Netfox Detective 2.0 - Nástroj pro síťovou forenzní analýzu

Netfox Detective 2.0

Technická zpráva FIT VUT v Brně

Jan Pluskal

Technická zpráva č. FIT-TR-2017-06
Fakulta informačních technologií, Vysoké učení technické v Brně

Naposledy změněno: 12. ledna 2018

Netfox Detective 2.0 - Nástroj pro síťovou forenzní analýzu

Jan Pluskal

Vysoké učení technické v Brně, email: ipluskal@fit.vutbr.cz

Abstrakt Tato technická zpráva popisuje novinky implementované v nástroji Netfox Detective 2.0. Podrobně je diskutovaná architektura nástroje a ospravedlňuje propojení jednotlivých modulů. Unikátní řešení schopné rekonstruovat zachycenou komunikaci i v případě, že záchyt neobsahuje všechna data je implementováno jako součást nástroje Netfox Framework. Provedená refaktorizace řešení jako nejvýznamnější odlišnost od první verze připravuje nástroj na použití v distribuovaném prostředí spolu s pevně definovanou persistenční vrstvou aktuálně využívající SQL databázi. Z vizualizační části text popisuje použití Inversion of Control kontejneru, který umožňuje implementaci plug-in analyzátorů, které mohou získávat již zpracovaná data, případně kompletně ovládat samotný nástroj. Tento text je určený jako programová dokumentace a předpokládá základní znalost architektury počítačových sítí, programování na platformě .NET a použitých návrhových vzorů.

1 Úvod

Síťová forenzní analýza je nezbytnou součástí jak moderních vyšetřovacích technik zločinů pro složky činné v trestním řízení, tak i administrátorů vyšetřujících incidenty na počítačové síti. Inspekcí síťového provozu je možné najít odpověď na otázky týkající se obsahu komunikace, identity komunikujících stran a přesného časového horizontu, kdy k incidentu došlo. Pro zodpovězení těchto otázek za použití ruční analýzy zachycené síťové komunikace je nezbytná znalost architektury počítačové síťě, která, zpravidla implementující TCP/IP síťový model, se skládá z rozličných technologií na libovolné úrovni [1]. Ruční analýza se pak stává expertní záležitostí a velmi zdlouhavým procesem zahrnující zkoumání enormního množství dat s velmi nejistými výsledky. Jejich výhodou je však vysoká přesnost právě díky expertní znalosti vyšetřovatele, který je schopen rozpoznat např. vektor útoku třeba DDos, SYN Flood, či steganograficky skrytou informaci [4] z nestandardního chování síťových protokolů, které primárně slouží pro zcela jiný účel.

Vzhledem k množství dat běžně tekoucí po síti i pro běžného domácího uživatele, o firemní síti nemluvě, se možnost ruční analýzy veškeré komunikace stala

[1] V tomto textu bude použita jako referenční architektura ISO/OSI pro její jednoznačnost v definici chování na jednotlivých vrstvách a lepší segregaci.

již delší dobu časově nedostupnou. S množství incidentů a nedostatkem lidských zdrojů je tlak na urychlení celého procesu vyšetřování a získávání výsledku pomocí automatizovaných či poloautomatizovaných řešení jako jsou síťové monitorovací nástroje (NSM - Network Security Monitoring) a síťové forenzní nástroje (NFAT - Network Forensic Analysis Tools).

Monitorovací nástroje jako je např. tcpdump, tshark, wireshark, Microsoft Network Monitor či Message Analyzer jsou určené spíše pro síťové administrátory při hledání špatné konfigurace síťě či bezpečnostních incidentů, ale stejně tak mohou být použity i expertními vyšetřovateli, a slouží pro rychlé vytvoření představy o zachycené komunikaci. Složitější, vesměs grafické, nejsou určené pro práci s velkým množstvím zachycených dat, typicky v rozsahu do stovek MB či jednotek GB v závislosti na velikosti operační paměti [7] stroje, na kterém je nástroj provozován. Ovšem jednodušší konzolové nástroje jako je tcpdump či tshark mohou být použity jako moduly či filtery pro složitější NFAT nástroje a při rozumné implementaci problémem s množstvím souborů trpět nemusí. Tyto nástroje jsou schopny identifikovat síťové protokoly na vrstvách L1-L4, tedy fyzické, spojové, síťové a transportní [9–11], protože vždy nižší vrstva nese informaci o protokolu vrstvy následující. Výjimku z tohoto pravidla však mohou tvořit různá zapouzdření při vytváření overlay sítí [1, 3, 6, 12, 13], která analýzu velice komplikují a tyto nástroje nemusí být schopny se s nimi vypořádat.

v kontrastu s monitorovacími nástroji se vyznačují přívětivějším ovládáním pro uživatele s nižší expertní znalostí a produkují jasně definované výsledky zpravidla ve formě rekonstruovaných objektů získaných z komunikace, mapou komunikujících stran či kombinací informací z vícero protokolů, jako např. přiřazení doménového jména získaného ze zpracovaného zachyceného DNS požadavku na rezoluci a jeho přiřazení k IP adrese identifikující komunikující stranu. Popularitu použití těchto nástrojů zvedá fakt, že takto získaný důkaz bývá často velmi snadno akceptovatelný v trestním řízení a nebývá třeba expertního znalce pro bližší vysvětlení. Nevýhodou naopak může být právě automatizace samotná, protože při použití obfuskovacích metod nemusí být nástroj schopný onen důkaz nalézt.

Po pečlivé analýze síťových monitorovacích nástrojů i síťových forenzních nástrojů jsme identifikovali možné výzvy, které jsme se rozhodli řešit.

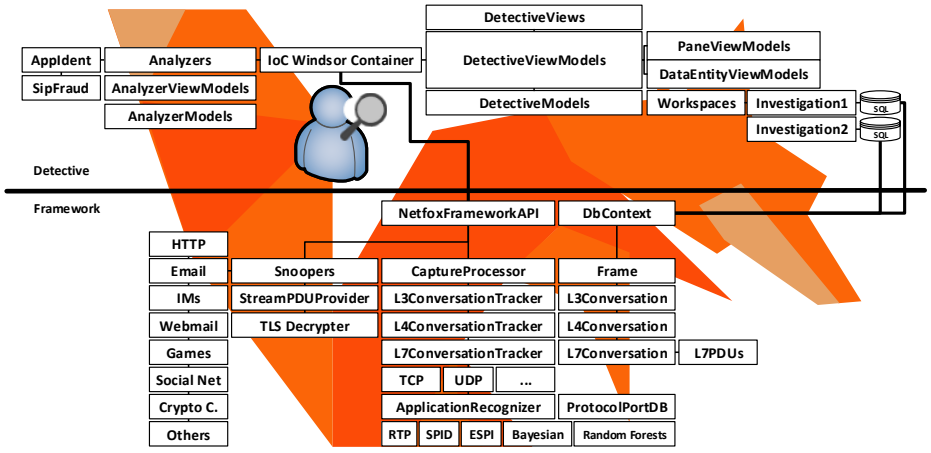
- Korektní zachycení síťových dat bez ztráty informace, která byla přenesena mezi hosty.
- Správné znovuustavení konverzací nad TCP transportním protokolem, obzvlášť pokud se rekonstruují neúplná data.
- Rozpoznání a správné zpracování dat při použití overlay sítí.
- Extrakce informací z aplikačních protokolů.
- Identifikace RTP přenosu bez znalosti signalizace.
- Identifikace použitého aplikačního protokolu pomocí statistických metod či strojového učení.
- Paralelní zřetězené zpracování síťových dat neomezené systémovými zdroji.

Tato technická zpráva vznikla v rámci bezpečnostní výzvy ministerstva vnitra, projektu Tarzan – Integrovaná platforma pro zpracování digitálních dat z

bezpečnostních incidentů[2], VI20172020062. Projekt je zaměřen na detekci a analýzu nových forem kybernetické kriminality v prostředí Internetu věcí, mobilních a komunikačních aplikací. Cílem projektu je výzkum nových metod založených na dolování dat, strojovém učení, vizuální analýze a vytvoření funkčního vzorku integrující tyto metody pro efektivní vyšetřování incidentů. Tato technická zpráva popisuje způsoby řešení jednotlivých výzev za pomoci druhé generace síťového forenzního nástroje *Netfox Detective 2.0*, jenž byl vyvinut taktéž v rámci tohoto projektu. Dále pak, v rámci nástroje Netfox Detective 2.0 byl implementován modul AppIdent [8], zajišťující identifikaci aplikačních protokolů pomocí metod strojového učení. Oba výstupy jsou součástí hlavního výstupu – Integrovaná platforma pro zpracování digitálních dat z bezpečnostních incidentů.

2 Architektura nástroje

Nástroj Netfox Detective je vytvořený pro operační systém Windows 7 a novější, v jazyce C# s použitím .NET Frameworku 4.7. Nástroj je členěný do dvou základních celků: Framework a Detective, viz Obrázek 1. Na spodní úrovni se nalézá sada modulů tvořících Netfox Framework, který zajišťuje zpracování komunikace zachycené v PCAP souborech. Na vrchní úrovni, zajišťující interakci s uživatelem, je grafické uživatelské rozhraní aplikace Netfox Detective, která využívá Netfox Framework pro zpracování dat a výsledky zobrazuje v přívětivé analytické formě uživateli.



Obrázek 1. Abstraktní znázornění architektury nástroje Netfox Detective. V horní části nad tlustou čárou se vyskytují komponenty implementující grafickou uživatelskou část nástroje. Pod čarou naopak komponenty Netfox Framework zpracovávající zachycenou komunikaci.

[2] http://www.fit.vutbr.cz/~ipluskal/grants.php?id=1063

2.1 Netfox Framework

Zpracováním síťové komunikace tímto frameworkem je myšleno čtení PCAP souboru rámec po rámci, parsování protokolů síťového stacku, roztřídění do konverzací reprezentující úrovně síťového stacku a to internetovou (L3), transportní (L4) a aplikační (L7). Zpracování L4 vyžaduje zpravidla, provedení reassemblingu dat této vrstvy s případnou IP defragmentací. Aplikováním heuristik [5] při zpracování L4 vrstvy jsou vytvořeny konverzace na vrstvě aplikační. Při tomto procesu jsou dále vytvořeny aproximací aplikačních zpráv (L7PDU). Netfox Framework zajišťuje na vyžádání zpracování L7PDU a extrakci dat z podporovaných aplikačních protokolů.

Na Obrázku 1 je ve spodní části vyznačena abstraktně architektura Frameworku. Komunikaci s vyššími vrstvami zajišťuje *NetfoxFrameworkAPI*, které řídí zpracování nového PCAP souboru a následnou extrakci informací z aplikační vrstvy. Samotná implementace je rozdělena na část *exekuční* a *modelovou*.

Exekuční část v Obrázku 1 se nachází pod a vlevo od *NetfoxFrameworkAPI*. Tyto moduly zajišťují svojí modulární skladbou polymorfní chování a rozšiřitelnost v případě potřeby implementovat nový síťový protokol. Komunikace mezi jednotlivými moduly je dána pevným rozhraním každého modulu, které definuje jeho vstupy a výstupy, které předávají data ve formě *modelů*.

Modelová část je ve zjednodušené podobě znázorněna pod modulem *DbContext*. Modely jsou v době svého vytvoření drženy v paměti a slouží pro uložení stavových informací, jako např. pro L3 konverzaci (L3 Conversation) mohou být IP adresa zdroje a cíle, a dále mohou seskupovat jiné modely, jako např. L3 konverzace obsahuje kolekci rámců (PmFrame), které mají stejnou zdrojovou a cílovou IP adresu jako konverzace samotná. Pro zajištění snadné rozšiřitelnosti a rychlosti vývoje jsou modely poskytovány vyšším vrstvám přes databázové připojení pomocí DbContextu[3].

Pro urychlení zpracování je využita knihovna Task Parallel Library (TPL). Převážně je využito Data Flow[4] zápisu, který zajišťuje členění do funkčních bloků, které jsou následně paralelně vykonávané. Framework implementuje funkcionalitu bufferovacích a exekučních TPL bloků, které kombinuje do komponent. Tímto přístupem je zajištěna modulárnost kódu a efektivní přístup ke zdrojům při zpracování. Modulárnost - exekuční kód je možné rozčlenit do logických celků, které spolu komunikují pomocí rozhraní tvořené bufferovacími bloky. Efektivita - protože bufferovací bloky mohou mít omezenou kapacitu, což v kombinaci s blokujícím voláním v exekučním bloku zajistí zastavení jeho vykonávání a vyčkání, než se místo v bufferu uvolní. Tato kombinace řeší i správu paměti a umožňuje používat bloky, které neprovádí výpočetní operaci se stejnou rychlostí.

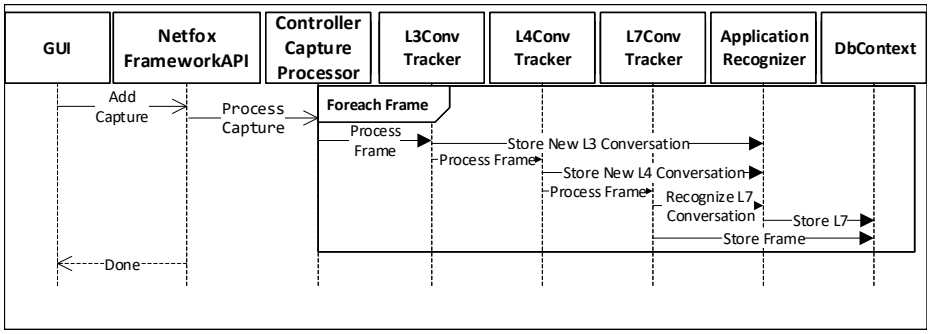
[3] Jedná se o komponentu Entity Frameworku, což je framework zajišťující Objektově-relační-mapování (ORM) a persistenci s podporou SQL databází.

[4] https://msdn.microsoft.com/cs-cz/library/hh228603(v=vs.110).aspx

Zpracování PCAP souboru

Zpracování PCAP souboru je inicializováno voláním modulu metody *AddCapture* v *NetfoxFrameworkAPI* a předáním cesty k souboru. V aktuální implementaci nástroj umožňuje zpracovávat pouze již zachycená data s uzavřeným souborem ve formátech libpcap, pcap-ng a Microsoft Network Monitor (mnm cap). Obrázek 2 popisuje sekvenci exekučních volání a předávání modelů abstraktně vysvětlující logickou strukturu zpracování.

Moduly jsou navrženy tak, aby jednotlivé fáze zpracování byly paralelizovatelné a byl umožněn jejich běh ve více instancích, jak táscích[5], procesech, tak i uzlech s NUMA přístupem. Této funkcionality je docíleno použitím TPL a možnosti spojovat bloky dle potřeby, což i umožňuje zpracovávat různorodá síťová zapouzdření. Architektura modulů spolu s propojeními exekučních a bufferovacích bloků je zobrazena na obrázku 3.

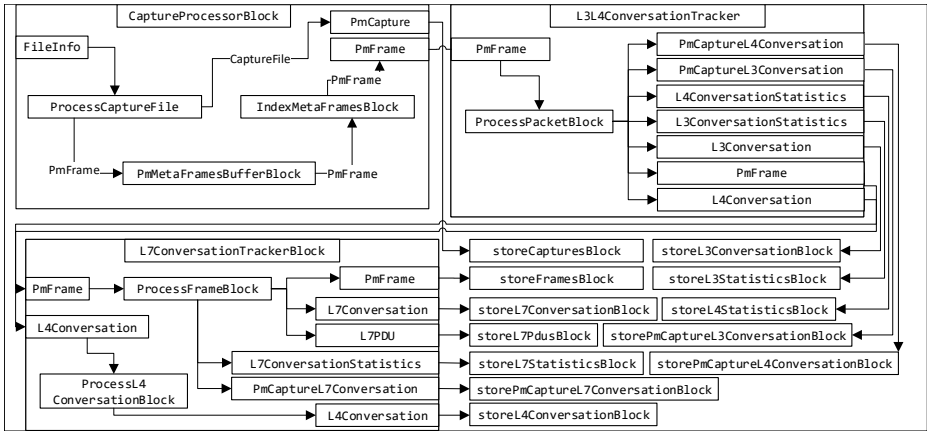


Obrázek 2. Zjednodušené schéma zpracování PCAP souboru sekvenčním průchodem. Nejsou uvažovány datové závislosti mezi modely. Nové konverzace se tvoří a ukládají v reakci na nalezení prvního rámce, který do vzniklé konverzace náleží.

ControllerCaptureProcessor provádí řízení procesu zpracování. Tento modul propojuje jednotlivé moduly bloků dle síťového zapouzdření a určuje stupeň paralelizace a kapacitu bufferů podle dostupných zdrojů. Při požadavku na současné zpracování jednoho či více PCAP souborů je nejprve vytvořena vždy nová procesní pipeline, dle obrázku 3. Popisovač souboru(ů) je předán bloku *CaptureProcessorBlock*, který zajistí naparsování rámců a vytvoří objekt popisující PCAP soubor–*PmCapture*. Používá sekvenční streamované čtení na úrovni PCAPu, což znamená, že se pouze identifikuje začátek packetu a jeho délku v uloženém souboru. Pro každý takto identifikovaný packet se vytvoří objekt

[5] https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx

PmFrame. Tento objekt obsahuje pouze výše zmíněné informace o své pozici v PCAPu a je třeba naplnit jeho vnitřní struktury blokem *IndexMetaFramesBlock*. Exekuční kód tohoto bloku je vykonáván s maximální možnou paralelizací a provádí paralelní, neblokující načtení dat z PCAPu a naparsování rámce ve vrstvách (L2), L3, L4 - vrstva L2 nemusí být vždy přítomná.



Obrázek 3. Schéma Task-Parallel-Library bloků paralelně zpracovávajících zachycenou komunikaci od otevření souboru, přes parsování rámců, trackování konverzací, identifikaci aplikačního a tunelujícícho protokolu po uložení. Bloky se mohou skládat z dalších zanořených bloků, obsahují vstupní a výstupní buffery pro kontrolu toku dat.

L3L4ConversationTracker zajišťuje agregaci rámců do příslušných konverzací na odpovídajících si vrstvách L3 a L4 pomocí bloku *ProcessPacketBlock*. Tedy pro rámce se stejnou IP adresou zdrojovou či cílovou, nebo obráceně, vytvoří objekt *L3Conversation*. Dále pak tato L3 konverzace bude kolekcí drobnějších agregací ve formě L4 konverzací, které opět seskupují rámce se stejnou IP zdrojovou či cílovou adresou (vyplývá ze vztahu 1 ... N L3Conversation ... L4Conversation), ale i stejnými zdrojovými či cílovými porty a tedy i stejným typem L4 protokolu.

Během vytváření nových konverzací a přiřazování rámů ke konverzacím je využito faktu, že rámce jsou načteny v paměti a jejich meta informace jsou přístupné. Proto komplementárně ke každé konverzaci na obou úrovních je také vytvořen objekt držící statistiky o dané konverzaci. Tento objekt je aktualizován o data z každého zpracovaného rámce v době vyhledání příslušné konverzace.

Komunikace bez IP vrstvy je součástí agregační L3 konverzace s adresami *0.0.0.0* a taktéž jediné L4 konverzace se stejnou IP adresou a porty *0*. Stejně tak,

IP komunikace s neznámým L4 protokolem je součástí patřičné IP konverzace a L4 konverzace s příslušnými IP adresami a opět porty *0*.

L7ConversationTracker je jádrem reassemblovacího enginu. Pomocí heuristických metod je možné separovat IP Flow komunikaci, tedy konverzace na L4 do jemnějších celků, a to konverzací na úrovni aplikačního protokolu.

Znamená to, že když uživatel(é) použije na jednom stroji aplikaci komunikující na TCP využívající pevných čísel portů, jak serveru tak klienta, na úrovni L4 konverzace se bude tato komunikace jevit jako jediná konverzace. Pokud je tento stroj sdílený a využívá ho více uživatelů, ze síťového pohledu nebude rozpoznatelné, že komunikace patří k více identitám. Tento problém je více znatelný, pokud zachycená data nejsou kompletní a ztrácejí datagramy nesoucí v metadatech signalizační informace, jako např. *TCP SYN, FIN, RST flagy*.

Modul zpracovává příchozí rámce paralaleně v *ProcessFrameBlock* a to tak, že pro každou instanci L4 konverzace vytvoří samostatný Task, který indexuje pomocí klíčových hodnot konverzace. Konkrétní implementace, dle použitého protokolu transportní vrstvy zajistí konkrétní zpracování, jako např. korektní TCP reassembling a IPv4 defragmentaci, či pouze defragmentaci v případě UDP. Když tento modul zjistí ze signalizace transportního protokolu, že uživatelské sezení skončilo, případně s využitím heuristických metod[5], vytvoří novou L7 konverzaci, pro kterou spočítá L7 konverzační statistiky, a předá na výstup.

Modul dále také využívá zmíněné heuristické metody, aby vytvořil abstrakce aplikačních zpráv, *L7PDU*. Většinou se jedná o souvislý datový blok tvořený jedním či více rámci, který přísluší k jedné akci. Můžeme tedy říct, že např. když chatující uživatel odešle textovou zprávu, tak ji považujeme za aplikační zprávu a je tvořena seskupením payloadů více datagramů. Z našeho pohledu jedna aplikační zpráva může být tvořena i více L7PDU, a to zejména v případě, že je velmi dlouhá. Nikdy však nenastane situace, že by jedno L7PDU obsahovalo dvě a více aplikačních zpráv. Tohoto omezení se pak dále využívá při tvorbě a použití parserů aplikačních protokolů.

Storage bloky zajišťují perzistenci vytvořených objektů. Všechny objekty se ukládají do SQL databáze pomocí *bulk insertů*, což značně zvyšuje propustnost databázové vrstvy. Tato metoda vyžaduje bufferování objektů a současné ukládání až při dosažení dostatečného počtu instancí. Aby nebylo třeba čekat na databázi, probíhá bufferování a ukládání do databáze odděleně v samostatných táscích. Každý storage blok má tedy dva tásky, které se v ideálním případě neblokují.

Tento velmi rychlý způsob zpracování je vykoupený složitějším návrhem databázové vrstvy. Použité modely, které jsou nositeli extrahovaných informací musí řešit datové závislosti ve své režii. Nemohou se spolehnout na Entity Framework, který v ostatních případech zajišťuje objektově relační mapování, protože overhead by byl násobný než zpracování samotné a tedy naprosto neakceptovatelný.

Pro vyřešení datových závislostí je použito generování identitních klíčů objektů typu Guid v aplikaci, místo generování databází jak je běžnější. Tímto krokem odpadne nutnost zpětné vazby, kdy by databáze sdělila identitní klíč nově vloženého objektu zpět aplikaci. Další nutností je nastavení cizích klíčů (FK) na objektech, které se vzájemně referencují a pokud je vazba mezi objekty N ... M, jako je tomu např. u L4 konverzace a zachyceného souboru, protože je možné tvořit konverzace napříč různými soubory, je třeba vložit záznam do propojující vazební tabulky. Tento záznam je generovaný během zpracování, např. blok *PmCaptureL4Conversation* a je následně uložený do databáze blokem *StorePmCaptureL4ConversationBlock*.

Zpracování je ukončeno a kontrola je navrácena přes *ControllerCaptureProcessor* a následně *NetfoxFrameworkAPI* volající aplikace. Mezi Netfox Framework a volající aplikací se nepřenáší žádné objekty. Aplikace pro získání dat musí využít *DbContext*, který modely zpřístupní včetně potřebných referencí. Programátor aplikace využívající Netfox Framework si může vybrat ze dvou přístupů. Buď použije přímo DbContext a vytvoří přes něj výkonově optimálnější dotaz na získání potřebných dat, nebo využije virtualizace za pomoci *VirtualizingObservableDBSetPagedCollection*, která poskytuje možnosti stránkování, notifikuje o změnách v databázi a zařídí načtení všech referencovaných modelů z databáze, ovšem za cenu vyšší režie.

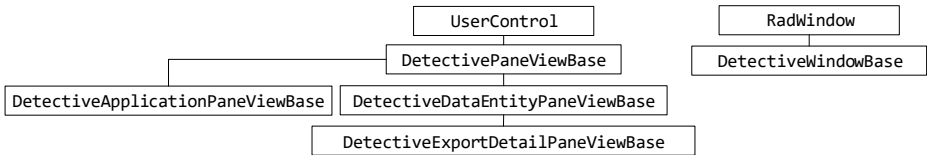
2.2 Netfox Detective

Netfox Detective je aplikace zobrazující informace získané analýzou zachycené komunikace nástrojem Netfox Framework. Architektura této aplikace je zobrazena na obrázku 1 ve vrchní části. Aplikace využívá pro tvorbu uživatelského rozhraní *Windows Presentation Foundation (WPF)*, a proto dle doporučení je vytvořena na základě třívrstvé architektury.

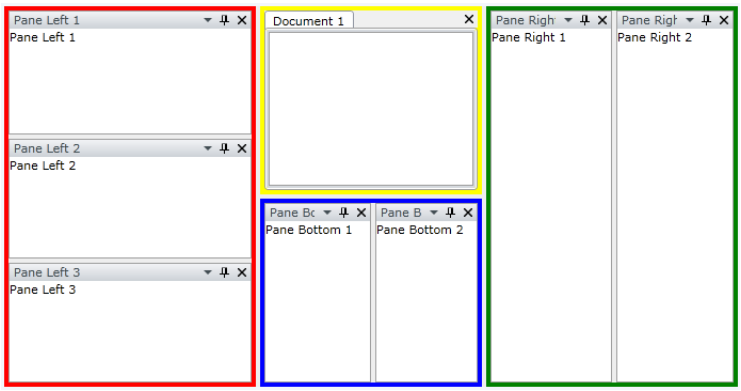
Tato architektura je založena na návrhovém vzoru *Model–View–ViewModel* tedy *MVVM*. Pro zajištění dynamičnosti je dále využit návrhový vzor *Dependency Injection* pomocí *Dependency Injection Container*, implementovaný *Windsor Container*. Pro zajištění modulárnosti a dynamického načítání komponent je vytvořena hierarchická struktura bázových tříd pro jednotlivé vrstvy MVVM. Na nejvyšší úrovni stojí pro každou vrstvu *DetectivePaneViewBase* a *DetectiveWindowBase*, *DetectiveViewModelBase*, a modelová vrstva bázi nemá.

Hierarchická dědičnost pro pohledovou vrstvu, viz obrázek 4, je dále rozdělena podle toho, zdali se jedná o okno, tedy *DetectiveWindowBase*, nebo přemístitelný, dockovatelný tab *DetectivePaneViewBase*. Taby je možné dockovat do mřížkové struktury, viz obrázek 5, což umožní vyšetřovateli rozdělit panely dle potřeby a aktuální úlohy. Dále pak, bázové třídy limitují dokovatelnost podle svého typu a to tak, že *DetectiveApplicationPaneViewBase* je umístitelný pouze do aplikačního okna, *DetectiveDataEntityPaneViewBase* je pohled spjatý s konkrétní instancí modelu a je umístitelný libovolně po aplikaci a *DetectiveExportDetailPaneViewBase* je taktéž spjatý s instancí modelu extrahovaných dat, ale

jeho umístění je limitováno pouze do prohlížeče exportních událostí. Okno samotné aplikace a další nemodální okna jsou založena na typu *DetectiveWindowBase*.



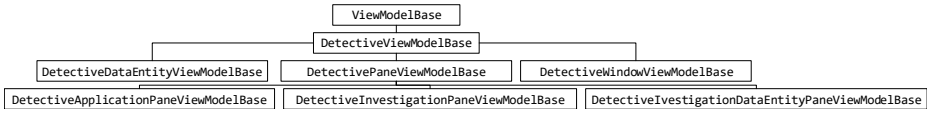
Obrázek 4. Diagram třídní dědičnosti bázových tříd pro pohledy. Komponenty určené pro zobrazení v uvnitř dokovatelných oblastí jsou děděny z *DetectivePaneViewBase*. Třídy reprezentují aplikační a nemodální okna jsou děděny z *DetectiveWindowBase*.



Obrázek 5. Obrázek popisuje princip dokování. Hlavní okno, případně jiná dokovací oblast, se rozdělí v mřížkové struktuře na podčásti, které jsou schopné hostit zadokovaný pohled. Tímto způsobem je docíleno dynamického zobrazení obsahu přesně podle potřeb vyšetřovatele.

MVVM vzor , jehož je View–Modelová vrstva starající se o předzpracování modelových objektů určených pro zobrazení je základní pilíř aplikace. Tato VM vrstva má velmi podobné členění jako zobrazovací vrstva. Jsou rozlišeny bázové třídy pro okno – *DetectiveWindowViewModelBase*, panel – *DetectivePaneViewModelBase* a zapouzdřující model – *DetectiveDataEntityViewModelBase*, viz obázek 6. Dále pak je třeba rozdělit objekty na ty, které jsou svojí životností spjaty s životním cyklem – *DetectiveApplicationPaneViewModelBase* a potom s

jedním vyšetřováním – *DetectiveInvestigationPaneViewModelBase* a *DetectiveInvestigationDataEntityPaneViewModelBase*.



Obrázek 6. Diagram třídní dědičnosti bázových tříd reprezentujících View–Modely. Stejně jako u obrázku 4 je použito dělení na View–Modely reprezentující okno a dockovatelné oblasti **Pane** View–Modely. Dále pak je zajištěna kontrola životnosti a závislosti na modelu pro typy **DataEntity**, které váží svoji životnost právě k zapouzdřeným modulům.

Vytváření instancí tříd z vrstev View a View–Model je zajištěno pomocí Windsor Containeru. Každý modul, assembly, obsahuje implementaci rozhraní *IWindsorInstaller*, která zajistí korektní registraci všech typů, závislostí a zajistí správnou životnost objektu. Registrace probíhá při spuštění aplikace spolu s instanciací všech objektů, které jsou dostupné z kořene aplikace. Pro instanciaci typů, které jsou spjaté s modely, jsou použity automaticky generované abstraktní továrny.

Dependency Injection – aplikace je navržena tak, že propoj mezi View a View–Modelem je řešen též pomocí dependency injection konteineru, a to injektací instance patřičného objektu View do konstruktoru View–Modelu určeného pomocí deklarovaného rozhraní. Tento mechanismus zajistí, aby nedocházelo k referenční závislosti mezi modulem nesoucím View a jiným nesoucím View–Model. Dále pak je možné programově kontrolovat, zdali je možné konstruovat kompletní strom závislostí, či nějaká načtená komponenta není kompatibilní s aktuální konfigurací.

Kvůli požadavku na vysoce dynamické schopnosti aplikace načítat externě vytvořené moduly není možná statická typová kontrola jazyka během kompilace. Použití dependency injection kontejneru jsme schopni ji částečně nahradit a zajistit korektní instanciaci a inicializaci objektů a spravovat jejich životnost v rámci celé aplikace, či pouze vázané na určitý, jiný typ objektu.

Architektura z uživatelského pohledu je zvolena na základě podobnosti s vývojovými prostředími (IDE), kdy zpravidla ucelený blok práce (project) je zařazení do nějakého řešení (solution). Pro přizpůsobení forenznímu prostředí proto tyto celky nazýváme vyšetřování (investigation), které můžeme seskupovat pod jedním velkým celkem pracovního prostředí (workspace). Vyšetřovatel si zvolí s jakým vyšetřováním aktuálně pracuje a přidá soubory se zachycenou komunikací, spustí extrakce informací a provede analýzu. Během práce může přidat další soubory a extrahovat další informace. Mezi jednotlivými vyšetřováními

je možné se v rámci jednoho pracovního prostředí rychle přepínat a ponechat si otevřená jednotlivá zobrazení pro korelaci získaných informací. Všechna data jsou fyzicky uložena jako součást jednoho pracovního prostředí, konkrétně v podsložce s vyšetřováním, a to včetně databáze. Jednotlivé pracovní prostředí je možné mezi stroji přenášet a následně pomocí souboru popisujícího pracovní prostředí (*.nfw) otevřít v nainstalovaném nástroji.

Analyzátory , jakožto plug-in rozšíření, je možné vyvíjet nezávisle na samotném nástroji Netfox Detective pouze za dodržení konvencní pro bázové typy View a View–Modelových tříd a instálatoru těchto typů do dependency injection kontejneru. Analyzátory můžou být spjaté s aplikací, jako je *SipFraud* analyzátor, nebo s vyšetřováním jako je *AppIdent* analyzátor. Analyzátory mají přístup ke všem typům, které jsou registrované v dependency injection konteineru a mohou rozšiřovat funkcionalitu nástroje, nebo jej dokonce ovládat.

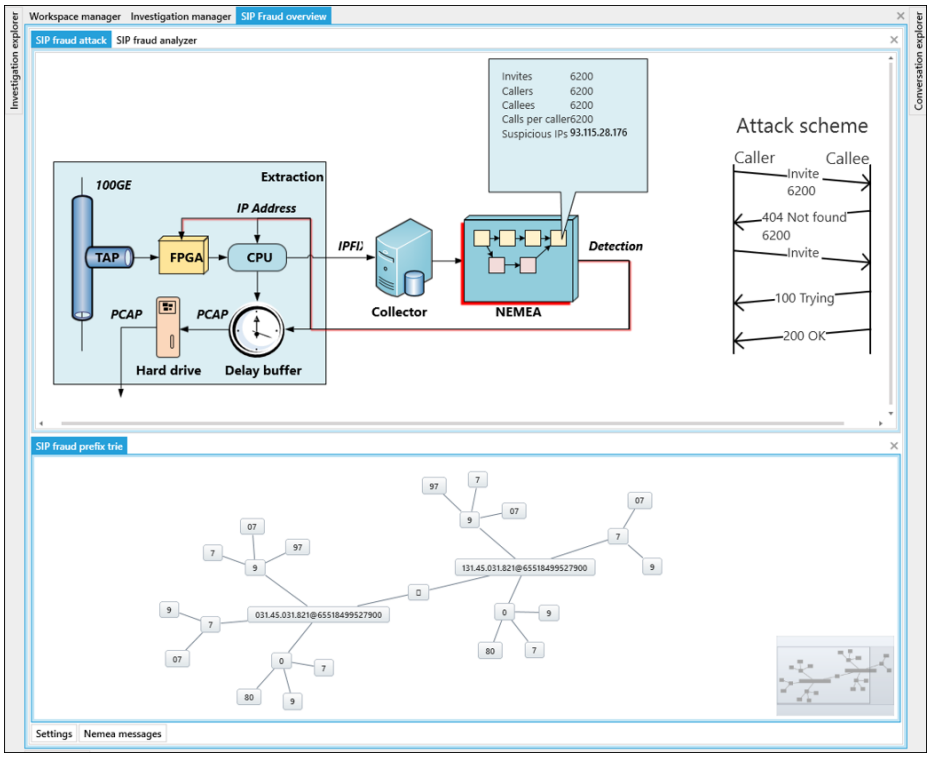
SipFraud analyzátor byl vyvinutý ve spolupráci se zájmovým združením CESNET z.s, aby pomocí jejich systému na detekci anomálií z netflow dokázal ověřit, že zachycená komunikace podezřelá z provádění útoku *SIP Fraud* jej opravdu obsahuje. Analyzátor komunikuje s NEMEA systémem [2], který mu oznámí stav ve kterém se nachází. Při detekci útoku umožní analyzátoru si stáhnout soubor s právě zachycenou komunikací a analyzáto převezme kontrolu nad nástrojem Netfox Detective tak, že přidá stažený PCAP soubor, extrahuje aplikační informace týkající se SIP a RTP protokolů a zobrazí analytický pohled. Tento pohled přehledně zobrazí všechny SIP zprávy, umožní filtrování a vypočítá strom volaných prefixů, aby se vyšetřovatel rozhodl, zdali se opravdu jedná o detekovaný útok. Ukázka takové analýzy je ve zjednodušeném pohledu viditelná na obrázku 7.

AppIdent analyzátor byl vyvinut pro ověření výkonosti nově vyvinuté metody vylepšující již existující statistické metody (ESPI – Enhanced Statistical Probability Identification) eliminovaného feature setu a porovnání s Bayesovským a Random Forests klasifikátory. Detaily o výzkumu spolu s implementací, naměřenými výsledky a jejich porovnáním jsou veřejně přístupné[6][8]. Výzkum ukázal, že je možné identifikovat nejen aplikační protokoly, ale za předpokladu, že máme k dispozici dostatečné množství ténovacích dat také i aplikace, které danou komunikaci vygenerovali.

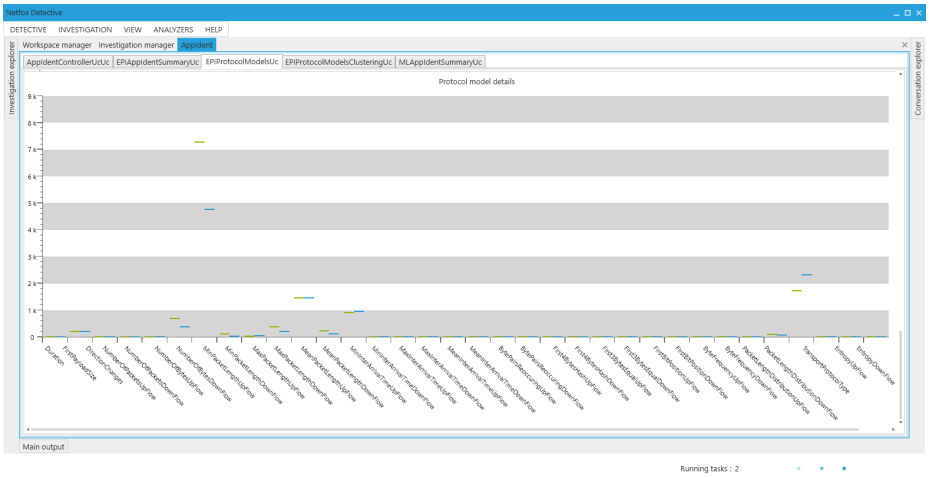
3 Porovnání existujících nástrojů pro forenzní analýzu

Síťová forenzní analýza je vysoce komplexní proces, jehož výsledná kvalita je velmi ovlivněna expertní znalostí vyšetřovatele, časovému fondu na vyřešení případu a nástrojům, které vyšetřovatel k získání výsledků využije. Pokud je expertní znalost vysoká a zároveň dostatečný čas na provedení detailní analýzy, získané výsledky budou velmi kvalitní. Ve skutečnosti se ale naopak setkáváme

[6] https://pluskal.github.io/AppIdent/



Obrázek 7. Snímek obrazovky analyzátoru *SIPFraud* zobrazující průběh analýzy zachytávané komunikace pomocí NEMEA detekčního nástroje. Ve spodní části je pohled na již nalezený incident a zobrazení stromu prefixů, kdy se útočník pokoušel hádat číselný prefix, který se používá k volání do PSTN (publik switched telephone network).



Obrázek 8. Ukázka vizualizace naučeného modelu *HTTP* protokolu, který byl získán pomocí analyzátoru *AppIdent*. Pohled zobrazuje na x ose popisky jednotlivých vyextrahovaných vlastností získaných z analýzy zachycené komunikace v učící fázi algoritmu *ESPI – Enhanced Statistical Probability Identification* a pomocí boxplot je zobrazuje do oboru hodnot na ose y.

s případy, kdy na vyřešení je pouze velmi omezený čas a expertní znalost už nehraje takovou roli, protože vyšetřovatel je nucený sáhnout po automatizovaném nástroji, který mu prezentuje zjištěné informace a vyšetřovatel pouze koriguje směr, kterým se bude ubírat.

Vysoce expertní vyšetřovatel s dostatkem času sáhne po nástrojích, které jsou velmi jednoduché, jednoúčelové, což mu umožní elegantní práci při manuální analýze. Vyšetřovatel tak bude mít naprostou kontrolu nad prováděnými operacemi, protože svým pochopením problému a zvolením vhodného jednoúčelového nástroje minimalizuje komplexitu a nestane se, že by nástroj ignoroval data, která nedokáže interpretovat, protože vyšetřovatel díky své expertní znalosti zvolil právě takový nástroj, který je nejvhodnější. Tento proces je však extrémně časově náročný, data týkající se případu mohou být obrovská a vyšetřovatel se stane naprosto zahlceným. Nástroje, které takto expertní vyšetřovatel používá jsou např. *TCPDump*, *ncat*, *SSLSplit*, *TShark*, *Wireshark* a další IDS a IPS systémy různé komplexity.

Kvůli tlaku na provedení analýzy co možná největšího množství dat za co nejkratší čas je častokrát vyšetřovatel nucen sáhnout po automatizovaném nástroji, který je přímo určen pro síťovou forenzní analýzu a je schopný provést extrakci užitečných informací ze zachyceného síťového provozu s co možná nejnižšími nároky na expertní znalosti vyšetřovatele a čas strávený vyšetřovatelem při analýze výstupů. Moderní forenzní nástroje došly tak daleko, že dokonce vytváří tiskové sestavy, které jsou poté přímo použitelné jako důkazy při soudním jednání. Snad nejznámějším zástupcem této kategorie nástrojů je *EnCase* pro di-

gitální forenzní analýzu. Naopak, pro síťovou forenzní analýzu můžeme uvažovat nástroje jako je *XPlico*, *Network Miner*, *PyFlag* a právě náš *Netfox Detective*. Porovnání vlastností nástrojů je uvedeno v tabulce 1.

Nástroj / Vlastnost	**Netfox Detective**	**Network Miner**	**XPlico**	**PyFlag**
Živý záchyt dat	NE	ANO	ANO	NE
Podpora PCAP	libPcap, pcap-ng, mnm	libPcap, pcap-ng	libPcap	libPcap
IPv6	ANO	ANO	ANO	NE
Tunelovaci protokoly	GRE, GSE	GRE, 802.1Q, PPPoE, LLMNR, VXLAN, OpenFlow, SOCKS, MPLS and EoMPLS	L2TP, VLAN, PPP	NE
OS Fingerprinting	ANO, na zaklade typickych aplikaci	ANO	NE	NE
Identifikace aplikacnich protokolu	SPID, NBAR, ESPI, Bayessian, Random Forests	SPID, PIPI	PIPI	NE
Identifikace bezicich aplikaci	ANO	NE	NE	NE
Aplikacni protokoly	HTTP, SSL/TLS, MAFF. XMPP, YMSG, OSCAR, Facebook Messanger, Hangouts, Twitter, XChat, IMAP, POP3, SMTP, Gmail, Yahoo, RTP, SIP, Minecraft, Warcraft, Facebook, Stratum, DNS, FTP, SPDY, MQTT	FTP, TFTP, HTTP, SMB, SMB2, SMTP, POP3, IMAP, YouTube	HTTP, POP3, SMTP, IMAP, SIP, RTP, SDP, FTP, DNS, IRC, IPP, PJL, MMS, SLL	DNS, HTTP, MSN, Gmail
Extrakce Oprávnění	Facebook, IMAP, SMTP, POP3	SMTP, HTTP Digest Autentizace	NE	NE
Zpracovani poskozene a neuplne komunikace	TCP vypadky, IPv4 fragmentace	?	NE	NE
Rychlost zpracování	33 MB/s	1.49-2.31 MB/s	?	?
Paralelní zpracování	ANO	NE	ANO	NE
Pokročilé analytické pohledy	ANO	NE	ANO	NE
Perzistentní úložiště	SQL	CSV / Excel / XML / CASE / JSON-LD	SQL	VFS
Dotazování	samostatně nad SQL databází	vyhledávání klíčových slov	samostatně nad SQL databází	ANO
PCAP-over-IP	NE	ANO	ANO	NE

Tabulka 1. Přehledové porovnání nástrojů pro síťovou forenzní analýzu. Byly vybrány nejznámější open-source či zdarma poskytované nástroje. Údaje uvedené v této tabulce jsou získány z propagačních materiálů, dokumentací a studiem zdrojových kódů. Není vyloučené, že během vývoje se vlastnosti změní.

4 Závěr

Tato technická zpráva popisuje aktuální stav implementace síťového forenzního nástroje Netfox Detective ve verzi 2.0 navazujíce na výsledku předchozího bezpečnostního výzkumu MV - VG20102015022. Byly nastíněny jednotlivé výzvy, viz sekce 1, které vidíme jako aktuální a jejich řešení přínosné pro zlepšení stavu na poli moderních nástrojů pro síťovou forenzní analýzu. Jako reakci na tyto výzvy jsme inovovali nástroj Netfox Detective o námi navržená řešení.

Převážně jsme se zaměřili na urychlení zpracování, proto jsme inovovali architekturu nástroje, abychom odstranili nepotřebné závislosti mezi moduly samotnými a zajistili jak budoucí rozšiřitelnost, tak i škálovatelnost řešení. Dalším krokem vývoje tohoto nástroje bude využití této architektury při návrhu distribuovaného řešení založeného nad Docker containery. Abychom odstranili závislost na proprietárním operačním systému Windows a .NET 4.7 frameworku, bude nutné jádro této aplikace Netfox Framework přepsat, aby využívalo odlehčeného .NET Core 2.0 frameworku. Tento krok nám zajistí snadnou přenositelnost jak mezi platformami díky konteinerizaci, tak i použitím open-source bázového obrazu pro kontejner samotný.

Samostatná standalone aplikace Netfox Detective si ovšem závislost na Windows i plném .NET frameworku ponechá, kvůli absenci alternativy pro WPF. Ovšem přibude nová webová aplikace zajišťující alternativní funkcionalitu jako nynější Netfox Detective, pouze ve webovém prostředí. Tento krok zajistí využití již otevřeného prostředí Netfox Framework a zpřístupní ho na libovolné platformě.

Poslední plánovanou změnou v nadcházející iteraci projektu je inovace persistentního úložiště. Zkušenosti s aktuálním řešení identifikují persistentní úložiště jako úzké hrdlo, protože množství zpracovaných dat aplikací roste a úložiště musí být schopné na změny reagovat. Z tohoto důvodu bylo rozhodnut přechod z SQL databáze na NoSQL či Key-Value databázi. Určení konkrétního typu databáze a implementace bude záviset na výkonnostních testech navržených jako nejčastější operace v této aplikaci a otestované na kandidátních řešeních jako je *Redis*, *Apache Ignite*, *ArangoDB*, *RocksDB* a další.

Tento projekt je vyvíjený za podpory projektu ministerstva vnitra - Integrovaná platforma pro zpracování digitálních dat z bezpečnostních incidentů, VI20172020062 a programová část je volně dostupná ve formě zdrojových kódů ve veřejném repozitáři projektu na GitHubu[7].

[7] https://github.com/nesfit/NetfoxDetective

Odkazy

[1] D. Black et al. *An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3)*. RFC8014. Pros. 2016. URL: http://tools.ietf.org/rfc/rfc8014.txt.

[2] Tomas Cejka et al. „NEMEA: A framework for network traffic analysis“. In: *Network and Service Management (CNSM), 2016 12th International Conference on*. IEEE. 2016, s. 195–201.

[3] D. Farinacci et al. *Generic Routing Encapsulation (GRE)*. Tech. zpr. 2784. Updated by RFC 2890. Břez. 2000. URL: http://www.ietf.org/rfc/rfc2784.txt.

[4] Radek Hranický. „Podvržená data v počítačových sítích“. czech. Bakalářská práce. Brno, CZ: Vysoké učení technické v Brně, Fakulta informačních technologií, 2012. URL: http://www.fit.vutbr.cz/study/DP/BP.php?id=13201.

[5] Petr Matoušek et al. „Advanced Techniques for Reconstruction of Incomplete Network Data“. Angl. In: *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* 2015.157 (2015), s. 69–84. ISSN: 1867-8211. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10864.

[6] T. Narten et al. *Problem Statement: Overlays for Network Virtualization*. Tech. zpr. 7364. Říj. 2014. URL: http://www.ietf.org/rfc/rfc7364.txt.

[7] Jan Pluskal. „Framework for Captured Network Communication Processing“. Angl. Dipl. Brno, CZ: Vysoké učení technické v Brně, Fakulta informačních technologií, 2014. URL: http://www.fit.vutbr.cz/study/DP/DP.php?id=16748.

[8] Jan Pluskal, Ondrej Lichtner a Ondřej Ryšavý. *Netfox Detective - Identifikace aplikačních protokolů pomocí algoritmů strojového učení*. czech. Tech. zpr. FIT-TR-2017-05, CZ, 2017, s. 19. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11566.

[9] J. Postel. *Internet Protocol*. Tech. zpr. 791. Updated by RFCs 1349, 2474, 6864. Zář. 1981. URL: http://www.ietf.org/rfc/rfc791.txt.

[10] J. Postel. *Transmission Control Protocol*. Tech. zpr. 793. Updated by RFCs 1122, 3168, 6093, 6528. Zář. 1981. URL: http://www.ietf.org/rfc/rfc793.txt.

[11] J. Postel. *User Datagram Protocol*. Tech. zpr. 768. Srp. 1980. URL: http://www.ietf.org/rfc/rfc768.txt.

[12] H. Song et al. *Peer-to-Peer (P2P) Overlay Diagnostics*. RFC7851. Květ. 2016. URL: http://tools.ietf.org/rfc/rfc7851.txt.

[13] F. Templin. *The Internet Routing Overlay Network (IRON)*. Tech. zpr. 6179. Břez. 2011. URL: http://www.ietf.org/rfc/rfc6179.txt.