, které nemají záznam v databázích negativních reputací adres. Tuto informaci lze dále využít např. k automatizované tvorbě seznamu útočících adres a následně zavést filtraci takového provozu na úrovni celé sítě. Pomocí detekce použité autentizační metody lze validovat nastavené bezpečnostní politiky organizace a rychle odhalovat nesoulady, které by mohly vést ke snížení požadované úrovně zabezpečení. Neočekávané datové přenosy mohou signalizovat např. úniky dat, proto je jejich monitorování užitečné.

V budoucnu je možné na tuto práci navázat rozpoznáním dalších typů SSH provozu jako např. vzdálené terminály a SSH tunely, nebo v případě datových přenosů dokonce odlišovat konkrétní spouštěné programy.

Seznam použité literatury

1. RODRIGUES, Gabriel; ALBUQUERQUE, Robson; DEUS, Flavio; SOUSA JUNIOR, Rafael de; JÚNIOR, Gildásio; GARCÍA VILLALBA, Luis; KIM, Tai-Hoon. Cybersecurity and Network Forensics: Analysis of Malicious Traffic towards a Honeynet with Deep Packet Inspection. *Applied Sciences*. 2017, roč. 7, s. 1082. Dostupné z DOI: 10.3390/app7101082.
2. *Service Name and Transport Protocol Port Number Registry* [online]. Internet Assigned Numbers Authority, 2021-04-29 [cit. 2021-05-03]. Dostupné z: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.
3. KOZÁK, Bc. Ondřej. *Pokročilá analýza SSH provozu* [online]. 2018 [cit. 2021-05-03]. Dostupné z: https://is.muni.cz/th/uf67z/dp-kozak.pdf#section.5.3. Dipl. Masarykova univerzita.
4. YLONEN, Tatu. *SSH Secure Shell home page, maintained by SSH protocol inventor Tatu Ylonen. SSH clients, servers, tutorials, how-tos.* [Online] [cit. 2021-05-03]. Dostupné z: https://www.ssh.com/academy/ssh.
5. YLONEN, Tatu. *SSH protocol is the standard for strong authentication, secure connection, and encrypted file transfers. We developed it.* [Online] [cit. 2021-05-03]. Dostupné z: https://www.ssh.com/academy/ssh/protocol.
6. YLONEN, Tatu. *SSH port forwarding/tunneling use cases and concrete examples. Client command, server configuration. Firewall considerations.* [Online] [cit. 2021-05-03]. Dostupné z: https://www.ssh.com/academy/ssh/tunneling/example.
7. YLONEN, Tatu. *The Secure Shell (SSH) Protocol Architecture* [Internet Requests for Comments]. 2006-01 [cit. 2021-05-03]. RFC. Dostupné z: https://tools.ietf.org/html/rfc4251.
8. *Wireshark · Go Deep.* [Online] [cit. 2021-05-03]. Dostupné z: https://www.wireshark.org.

9. YLONEN, Tatu. *Session key in cryptographic protocols is a per-session key used for encryption and integrity checking.* [Online] [cit. 2021-05-03]. Dostupné z: `https://www.ssh.com/ssh/session-key`.

10. YLONEN, Tatu. *The Secure Shell (SSH) Transport Layer Protocol* [Internet Requests for Comments]. 2006-01 [cit. 2021-05-03]. RFC. Dostupné z: `https://tools.ietf.org/html/rfc4253`.

11. YLONEN, Tatu. *The Secure Shell (SSH) Authentication Protocol* [Internet Requests for Comments]. 2006-01 [cit. 2021-05-03]. RFC. Dostupné z: `https://tools.ietf.org/html/rfc4252`.

12. YERGEAU, F. *UTF-8, a transformation format of ISO 10646* [Internet Requests for Comments]. 2003-11 [cit. 2021-05-03]. RFC. Dostupné z: `https://tools.ietf.org/html/rfc3629`.

13. KHEIRKHAH, Esmaeil; AMIN, SM Poustchi; SISTANI, HA Jahanshahi; ACHARYA, Haridas. An experimental study of ssh attacks by using honeypot decoys. *Indian Journal of Science and Technology* [online]. 2013, roč. 6, č. 12, s. 5567–5578 [cit. 2021-05-03]. Dostupné z: `https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2013/Issue-12/Article8.pdf`.

14. YLONEN, Tatu. *The Secure Shell (SSH) Connection Protocol* [Internet Requests for Comments]. 2006-01 [cit. 2021-05-03]. RFC. Dostupné z: `https://tools.ietf.org/html/rfc4254`.

15. *Suricata — Open Source IDS / IPS / NSM engine* [online]. Open Information Security Foundation [cit. 2021-05-03]. Dostupné z: `https://suricata-ids.org`.

16. *Snort - Network Intrusion Detection & Prevention System* [online] [cit. 2021-05-03]. Dostupné z: `https://www.snort.org`.

17. BULAJOUL, Waleed; JAMES, Anne; PANNU, Mandeep. Network Intrusion Detection Systems in High-Speed Traffic in Computer Networks. In: *2013 IEEE 10th International Conference on e-Business Engineering.* 2013, s. 168–175. Dostupné z DOI: `10.1109/ICEBE.2013.26`.

18. NAIDU, R.China; P.S.AVADHANI. An Effective Evolution of Packet Loss With SNORT. *Indian Journal of Science and Technology.* 2013, roč. 4, č. 3.

19. SHAH, Syed; ISSAC, Biju. Performance Comparison of Intrusion Detection Systems and Application of Machine Learning to Snort System. *Future Generation Computer Systems.* 2018, roč. 80, s. 157–170. Dostupné z DOI: `10.1016/j.future.2017.10.016`.

20. DREGER, Holger; FELDMANN, Anja; PAXSON, Vern; SOMMER, Robin. Operational experiences with high-volume network intrusion detection. In: *Proceedings of the 11th ACM conference on Computer and communications security.* 2004, s. 2–11.

21. DAY, David; BURNS, Benjamin. A performance analysis of snort and suricata network intrusion detection and prevention engines. In: *Fifth international conference on digital society, Gosier, Guadeloupe*. 2011, s. 187–192.

22. *4x Suricata Performance Increase for Intel PAC* [online] [cit. 2021-05-03]. Dostupné z: `https://www.napatech.com/support/resources/solution-descriptions/4x-suricata-performance-increase-for-intel`.

23. *All features — Suricata* [online]. Open Information Security Foundation [cit. 2021-05-03]. Dostupné z: `https://suricata-ids.org/features/all-features`.

24. *About Zeek – Book of Zeek (git/master)* [online] [cit. 2021-05-03]. Dostupné z: `https://docs.zeek.org/en/master/about.html`.

25. HOFSTEDE, Rick; ČELEDA, Pavel; TRAMMELL, Brian; DRAGO, Idilio; SADRE, Ramin; SPEROTTO, Anna; PRAS, Aiko. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys Tutorials*. 2014, roč. 16, č. 4, s. 2037–2064. Dostupné z DOI: `10.1109/COMST.2014.2321898`.

26. *GitHub - CESNET/ipfixprobe* [online]. Czech Educational a Research Network [cit. 2021-05-03]. Dostupné z: `https://github.com/CESNET/ipfixprobe`.

27. CEJKA, Tomas; BARTOS, Vaclav; SVEPES, Marek; ROSA, Zdenek; KUBATOVA, Hana. NEMEA: a framework for network traffic analysis. In: *2016 12th International Conference on Network and Service Management (CNSM)*. 2016, s. 195–201.

28. *GitHub - CESNET/Nemea: System for network traffic analysis and anomaly detection* [online]. Czech Educational a Research Network [cit. 2021-05-03]. Dostupné z: `https://github.com/CESNET/NEMEA`.

29. *NEMEA* [online]. Czech Educational a Research Network [cit. 2021-05-03]. Dostupné z: `https://nemea.liberouter.org`.

30. *Nemea Framework* [online]. Czech Educational a Research Network [cit. 2021-05-03]. Dostupné z: `https://github.com/CESNET/Nemea-Framework`.

31. *OpenSSH* [online] [cit. 2021-05-03]. Dostupné z: `https://www.openssh.com`.

32. *PuTTY* [online] [cit. 2021-05-03]. Dostupné z: `https://www.putty.org`.

33. D. MILLER, S. Josefsson. *The chacha20-poly1305@openssh.com authenticated encryption cipher draft-josefsson-ssh-chacha20-poly1305-openssh-00* [online]. 2015-11 [cit. 2021-05-03]. Tech. zpr. Dostupné z: `https://tools.ietf.org/html/draft-josefsson-ssh-chacha20-poly1305-openssh-00`.

34. *AbuseIPDB - IP address abuse reports - Making the Internet safer, one IP at a time* [online] [cit. 2021-05-03]. Dostupné z: `https://www.abuseipdb.com`.

35. *NERD - Network Entity Reputation Database* [online] [cit. 2021-05-03]. Dostupné z: `https://nerd.cesnet.cz`.

36. *sklearn.cluster.OPTICS* [online] [cit. 2021-05-03]. Dostupné z: `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html`.

Příloha **A**

Seznam použitých zkratek

SSH – Secure Shell
MAC – Message Authentication Code
ETM – Encrypt Than MAC
NEMEA – Network Measurements Analysis
TCP – Transmission Control Protocol
MTU – Maximum transmission unit
VPN – Virtual Private Network
IDS – Intrusion Detection System
SIEM – Security Information and Event Management
RFC – Request for Comments
IANA – Internet Assigned Numbers Authority
HTTP – Hypertext Transfer Protocol
DNS – Domain Name System
SSL – Secure Sockets Layer
TLS – Transport Layer Security
ECDSA – Elliptic Curve Digital Signature Algorithm
PCAP – Packet Capture
IDPContent – Initial Data Packets Content
PSTATS – Packets Statistics
PHISTS – Packets Histograms
TCP SYN – Transmission Control Protocol - Synchronize
TCP ACK – Transmission Control Protocol - Acknowledgment
TCP FIN – Transmission Control Protocol - Finish
NERD – Network Entity Reputation Database

Příloha **B**

Obsah přiložené SD karty

- `readme.txt` — stručný popis obsahu SD karty
- `analysis/` — podklady pro analýzu
  - `analysis-graphs.ipynb` — generované grafy z analytické sady
  - `analysis-graphs.pdf` — generované grafy z analytické sady
  - `analysis-idpcontent.csv` — analytická datová sada
  - `analysis-phists.csv` — analytická datová sada
  - `analysis-pstats.csv` — analytická datová sada
- `evaluation/` — podklady pro vyhodnocení přesnosti
  - `reputation-lookup.csv` — anonymizované reputace IP adres
  - `results.csv` — výstup detekčního modulu
  - `testing.ipynb` — vyhodnocení prototypu
  - `testing.pdf` — vyhodnocení prototypu
  - `testing-annotation.csv` — anotovaná testovací sada
  - `testing-negative-reputation.csv` — sada negativních reputací
  - `testing-idpcontent.csv` — testovací sada
  - `testing-phists.csv` — testovací sada
  - `testing-pstats.csv` — testovací sada
- `src/`
  - `ssh.py` — implementace detekčního modulu
- `thesis/`
  - `bibliography.bib` — použitá bibliografie
  - `cvut-logo-bw.pdf` — logo použité v práci
  - `DP_smejkal_radek_2021.pdf` — text práce ve formátu PDF
  - `DP_smejkal_radek_2021.tex` — zdrojová forma práce ve formátu Latex
  - `FITthesis.cls` — použitá Latex šablona
  - `prohlaseni5.txt` — použité prohlášení
  - `zadani.pdf` — zadání práce
  - `figures/` — adresář s obrázky použitými v práci