FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Využití virtualizace pro zabezpečení systému automatizovaného hodnocení úloh z programování
Student:	Jan Kuběna
Vedoucí:	Mgr. Jakub Růžička
Studijní program:	Informatika
Studijní obor:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Popište obecné problémy spojené se spouštěním cizího kódu z pohledu bezpečnosti. Analyzujte možné využití virtualizace pro řešení těchto problémů. Porovnejte bezpečnost a použitelnost řešení využívající virtualizaci oproti stávajícímu řešení (program Sandboxie) používaného v systému automatizovaného hodnocení domácích úloh z programování SharpTest, jehož funkční prototyp byl vytvořen v rámci projektů BI-SP1/2. V návaznosti na to proveďte průzkum a výběr vhodného virtualizačního nástroje a jeho následnou integraci do stávajícího systému. Otestujte systém pomocí integračních testů, včetně testů předpokladů zabezpečení virtualizovaného řešení.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 17. prosince 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Využití virtualizace pro zabezpečení systému automatizovaného hodnocení úloh z programování

Jan Kuběna

Katedra počítačových systémů
Vedoucí práce: Mgr. Jakub Růžička

30. července 2020

Poděkování

Rád bych touto cestou poděkoval Mgr. Jakubovi Růžičkovi za jeho cenné rady, podporu a připomínky při vedení této práce. Dále bych chtěl poděkovat své rodině, bez které bych studium nezvládl a mým přátelům i kolegům z práce, zejména však Jiřímu Zikánovi a Davidu Hornofovi za veškeré konzultace.

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. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. července 2020 .....

Odkaz na tuto práci

Kuběna, Jan. *Využití virtualizace pro zabezpečení systému automatizovaného hodnocení úloh z programování.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Cílem této práce je využít virtualizaci k zabezpečení existujícího systému SharpTest, což je systém pro automatizované hodnocení domácích úloh z programování. Zabezpečení tohoto systému obnáší zejména bezpečné spouštění nedůvěryhodného kódu v testovacím prostředí a separaci systému do izolovaných částí. V první části práce jsou popsány problémy se spouštěním nedůvěryhodného kódu a je provedena rešerše technologií virtualizace. Také je provedena analýza systému SharpTest, zabývající se zejména částí systému starající se o spouštění nedůvěryhodného kódu a automatizované hodnocení úloh. V druhé části práce je navrženo řešení využívající hardwarovou virtualizaci na virtualizační platformě XCP-ng s hypervisorem Xen a je proveden návrh její integrace do systému SharpTest. Tento návrh je realizován a jeho správná integrace je otestována, příčemž jsou vytvořeny i automatizované testy.

Klíčová slova virtualizace, nedůvěryhodný kód, bezpečné spouštění kódu, automatizované hodnocení programovacích úloh, Xen, XCP-ng

Abstract

The aim of this thesis is to utilize virtualization in order to secure an existing solution SharpTest (a system for automated evaluation of programming tasks). Main focus is on secure execution of untrusted code in testing environment and separation and isolation of the system's components. The first part of this thesis introduces problems with execution of untrusted code followed by literature review on virtualization technology. SharpTest is also analyzed with emphasis on the components responsible for untrusted code execution and automated evaluation. In the latter part of this thesis, a solution utilizing hardware virtualization on virtualization platform XCP-ng with hypervisor Xen and it's integration into SharpTest is introduced. Proposed solution is then implemented, integrated and finally tested, with automated tests included as well.

Keywords virtualization, untrusted code, safe code execution, automated evaluation of programming tasks, Xen, XCP-ng

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Automatizované hodnocení programovacích úloh může sloužit jako efektivní způsob procvičování syntaxe, algoritmů nebo ošetřování vstupů. Takovýto systém dokáže učiteli ušetřit hodiny opravování studentských úloh a student se zároveň mnohem rychleji dozví, zda úlohu vyřešil správně. Pro automatizované hodnocení úloh je však většinou potřeba zásadní funkcionalita – bezpečné spouštění cizího kódu. Z toho bohužel plyne řada problémů, protože cizí kód může provádět celou řadu škodlivých činností, ať už jde o ovlivnění výsledků testů, ovládnutí stroje na kterém je kód spuštěn nebo případnou krádež osobních dat studentů.

Tyto problémy pak vyžadují robustní vícevrstvou bezpečnost, která zahrnuje například dobrý návrh samotného systému spouštění jednotlivých testů, správné nastavení přístupových práv do databází a jiných integrovaných služeb, předávání pouze nejmenšího nutného množství informací a nebo právě spouštění kódu v bezpečném prostředí. Virtualizace zde nabízí velice dobré bezpečnostní vlastnosti, protože kód, který je spouštěn ve virtuálním stroji, má značně omezené možnosti přímého ovlivnění hosta, tedy fyzického stroje, na kterém je virtuální stroj spuštěn. Navíc může host virtuální stroj libovolně vypínat/zapínat, přemazávat a jasně určovat všechny přidělené prostředky[1]. Lze pak velice dobře oddělit jednotlivé části procesu automatizovaného hodnocení, což může drasticky snížit počet potenciálních vektorů útoků.

Cílem této práce je zejména vytvoření nového bezpečného prostředí pro spouštění cizího kódu pro systém automatizovaného hodnocení úloh z programování SharpTest, jehož funkční prototyp byl vytvořen v rámci projektů v předmětech BI-SP1/2, ale i oddělení jednotlivých částí tohoto systému. Jelikož má tato práce vylepšit systém SharpTest, je jejím cílem i provedení porovnání možných výhod a nevýhod virtualizace oproti stávajícímu řešení využívající sandbox Sandboxie. Bude proveden výběr vhodné virtualizační platformy, která bude poskytovat potřebné funkcionality pro bezpečný běh programu a tato platforma bude plně integrována do systému SharpTest.

Pomocí virtualizace pak bude nahrazeno stávající testovací prostředí a bude navržen model nasazení systému SharpTest ve virtualizovaném prostředí. U takto integrovaného řešení pak budou provedeny a vytvořeny automatizované testy ověřující jeho funkčnost a předpoklady pro jeho zabezpečení.

Kapitola 1

Problémy se spouštěním nedůvěryhodného kódu

Nedůvěryhodný kód je takový kód, který je získán z nedůvěryhodného zdroje a/nebo nedůvěryhodnou cestou a u kterého nevíme, co přesně dělá[2]. Cizí kód, tedy kód pocházející z neznámého zdroje, by pak měl být automaticky považován za nedůvěryhodný a to i přes to, že neplatí, že nedůvěryhodný (a tedy i cizí) kód je nutně škodlivý. S nedůvěryhodným kódem se mohou setkat jak běžní uživatelé, když se jim při prohlížení webových stránek stahují a spouštějí skripty v jejich prohlížeči, tak například hostingové služby provozující zákaznické aplikace. Nedůvěryhodný kód je však často spouštěn i vědomě za účelem analýzy jeho chování.

Jedním z největších problémů nedůvěryhodného kódu je to, že u dostatečně komplexního kódu je těžké bez spuštění přesně určit, zda bude mít jeho spuštění negativní dopady. Při zkoumání nedůvěryhodného kódu se využívá statická a dynamická analýza. Statická analýza je prováděna přímo nad kódem samotným například detekcí podezřelých kombinací užívaných systémových funkcí nebo porovnáním signatury kódu se známými databázemi. Dynamická analýza spočívá ve spuštění kódu v omezeném prostředí a následném sledování jeho chování a vnitřních stavů[3].

Existují mechanismy, které se snaží kód udělat více důvěryhodným, ať už se jedná o zajištění bezpečné komunikační cesty, ověření identity odesílatele nebo elektronický podpis samotného kódu. I tyto mechanismy však mají potencionální slabé články[4, 5] a samotná hranice, kdy se kód stává důvěryhodný není přesně specifikována. V některých případech je jako nedůvěryhodný kód označován veškerý kód spuštěný uživatelem a za důvěryhodný kód považován jen operační systém [6].

Bezpečné spouštění nedůvěryhodného kódu je důležitý a zároveň obtížný úkol. Navíc se takovýto kód šíří v různých formách, od prostého textu přes binární reprezentaci až po jeho zakódovaný ekvivalent v obrázku[7] nebo souboru PDF. V této kapitole jsou na příkladech uvedeny možné dopady spou-

štění nedůvěryhodného kódu. Dále jsou analyzovány skutečnosti umožňující vznik těchto negativních dopadů a popsány potřebné prerekvizity k minimalizaci a případné eliminaci zmíněných dopadů.

1.1 Dopady spouštění nedůvěryhodného kódu

Při spouštění nedůvěryhodného kódu bez jakékoliv ochrany je systém a jeho uživatel vystaven mnoha rizikům. Spouštěný kód totiž může být malware (Malicious Software), tedy záměrně škodlivý software. Malware může způsobit značné škody jak jednotlivým uživatelům, tak velkým firmám. Mezi běžné dopady spuštění malwaru, které jsou navíc často kombinovány, patří[8]:

Stahování dalšího malwaru v závislosti na dalších úmyslech útočníka.

Vydírání, které se na popředí dostalo hlavně v poslední době ve formě různých zamykačů (locker). Takovýto malware znemožní přístup k datům nebo k celému systému a uvolní je až po zaplacení výkupného.

Vzdálené ovládání za účelem vytvoření botnetu, tedy sítě vzdáleně ovládaných počítačů, nebo dlouhodobější infiltrace pomocí zadních vrátek do systému.

Krádež citlivých dat a identity, která může probíhat zachytáváním kláves, prohledáváním souborů nebo aplikováním různých technik sociálního inženýrství přímo skze počítač oběti.

Persistentní setrvání malwaru na počítači, a to někdy i přes pokusy o jeho odstranění.

Obecně lze říci, že spuštění malwaru bez jakékoliv ochrany může způsobovat vážné škody, které závisí hlavně na kreativitě a cíli tvůrce tohoto malwaru.

1.1.1 Přístup k systémovým prostředkům

Všechny procesy, tedy spuštěný kód, v operačním systému vyžadují určité kvantum systémových prostředků, se kterými pak operují a snaží se dosáhnout nějakého cíle. Takovýmito prostředky může být procesorový čas a paměťové místo, ale také sockety pro síťovou komunikaci nebo přístup k souborovému systému. Přístup k těmto prostředkům je pak omezen podle práv, se kterými byl proces spuštěn a podle dalšího nastavení systému, například firewallu. Omezením přístupu k systémovým prostředkům je možné limitovat malware v tom, jak velké škody může napáchat.

1.1.2 Vyčerpání systémových prostředků

Vytížením jednoho či více systémových prostředků vzniká jejich nedostatek pro ostatní procesy, které tyto prostředky potřebují pro své správné fungování. To může vyústit v částečné omezení funkčnosti systému jako celku nebo i k jeho kolapsu.

Prevence proti takovému vytížení je omezení množství prostředků, které může proces využívat. Ne vždy je však možné toto nastavit ve všech operačních systémech pro určitý proces a pro všechny systémové prostředky.

1.1.3 Škodlivá manipulace se soubory

Z hlediska bezpečnosti je přístup k souborům problematický a nabízí velké množství potencionálních vektorů pro útok. Jako známý příklad lze uvést třeba CryptoLocker, což je škodlivý software, který zašifruje uživatelské soubory a požaduje výkupné. I prostý malware vyhledávající a odesílající citlivé firemní dokumenty nebo zdrojové kódy na počítači oběti může v závislosti na citlivosti souborů způsobit značné škody. S dostatečnými právy přístupu k souborům pak lze často zajistit i persistenci, tedy přežití škodlivého software po restartu, a to třeba nahrazením již zavedeného programu spouštěného po restartu, přidáním škodlivých maker do uživatelských souborů a podobně.

Obecně se dá říct, že by proces měl by mít přidělená co nejrestriktivnější práva tak, aby stále mohl vykonávat jen a pouze zamýšlenou funkci. Lze přidat i další vrstvy ochrany, jakou je například záloha dat a zaznamenávání přístupů k souborům. Jednou z nejúčinnějších metod je však citlivé soubory vůbec v systému nemít, což však není vždy možné.

1.1.4 Ovládnutí operačního systému

Jedním z problémů může být jakákoliv nechtěná kontrola nad operačním systémem (OS), ve kterém je kód spuštěn. Negativní dopady takovéto kontroly pak závisí zejména na tom, které prostředky a soubory jsou útočníkovi přístupné.

Ovládnutí OS lze dosáhnout dosáhnout mnoha způsoby, přičemž efektivní obranou je udržovat OS aktualizovaný a spouštět nedůvěryhodný kód s co nejmenšími právy nebo nejlépe vůbec.

V případě přístupu k internetu může potencionální útočník ovládat počítač i vzdáleně, což mu umožňuje lépe interagovat s počítačem oběti. Může pak zcizit vybraná citlivá data nebo se snažit o další zvýšení práv v OS a infikování dalších počítačů na lokální síti-.

Zamezení vzdáleného ovládání pak vyžaduje omezení přístupu k internetu. Takovéto omezení může být úplné nebo na úrovni vybraných IP adres. Lze ho aplikovat jak na úrovni OS, tak vzdálenějších síťových prvků, které pak mohou analyzovat i obsah komunikace.

1.1.5 Rozšíření do lokální sítě

Přístup k lokální síti může útočníkovi poskytnout další cíle pro své následující útoky. Navíc se často stává, že je většina ochran zaměřena na perimetr lokální sítě a samotné zabezpečení a separace počítačů v lokální síti už není prioritou.

V lokální síti spolu většinou počítače komunikovat potřebují, tuto komunikaci však lze omezit. Z pohledu OS to může být správné nastavení firewallu. Na úrovni síťových prvků pak lze komunikaci filtrovat po rozdělení počítačů do vlastních podsítí nebo nastavení VLAN (Virtual Local Area Network).

1.2 Omezení negativních dopadů

Negativní dopady lze omezit použitím doporučených strategií, postupů a nástrojů. Tato doporučení by měla ovlivňovat rozhodování o použitých technologiích a postupech.

Obrana počítačových systémů před útoky musí postupovat tak, aby byla vytvořená bezpečnostní strategie robustní a zároveň nabízela útočníkům co nejmenší prostor pro útoky. Výsledkem je snaha co nejlépe zabezpečit všechny aspekty serverů a aplikací, ať ze strany technické či procesní. Vynaložená snaha na obranu však nemusí být vždy přímo úměrná nutné vynaložené snaze na provedení útoku. Útočníci si totiž mohou vybírat místo a čas na útok a často jim stačí najít jeden slabý článek, který pak naruší bezpečnost všech systémů. K zabezpečování je tedy nutné přistupovat systematicky, správně analyzovat kritická místa pro případný útok a ty se snažit zabezpečit.

1.2.1 Defense In Depth

Defense In Depth je strategie, při které je rozkládáno zabezpečení do více vrstev. Pokud pak jedna vrstva selže, další vrstvy toto selhání kompenzují. Tato kompenzace pak musí být dostatečně dlouhá na to, aby byl případný útok zaregistrován a byla na něj provedena odpovídající reakce[9]. Konkrétní rozdělení vrstev není pevně dáno, ale pro představu se může jednat o:

Fyzickou bezpečnost zajišťující omezení přístupu k hardware (HW).

Vnější síťový perimetr filtrující komunikaci z a do internetu.

Vnitřní síť oddělující jednotlivé servery, jejich aplikace. Může aktivně detekovat podezřelou komunikaci.

Operační systém s lokálním firewallem, AV a zapnutým logováním.

Aplikace správně validující příchozí data a ukládající záznamy o přístupech.

Data, které mohou být zálohovány nebo šifrovány.

Pří potencionálním útoku se tak útočník musí probojovat skrze všechny vrstvy, což ho stojí čas a úsilí.

1.2.2 Princip minimálních privilegií

Princip minimálních privilegií (principle of least privileges) říká, že každá entita by měla mít nastavená právě taková práva, aby mohla plnit svou funkci a nic víc. Takovouto entitou jsou pak nejčastěji uživatelské účty, role a procesy[10, 11].

V OS je možnost omezení hlavně na úrovni uživatelů a jejich rolí, kterým je přiřazována celá sada práv. Tyto práva se týkají například přístupu k souborovému systému, registrům nebo změnám nastavení OS. Tyto omezené role pak mohou být využívány jak uživateli, tak službami spuštěnými pod určitým uživatelským účtem. Využití však není jen v OS samotném, ale i na něm běžících službách poskytujících API (Application Programming Interface). Tyto služby totiž mohou mít své vlastní implementace účtů a oprávnění.

Pro dodržení principu minimálních privilegií je nutné zmapování požadovaných práv, tedy třeba seznamu souborů nebo serverů, ke kterým je přistupováno. Při dodržení tohoto principu je pak případný útočník značně omezen v jeho dalším konání.

1.2.3 Aktualizace

Udržení provozovaných systémů aktuálních může být sice náročné, je však důležité, protože aktualizace často opravují nově nalezené bezpečnostní chyby. Aktualizace jsou ale často opomíjeny kvůli nutnosti ověřování funkčnosti nasazených systémů a aplikací.

1.2.4 Logování

Logování je zásadní složkou dobrého zabezpečení, protože pomáhá při zjišťování zdroje útoků. Takovýto zdroj pak lze rychle izolovat od dalších systémů. Samotný útok je s pomocí logů možné analyzovat a provést nápravy, které mu pro příště zabrání, nebo alespoň na podobný útok dříve upozorní.

1.2.5 Nástroje

Kromě strategií a principů je vhodné k posílení bezpečnosti používat i k tomu určené nástroje. Například Antivirus (AV) je software, který dokáže detekovat a odstraňovat malware na základě automatické statické a dynamické analýzy.

Dalším vhodným nástrojem je Intrusion Detection System (IDS), který monitoruje systém a/nebo síťový provoz a detekuje podezřelé aktivity. Na základě nich pak upozorňuje administrátora, který tyto podezřelé aktivity prověří a rozhodne o dalším postupu.

Posledním zmíněným typem nástrojů budou nástroje pro management bezpečnostních informací a událostí (SIEM), pomocí kterých lze soustředit logy na jednom místě a hromadně je vyhodnocovat. Toto soustředění a hro-

madné vyhodnocování umožňuje nasbírané informace propojovat a zároveň zajišťuje jejich uchování mimo napadený systém.

1.3 Negativní dopady pro systémy automatického hodnocení úloh z programování

Z výše zmíněných dopadů se systémů automatického hodnocení úloh z programování týká většina, konkrétně si ale takovýto systém musí poradit s následujícími hrozbami:

Krádež citlivých dat studentů, kterými mohou být jména, příjmení, studijní výsledky a podobně.

Ovládnutí serverů systému a zneužití ke kyberútokům na jiné počítače v lokální síti, ale i na internetu.

Získání referenčního řešení úlohy, která je použita pro kontrolu výstupu studentského řešení.

Omezení služby (Denial of Service) způsobující nefunkčnost systému na různě dlouhou dobu.

Ovlivnění výsledků ohodnocování úlohy tak, aby student dosáhl lepšího výsledku.

Obranu proti těmto hrozbám lze implementovat na více úrovních. Některých negativních dopadům lze předcházet dobrým návrhem tohoto systému pro automatické hodnocení, tedy třeba ukládáním pouze opravdu potřebných uživatelských dat. Tato práce je zaměřená na využití virtualizace jako součásti této obrany, a to zejména při bezpečném spouštění studentského kódu, ale i rozdělením takovéhoto systému na více izolovaných částí.

Kapitola 2

Virtualizace

Hlavní myšlenkou virtualizace je abstrahování fyzických prostředků počítače za účelem vytváření tzv. virtuálních strojů, které tyto prostředky v různých formách sdílejí[1]. Přidaná abstrakce umožňuje simulovat, omezovat a celkově velice dobře ovládat takto vzniklé virtuální prostředky.

První koncepty virtualizace se začaly objevovat již na konci 60. let v laboratořích IBM při vývoji mainframe CP-40, později CP-67. Na rozdíl od ostatních řešení své doby nenabízel jen sdílení prostředků mezi uživatelskými procesy, ale i separované virtualizované operační systémy[12]. Dnes již mohou virtualizaci využívat jak velké společnosti, tak jednotlivci a typů virtualizace je hned několik. Virtualizace nabízí zjednodušení správy serverů díky konsolidaci více malých fyzických serverů pro jednotlivé služby do jednoho fyzického serveru s více virtuálními stroji. To značně snižuje náklady na hardware samotných serverů, umožňuje velice rychlé nasazování dalších virtuálních strojů a zjednodušuje obnovu systémů při zotavování po haváriích[13].

V oblasti virtualizace se pak používají s následující pojmy[1, 14]:

Virtuální stroj, označován jako VM (Virtual Machine), obecně označuje virtuální prostředí, ve kterém jsou spuštěny virtualizované aplikace a operační systémy. Typicky je realizován sadou souborů, které popisují potřebnou konfiguraci prostředků virtuálního stroje.

Kontejner, také nazýván jail nebo zóna, je ekvivalentem virtuálního stroje ve virtualizaci na úrovni operačního systému.

Hostitel je označení pro fyzický stroj, na kterém je spuštěn hypervisor a virtuální stroje.

Hypervisor je součást virtualizační platformy, která se stará převážně o správu prostředků poskytovaných virtuálním strojům.

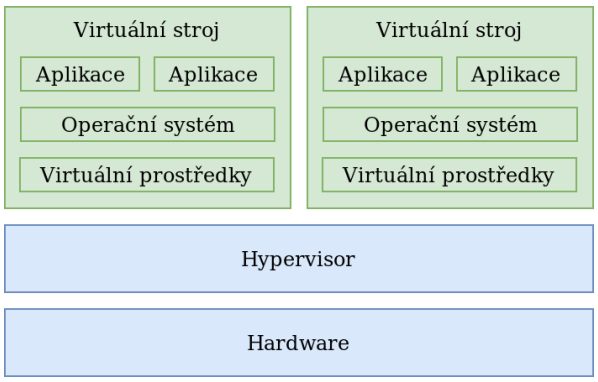
Virtualizační plaftorma je ucelené řešení pro virtualizaci, skládající se z hypervisoru a dalších podpůrných součástí, jako například nástrojů ke správě a uživatelského API.

2.1 Hypervisor

Hypervisor, někdy označován jako Virtual Machine Monitor, je jednou z hlavních komponent virtualizační platformy. Je to vrstva mezi hardwarem a virtuálním strojem, která se stará o přidělování prostředků a vyřizování požadavků od virtuálních strojů. Běžně se stará například o zprostředkování síťové komunikace nebo vstupně výstupní komunikace s disky. Může poskytovat i další pokročilé funkcionality, jako například zajištění vysoké dostupnosti[14]. Typicky se rozlišují dva typy hypervisorů:

Typ 1 je někdy označován také Native nebo Bare-Metal a je spuštěn přímo na hostitelově hardware, což mu umožňuje efektivněji spravovat systémové prostředky poskytované virtuálním strojům. Architekturu tohoto typu hypervisoru lze vidět na obrázku 2.1. Tento typ hypervisoru se využívá zejména při virtualizaci serverů a implementují ho například VMware ESXi, Hyper-V nebo Xen.

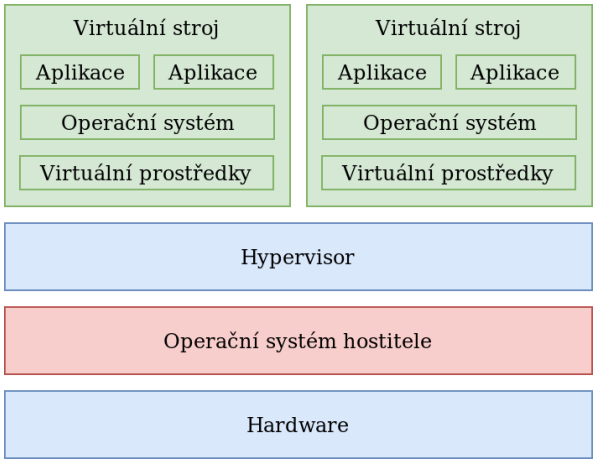
Obrázek 2.1: Hypervisor typu 1



Typ 2 je typicky proces spuštěný v operačním systému a tedy sdílí zdroje s ostatními procesy. Na rozdíl od typu 1 obsahuje další vrstu, a to operační systém hostitele, jak lze vidět na obrázku 2.2 Je vhodný zejména pro běžné uživatele, kteří chtějí virtuální stroje spouštět na běžných operačních systémech. Příkladem může být VirtualBox a QEMU.

Určení typu konkrétního hypervisoru dnes není však tak jednoduché, některé hypervisory totiž kombinují výhody obou typů.

Obrázek 2.2: Hypervisor typu 2



2.2 Typy virtualizace

K virtualizaci lze přistupovat hned několika způsoby v závislosti na tom, jaké jsou požadavky na rychlost, bezpečnost nebo operační systém ve virtuálním stroji. Virtualizační technologie se běžně dělí na HW virtualizaci a virtualizaci na úrovni operačního systému. HW virtualizaci pak představuje úplná virtualizace, paravirtualizace a HW asistovaná virtualizace a jejich kombinace.

2.2.1 Úplná virtualizace

U tohoto typu virtualizace dochází k úplné abstrakci veškerých fyzických prostředků a jejich oddělení od operačního systému bežícího ve virtuálním stroji[15]. Oddělení zajišťuje a řídí hypervisor, který se také stará o ošetření tzv. nebezpečných instrukcí, což jsou instrukce jejichž vykonání není žádoucí. Takovéto instrukce musí hypervisor nahradit za jejich bezpečné ekvivalenty. Hypervisor také emuluje legacy boot (start v 16-bitovém módu), přerušení, podporu pro správu paměti a vstupně-výstupní zařízení jako jsou disky a síťové karty. Tato úplná emulace je však poměrně výkonově náročné[16]. OS uvnitř VM pak ale může běžet bez dalších úprav (na rozdíl od paravirtualizace) a VM jsou od sebe velice dobře izolované.

2.2.2 Paravirtualizace

Paravirtualizace se od plné virtualizace liší tím, že do OS VM jsou přidány speciální ovladače zařízení, které pak komunikují přímo s hypervisorem. Díky těmto ovladačům je pro hypervisor snazší ošetřit některé z nebezpečných instrukcí[17], což vede ke zrychlení zejména vstupně výstupních operací. Jistou nevýhodou je právě nutnost modifikace jádra operačního systému virtuálního stroje. Většina virtualizačních platforem s podporou paravirtualizace nabízí

paravirtualizační ovladače pro běžně používané operační systémy. Problém nastává hlavně při použití starších a nepodporovaných operačních systémů.

2.2.3 HW asistovaná virtualizace

HW asistovaná virtualizace, někdy zvaná „nativní", využívá podpory virtualizace přímo v procesoru. Konkrétně se jedná o technologie VT-x pro procesory od firmy Intel a AMD-v pro procesory od firmy AMD. Obě tyto technologie se zaměřují na úzké hrdlo, kterým je ošetřování nebezpečných instrukcí hypervisorem, a to přidáním nových instrukcí určených specificky pro virtualizaci.

2.2.4 HW asistovaná s paravirtualizovanými drivery (PVHVM)

Tento typ virtualizace je spojením HW asistované virtualizace a paravirtualizace. Rozdíl oproti HW asistované virtualizaci je v tom, že ve VM jsou nainstalovány speciální ovladače podobně jako při paravirtualizaci. Díky tomu je typ PVHVM rychlejší než samotná HW asistovaná virtualizace.

2.2.5 Virtualizace na úrovni operačního systému

Virtualizace na úrovni OS je na rozdíl od předchozích typů založená na principu virtualizace OS systému namísto virtualizace fyzických prostředků. Jelikož nejsou virtualizovány přímo fyzické prostředky, používá se místo označení virtuální stroj například pojem kontejner, zóna nebo jail, v závislosti na konkrétních použitých technologiích. Virtualizace na úrovni OS je závislá na podpoře v operačním systému hostitele, protože kontejnery sdílí stejné jádro OS. Tato podpora umožňuje sdílený přístup k fyzickým prostředkům, ale zároveň zajišťuje separaci jednotlivých kontejnerů. Díky sdílenému jádru operačního systému pak mají kontejnery oproti virtuálním strojům menší nároky zejména na paměť, ale i na procesor. Místo celého operačního systému tedy v kontejneru běží jen aplikace, jak lze vidět na obrázku 2.3.

2.3 Virtualizace sítí

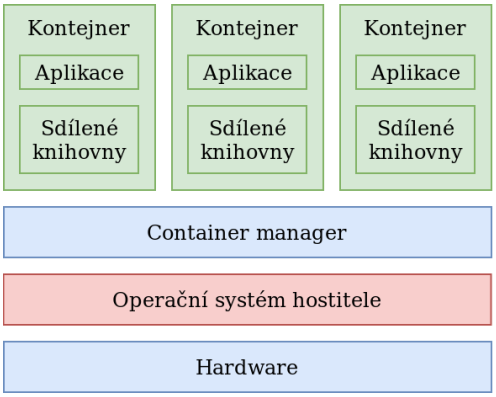
Jak už bylo naznačeno, hypervisor musí umět virtualizovat i síťová zařízení, aby mohlo VM komunikovat mezi sebou, ale i se servery mimo server hostitele[14]. Základními prvky, které lze nalézt i na obrázku 2.4 jsou:

Virtuální síťové rozhraní, označováno jako VIF (Virtual Network Interface) je síťové rozhraní na straně virtuálního stroje, které virtuální stroj používá k síťové komunikaci.

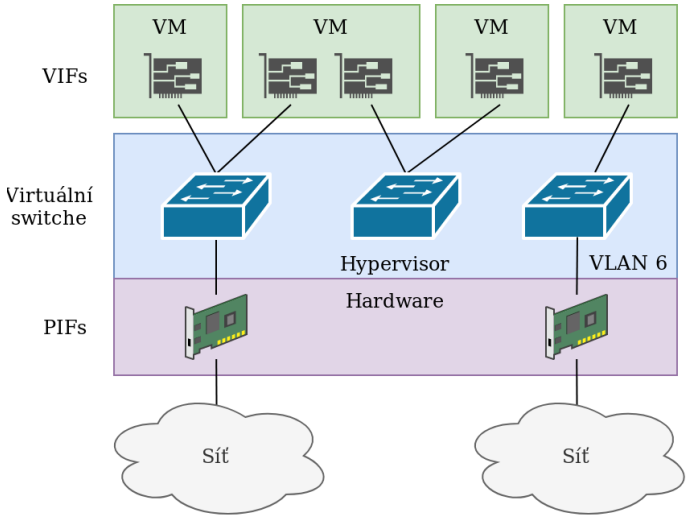
Fyzické síťové rozhraní, označováno jako PIF (Physical Network Interface) je síťové rozhraní hostitele, které může být použito pro zprostředkování komunikace z VM do vnější sítě.

Virtuální síť je realizována virtuálním switchem, který je většinou součástí hypervisoru. Propojuje více VIF a jejich VM a případně také připojuje PIF.

Obrázek 2.3: Virtualizace na úrovni operačního systému



Obrázek 2.4: Ilustrace virtuálních zařízení, sítí a jejich propojení s hypervisorem



Většina virtuálních switchů podporuje i tzv. VLAN tagging, kterým se dá označit komunikace vycházející z virtuální sítě přes PIF do sítě vnější. Využívá se zejména pro směrování a filtrování síťovými prvky. Jelikož je komunikace

v dnešním světě kritická, požadujeme po virtuálních sítích hlavně rychlost, stabilitu a dobrou izolaci virtuálních sítí.

Některými z dalších funkcionalit virtualizace sítí mohou být:

- Speciální virtuální síť (tzv. Management Network) pro komunikaci VM s hostitelem.
- Omezení přidělených IP adres VM na úrovni VIF.
- Propojení virtuálních sítí mezi více hostiteli.
- Spojení více PIF do jednoho (tzv. bonding) za účelem zvýšení propustnosti.

2.4 Virtualizace úložiště

Virtualizace úložiště je podobně jako virtualizace sítě většinou řízena hypervisorem, který zpřístupňuje dané fyzické prostředky. Díky virtualizaci lze jako úložiště pro disky VM použít například síťového úložiště NAS (Network Attached Storage). Dále umožňuje důležité funkcionality, jako snapshoty nebo tzv. thin provisioning, který umožňuje na hostiteli alokovat pouze aktuálně využité místo ve VM. Není tedy potřeba dopředu alokovat veškeré místo, které bylo virtuálnímu disku VM přiděleno.

2.5 Snapshot

Snapshot je obrazem stavu VM v okamžiku jeho vytvoření a jeho obsahem jsou všechny prvky nutné ke zpětné rekonstrukci tohoto stavu. S pomocí snapshotu lze jednoduše vrátit změny provedené ve VM na všech úrovních ať už to je konfigurace na straně virtualizační platformy nebo obsah disku a paměti uvnitř VM. Zpravidla snapshot obsahuje zejména[18]:

- Obsah disků.
- Obsah paměti (pokud byl stroj v době snapshotu zapnut).
- Konfiguraci.
- Stav VM (zapnutý/pozastavený/vypnutý).

V některých virtualizačních platformách, konkrétně například v XCP-ng a XenServer, se také rozlišuje pojem snapshot a checkpoint. Snapshot je v těcho virtualizačních platformách bez obsahu paměti a checkpoint je s obsahem paměti VM v době vytvoření checkpointu.

2.6 Virtualizace pro bezpečnost

Samotná virtualizace bezpečnost nezajistí, ale v kombinaci s dalšími opatřeními se z ní stává efektivní nástroj, kterým lze zvýšit úroveň celkové bezpečnosti. Virtualizaci lze použít zejména pro[19]:

- Logickou separaci a izolaci operačních systémů a jejich aplikací.
- Jasné rozdělení systémových prostředků.
- Rychlou obnovu při infikování VM/kontejnerů.
- Definování komunikačních cest díky virtuálním sítím.
- Zjednodušení zálohování.

Toto lze využít k rozdělení komplexního systému do více částí, kde každá část může mít jasně definované prostředky a přístupy přes síť, které může využívat. K jednotlivým částem pak lze přistupovat jinak podle jejich důležitosti. To může znamenat například častější zálohování, přidělení většího či menšího kvanta systémových prostředků nebo striktnější omezení přístupu k serverům s citlivými daty.

Virtualizace však s sebou přináší i nová bezpečnostní rizika, kterými může být absence fyzické separace, sdílení úložišť mezi více VM/kontejnery nebo falešný pocit bezpečí způsobený použitím virtualizace.

2.7 Sandbox a virtualizace

Sandbox je omezené izolované prostředí separující běžící programy za účelem zmírnění případných dopadů škodlivého kódu[9]. Míra kontroly se liší v závislosti na implementaci. Některé sandboxy využívají virtualizace na úrovni OS, a jiné, jako například Cuckoo Sandbox[1], využívají HW virtualizaci. Pojem sandbox je také často používán i pro jeho implementace, příkladem budiž sandbox v programovacím jazyce Java[20].

2.7.1 Sandboxie

Sandboxie je sandbox pro systém Windows umožňující jednoduše spouštět nedůvěryhodné programy prostřednictvím virtualizace na úrovni OS. Ještě donedávna (duben 2019) byl Sandboxie pro komerční použití placeným softwarem. To se však změnilo po uveřejnění zdrojových kódů[2] pod licencí GPLv3 firmou Sophos, která Sandboxie od roku 2017 vlastnila a vyvíjela. Firma Sophos ale také oznámila konec vývoje Sandboxie a veškeré změny ponechala na open source vývojářích. Tím učinila budoucnost Sandboxie prozatím nejistou a bude záležet zejména na iniciativě open source komunity.

[1] `https://cuckoosandbox.org/`

[2] `https://github.com/sandboxie/sandboxie`

Kapitola 3

Analýza systému SharpTest

V rámci předmětů BI-SP1/2 byl vyvinut systém SharpTest sloužící jako proof-of-concept zejména k demonstraci budoucímu zákazníkovi a k testům uživatelského rozhraní samotnými uživateli. Cílem systému je zlepšit a zjednodušit výuku programování tím, že studenti budou moci dostávat více lehkých domácích úkolů a hned na ně dostanou zpětnou vazbu ve formě automatického hodnocení. To jim napoví, ve kterém z testů definovaných učitelem jejich program neprochází. Takovéto automatické ohodnocování by mělo vést nejen k důkladnému seznámení se a procvičení základních programovacích příkazů, ale díky podpoře unit testů také k osvojení základů OOP (Objektově Orientované Programování). Učitelé pak budou mít více prostoru na výuku samotných konceptů programování, protože nebudou muset tolik času věnovat opravování domácích úkolů.

Systém SharpTest se skládá z 5 projektů, a to z:

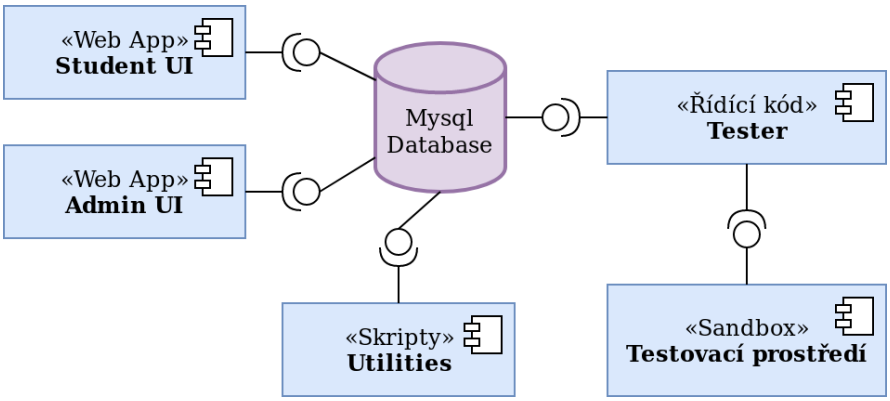
- Uživatelského rozhraní pro studenty **Web**[3].
- Administračního rozhraní pro lektory **Administration**[4].
- Relační databáze **Database**[5].
- Systému pro automatické hodnocení úloh **Tester**[6].
- Pomocných skriptů **Utilities**[7].

Web pro studenty umožňuje zobrazovat úlohy, nahrávat k nim řešení, zobrazovat hodnocení studentských odevzdání a případné zjištěné chyby při jejich automatickém opravování. Nově vyvíjené rozhraní pro administrátory bude

[3] `https://gitlab.fit.cvut.cz/sharptest/web`
[4] `https://gitlab.fit.cvut.cz/sharptest/administration`
[5] `https://gitlab.fit.cvut.cz/sharptest/database`
[6] `https://gitlab.fit.cvut.cz/sharptest/tester`
[7] `https://gitlab.fit.cvut.cz/sharptest/utilities`

umožňovat správu předmětů, studentů a úloh. Tester se stará o opravování řešení úloh v jazyku C# se vstupy ve formě souborů se vstupními daty nebo unit testy. Relační databáze je použita jako propojovací vrstva mezi Webem a Testerem a podpůrnými nástroji, které slouží hlavně k importování studentů do databáze a exportu grafů se statistikami. Provázání těchto projektů v systému SharpTest lze vidět na obrázku 3.1.

Obrázek 3.1: Diagram komponent systému SharpTest



Jedním z cílů této práce je vytvořit a integrovat nové prostředí pro spouštění studentského kódu pomocí virtualizace. Proto zde bude analyzován hlavně projekt Tester a jeho napojení na databázi.

3.1 Databáze

Jako databáze je v systému SharpTest použita relační databáze MySQL. Právě v ní jsou uloženy úlohy, odevzdané studentské řešení a hodnocení těchto úloh. Celé schéma databáze lze vidět na obrázku 3.2.

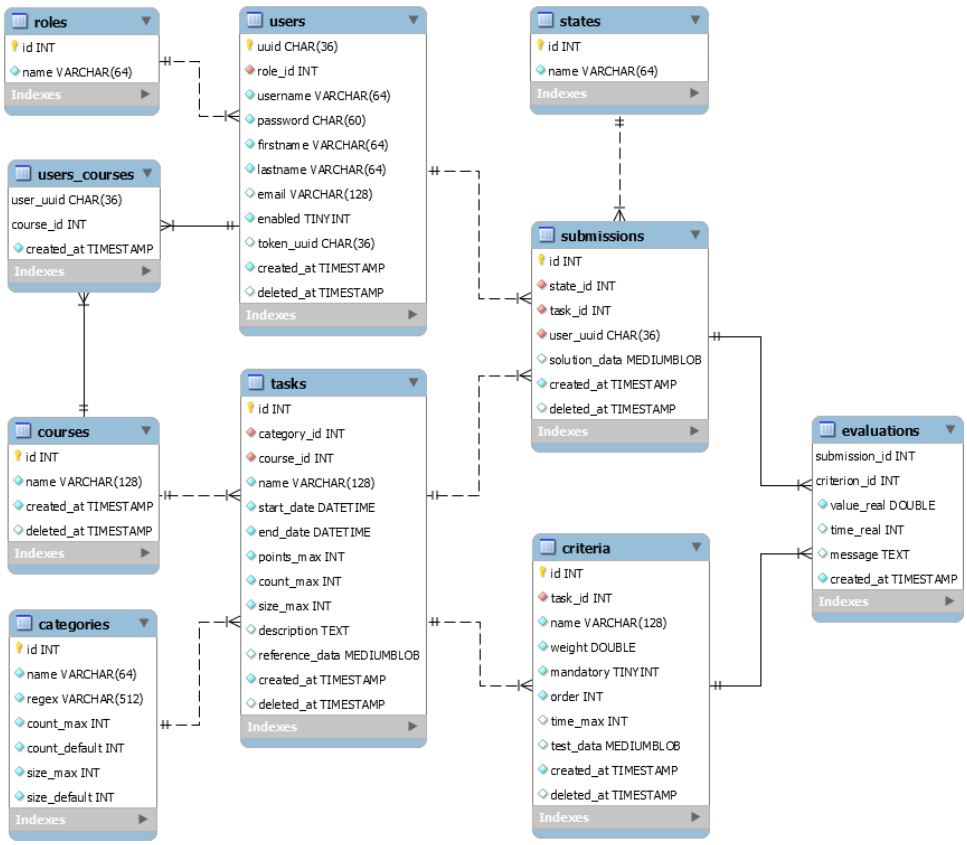
V rámci jasného určení pravomocí používá Tester pro přístup do databáze účet `tester_user`, který má oprávnění uvedená v tabulce 3.1.

Tabulka 3.1: Oprávnění uživatele tester_user v databázi

	Select	Update	Insert
tasks	X		
criteria	X		
submissions	X	X	
evaluations			X

Právě tabulky `tasks`, `criteria`, `submissions` a `evaluations` jsou těmi, ke kterým Tester přistupuje. V tabulce `tasks` jsou uloženy informace o úlohách pro studenty a referenční řešení těchto úloh. Každá úloha může mít více testů, například test správného ošetření vstupů, test rychlosti výpočtu a podobně.

Obrázek 3.2: Schéma databáze v MySQL[21]



Tyto testy jsou uloženy v tabulce `criteria`, ve které je uloženo zejména pořadí testů, indikátor toho, zda pokračovat, pokud řešení testem neprojde, maximální doba běhu řešení a generátor vstupů. Samotné odevzdané studentské řešení je uloženo v tabulce `submissions` a jeho ohodnocení Tester ukládá do tabulky `evaluations`.

3.2 Tester

Projekt Tester zahrnuje jak omezené testovací prostředí pro testování odevzdaných řešení, tak kód, který toto testování plánuje, řídí a vyhodnocuje. Jeho hlavním úkolem je bezpečné a přijatelně rychlé testování studentských řešení zadaných úloh.

3.2.1 Struktura projektu

Testovací prostředí je realizováno sandboxem Sandboxie, ve kterém jsou nastaveny zejména restrikce přístupu k síti a souborům. Adresářovou strukturu pro-

jektu Tester lze vidět na obrázku 3.3. Nastavení tohoto sandboxu je umístěno v adresáři `./sandboxie`. Dále je v adresáři `./sharpen` sada skriptů a doporučení na zabezpečení systému Windows, které jsou aplikovány po instalaci OS testovacího prostředí. Samotný řídicí kód v jazyce Python je v adresáři `./tester`.

Obrázek 3.3: Adresářová struktura projektu Tester

```
sandboxie ..... nastavení sandboxu Sandboxie
sharpen ..... sada skriptů na zabezpečení Windows
tester ..... řídicí kód Testeru
    test_modules ..... testové moduly
    unit_tests ..... unit testy
```

3.2.2 Řídicí kód Testeru

Řídicí kód Testeru je v jazyce Python verze 3.6.8 a dodržuje styl kódu PEP8[22]. To je automaticky kontrolováno nástroji `pycodestyle`[8], `flake8`[9] a `pylint`[10]. Mimo to je pro statickou analýzu kódu použit nástroj `bandit`[11] hledající časté bezpečnostní chyby v kódu a `vulture`[12] upozorňující na části kódu, které nejsou používány.

Tento kód lze logicky rozdělit na tři části, a to `DbCom` zajišťující komunikaci s databází, ovladač instance testovacího prostředí `Runner` a jádro `Core`, které zajišťuje organizaci testů a propojuje `Runner` a `DBCom`. Princip fungování řídicího kódu Testeru ilustruje obrázek 3.4

3.2.2.1 Core

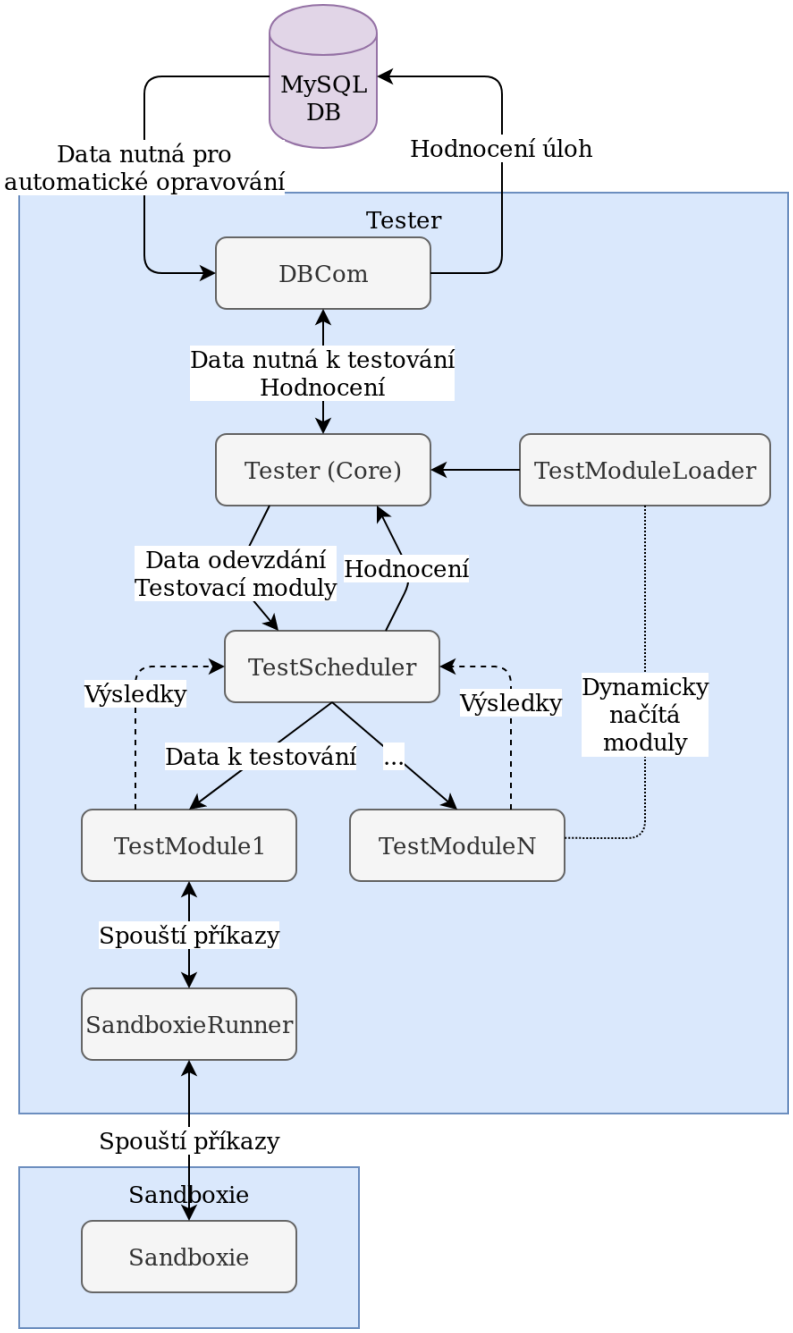
Část `Core` je jádrem projektu Tester, ve kterém probíhá organizace samotných testů, jejich vyhodnocování a komunikace jak s testovacím prostředím, tak s databází.

Hlavní třídou je `Tester` z modulu `tester`, jejímž úkolem je obstarat spouštění a ukončování celého Testeru, včetně načtení konfigurace a její validace. S pomocí modulu `test_module_loader` také načítá testovací moduly a inicializuje testovací prostředí a komunikaci s databází. Plánování a spouštění testů má na starosti třída `TestScheduler` z modulu `test_scheduler`.

Součástí `Core` jsou i testovací moduly `csharp_io` a `csharp_unit`, pomocí kterých se provádí samotné testování. Tyto testovací moduly jsou pouze demonstrační a nevyužívají možnosti paralelního spouštění referenčního a studentského řešení.

[8] `https://pypi.org/project/pycodestyle/`
[9] `https://flake8.pycqa.org/en/latest/`
[10] `https://www.pylint.org/`
[11] `https://pypi.org/project/bandit/`
[12] `https://pypi.org/project/vulture/`

Obrázek 3.4: Ilustrace fungování Testeru převzata z Wiki Testeru[23]



3.2.2.2 Runner

`Runner` je třída poskytující rozhraní k ovládání testovacího prostředí a jednotlivých *Workerů*, což je označení pro instance testovacího prostředí.

V aktuálním řešení je to konkrétně třída `SandboxieRunner`, která je určena pro ovládání sandboxu Sandboxie. Pro ovládání samotného sandboxu Sandboxie je použit již existující modul `sandboxie`[13].

Již zmíněné testovací moduly na instancích třídy `Runner` (které budeme stejnojmenně označovat jako *Runnery*) spouštějí příkazy v příkazovém řádku a přesouvají soubory dovnitř a ven z *Workerů*). Díky jednotnému rozhraní třídy `Runner`, které využívají všechny testovací moduly, je implementace testovacích modulů téměř nezávislá na konkrétní implementaci testovacího prostředí. Hlavní závislostí je použitý OS a specifické přizpůsobení spouštěných příkazů kvůli způsobu, jakým Sandboxie tyto příkazy v sandboxu spouští.

3.2.2.3 DbCom

Třída `DbCom` z modulu `dbcom` zajišťuje a abstrahuje komunikaci s databází. Součástí tohoto modulu je i třída `DbLogger`, která má sloužit k logování do databáze, nicméně zatím není v řídicím kódu implementována.

3.2.3 Testovací prostředí

Testovací prostředí je realizováno na operačním systému Windows 10 Pro se sandboxem Sandboxie. Pro kompilaci projektu v jazyce C# používá Tester software MSBuild[14], který je normálně součástí vývojového prostředí Visual Studio[15]. Lze ho však používat i samostatně. Další nezbytnou součástí testovacího prostředí je .NET Framework[16], který umožňuje spouštět kompilovaný kód.

Pro samotné testování jsou pak potřeba dva typy *Workerů*. První z nich je nazýván Prep a slouží pro kompilaci a spouštění referenčních řešení a generátorů vstupů. Tím druhým je Jail, který je určen pouze pro spouštění studentského řešení. Je tedy separováno referenční a studentské řešení. Pro oba typy *Workerů* platí podobná omezení a pro jejich ovládání je použita stejná třída `SandboxieRunner`.

Sandboxie jako sandbox určený ke spouštění nedůvěryhodného kódu pak disponuje celou řadou funkcionalit, z nichž systém SharpTest využívá:

- Omezení přístupu k síti na úrovni Ano/Ne.
- Více spuštěných instancí Sandboxie najednou.
- Omezení přístupu k souborovému systému.
- Rychlé smazání změn provedených programem spuštěných v Sandboxie v dané instanci sandboxu.

[13] `https://pypi.org/project/sandboxie/`
[14] `https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2019`
[15] `https://visualstudio.microsoft.com/`
[16] `https://docs.microsoft.com/en-us/dotnet/framework/`

- Omezení přístupu k registrům.

3.2.4 Příklad průběhu testu

V následujícím příkladu je předpokládána úloha s testy vstupů a výstupů, konkrétně se jedná o testovací modul `csharp_io`.

Student nejprve odevzdá přes webové rozhraní projekt vytvořený ve vývojovém prostředí Visual Studio. Ten je v databázi uložený jako archiv ve formátu ZIP. Tester dále zjistí, že v databázi je neohodnocené odevzdané řešení od studenta a začne stahovat všechna důležitá data pro testování. Jedná se o řešení studenta, referenční řešení a generátory vstupů. Neohodnocené odevzdané řešení přidá do fronty neohodnocených odevzdaných řešení, kde řešení setrvá, než pro něj bude volná instance testovacího prostředí.

Jakmile se testovací prostředí uvolní, předá se toto neohodnocené odevzdané řešení testovacímu modulu, který má již přidělený potřebný pár testovacích prostředí Prep a Jail. Dále jsou kompilovány generátory vstupů, referenční řešení a studentské řešení. Vstupy, které jsou generovány generátory vstupů, jsou poslány do referenčního i studentského řešení a následně porovnány. Podle výsledků porovnání pak mohou nastat tyto možnosti:

- Výstupy byly stejné a test je splněn.
- Výstupy byly rozdílné a test je povinný, pak se testování zastaví.
- Výstupy byly rozdílné ale test je nepovinný.

Podle toho, jak velká část výstupů se shodovala, dostane student příslušné skóre testu. Každý test má i časový limit, při jehož překročení je skóre testu nulové. Takto se pokračuje dokud nebyly spuštěny všechny testy, načež je studentovi spočítáno finální skóre, které závisí na váze jednotlivých testů. Po dokončení všech testů jsou prostředí vyčištěna smazáním všech souborů a ukončením všech procesů v sandboxu.

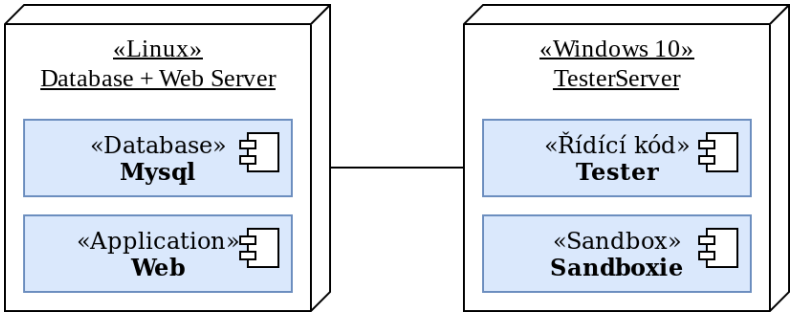
3.2.5 Logování

V celém projektu Tester je konzistentně používán standardní modul `logging` pro účely logování do souboru `tester.log`. Jednotlivé moduly v Testeru pak mají přiřazená rozdílná jména pro lepší rozlišení zdroje záznamů v logu. Mimo to je připravené rozhraní již zmíněné třídy `DbLogger`, která zatím není používána.

3.3 Model nasazení

Model nasazení systému SharpTest předpokládá rozdělení systému na 2 části. Na serveru s OS Linux je Databáze a Web a na počítači s OS Windows 10 je Tester se Sandboxie. Toto rozdělení je ilustrováno na obrázku 3.5.

Obrázek 3.5: Model nasazení systému SharpTest



V Sandboxie jsou pak vytvořeny příslušné kontejnery představující *Workery* typů Prep a Jail.

3.3.1 Konfigurační soubor

Konfigurace Testeru probíhá v konfiguračním souboru `tester.cfg`, který je načítán standardní třídou `ConfigParser` z modulu `configparser`. Ukázku konfigurace lze najít ve výpisu 1. V konfiguraci lze najít obecné nastavení (sekce tester, dbcom a testrunner), konkrétní konfigurace sandboxů (sekce runner1_prep_snb a runner1_jail_snb) a konfigurace jednotlivých testovacích modulů (sekce mod_csharp_io a mod_csharp_unit).

```
[tester]
poll_time=3

[dbcom]
db_host=192.168.56.1
db_user=tester_user
db_passwd=password
db_name=sharptest

[testrunner]
testrunner_data_dir = C:\TestRunnerData\

[runner1_prep_snb]
name=prep1
box=prep
data_dir=C:\Sandbox\admin\prep
cmd_out_path=C:\SharpTestData\out.txt

[runner1_jail_snb]
...

[mod_csharp_io]
db_index=1
mod_name=csharp_io
msbuild_path=C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe

[mod_csharp_unit]
...
vstest_path=C:\PROGRA~2\MICROS~2\2019\Community\Common7\IDE\CommonExtensions\
↪  Microsoft\TestWindow\vstest.console.exe
```

Výpis 1: Ukázka konfigurace projektu Tester z repozitáře Testeru[24]

Kapitola 4

Návrh integrace virtualizace do systému SharpTest

V předchozích kapitolách bylo provedeno seznámení se s virtualizací a systémem SharpTest, projektem Tester a fungováním testovacího prostředí a jeho instancí (*Workerů*). V této kapitole bude vybrán typ virtualizace a konkrétní virtualizační platforma. Následně bude navržena její integrace do systému SharpTest na základě analýzy z předchozí kapitoly. Integrace virtualizace do systému SharpTest spočívá v naprogramování nové třídy realizující *Runner*, návrhu a implementaci efektivního systému čištění použitých *Workerů*. Dále budou navrženy postupy a skripty pro vytváření a mazání těchto virtualizovaných *Workerů*. Nakonec dojde i na návrh nového modelu nasazení systému SharpTest se separovanými komponentami.

4.1 Výběr typu virtualizace

Virtualizace nabízí kontrolu nad prostředky jednotlivých VM a kontejnerů a zároveň poskytuje dobré vlastnosti z pohledu izolace. Při řešení problémů se spouštěním cizího kódu je pak na místě volba konkrétního typu virtualizace. V systému SharpTest jsou od virtualizace požadovány co nejlepší bezpečnostní vlastnosti týkajících se především izolace, ale také rozumně dobrý výkon. Tyto dva požadavky jdou částečně proti sobě, protože instancí kontejnerů ve virtualizaci na úrovni OS lze spustit více než VM se stejnými prostředky a dají se tedy považovat za výkonnější[25, 26]. HW virtualizace a její VM jsou ale zpravidla lépe izolované[27, 28] při použití jako sandbox. Jeví se tedy z pohledu bezpečnosti jako vhodnější oproti kontejnerům[30, 31, 29]. Například technologie Windows containers nabízí dva módy izolace, a to na úrovni kontejnerů a na úrovni VM. O izolaci na úrovni VM pak tvrdí, že je bezpečnější ale také náročnější pro hostitele[32].

Jelikož systém bude používán v organizacích zpracovávajících citlivá data

studentů, je jedním ze zásadních požadavků na systému SharpTest jeho bezpečnost. Na základě tohoto požadavku byla po diskuzi se zadavatelem upřednostněna bezpečnost před výkonem a ke spouštění cizího kódu bude využívána HW virtualizace. Takové rozhodnutí znamená zejména omezení testovacích kapacit díky větší náročnosti VM na paměť a start systému.

4.2 Výběr virtualizační platformy

Z pohledu uživatele nabízí většina dnešních virtualizačních platforem velice podobné funkcionality. Při výběru platformy je důležitá hlavně hlavně funkčnost týkající podpory jednotlivých OS, dobrá síťová separace virtuálních strojů a dostatečně obsáhlé ovládací API.

4.2.1 Xen Project

Xen Project je open source projekt s licencí GPL verze 2 který vznikl na půdě University of Cambridge. Později ho převzala firma Citrix Systems pro svůj Citrix XenServer a v současné době je zaštiťován organizací Linux Foundation jako kolaborativní projekt[34, 33]. Projekt zahrnuje hypervisor typu 1 a dále se v rámci něj vyvíjejí PV drivery a nástroje Xen Project management API (XAPI).

4.2.1.1 Citrix Hypervisor

Citrix Hypervisor, původně Citrix XenServer, je jednou z hlavních virtualizačních platforem využívajících Xen hypervisoru. Nabízí jednoduchou správu virtuálních strojů, jejich migrací, záloh, sítí a podobně[35]. Jednou z výhod tohoto řešení je to, že za ním stojí firma Citrix Systems, která své další řešení integruje s touto virtualizační platformou a nabízí také profesionální podporu.

4.2.1.2 Xen Cloud Platform - next generation (XCP-ng)

XCP-ng vznikl jako odpověď na zrušení verze Citrix XenServer poskytované zdarma a nabízí téměř stejné funkcionality jako Citrix Hypervisor. Při vzniku byl proveden fork poslední open source verze Citrix Hypervisoru, díky čemuž je možný například přímý upgrade z něj na XCP-ng[36]. XCP-ng je součástí skupiny projektů Xen Project. Uživatelské prostředí pak zajišťuje open source projekt Xen Orchestra[17]. Obě tato řešení nabízejí i placenou podporu.

4.2.2 VirtualBox

VirtualBox je oblíbeným řešením zejména běžných uživatelů pro domácí použití kvůli jeho jednoduché instalaci, kompatibilitě a poměrně široké

[17] `https://xen-orchestra.com/`

nabídce funkcionalit. Obsahuje hypervisor typu 2 a je možné jej provozovat na téměř všech moderních OS[37]. V současné době ho vyvíjí firma Oracle, přičemž jeho hlavní jádro je open source pod licencí GPLv2. VirtualBox Extension Pack pak nabízí podporu USB 2.0 a USB 3.0, VirtualBox RDP, šifrování disků a podobně. Komerční licence umožňující ho zdarma využívat, je však pouze pro osobní a vzdělávací použití. Pro komerční použití je nutné zakoupit enterprise licenci[38].

4.2.3 Kernel-based Virtual Machine (KVM)

KVM je plně open source software s licenci GPL zahrnutý v jádru Linuxu od verze 2.6.20. Je brán jako primární řešení pro virtualizaci v linuxových OS a jeho hypervisor je typu 2. Implementován je však jako kernel modul, což ho efektivně dělá hypervisorem typu 1[39].

4.2.3.1 Proxmox VE

Proxmox VE je open source virtualizační platforma s licenci GNU AGPL v3 využívající hypervisor KVM. Má integrovaný webový interface a rozsáhlou dokumentaci[40].

4.2.4 Microsoft Hyper-V

Microsoft Hyper-V je vyvíjen firmou Microsoft a je dostupný pro Windows Server a pro některé 64-bitové verze OS Windows jako doinstalovatelný doplněk[41]. Hyper-V je také použit v technologii Microsoft containers jako bezpečnější varianta z pohledu izolace[26].

4.2.5 VMware vSphere

VMware vSphere je virtualizační platforma s hypervisorem VMware ESXi typu 1. Je určena zejména pro komerční použití, nicméně existuje i omezená verze zdarma[42]. Vyznačuje se hlavně dobrou stabilitou a podporou a je jednou z nejrozšířenějších virtualizačních platforem.

4.2.6 Porovnání

Výběr virtualizačních platforem je poměrně široký a virtualizační platformy jsou si v základních poskytovaných funkcích velice podobné. Byly tedy specifikovány žádoucí a nežádoucí vlastnosti vybírané virtualizační platformy. Mezi tyto vlastnosti byl zvolen výkon, možnost komerčního využití a fakt, zda je platforma open source.

Všechny platformy v tabulce 4.1 pak podporují virtualizaci systému Windows 10 a snapshoty s pamětí.

Tabulka 4.1: Srovnání virtualizačních platforem

Platforma	Výkon [43]	Open source	Komerční využití
Proxmox	+	Ano	Ano
XCP-ng	+	Ano	Ano
Citrix XenServer	+	Ne	Placeně
VirtualBox	-	Částečně	Placeně
Hyper-V	++	Ne	Placeně
VMware vSphere	+	Ne	Částečně

Jako virtualizační plaforma bylo po předběžném testování vybráno XCP-ng, a to zejména proto, že je to open source projekt s aktivní komunitou a hypervisorem Xen, který se vyznačuje dobrou bezpečností hlavně díky relativně krátkému kódu[44]. Xen byl také velice dlouhou dobu používán (a ještě částečně je) v Amazon Web Services, je použit v OS orientovaném na bezpečnost QubesOS[18] a systému pro automatickou analýzu malware DRAKVUF[19].

Samotná virtualizace pak bude typu PVHVM, který by měl poskytovat dostatečně výkonné řešení[45].

Hypervisor Xen je specifický svou architekturou, kde část role hypervisoru přebírá speciální VM nazývané Dom0, které se stará zejména o zpřístupnění disků a o řízení komunikace ve virtuálních sítích. Také je v hypervisoru Xen rozlišováno mezi snapshotem a checkpointem.

4.2.7 Očekávané výhody a nevýhody oproti programu Sandboxie

Důvodů pro vytvoření nového testovacího prostředí je více, API pro ovládání Sandboxie totiž není úplně stabilní, zejména při čištění sandboxu v některých případech nebylo plně dokončeno a byl nutný ruční zásah. U jednotlivých instancí sandboxů navíc nelze limitovat procesorový čas ani paměť, což může vést k útokům odepření služby a negativního ovlivnění hodnocení jiných odevzdání. V neprospěch Sandboxie jsou i zvýšené požadavky na izolovanost celého systému a kompatibilita s jinými OS, na kterých by v budoucnu mohl SharpTest testovat úlohy.

Použití HW virtualizace do budoucna umožní zejména podporu více OS a více propojených testovacích prostředí například pro síťové úlohy typu klient-server. Očekávanými výhodami HW virtualizace pro aktuální podobu systému SharpTest jsou zejména:

- Konkrétnější omezení přístupu k síti jednotlivých VM.

[18] `https://www.qubes-os.org/`
[19] `https://drakvuf.com/`

- Nastavení konkrétních systémových prostředků pro *Workery*.

- Lepší separace a izolace *Workerů*, ale i jednotlivých částí celého systému SharpTest.

Nevýhodou bude již zmiňovaný nižší výkon celého systému ve smyslu počtu najednou opravovaných studentských řešení. Ten bude způsoben menším počtem *Workerů* kvůli nutnosti vyhradit prostředky navíc pro OS každého *Workera*.

4.3 Instance testovacího prostředí

Instance testovacího prostředí, tedy *Worker*, bude realizována pomocí VM s OS Windows 10 Pro kvůli jeho ověřené kompatibilitě se stávající verzí systému SharpTest. Do budoucna pak zůstává možná optimalizace výběru OS, který zajistí lepší výkon systému (tedy bude mít menší nároky na prostředky) se stejnou kompatibilitou.

Samotné VM bude mít k dispozici jedno procesorové jádro a 1,25 GB operační paměti, což bude umožněno použitím 32-bitových Windows 10, které vyžadují minimálně 1 GB operační paměti[46]. Díky tomu bude spuštěn vyšší počet *Workerů*, než při použití 64-bitové verze OS. Navíc se při obnovování checkpointu bude pracovat s menšími objemy dat. Při vytváření *Workerů* pak bude využíváno principu Copy-on-Write (CoW), díky kterému budou menší nároky na úložný prostor než při plné kopii virtuálního úložiště.

4.4 Efektivní čištění použitých testovacích prostředí

Po každém testování probíhá vyčištění testovacího prostředí, tedy konkrétních *Workerů*, což s použitím Sandboxie znamenalo smazání souborů a ukončení všech procesů. Na rozdíl od Sandboxie je takovéto čištění virtuálního stroje náročnější, protože se musí pročistit celý OS.

K vyčištění disku lze využít snapshot, po jehož obnovení se disk vrátí do stavu, kdy byl snapshot vytvořen. Tento postup ale vyžaduje restart, který by čas na vyčištění značně zvýšil. Lepším řešením je checkpoint, který ukládá i paměť běžícího VM a po jeho obnovení je již VM spuštěno. Při průběžném testování se ukázalo, že i obnovování starších checkpointů (14 dní) nemá negativní vliv na funkčnost OS ve VM. Vyčištění bude probíhat pomocí checkpointů. Pokud se v budoucnu objeví problémy s jejich zastaráváním, využije se nově přidané funkcionality dynamického přidávání a odebírání *Workerů* k doprogramování periodického vytváření checkpointů.

4.5 Navrhované změny v řídicím kódu Testeru

Hlavní změnou v řídicím kódu Testeru bude náhrada třídy `SandboxieRunner` za třídu `XenRunner`. Dále budou provedeny i některé menší změny, jako je automatické načítání a ukládání konfigurace *Runnerů* z konfiguračního souboru a příprava na budoucí rozšíření Testeru tam, kde se to týká třídy `XenRunner`. Při implementaci bude dbáno i na udržení kvality kódu, která je automaticky kontrolována nastavenými nástroji.

4.5.1 XenRunner

Pro ovládání a komunikaci s *Workerem* bude sloužit třída `XenRunner`, která nahradí třídu `SandboxieRunner`. V rámci toho bude do Testeru přidána i abstraktní třída `Runner`, která bude předepisovat všechny metody potřebné k nahrazení `SandboxieRunner`. Třída `XenRunner` pak bude od třídy `Runner` tyto metody dědit a implementovat. To v budoucnu napomůže při implementování jiných typů třídy `Runner`.

4.5.1.1 PowerShell Remoting Protocol

Pro správné fungování třídy `XenRunner` je nutná zejména možnost spouštět příkazy a stahovat/nahrávat soubory. Na to bude využit plugin PSRP (PowerShell Remoting Protocol)[47] do WinRM (Windows Remote Management)[48]. Ten umožňuje vzdálené spouštění příkazů v příkazové řádce i skriptů v jazyce PowerShell. Pro komunikaci s PSRP pak existuje modul `pypsrp`[49], který nabízí právě možnost spouštění jak příkazů v příkazové řádce, tak skriptů v jazyce PowerShell a potřebné stahování a nahrávání souborů. Tento modul využívá k připojování protokol WSMan (Web Services for Management), což je protokol postavený nad specifikacemi HTTPS (Hypertext Transfer Protocol Secure), SOAP over HTTP a dalších. WSMan umožňuje autentizaci pomocí NTLM (NT Lan Manager), certifikátu, CredSSP, Kerberos nebo HTTP Basic, přičemž pro automatizovanou komunikaci se hodí především autentizace pomocí certifikátu[50]. Autentizace certifikátem znamená, že uživateli, pod kterým bude Tester řešení spouštět, se vygeneruje veřejný a privátní klíč. Veřejný klíč pak bude na všech *Workerech* a privátní klíč bude mít Tester a bude ho používat k přihlašování na *Workery*. Komunikace bude probíhat přes protokol HTTPS, kde každý server dostane při jeho nastavení certifikát a privátní klíč vygenerovaný lokální certifikační autoritou. Generování a nastavování těchto certifikátů bude automatizováno.

Alternativou k tomuto řešení by byl vlastní klient v každém virtuálním stroji, který by komunikoval přes síť s Testerem a poskytoval mu potřebné API ke spouštění příkazů a práci se soubory. Takovýto klient by byl vhodný zejména pokud by bylo potřeba i jiných funkcionalit. PSRP však poskytuje

právě potřebné funkcionality a jedná se o ověřené standardní řešení použité například v populárním nástroji Ansible[51].

4.5.1.2 XenAPI k čištění Workerů

Jelikož čištění *Workerů* bude probíhat obnovením checkpointu, bude mít třída `XenRunner` přístup i k XenAPI, pomocí kterého bude obnovování řídit. Pro komunikaci s XenAPI pak bude použit již existující stejnojmenný Python modul `XenAPI`[20].

V dokumentaci[52] lze najít, že na vytvoření checkpointu slouží metoda `checkpoint`, na jeho obnovení zase metoda `revert`, obě ze třídy `VM`. Obě metody vyžadují reference na virtuální stroj a metoda `revert` požaduje dále referenci na checkpoint. Referenci na VM vrací metoda `get_by_uuid` ze třídy `VM` podle uuid, které musí být předem specifikováno. U checkpointu lze získat referenci na ten poslední pouze načtením všech checkpointů metodou `get_snapshots` a následným seřazením podle výstupu metody `get_snapshot_time`. Obě tyto metody jsou ze třídy `VM`.

Pro Tester pak bude vytvořen odpovídající účet, který co nejvíce omezí přístup k XenAPI, čímž bude docíleno dodržení principu nejmenších možných oprávnění.

4.5.2 Další úpravy související se třídou Runner

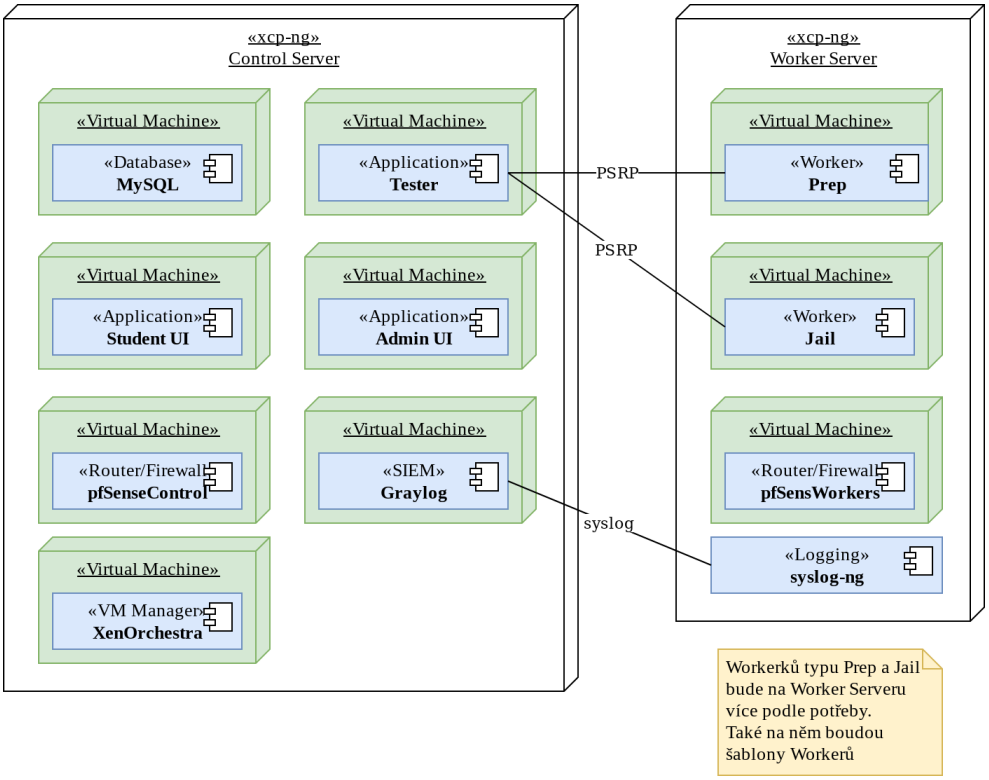
V plánu budoucího vývoje řídicího kódu Testeru je vytvoření CLI (Command Line Interface) a API pro ovládání a monitorování Testeru za běhu. Proto bude v rámci implementace obecné třídy `Runner` upravena i třída `TestScheduler`, která obsahuje seznam aktivních *Runnerů*. Bude učiněna příprava na přidávání a odebírání *Runnerů* za běhu ve formě přidání dodatečných metod. Jedním z nedodělků řídicího kódu Testeru bylo i statické načítání *Runnerů* z konfigurace. To bude nyní napraveno přidáním automatického načítání a ukládání této konfigurace. Zároveň bude pro přehlednost oddělena obecná konfiguraci Testeru a konfigurace *Runnerů*.

4.6 Návrh modelu nasazení

Jednou z podstatných nevýhod aktuálního řešení se Sandboxie je nedostatečná separace mezi řídicím kódem a sandboxem samotným. Pokud by se případnému útočníkovi podařilo uniknout ze sandboxu, měl by přístup k celému Testeru a jeho části databáze. Mohl by pak například modifikovat kód Testeru nebo číst referenční a studentské řešení z databáze. V navrhovaném modelu nasazení je testovací prostředí od zbytku systému odděleno tak, aby útočník musel projít více vrstvami zabezpečení a v každé vrstvě měl co nejmenší potenciál na napáchání škod.

[20] `https://pypi.org/project/XenAPI/`

Obrázek 4.1: Model nasazení systému SharpTest



Na obrázku 4.1 lze vidět navrhované nasazení v produkci s testovacím prostředím fyzicky odděleným od zbytku systému. *Control Server* zde označuje stroj, na kterém budou virtualizovány všechny části systému SharpTest právě kromě testovacího prostředí. Pro maximální efektivitu zamýšlené separace budou všechny části ve vlastních virtuálních strojích. To umožní jasně definovat a omezovat jejich přístup k ostatním virtuálním strojům. Testovací prostředí je na jiném fyzickém serveru nazývaném *Worker Server*. Ten se bude starat hlavně o provozování virtuálních strojů určených k testování, tedy *Workerů*. Díky SDN (Software Defined Network) lze spojit více virtuálních sítí v jednu, čehož bude využito při propojování Testeru a jeho *Workerů*. Takováto síť se v XCP-ng označuje jako „privátní“. Konkrétně bude při použití fyzické separace propojena síť TesterNet. Ta bude na *Worker Serveru* napojena do pfSenseWorkers, který se lokálně postará o filtrování komunikace na tomto serveru.

Pro účely vývoje a testování je možné tento model nasazení zjednodušit tak, že bude použit pouze jeden fyzický server. V produkčním nasazení pak bude realizována fyzická separace.

4.6.1 Podpůrné nástroje

Kvůli propojení sítě TesterNet se sítěmi JailNet a PrepNet bude mezi těmito sítěmi použit router a firewall, přičemž pro tento účel je vhodný pfSense. Ten poskytuje všechna potřebná nastavení a jedná se o open source s licencí Apache 2.0 umožňující komerční využití[53]. V modelu nasazení jsou uvedeny hned dva pfSense, kde pfSenseControl obsluhuje *Control Server* a pfSenseWorkers zase *Worker Server*. Tato redundance pak zajišťuje, že VM z *Worker Serveru* nebudou zbytečně vytěžovat síť *Control Serveru*.

V rámci návrhu modelu nasazení byly uvedeny i AV, IDS a SIEM, které se sice netýkají přímo virtualizace, ale mohou mít velkou přidanou hodnotu a jsou tedy v návrhu zahrnuty. Tyto nástroje budou nasazeny na všech virtuálních strojích kromě *Workerů*. Instalace dalších nástrojů do dom0 není doporučována kvůli možnému snížení výkonu, ale je možné nechat přeposílat syslog z dom0 do nástroje SIEM.

Konkrétními nástroji pak jsou:

ClamAV [21] je open source AV detekující malware a další hrozby. Podporuje jak on-demand tak on-access skenování.

Open Source TripWire [22] je open source verze IDS TripWrire, která je dostupná pro OS Linux. Zajistí hlavně integritu souborů na VM.

GrayLog [23] je open source SIEM, díky kterému bude možné zpracovávat hromadně výstupy z nástrojů ClamAV a TripWire, ale také z logů OS ve VM.

4.7 Návrh realizace sítí

Vnitřní sítě systému SharpTest by měly být zvoleny tak, aby byly přehledné a funkční a umožňovaly jednoduché pozdější napojení dalších sítí a přidávání VM do sítí. Vzhledem k těmto požadavkům bylo zvoleno rozdělení na podsítě pro všechny části systému SharpTest, které lze vidět na obrázku 4.2.

Sítě DatabaseNet, WebNet a TesterNet reprezentují sítě pro dané komponenty systému SharpTest. Síť TemplateNet bude využívána při vytváření nových *Workerků*. JailNet a PrepNet pak budou sloužit k separaci *Workerů* typu Jail a Prep.

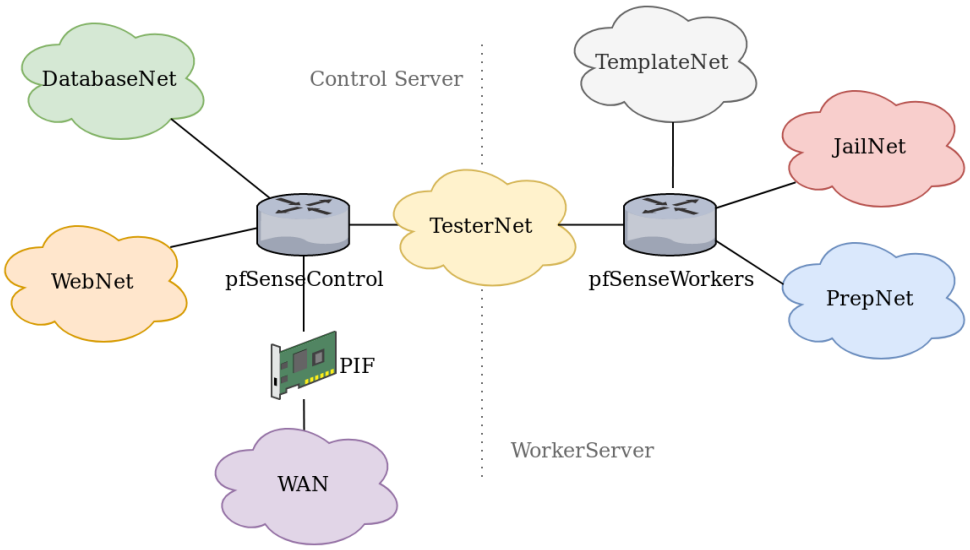
4.7.1 Separace Workerů

Při rozdělování sítí v XCP-ng je možné volit z více variant úrovně separace jednotlivých *Workerů*. První z těchto možností je každému *Workeru* přiřadit

[21] `https://www.clamav.net/`
[22] `https://github.com/Tripwire/tripwire-open-source`
[23] `https://www.graylog.org/`

vlastní virtuální síť, přes kterou by pak komunikoval s Testerem. Toto řešení je sice možné a přidává jistou míru izolace, ovšem má i několik nevýhod. Jednou z nich je už to, že se v administračním rozhraní virtualizační platformy (a v routeru/firewallu) bude vyskytovat opravdu velké množství virtuálních sítí. To rychle může vést k nechtěným chybám při jejich administraci. Pokud bychom toto řešení dále implementovali tak, že VM Testeru bude mít tolik VIF, kolik je *Workerů*, budeme omezeni limitem počtu VIF v XCP-ng (tento limit je i v ostatních virtualizačních platformách). K řešení lze využít tzv. VLAN trunking[54]. VIF *Workerů* by byly svázány s PIF serveru a každá virtuální síť by měla svou VLAN. VLAN lze vytvořit až 4096, což už je dostatečný počet *Workerů*.

Obrázek 4.2: Oddělení sítí v systému SharpTest



Zmíněným řešení by byl dán přístup *Workerům* k PIF na *Worker Serveru*, čímž by se mohla otevřít možnost útoků na něj a na služby, které na něm naslouchají a které při instalaci nebyly správně omezeny. Dále ani VLAN nejsou naprosto bezpečné a tímto jednoduchým spojením všech *Workerků* se vystavujeme dalším nebezpečím. Jedná se například o ovlivnění komunikace *Workeru* typu *Prep*, na kterém je referenční řešení, *Workerem* typu *Jail*[55]. Nabízí se použití dvou PIF pro rozdělení *Workerů Jail* a *Prep*, nicméně přidaná komplexita takového řešení je opravdu značná.

Druhým řešením je vytvoření dvou oddělených sítí JailNet a PrepNet a nastavení firewallů na všech *Workerech* pouze pro komunikaci s Testerem. Toto řešení je oproti tomu prvnímu mnohem přehlednější. Sice je pravdou, že jsou *Workeři* vystaveni útokům na společné síti, nicméně nebezpečí není ve skutečnosti tak velké. Útočník by musel dokázat spustit na jednom *Workerovi* kód, který překoná omezení OS Windows a útočí na okolní *Workery*, kde

musí překonat jejich firewall. Pokud by se však útočníkovi podařilo ovládnout jiného *Workera* před tím, než bude jeden z nich po testování vyčištěn, mohl by získat zkompilované řešení jiného studenta nebo způsobit odepření služby. V současné chvíli nejsou studentům v systému SharpTest vraceny žádné výstupy jejich programů. Získání tohoto zkompilovaného řešení by vyžadovalo využití dalších případných chyb v Testeru samotném. Odepření služby touto cestou by bylo možné, nicméně také by bylo poměrně jednoduše napravitelné vrácením připraveného checkpointu a zablokováním uživatele. Jelikož nemají tyto uvažované možnosti přímý negativní vliv na citlivá data uživatelů, bylo zvoleno právě toto řešení.

Každý *Worker* bude mít také omezení na přidělenou IP adresu na VIF na úrovni virtuálního switche a v XCP-ng bude vypnuta podpora IPv6 na všech VIF. IPv6 v síti není potřeba a jeho ponechání by nepřinášelo žádné výhody.

4.7.2 Firewall pfSense

Ve firewallu pfSenseWorkers a pfSenseControl pak budou nastavena omezení pro komunikaci z TesterNet do JailNet a PrepNet. Bude povolena pouze komunikace přes WinRM na portu 5986, na kterém běží služba WSMan přes HTTPS. Na úrovni obou pfSense pak bude také vypnuto směrování IPv6 adres.

4.8 Návrh nastavení OS Workerů

Samotné testování bude probíhat pod uživatelským účtem `user` s již zmíněným přihlašováním přes certifikát. Tento uživatelský účet bude mít takové role a oprávnění, aby se mohl vzdáleně přihlašovat k PSRP, přes které přihlašování probíhá. To znamená nastavení OS, rolí uživatele a oprávnění uživatele podle dostupné dokumentace[56].

Při přípravě OS *Workerů* proběhne rozdělení na systémy typu Jail a Prep. V OS typu Prep oproti Jail jsou přidány nástroje MSBuild[24] pro kompilaci generátorů vstupů, referenčních řešení a studentských řešení. Na těchto OS bude také v největší možné míře aplikována již existující sada omezení převzatá z repozitáře Testeru[24] umístěná v adresáři `sharpen`, která byla vytvořena pro původní podobu testovacího prostředí.

4.9 Podpůrné skripty pro administraci

K efektivní administraci nového testovacího prostředí budou vytvořeny dva skripty, jeden na hromadné vytváření *Workerů* a druhý na jejich mazání.

[24] `https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild`

4.9.1 Hromadné vytváření Workerů

Vytváření více *Workerů* se může hodit zejména při potřebě aktualizovat OS nebo v něm provést změny nastavení a ty promítnout mezi *Workery*. Při tomto hromadném vytváření bude využito šablony pro OS typu Prep a Jail, která pak bude naklonována a budou jí nastaveny potřebné certifikáty pro službu WinRM. K tomuto hromadnému vytváření bude patřit i vygenerování nové konfigurace *Runnerů*.

4.9.2 Hromadné mazání Workerů

Hromadné mazání lze sice provádět z administračního rozhraní Xen Orchestra, nicméně vytvořením skriptu se částečně zamezí chybám při tomto ručním mazání.

4.10 Návrh testů

Testování proběhne na více úrovních s cílem zaručit funkčnost řešení a jeho integrace do stávajícího systému. Na nejnižší úrovni to budou unit testy, které budou testovat funkčnost jednotlivých částí implementovaného kódu. Dále se bude jednat o testy integrační, které ověří funkčnost větších části systému SharpTest a jejich provázání. V tomto případě se bude testovat zejména:

- Komunikace třídy `XenRunner` s *Workerem*.
- Integrace třídy `XenRunner` s ostatními komponentami Testeru.
- Ověření funkčnosti čištění *Workerů*.

Systémové testování pak proběhne ve formě ručního ověření funkčnosti systému SharpTest jako celku. Poslední skupinou testů budou testy předpokladů zabezpečení, jejichž účelem bude demonstrace funkčnosti omezení provedených za účelem izolace *Workerů*, konkrétně se bude jednat o ověření:

- Síťové separace mezi *Workery*.
- Správného čištění *Workerů* z pohledu zanechaných artefaktů, tedy souborů a spuštěných procesů.
- Funkčnosti nastavených pravidel v pfSense.
- Správnosti nastavených práv uživatele `user`.

Kapitola 5

Integrace virtualizace do aktuálního řešení

V této kapitole je popsán realizovaný návrh z kapitoly minulé, a to z pohledu aktuálního stavu změn v řídicím kódu Testeru a konkrétní konfigurace virtualizační platformy XCP-ng a OS *Workerů*.

5.1 Změny v řídicím kódu Testeru

Hlavní změny v řídicím kódu proběhly podle návrhu, přičemž při implementaci byly provedeny i jiné drobné úpravy. Všechny změny jsou k dohledání na přiloženém fyzickém médiu, ve kterém je soubor `diff.txt` obsahující rozdíl verze před a po změnách v řídicím kódu. V rámci těchto úprav byla zvýšena verze Pythonu z 3.6.8 na 3.7.3. Samotný kód je okomentován, a to na úrovni modulů, tříd a metod, které jsou určeny jako veřejné. Níže popsané změny pak ilustruje obrázek 5.1.

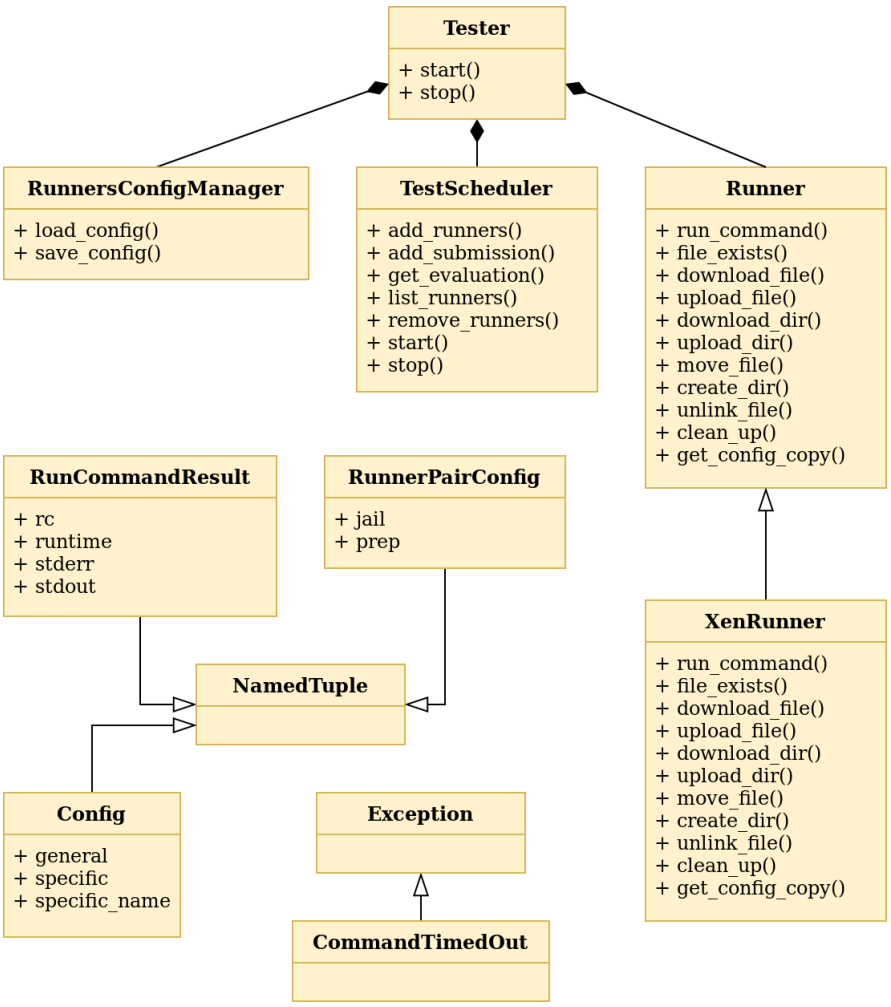
5.1.1 Třídy Runner a XenRunner

Ze třídy `SandboxieRunner`, která byla odstraněna, vznikla abstraktní třída `Runner`. Tato třída má navíc přidány metody `move_file` pro přesun souborů a `get_config_copy` pro účely automatického ukládání konfigurace. Dále byl změněn způsob předávání konfigurace *Runnerům* tak, aby v této konfiguraci mohlo být více sekcí a byla lépe členěna. Konfigurace je nově reprezentována pomocí NamedTuple[25] `Config`, který v sobě má povinné i nepovinné nastavení pro třídy typu `Runner`. Kvůli použití PSRP, kterým lze získat návratový kód a výstup na standardní chybový výstup (stderr), byl také modifikován NamedTuple `RunCommandResult`. Ten je používán v metodě `run_command`

[25]Zde je myšlena třída, která dědí od třídy NamedTuple, sloužící k vytváření n-tic s pojmenovanými datovými složkami

jako obálka, přes kterou metoda vrací výsledky své činnosti. Poslední změnou oproti `SandboxieRunner` je přidání výjimky `CommandTimedOut`, která je vyvolána, pokud příkaz spuštěný na *Workerovi* nestihl doběhnout a následně se ho nepodařilo ukončit standardním způsobem. Tato výjimka bude použita při vytváření produkčních testovacích modulů.

Obrázek 5.1: Diagram nově přidaných nebo změněných tříd v Testeru



5.1.2 Načítání a ukládání konfigurace Runnerů

Tato funkcionalita je implementována novou třídou `RunnersConfigManager` z nového modulu `runners_conf_manager`. Tato třída se stará o načítání a ukládání konfigurace, a to metodami `save_config` a `load_config`. Dále byl přidán do stejného modulu NamedTuple `RunnerPairConfig`, který slouží k uložení konfigurace pro pár *Runnerů* typu Jail a Prep. Třídu

`RunnersConfigManager` pak používá třída `Tester` k načtení konfigurace a vytvoření instancí třídy `XenRunner`. Při vypínání Testeru je pak aktuální konfigurace uložena.

```
[xenapi]
ip = 169.254.0.2
username = tester_user
password = password

[runner1_prep]
name = xen_prep1
username = user
ip = 192.168.129.10
worker_uuid = 8530cd62-0454-1712-9251-b864674b8f57
user_cert_key = user.key
user_cert_crt = user.crt

[runner1_jail]
...
```

Výpis 2: Ukázka oddělené konfigurace Runnerů vygenerované skriptem `create_runners.py`

5.2 Realizace sítí

Navrhované rozdělení sítí je realizováno přidáním sítí `DatabaseNet`, `WebNet`, `TesterNet`, `JailNet`, `PrepNet` a `TemplateNet`. Pro sítě jsou zvoleny rozsahy IP adres uvedené v tabulce 5.1 a konkrétní IP adresy VM uvedené v tabulce 5.2. Pro Workery je zvolena IP adresa podle jejich čísla tak, aby první Worker začínal adresou 192.168.x.10. VM TesterServer má pak VIF na Management síti, díky které může komunikovat s XenAPI *Control Serveru*. Při použití fyzické separace TesterServer není napojen na lokální Management síť *Control Serveru*, ale k jiné síti, přes kterou má přístup k XenAPI na *Worker Serveru*.

Firewall v pfSenseWorkers je nastaven tak, aby propustil pouze komunikaci z TesterNet do sítí PrepNet, JailNet a TemplateNet. Toto nastavení odpovídá tabulce 5.3.

Firewall v pfSenseControl propustí potřebnou komunikaci ze sítě WebNet a TesterNet do DatabaseNet. Dále pak obsahuje pravidla pro vzdálenou správu pomocí SSH a také pravidla povolující komunikaci do sítě TemplateNet přes WSMan a do sítě WebNet přes HTTP a HTTPS. Takto nastavená pravidla jsou sepsána v tabulce 5.4.

Tabulka 5.1: Rozsahy IP adres

Virtuální síť	Rozsah IP adres
WebNet	192.168.64.0/24
DatabaseNet	192.168.65.0/24
TesterNet	192.168.66.0/24
PrepNet	192.168.128.0/24
JailNet	192.168.129.0/24
TemplateNet	192.168.130.0/24

Tabulka 5.2: Přiřazené IP adresy

Virtuální síť	Název stroje	IP adresa
WebNet	pfSenseControl	192.168.64.1
WebNet	WebServer	192.168.64.2
DatabaseNet	pfSenseControl	192.168.65.1
DatabaseNet	DatabaseServer	192.168.65.2
TesterNet	pfSenseControl	192.168.66.1
TesterNet	pfSenseWorkers	192.168.66.2
TesterNet	TesterServer	192.168.66.3
Management	TesterServer	169.254.0.2
PrepNet	pfSenseWorkers	192.168.128.1
PrepNet	WorkerPairN_Jail	192.168.128.10-192.168.128.240
JailNet	pfSenseWorkers	192.168.129.1
JailNet	WorkerPairN_Prep	192.168.129.10-192.168.129.240
TemplateNet	pfSenseWorkers	192.168.130.1
TemplateNet	WorkerTemplate_Prep	192.168.130.2
TemplateNet	WorkerTemplate_Jail	192.168.130.3

Tabulka 5.3: Nastavení firewallu v pfSenseWorkers

Ze sítě	Do sítě	Porty
TesterNet	JailNet	5986
TesterNet	PrepNet	5986
TesterNet	TemplateNet	5985, 5986

Tabulka 5.4: Nastavení firewallu v pfSenseControl

Ze sítě	Do sítě	Porty
TesterNet	DatabaseNet	3306
WebNet	DatabaseNet	3306
WAN	*	22
WAN	WebNet	80,443

Dále je na každém VM nastaven firewall, který povolí pouze komunikaci na portech uvedených v tabulce 5.5.

Tabulka 5.5: Nastavení firewallů VM

Název VM	Porty	Z IP adres
DatabaseServer	22	*
DatabaseServer	3306	*
TesterServer	22	*
WebServer	22	*
WebServer	80	*
WebServer	443	*
WorkerPairN_Prep	5986	192.168.66.3
WorkerPairN_Jail	5986	192.168.66.3
WorkerTemplate_Prep	5985, 5986	192.168.66.3
WorkerTemplate_Jail	5985, 5986	192.168.66.3

Při aktualizacích OS ve VM jsou pak v obou pfSense dočasně povolena pravidla umožňující tuto aktualizaci. Ve všech OS VM jsou pak přidány cesty (routes) potřebné ke komunikaci s jinými VM.

5.3 Fyzická separace

Při použití fyzické separace je mezi hostiteli vytvořena privátní síť za pomoci SDN, která mezi těmito hostiteli spojí danou virtuální síť. Privátní virtuální sítí spojující oba hostitele je síť TesterNet. Díky tomu může pfSenseWorkers filtrovat požadavky ze sítí JailNet, PrepNet a TemplateNet ještě před tím, než jsou přeposlány na *Control Server*. Tester pak používá pro ovládání VM na serveru *Worker Server* jinou síť, na které má *Worker Server* nastaveno management rozhraní.

5.4 Nastavení operačních systémů virtuálních strojů

Pro VM TesterServer, DatabaseServer a WebServer byl zvolen OS Debian 10 s firewallem `firewalld` nastaveným podle tabulky 5.5. Na těchto VM je nainstalován a nastaven SSH server. Pro přihlašování na tyto stroje přes SSH je určen uživatel `user`, který na TesterServeru slouží také pro spuštění řídicího kódu Testeru.

Ostatní VM, konkrétně tedy WorkerTemplate_Prep, WorkerTemplate_Jail, WorkerPairN_Prep a WorkerPairN_Jail, mají podle návrhu OS Windows 10 Pro, přičemž pro demonstraci je zatím nainstalován v evaluační verzi. I na

těchto serverech je uživatel `user`, který slouží k jako omezený uživatel pro účely hodnocení odevzdaných úloh.

Router a firewall pfSense byl nastaven tak, aby byla povolena pouze komunikace přes protokol HTTPS. Dále bylo vypnuto směrování IPv6.

5.5 Konfigurace XCP-ng

V XCP-ng byla podle návrhu vypnuta podpora IPv6 a také byl nastaven uživatel `tester_user` s rolí `read-only`. Tento uživatel byl modifikován tak, aby měl nejmenší možná práva. Konkrétně se jedná o přístup k těmto metodám v XAPI:

- `VM.get_by_uuid`
- `VM.checkpoint`
- `VM.get_snapshots`
- `VM.get_snapshot_time`
- `VM.revert`
- `VM.resume`
- `VM.get_power_state`
- `VM.get_guest_metrics`
- `VM_guest_metrics.get_PV_drivers_detected`
- `VM_guest_metrics.get_PV_drivers_version`

Informace o takto vytvořeném uživateli lze najít ve výpisu 3.

Bohužel nelze v XCP-ng tato práva vztahovat na konkrétní VM. Pokud se tedy útočník zmocní uživatele `tester_user`, bude moci provádět dané operace na všech VM daného hostitele. Toto lze řešit například vytvořením samostatné služby, která bude poskytovat rozhraní pro ovládání VM. Na úrovni takovéhoto rozhraní už půjde ověřovat validita požadavku na detailnější úrovni. Při použití fyzické separace je uživatel vytvořen pouze na hostiteli *Worker Server* a může ovládat pouze VM na tomto hostiteli.

Na serveru byla také nakonfigurována autentizace pomocí PAM (Pluggable Authentication Module) pro XenAPI. Toho bylo docíleno vytvořením souboru `/etc/pam.d/xapi` s obsahem ve výpisu 4. Obsah souboru `/etc/xapi_allow` pak lze nalézt ve výpisu 5.

```
uuid ( RO)                  : d3d469d4-3d80-7bee-48c7-98487f181736
    subject-identifier ( RO): u1001
           other-config (MRO): subject-name: tester_user; subject-uid: u1001;
           ↪  subject-gid: g1001; subject-gecos: ; subject-displayname:
           ↪  tester_user; subject-is-group: false; subject-account-disabled:
           ↪  false; subject-account-expired: false; subject-account-locked:
           ↪  false; subject-password-expired: false
                  roles (SRO): read-only;
                  ↪  vm_guest_metrics.get_pv_drivers_version;
                  ↪  vm_guest_metrics.get_pv_drivers_detected;
                  ↪  vm.get_guest_metrics; vm.get_power_state; vm.resume;
                  ↪  vm.revert; vm.get_snapshot_time; vm.get_snapshots;
                  ↪  vm.checkpoint; vm.get_by_uuid
```

Výpis 3: Výpis informací o uživateli `tester_user` v XCP-ng

```
auth required pam_listfile.so item=user sense=allow file=/etc/xapi_allow

auth           include     system-auth
account    include     system-auth
password   include     system-auth
```

Výpis 4: Obsah souboru `/etc/pam.d/xapi`

```
root
tester_user
```

Výpis 5: Obsah souboru `/etc/xapi_allowed`

5.6 Podpůrné skripty

Pro podpůrné skripty byl zvolen jazyk Python 3 a Bash a jsou umístěny v adresáři `management_scripts`. Skripty napsané v Pythonu využívají standardní modul `argparse`, který poskytuje stručnou nápovědu a seznam argumentů. Tyto skripty nepodléhaly tak přísné kontrole kvality kódu jako řídicí kód Testeru. Jejich hlavním účelem je zrychlení procesu přípravy testovacího prostředí pro administrátora.

5.6.1 Hromadné vytváření Workerů

Hromadné vytváření Workerů se skládá ze dvou skriptů, přičemž tím hlavním je `create_workers.py`, který využívá `gen_srv_cert.sh` umístěný v adresáři

`management_scripts/certs`. V tomto adresáři musí být umístěn také adresář s nástroji EasyRSA[26] s vygenerovanou certifikační autoritou a také v něm musí být konfigurační soubor `req_conf.cnf`. Ke správnému fungování také musí být na OS dostupný příkaz `openssl`, který je používán ke generování certifikátu.

Skript `create_workers.py` vytvoří zadaný počet párů *Workerů*. Takto vytvořené VM budou mít název ve tvaru `WorkerPairN_Prep`, resp. `WorkerPairN_Jail`, kde N je index, podle kterého se určí IP adresa. Samotný proces vytváření probíhá tak, že se naklonuje WorkerTemplate_Prep, resp. WorkerTemplate_Jail, nastaví se jim IP adresy v síti TemplateNet a následně samotné certifikáty. Poté jsou VIF obou VM přiřazeny do správné sítě (PrepNet/JailNet) a je jim přiřazena jejich konečná IP adresa. Tento skript vytváří více *Workerů* najednou na úrovní párů, tedy využívá více vláken podle specifikace v parametrech. Nápověda ke skriptu je ukázána ve výpisu 6. Ukázkové použití, při kterém budou vytvořeny 3 páry *Workerů*, je vidět ve výpisu 7. Výstupem tohoto skriptu je soubor s konfigurací pro vytvořené *Workery*. Tato konfigurace se pak použije v souboru `runners.cfg` pro Tester. Vygenerovaná konfigurace neobsahuje specifické sekce, tedy například konfiguraci přístupu k XenAPI.

```
1  usage: create_workers.py [-h]
2                           xapi_ip xapi_username prep_template_name
3                           jail_template_name from_n to_n config_path thread_cnt
4
5  positional arguments:
6    xapi_ip             IP for XenAPI
7    xapi_username       Username for XenAPI
8    prep_template_name  Prep VM template name
9    jail_template_name  Jail VM template name
10   from_n              Starting number of WorkerPair
11   to_n                Ending number of WorkerPair
12   config_path         Path to config file that will be overwritten with new
13                       runners config
14   thread_cnt          Number of threads to use for setup (Worker VMs that will
15                       be running at once)
16
17 optional arguments:
18   -h, --help          show this help message and exit
```

Výpis 6: Nápověda ke skriptu `create_workers.py`

[26] `https://github.com/OpenVPN/easy-rsa`

```
1 python3 create_workers.py 169.254.0.1 root WorkerTemplate_Prep
  ↪ WorkerTemplate_Jail 1 3 runners.cfg 3
```

Výpis 7: Ukázka použití skriptu `create_workers.py`

5.6.2 Hromadné mazání Workerů

Pro hromadné mazání *Workerů* slouží skript `remove_workers.py`. Příklad jeho použití lze vidět ve výpisu 8.

```
1 python3 remove_workers.py 169.254.0.1 root 1 3
```

Výpis 8: Ukázka použití skriptu `remove_workers.py`

5.7 Porovnání s původním řešením

Na závěr implementace bylo nové řešení porovnáno s řešením původním, a to z pohledu bezpečnosti, rychlosti a stability.

5.7.1 Bezpečnost

Z pohledu bezpečnosti je systém SharpTest lépe separován, a to jak z pohledu běžících procesů, tak z pohledu dat. V původním řešení byl řídicí kód Testeru a testovací prostředí na jednom serveru. To znamenalo, že pokud by se podařilo testovanému studentskému řešení uniknout ze Sandboxie, měl by přístupná referenční řešení, jiná studentská řešení a mohl by vkládat nepravdivé hodnocení do databáze. Při použití virtualizace bylo testovací prostředí a řídicí kód Testeru separován, a to jak na úrovni VM, tak na úrovni fyzické při budoucím nasazení do produkce. Testované řešení by tedy muselo uniknout z omezeného uživatele v OS, dále překonat hypervisor a v produkčním nasazení i fyzickou separaci, což znamená omezenou komunikaci mezi *Control Serverem* a *Worker Serverem*. Přínosy virtualizace, ale i lepšího návrhu nasazení, jsou tedy poměrně značné.

Okamžitým výsledkem použití virtualizace místo sandboxu Sandboxie je pak zejména zamezení útoku omezení služby, které v původním řešení nebylo dobře ošetřeno. Mohlo ohrozit jak ostatní testovaná řešení, tak samotný spuštěný řídicí kód Testeru, čímž mohlo dojít i k dlouhodobému omezení služby.

5.7.2 Výkon

Již od začátku bylo počítáno s tím, že výkon testování nového řešení bude horší, což se ukázalo být pravdou. Důvodem zhoršení výkonu je zejména to, že lze testovat méně odevzdaných řešení najednou kvůli zvýšeným požadavkům VM na operační paměť. Dále VM potřebuje více času pro vyčištění prostředí, které probíhá po ohodnocení úlohy. Na čas potřebný k opravě samotné toto velký vliv nemá, tudíž pokud má systém kapacity, uvidí uživatel řešení za téměř stejnou dobu, jako tomu bylo v řešení původním. Při zaplnění kapacit se k tomuto času ale musí přičíst i ono čištění, které zabírá v závislosti na výkonu hostitele průměrně 25 vteřin na výkonnějším domácím počítači. Dalších cca 5 vteřin zabere, než se OS ve VM dostane do takového stavu, že začne odpovídat na síťové požadavky. Úzkým hrdlem obnovování checkpointů bylo při testování čtení z SSD (Solid State Drive) disku. Pokud se navíc sejde více požadavků na obnovení checkpointu, potřebná doba se ještě prodlouží. Při požadavku na rychlejší obnovování checkpointu by mohlo být vhodné použít pro úložiště RAID (Redundant Array of Independent Disks).

5.7.3 Stabilita

Nové řešení využívající virtualizace je stabilnější hlavně při čištění *Workerů*, kde u Sandboxie docházelo k problémům, které vyžadovaly ruční intervenci. Stejně tak při vytížení prostředků *Workera* už nedochází k ovlivňování ostatních *Workerů*, a tedy i samotného systému SharpTest.

5.8 Testování

Popsané řešení je otestováno na několika úrovních, a to jak testy manuálními, tak automatizovanými. Při vývoji bylo využíváno zejména smoke testů a po implementaci jednotlivých celků kódu unit testů k ověření jejich funkčnosti. Po dokončení integrace byly také připraveny testy integrační, kterými byla testována specificky integrace třídy `XenRunner` s dalšími třídami popsanými v předchozích kapitolách. Manuálními systémovými testy pak byla otestována integrace virtualizace do celého řešení na úrovni systému.

5.8.1 Unit testy

Implementované rozšíření a změny v řídicím kódu Testeru pokrývá 24 unit testů rozdělených do modulů `test_xen_runner`, `test_test_scheduler`, `test_runners_conf_manager` a `test_helpers`. Pro lokální spuštění unit testů byl vytvořen skript `.run_unit_tests.py`, který je umístěn v adresáři `tester`. Při vývoji byly unit testy průběžně spouštěny za pomoci CI (Continuous Integration). Výsledek spuštění unit testů lze vidět ve výpisu 9.

```
...
test_add_remove_runners_ (unit_tests.test_test_scheduler.test_scheduling)
Tests add_runners method ... ok
test_add_runners_fails (unit_tests.test_test_scheduler.test_scheduling)
Tests add runners failures ... ok
test_list_runners (unit_tests.test_test_scheduler.test_scheduling)
Tests runners listing ... ok
test_start_stop_ok (unit_tests.test_test_scheduler.test_scheduling)
Tests start and stop functionality ... ok
test_work (unit_tests.test_test_scheduler.test_scheduling)
Tests whether the work method runs module.run ... ok
test_connect (unit_tests.test_xen_runner.test_xen_runner)
Tests connection attempt to psrp ... ok
test_create_dir (unit_tests.test_xen_runner.test_xen_runner)
Tests create_dir method ... ok
test_download_dir
...
```

Výpis 9: Část výstupu skriptu `.run_unit_tests.py`

5.8.2 Integrační testy

Integrační testy mají za úkol ověřit funkčnost mezi nově přidanými komponentami systému SharpTest. Jedná se o 12 automatických testů (je použit modul `unittest` pro jejich lehčí správu) ve dvou modulech. Testovací modul `test_xen_runner.py` testuje funkčnost všech metod třídy `XenRunner` na spuštěných *Workerech*, zatímco testovací modul `test_runner_with_modules` testuje funkčnost tříd `XenRunner`, `TestScheduler` a modulu `csharp_io`, který je použit pro testování úloh typu vstup/výstup, na spuštěných *Workerech*, což umožňuje testování bez přístupu k databázi nebo Webu. Pro lokální spuštění integračních testů byl vytvořen skript `.run_integration_tests.py`, který je umístěn v adresáři `tester`. Výsledek spuštění integračních testů lze vidět ve výpisu 10.

5.8.3 Testy předpokladů zabezpečení virtualizovaného řešení

Testy předpokladů zabezpečení jsou testy, které ověřují, že použitá virtualizace jako zabezpečení systému SharpTest funguje. Stejně jako testy integrační jsou testy předpokladů zabezpečení automatizovány.

Tyto testy jsou rozděleny do modulů `test_malicious_programs.py`, `test_net_separation.py` a `test_user_privileges.py`. Měly by být spouštěny například po vytvoření nových *Workerů* za použití skriptu `run_security_tests.py` pro ověření funkčnosti zabezpečení.

```
Tests a case where solution fails first test ... ok
test_ok (integration_tests.test_runner_with_modules.test_csharp_io)
Tests a case where solution is supposed to get 100% mark ... ok
test_cleanup_create_files (integration_tests.test_xen_runner.test_basic)
Creates a few files and checks if files remain after cleanup() ... ok
test_cleanup_start_calc (integration_tests.test_xen_runner.test_basic)
Starts a few calc.exe processes and checks if they remain after cleanup() ...
↪  ok
test_create_dir (integration_tests.test_xen_runner.test_basic)
Tests create_dir ... ok
test_file_exists (integration_tests.test_xen_runner.test_basic)
Tests if file exists on Worker ... ok
test_move_file (integration_tests.test_xen_runner.test_basic)
Tests move_file ... ok
test_overtime_kill (integration_tests.test_xen_runner.test_basic)
Tests if XenRunner will kill a command that runs longer than timeout ... ok
test_run_command (integration_tests.test_xen_runner.test_basic)
Tests run_command and confirms it's execution on Worker ...
ok
test_unlink_file (integration_tests.test_xen_runner.test_basic)
Tests ulink_file ... ok
test_upload_download_dir (integration_tests.test_xen_runner.test_basic)
Tests upload_dir and download_dir ... ok
test_upload_download_file (integration_tests.test_xen_runner.test_basic)
Tests upload_file and download_file ... ok

----------------------------------------------------------------------
Ran 12 tests in 320.128s

OK
```

Výpis 10: Výstup skriptu `.run_integrations_tests.py`

Modul `test_net_separation` zkouší komunikovat s ostatními *Workery* a se všemi servery v systému SharpTest. Dá se použít jako test nastavení firewallu *Workera* nebo test nastavení pfSenseWorkers, přičemž pro druhý zmiňovaný případ je nutné, aby byla povolena odchozí komunikace ve firewallu *Workera*. To lze udělat ručně na vybraných *Workerech*.

Druhým modulem těchto testů je `test_user_privileges`, kde jsou testována správná práva uživatele *user* na OS *Workera*.

Posledním modulem je `test_malicious_programs`, který obsahuje malou sadu testovacích škodlivých programů, které simulují pokusy o útok, přičemž dva z nich se snaží o omezení služby a další z nich o otevření portu, pomocí

kterého by útočník server vzdáleně ovládal. Účelem těchto testů je demonstrace funkčnosti omezení škodlivých dopadů nedůvěryhodného kódu, protože všechny tyto programy by mohly mít fatální následky, pokud by byly spuštěny na nezabezpečeném systému. Omezení služby (DoS) by například fungovalo i s použitím sandboxu Sandboxie a mohlo by znamenat až pád celé části Tester systému SharpTest. Výsledek spuštění integračních testů lze vidět ve výpisu 11.

```
1  test_calc_exhaustion
   ↪  (security_tests.test_malicious_programs.test_malicous_zips) ... ok
2  test_tcp_listener (security_tests.test_malicious_programs.test_malicous_zips)
   ↪  ... ok
3  test_threads_exhaustion
   ↪  (security_tests.test_malicious_programs.test_malicous_zips) ... ok
4  test_forbidden_servers
   ↪  (security_tests.test_net_separation.test_net_separation)
5  Tests communication to other servers of SharpTest system (which should be
   ↪  forbidden) ... ok
6  test_workers_ping (security_tests.test_net_separation.test_net_separation)
7  Tests communication between Workers with ping ... ok
8  test_workers_wsman (security_tests.test_net_separation.test_net_separation)
9  Tests communication between Workers with WSMan-Test ... ok
10 test_check_groups (security_tests.test_user_privileges.test_privileges)
11 Tests user groups ... ok
12
13 ----------------------------------------------------------------------
14 Ran 7 tests in 645.171s
15
16 OK
```

Výpis 11: Výstup skriptu `.run_security_tests.py`

Závěr

Cílem této bakalářské práce bylo využít virtualizaci jako nástroj k zabezpečení již existujícího systému automatizovaného hodnocení úloh z programování.

Za účelem splnění tohoto cíle byly popsány problémy se spouštěním cizího kódu a definovány potenciální rizika, ale i způsoby, jak těmto rizikům předejít. Následovala rešerše technologií virtualizace, kde byly popsány základní typy virtualizace a pojmy nutné k orientaci v této oblasti. Dále byl analyzován systém SharpTest se zaměřením na část Tester, která má za úkol spouštět studentská řešení úloh a automatizovaně je hodnotit. V rešerši i analýze byl postupně představen program Sandboxie a byly uvedeny důvody vedoucí k potřebě jeho nahrazení.

Po této přípravě byl proveden výběr typu virtualizace a virtualizační platformy. Byla upřednostněna lepší izolace HW virtualizace před vyšším výkonem virtualizace na úrovni OS. Za virtualizační platformu byl zvolen XCP-ng s hypervisorem Xen a byl proveden návrh integrace této virtualizační platformy do systému SharpTest. Tento návrh popisuje změny v řídicím kódu Testeru, zvolené technologie a nástroje pro ovládání XCP-ng, komunikaci s instancemi testovacího prostředí a správu síťové komunikace. Dále obsahuje nový model nasazení systému SharpTest, návrh virtuálních sítí a popis separace jednotlivých instancí testovacího prostředí.

Navržené řešení je realizováno a popsáno z pohledu změn v řídicím kódu Testeru a z pohledu konkrétní konfigurace virtualizovaného prostředí. Realizované řešení bylo otestováno na více úrovních. Testování probíhalo automatizovaně na úrovni unit testů a testů integračních, sloužících k ověření funkčnosti změn v řídicím kódu spolu s novým testovacím prostředím. Také byly vytvořeny automatizované testy předpokladů pro zabezpečení nově vytvořeného řešení. Funkčnost celého systému pak byla ověřena manuálními systémovými testy. HW virtualizace se ukázala jako použitelné, i když oproti virtualizaci na úrovni OS pomalejší, řešení pro tento systém. Při návrhu a implementaci řešení bylo využíváno přístupu Defense in depth a principu minimálních privilegií.

Výsledkem této bakalářské práce je funkční integrace HW virtualizace do systému SharpTest za účelem zlepšení jeho bezpečnosti a funkčnosti. Jako takový tedy považuji cíl práce za splněný.

Dalšími kroky navazujícími na tuto práci bude zejména vytvoření produkčních testovacích modulů. Poté bude systém testován v reálném prostředí se studenty střední školy. Na základě tohoto testování pak bude systém dále upravován podle nových požadavků.

Literatura

1. RAY, Edward; SCHULTZ, Eugene. Virtualization security. *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research Cyber Security and Information Intelligence Challenges and Strategies - CSIIRW 09*. 2009. Dostupné z DOI: 10.1145/1558607.1558655.
2. WARTELL, Richard; MOHAN, Vishwath; HAMLEN, Kevin W.; LIN, Zhiqiang. Securing untrusted code via compiler-agnostic binary rewriting. *Proceedings of the 28th Annual Computer Security Applications Conference on - ACSAC '12*. 2012. Dostupné z DOI: 10.1145/2420950.2420995.
3. KUNWAR, Rakesh Singh; SHARMA, Priyanka. Malware Analysis. *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '16*. 2016. Dostupné z DOI: 10.1145/2905055.2905361.
4. TREND MICRO INCORPORATED. *Fake antivirus solutions increasingly have stolen code-signing certificates* [online]. 2014 [cit. 2020-04-20]. Dostupné z: https://blog.trendmicro.com/fake-antivirus-solutions-increasingly-stolen-code-signing-certificates/.
5. LANGLEY, Adam. *Further improving digital certificate security* [online]. 2013 [cit. 2020-03-29]. Dostupné z: https://security.googleblog.com/2013/12/further-improving-digital-certificate.html.
6. LIGH, Michael Hale.; CASE, Andrew; LEVY, Jamie; WALTERS, AAron. *The art of memory forensics: detecting malware and threats in Windows, Linux and Mac memory*. Wiley, 2014.
7. OSBORNE, Charlie. *OpenJPEG zero-day flaw leads to remote code execution* [online]. ZDNet, 2016 [cit. 2020-03-24]. Dostupné z: https://www.zdnet.com/article/openjpeg-zero-day-flaw-leads-to-remote-code-execution/.

8. SIKORSKI, Michael; HONIG, Andrew. *Practical malware analysis: the hands-on guide to dissecting malicious software*. No Starch Press, 2012.
9. ANDRESS, Jason. *The basics of information security: understanding the fundamentals of InfoSec in theory and practice*. Syngress, 2015.
10. SHOSTACK, Adam. *Threat modeling: designing for security*. Wiley, 2014.
11. MOZILLA CORPORATION. *Security Principles* [online]. Mozilla [cit. 2020-04-02]. Dostupné z: `https://infosec.mozilla.org/fundamentals/security_principles.html`.
12. CONROY, Sean. *History of Virtualization* [online]. 2019 [cit. 2020-03-27]. Dostupné z: `https://www.idkrtm.com/history-of-virtualization/`.
13. POGARCIC, Ivan; KRNJAK, David; OZANIC, Davor. Business Benefits from the Virtualization of an ICT Infrastructure. *International Journal of Engineering Business Management*. 2012, roč. 4, s. 42. Dostupné z DOI: `10.5772/51603`.
14. PORTNOY, Matthew. *Virtualization essentials*. Sybex, a Wiley brand, 2016.
15. SCARFONE, Karen; SOUPPAYA, Murugiah; HOFFMAN, Paul. Guide to Security for Full Virtualization Technologies. *NIST Special Publication*. 2011, č. 800-125.
16. BARRETT, Diane; KIPPER, Gregory; LILES, Samuel. *Virtualization and forensics a digital forensic investigator's guide to virtual environments*. Syngress, 2010.
17. THE XEN PROJECT. *Understanding the Virtualization Spectrum* [online] [cit. 2020-06-20]. Dostupné z: `https://wiki.xenproject.org/wiki/Understanding_the_Virtualization_Spectrum`.
18. VMWARE INCORPORATED. *Understanding VM snapshots in ESXi* [online] [cit. 2020-07-05]. Dostupné z: `https://kb.vmware.com/s/article/1015180`.
19. KOMPERDA, Terry. *Virtualization Security* [online]. 2015 [cit. 2020-06-12]. Dostupné z: `https://resources.infosecinstitute.com/virtualization-security-2/`.
20. ORACLE CORPORATION. *Java Security Architecture* [online] [cit. 2020-07-12]. Dostupné z: `https://docs.oracle.com/javase/7/docs/technotes/guides/security/spec/security-spec.doc1.html`.
21. ZIKÁN, Jiří. *Database Gitlab repository* [online] [cit. 2020-04-05]. Dostupné z: `https://gitlab.fit.cvut.cz/sharptest/database`.

22. PYTHON SOFTWARE FOUNDATION. *PEP 8 – Style Guide for Python Code* [online] [cit. 2020-05-10]. Dostupné z: https://www.python.org/dev/peps/pep-0008/.

23. KUBĚNA, Jan; ŠMÍD, Radek. *Analysis Wiki sharptest/Tester* [online] [cit. 2020-04-05]. Dostupné z: https://gitlab.fit.cvut.cz/sharptest/tester/wikis/analysis.

24. KUBĚNA, Jan; ŠMÍD, Radek. *Tester Gitlab repository* [online] [cit. 2020-04-05]. Dostupné z: https://gitlab.fit.cvut.cz/sharptest/tester.

25. RAD, Babak Bashari; BHATTI, Harrison John; AHMADI, Mohammad. An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*. 2017, roč. 17, č. 3, s. 228–235.

26. BISSON, Simon. *Windows 10: Containers are the future, and here's what you need to know* [online]. TechRepublic, 2020 [cit. 2020-05-20]. Dostupné z: https://www.techrepublic.com/article/windows-10-containers-are-the-future-and-heres-what-you-need-to-know/.

27. PEVELER, Matthew; MAICUS, Evan; HOLZBAUER, Buster; CUTLER, Barbara. Analysis of Container Based vs. Jailed Sandbox Autograding Systems. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 2018. Dostupné z DOI: 10.1145/3159450.3162307.

28. MICROSOFT CORPORATION. *Containers vs. virtual machines* [online] [cit. 2020-04-04]. Dostupné z: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm.

29. OR-MEIR, Ori; NISSIM, Nir; ELOVICI, Yuval; ROKACH, Lior. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Computing Surveys*. 2019, roč. 52, č. 5, s. 1–48. Dostupné z DOI: 10.1145/3329786.

30. FREEMAN, Nick. *Container Escapes: An Exercise in Practical Container Escapology • Capsule8* [online]. 2020 [cit. 2020-07-13]. Dostupné z: https://capsule8.com/blog/practical-container-escape-exercise/.

31. PRIZMANT, Daniel. *Windows Server Containers Are Open* [online]. 2020 [cit. 2020-05-20]. Dostupné z: https://unit42.paloaltonetworks.com/windows-server-containers-vulnerabilities/.

32. CRWILHIT. *Isolation Modes* [online] [cit. 2020-06-24]. Dostupné z: https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/hyperv-container.

33. THE XEN PROJECT. *Xen.org History* [online] [cit. 2020-05-10]. Dostupné z: http://www-archive.xenproject.org/community/xenhistory.html.

34. THE XEN PROJECT. *Xen Project* [online] [cit. 2020-03-28]. Dostupné z: https://xenproject.org/.

35. CITRIX SYSTEMS INCORPORATED. *Citrix Hypervisor - Server Virtualization and Management Software - Citrix* [online] [cit. 2020-03-27]. Dostupné z: https://www.citrix.com/products/citrix-hypervisor/.

36. XCP-NG TEAM. *Turnkey Open Source Hypervisor* [online] [cit. 2020-05-15]. Dostupné z: https://xcp-ng.org/.

37. ORACLE CORPORATION. *Oracle VM Virtualbox* [online] [cit. 2020-04-26]. Dostupné z: https://www.virtualbox.org/.

38. ORACLE CORPORATION. *Licensing: Frequently Asked Questions* [online] [cit. 2020-03-28]. Dostupné z: https://www.virtualbox.org/wiki/Licensing%5C_FAQ.

39. THE LINUX KERNEL COMMUNITY. *KVM* [online] [cit. 2020-04-20]. Dostupné z: https://www.linux-kvm.org/page/Main_Page.

40. PROXMOX SERVER SOLUTIONS GMBH. *Open-Source Virtualization Platform* [online] [cit. 2020-05-10]. Dostupné z: https://www.proxmox.com/en/proxmox-ve.

41. MICROSOFT CORPORATION. *Introduction to Hyper-V on Windows 10* [online] [cit. 2020-03-28]. Dostupné z: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/.

42. VMWARE INCORPORATED. *ESXi: Bare Metal Hypervisor* [online]. 2020 [cit. 2020-04-07]. Dostupné z: https://www.vmware.com/cz/products/esxi-and-esx.html.

43. GRANISZEWSKI, Waldemar; ARCISZEWSKI, Adam. Performance analysis of selected hypervisors (Virtual Machine Monitors - VMMs). *International Journal of Electronics and Telecommunications*. 2016, roč. 62, č. 3, s. 231–236. Dostupné z DOI: 10.1515/eletel-2016-0031.

44. THE XEN PROJECT. *Why Xen Project?* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://xenproject.org/users/why-xen/.

45. LAMBERT, Olivier. *Xen virtualization modes* [online]. XO Blog, 2018 [cit. 2020-07-17]. Dostupné z: https://xen-orchestra.com/blog/xen-virtualization-modes/.

46. MICROSOFT CORPORATION. *Windows 10 system requirements* [online] [cit. 2020-07-12]. Dostupné z: https://support.microsoft.com/en-us/help/4028142/windows-10-system-requirements.

47. MICROSOFT CORPORATION. *[MS-PSRP]: PowerShell Remoting Protocol* [online] [cit. 2020-04-22]. Dostupné z: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-psrp/602ee78e-9a19-45ad-90fa-bb132b7cecec.

48. MICROSOFT CORPORATION. Windows Remote Management - Win32 apps. *Win32 apps — Microsoft Docs* [online] [cit. 2020-04-29]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/winrm/portal.

49. BOREAN, Jordan. *pypsrp* [online] [cit. 2020-04-18]. Dostupné z: https://pypi.org/project/pypsrp/.

50. JOEYAIELLO. *WinRMSecurity - PowerShell* [online] [cit. 2020-05-07]. Dostupné z: https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/winrmsecurity?view=powershell-7.

51. ANSIBLE CORE TEAM. *psrp – Run tasks over Microsoft PowerShell Remoting Protocol* [online]. 2020 [cit. 2020-05-27]. Dostupné z: https://docs.ansible.com/ansible/latest/plugins/connection/psrp.html.

52. XAPI TEAM. *Xapi Project Docs* [online] [cit. 2020-05-10]. Dostupné z: https://xapi-project.github.io/xen-api/classes/vm.html.

53. RUBICON COMMUNICATIONS, LLC. *pfSense® - World's Most Trusted Open Source Firewall* [online] [cit. 2020-06-03]. Dostupné z: https://www.pfsense.org/.

54. XCP-NG TEAM. *Guides - VLAN Trunking in a VM* [online]. 2020 [cit. 2020-07-03]. Dostupné z: https://xcp-ng.org/docs/guides.html.

55. INAMDAR, Mohammed Suhel; TEKEOGLU, Ali. Security Analysis of Open Source Network Access Control in Virtual Networks. *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 2018. Dostupné z DOI: 10.1109/waina.2018.00131.

56. MICROSOFT CORPORATION. *Installation and Configuration for Windows Remote Management - Win32 apps* [online] [cit. 2020-07-01]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/winrm/installation-and-configuration-for-windows-remote-management.

57. BAZARGAN, Fatma; YEUN, Chan Yeob; ZEMERLY, Mohamed Jamal. State-of-the-Art of Virtualization, its Security Threats and Deployment Models. *International Journal for Information Security Research*. 2013, roč. 3, č. 3, s. 335–343. Dostupné z DOI: 10.20533/ijisr.2042.4639.2013.0039.

58. PEARCE, Michael; ZEADALLY, Sherali; HUNT, Ray. Virtualization: Issues, Security Threats, and Solutions. *ACM Computing Surveys*. 2013, roč. 45, č. 2, s. 1–39. Dostupné z DOI: 10.1145/2431211.2431216.

59. PIETRO, Roberto Di; LOMBARDI, Flavio. Virtualization Technologies and Cloud Security: Advantages, Issues, and Perspectives. *Lecture Notes in Computer Science From Database to Cyber Security*. 2018, s. 166–185. Dostupné z DOI: 10.1007/978-3-030-04834-1_9.

60. THE XEN PROJECT. *Xen Project Documentation* [online] [cit. 2020-04-19]. Dostupné z: https://xenproject.org/help/documentation/.

61. PYTHON SOFTWARE FOUNDATION. *logging - Logging facility for Python* [online] [cit. 2020-04-16]. Dostupné z: https://docs.python.org/3.6/library/logging.html.

62. VMWARE INCORPORATED. *VMware ESXi: The Purpose-Built Bare Metal Hypervisor* [online] [cit. 2020-04-28]. Dostupné z: https://www.vmware.com/cz/products/esxi-and-esx.html/.

63. HESS, Ken. *Understanding Hardware-Assisted Virtualization "ADMIN Magazine* [online] [cit. 2020-06-15]. Dostupné z: https://www.admin-magazine.com/Articles/Hardware-assisted-Virtualization.

Příloha A

Instalační manuál

Instalační manuál popisuje zprovoznění realizované integrace virtualizace do systému SharpTest pomocí připravených VM.

A.1 Import VM

Takto připravené VM jsou určené k nasazení na jednoho hostitele. Budou však popsány i modifikace nutné k využití fyzické separace a tedy nasazení na dva hostitele. Exportované VM jsou na přiloženém fyzickém médiu v adresáři `vms`. Konkrétně se jedná o:

- `TesterServer.xva`
- `DatabaseServer.xva`
- `pfSenseWorkers.xva`
- `pfSenseControl.xva`
- `WorkerPair1_Prep.xva`
- `WorkerPair1_Jail.xva`
- `WorkerTemplate_Prep.xva`
- `WorkerTemplate_Jail.xva`

Na těchto VM je připravena databáze a řídící kód Testeru. Po jejich importu jsou automaticky vytvořeny propojující sítě. K dispozici je i jeden pár *Workerů*. Po importu připraveného páru *Workerů* je nutné jejich spuštění a vytvoření checkpointu. K vytvoření dalších je pak možné využít skript `create_workers.py`, který je umístěn na serveru VM TesterServer v adresáři `/home/user/tester/management_scripts`. Přidání dalšího páru *Workerů* je prerekvizitou pro správnou funkčnost testů předpokladů zabezpečení.

Připravené VM jsou importovány do nainstalovaného XCP-ng, přičemž pro testování s jedním hostitelem stačí 12 GB operační paměti i s VM Xen Orchestry. Úložiště by pak mělo mít velikost alespoň 80 GB.

Při aplikování fyzické separace je doporučeno 6 GB paměti na *Control Serveru* a 6 GB paměti na *Worker Serveru*. Po importu VM na jejich servery je pak nutné vytvořit sdílenou virtuální síť TesterNet mezi hostiteli[27]. Tato síť je napojena na VIF VM TesterServer, pfSenseControl a pfSenseWorkers.

Samotný import lze provést přes webové rozhraní VM Xen Orchestra[28]. Po importu je nutné změnit virtuální síť u prvního VIF VM pfSenseControl, která má být nastavena na virtuální síť s přístupem k PIF.

Uživatelé a jejich hesla do OS VM jsou uložena v poznámkách u jednotlivých VM, které lze nalézt v Xen Orchestra nebo pomocí příkazu:

```
xe vm-param-get param-name=name-description uuid=<vm_uuid>
```

A.2 Nastavení XCP-ng

Pro účely testování je doporučeno využít souborový systém ext podporující thin provisioning. Na hostiteli musí být vytvořen uživatel `tester_user` s příslušnými oprávněními. Uživatele lze vytvořit pomocí těchto příkazů:

```
useradd tester_user
passwd tester_user  # zde je nutné nastavit heslo
# povolí autentizaci pomocí modulu PAM
xe pool-enable-external-auth auth-type=PAM service-name=pam
xe subject-add subject-name=tester_user
# vypíše uživatele, je nutné vykopírovat uuid uživatele tester_user
xe subject-list
# tyto příkazy přidají uživateli potřebné role, je nutné nahradit <tu_uuid> za
↪  uživatelovo uuid
xe subject-role-add role-name=read-only uuid=<tu_uuid>
xe subject-role-add role-name=vm.get_by_uuid uuid=<tu_uuid>
xe subject-role-add role-name=vm.checkpoint uuid=<tu_uuid>
xe subject-role-add role-name=vm.get_snapshots uuid=<tu_uuid>
xe subject-role-add role-name=vm.get_snapshot_time uuid=<tu_uuid>
xe subject-role-add role-name=vm.revert uuid=<tu_uuid>
xe subject-role-add role-name=vm.resume uuid=<tu_uuid>
xe subject-role-add role-name=vm.get_power_state uuid=<tu_uuid>
xe subject-role-add role-name=vm.get_guest_metrics uuid=<tu_uuid>
xe subject-role-add role-name=vm_guest_metrics.get_pv_drivers_detected
↪  uuid=<tu_uuid>
xe subject-role-add role-name=vm_guest_metrics.get_pv_drivers_version
↪  uuid=<tu_uuid>
```

[27] Návod lze najít zde: `https://xen-orchestra.com/docs/sdn_controller.html`.

[28] Která je ke stažení zde: `https://xen-orchestra.com/`.

Také musí být na hostiteli povolena autentizace pomocí PAM, což lze provést příkazem:

```
xe pool-enable-external-auth auth-type=PAM service-name=pam
```

Dále musí být vytvořen soubor `/etc/pam.d/xapi` s obsahem:

```
auth required pam_listfile.so item=user sense=allow file=/etc/xapi_allow
auth           include     system-auth
account    include     system-auth
password   include     system-auth
```

Obsah souboru `/etc/xapi_allow` pak vypadá následovně:

```
root
tester_user
```

A.3 Testování

Spuštění Testeru je prováděno lokálně pod uživatelem `user` na serveru TesterServer, který má IP adresu 192.168.66.3. Na tento server se lze připojit přes po nastavení cest (route) připojit přes SSH, ale lze také použít přímý přístup v nástroji Xen Orchestra. V adresáři `/home/user/tester/tester` se pak nachází soubor `tester.py`, který lze spustit následujícím příkazem:

```
python3 tester.py
```

Tester lze zastavit pomocí CTRL+C a následným počkáním na uložení všech jeho stavů.

Funkčnost řešení pak lze ověřit pomocí automatizovaných testů spuštěním skriptů `.run_integration_tests.py` a `run_security_tests.py`. Výstupy těchto skriptů lze nalézt i ve výpisech 10 a 11. Tyto skripty lze spustit následovně:

```
python3 .run_integration_tests.py
python3 .run_security_tests.py
```

Druhou formou testování může být přidání studentských odevzdání do databáze. Pro tyto účely jsou na serveru DatabaseServer v adresáři `/home/user/` v souboru database.sql připravené příkazy pro MySQL s ukázkovými příklady vložení nového odevzdání. V tomto adresáři je několik ZIP archivů, které lze použít jako vstup do těchto příkazů. Typické použití za uživatele `user` by pak vypadalo následovně:

```
cd /home/user
cp *.zip /tmp/
su root  # zadání hesla Ab123456
```

```
4  mysql -u root  # interaktivní připojení k databázi
5  use sharptest;
6  insert into submissions (state_id, task_id, user_uuid, solution_data,
   ↪  created_at, deleted_at) values (1, 1,
   ↪  '11cba162-775b-11ea-8b5e-ac9e174aa25c', load_file('/tmp/ok.zip'), now(),
   ↪  null);
7  quit
```

Spuštěný Tester tato odevzdání začne opravovat. Pro účely testování je v Testeru zapnuto logování na úrovni debug zpráv. Tyto logy jsou v souboru `tester.log`.

Příloha B

Demonstrační videa

Na přiloženém fyzickém médiu jsou v adresáři `demo` k dispozici demonstrační videa. Tato videa ukazují funkčnost a správnou integraci vytvořeného řešení do Testeru a celého systému SharpTest.

Konkrétně se jedná o tato videa:

- `system_testing_demo.mp4` - ukazuje průběh systémových testů, ve kterých je do databáze systému SharpTest nahráno několik (10) odevzdání. Tato odevzdání jsou různého typu a jsou ve videu Testerem opravena.
- `management_scripts.mp4` - demonstruje dodané skripty pro hromadné vytváření a mazání *Workerů*.
- `tests_demo.mp4` - ukazuje funkčnost unit testů, integračních testů a testů předpokladů zabezpečení.

Příloha C

Seznam použitých zkratek

API Application programming interface

AV Antivirus

CI Continuous Integration

CoW Copy-on-Write

DoS Denial-of-Service

HVM Hardware Virtual Machine

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HW Hardware

IDS Intrusion Detection System

LAN Local Area Network

OOP Object Oriented Programming

OS Operating System

PAM Pluggable Authentication Module

PIF Physical Network Interface

PSRP PowerShell Remoting Protocol

PV Paravirtualization

PVHVM Paravirtualization on Hardware Virtual Machine

RAID Redundant Array of Independent Disks

SDN Software Defined Network

SSD Solid State Drive

SSH Secure Shell

VIF Virtual Network Interface

VLAN Virtual Local Area Network

VM Virtual Machine

WinRM Windows Remote Management

WSMan Web Services for Management

XAPI Xen Project management API

Příloha D

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `src` ..... výsledná část projektu Tester po změnách
  - `management_scripts` ..... pomocné skripty
    - `certs` ..... certifikáty použité na ukázkových strojích
  - `tester` ..... řídicí kód Testeru
    - `integration_tests` ..... integrační testy
    - `security_tests` ..... testy předpokladů zabezpečení
    - `test_modules` ..... testovací moduly Testeru
    - `unit_tests` ..... unit testy
- `vms` ..... virtuální stroje k otestování řešení
- `demo` ..... videa demonstrující funkčnost řešení
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF
- `diff.txt` ..... seznam změn v projektu Tester