VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DYNAMICKÁ ANALÝZA BEZPEČNOSTI APLIKACÍ S VYUŽITÍM OPENVAS

DYNAMIC ANALYSIS OF APPLICATION SECURITY USING OPENVAS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ ZÁLEŠÁK**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Zálešák Tomáš**
Program: Informační technologie
Název: **Dynamická analýza bezpečnosti aplikací s využitím OpenVAS**
Dynamic Analysis of Application Security Using OpenVAS
Kategorie: Informační systémy
Zadání:

1. Seznamte se s literaturou na téma hledání zranitelnosti systémů a seznamte se s OpenVAS - nástrojem pro hledaní zranitelnosti.

2. Seznamte se s nástroji pro průběžnou integraci (primárně s Atlassian Bamboo) a proveďte analýzu možnosti integrace OpenVAS s těmito nástroji.

3. Integrujte OpenVAS s Atlassian Bamboo (volitelně s Travis CI / CircleCI / Jenkins) a tím, že analýza se musí spouštět pravidelně, musí být dostatečně konfigurovatelná pro potlačování falešných detekcí a musí být použitelná pro regresní testování bezpečnosti, zvažte použití Greenbone Security Manager, nástroje pro správu zranitelnosti, který zahrnuje OpenVAS.

4. Ověřte implementované řešení na vybrané aplikaci s otevřeným kódem.

5. Vyhodnoťte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- http://www.openvas.org/
- dále dle pokynů vedoucího a konzultanta

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-2 zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Zemčík Pavel, prof. Dr. Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 5. listopadu 2019

Abstrakt

Tato bakalářská práce se zabývá dynamickou analýzou bezpečnosti aplikací a systémů a její automatizací. Ke hledání zranitelností je využit skener OpenVAS a pro automatizaci nástroj pro průběžnou integraci Atlassian Bamboo. Problém je vyřešen pomocí infrastruktury tří virtuálních počítačů, první s frameworkem Greenbone Vulnerability Management, který obsahuje skener OpenVAS, druhý s Atlassian Bamboo a třetí pro instalaci skenovaného systému. Jako virtualizační platforma je využit VirtualBox. Vytvořené řešení je plně automatizované, umožňuje automatickou detekci nových zranitelností a potlačení falešných detekcí. Funkčnost řešení byla ověřena na pravidelném skenování zranitelností systému Ubuntu 18.04 s nasazenou aplikací Rocket.Chat.

Abstract

This bachelor thesis covers the topic of dynamic analysis of application security and its automation. OpenVAS scanner is used to detect vulnerabilities and Atlassian Bamboo for automation. The problem is solved using the infrastructure of three virtual machines, the first with the Greenbone Vulnerability Management framework, which includes OpenVAS scanner, the second with Atlassian Bamboo, and the third for the installation of the scanned system. VirtualBox is used as a virtualization platform. The created solution is fully automated, enables automatic detection of new vulnerabilities and suppression of false detections. The functionality of the solution was verified by regularly scanning the vulnerabilities of the Ubuntu 18.04 system with the Rocket.Chat application deployed.

Klíčová slova

OpenVAS, Atlassian Bamboo, bezpečnost, dynamická analýza bezpečnosti, hledání zranitelností, průběžná integrace, automatizace, Greenbone Security Manager

Keywords

OpenVAS, Atlassian Bamboo, security, dynamic analysis of security, vulnerability scanning, continuous integration, automation, Greenbone Security Manager

Citace

ZÁLEŠÁK, Tomáš. *Dynamická analýza bezpečnosti aplikací s využitím OpenVAS*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Dr. Ing. Pavel Zemčík

Dynamická analýza bezpečnosti aplikací s využitím OpenVAS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Zálešák
10. června 2020

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce prof. Dr. Ing. Pavlu Zemčíkovi za cenné rady, připomínky a vedení práce. Dále bych chtěl poděkovat RNDr. Andriymu Stetskovi, Ph.D., Mgr. Davidu Formánkovi, Mgr. Janu Stárkovi a společnosti Y Soft za dobrou spolupráci při vypracování praktické části práce.

Obsah

Kapitola 1

Úvod

V poslední době se o počítačových chybách hodně mluví. Počítače se využívají čím dál víc a zasahují prakticky do všech průmyslových odvětví. Každý den si čteme internetové zprávy, komunikujeme přes sociální sítě, platíme pomocí internetového bankovnictví, voláme a spoustu dalších věcí. Počítače řídí vše a tím pádem firmy, které vyvíjí počítače a jejich programy, mají čím dál větší zodpovědnost. Jednou z možností, jak něco pokazit, je použití počítačového programu, jenž obsahuje chyby, které je možné zneužít. Jednotlivé chyby jsou postupně objevovány a dokumentovány, čímž je umožněna jejich detekce dříve, než je někdo zneužije.

Existují nástroje jako OpenVAS, které oskenují daný systém s nasazeným softwarem a detekují nalezené zranitelnosti. Týmy spolupracující na vývoji softwaru používají spoustu nástrojů na zjednodušení a urychlení vývoje jejich softwaru. Jeden z nástrojů, který tohle umožňuje je Atlassian Bamboo.

Cílem téhle práce je využít nástroje OpenVAS a Bamboo ke zdokonalení automatické detekce zranitelností tak, aby bylo možné odhalit zranitelnosti dřív, než se dostanou k zákazníkům.

Zadání téhle práce vzniklo jako spolupráce s firmou Y Soft na rozšíření nástrojů detekce zranitelností při vývoji produktu Y Soft SafeQ.

Práce obsahuje úvod do problematiky v kapitole 2, ve které uvede čtenáře do zranitelností systémů, skenování zranitelností a průběžné integrace. Kapitola 3 obsahuje zhodnocení dosavadního stavu, specifikaci požadavků a plán práce. V kapitole 4 uveden popis vlastní práce, jenž popisuje návrh řešení, jeho implementaci a ověření funkčnosti s interpretací výsledků. Výsledky práce jsou shrnuty v závěru práce v kapitole 5.

Kapitola 2

Shrnutí současného stavu

Tato kapitola obsahuje krátký úvod do hledání zranitelností systémů a použití nástrojů pro průběžnou integraci tak, aby čtenáře uvedla do dané problematiky, a tak nemusí nutně obsahovat všechny poznatky a nemá encyklopedický charakter. Je rozdělena do 4 podkapitol - zranitelnosti systémů, skenování zranitelností a OpenVAS, průběžná integrace a Atlassian Bamboo a dostupná řešení implementace OpenVAS s nástroji průběžné integrace.

2.1 Zranitelnosti systémů

Chyby se dějí i při návrhu softwarových systémů a programování. To, co po chybách zůstane, je označováno jako softwarové chyby. Softwarové chyby jako takové nemusí být ihned brány jako škodlivé. Ty softwarové chyby, které lze potencionálně zneužít jsou označovány jako zranitelnosti. Zranitelnost lze definovat jako chybu nebo slabinu v návrhu systému, implementaci, nebo provozu a řízení systému, které by mohly být využity k porušení bezpečnostní politiky systému [18].

Bezpečnostní slabiny

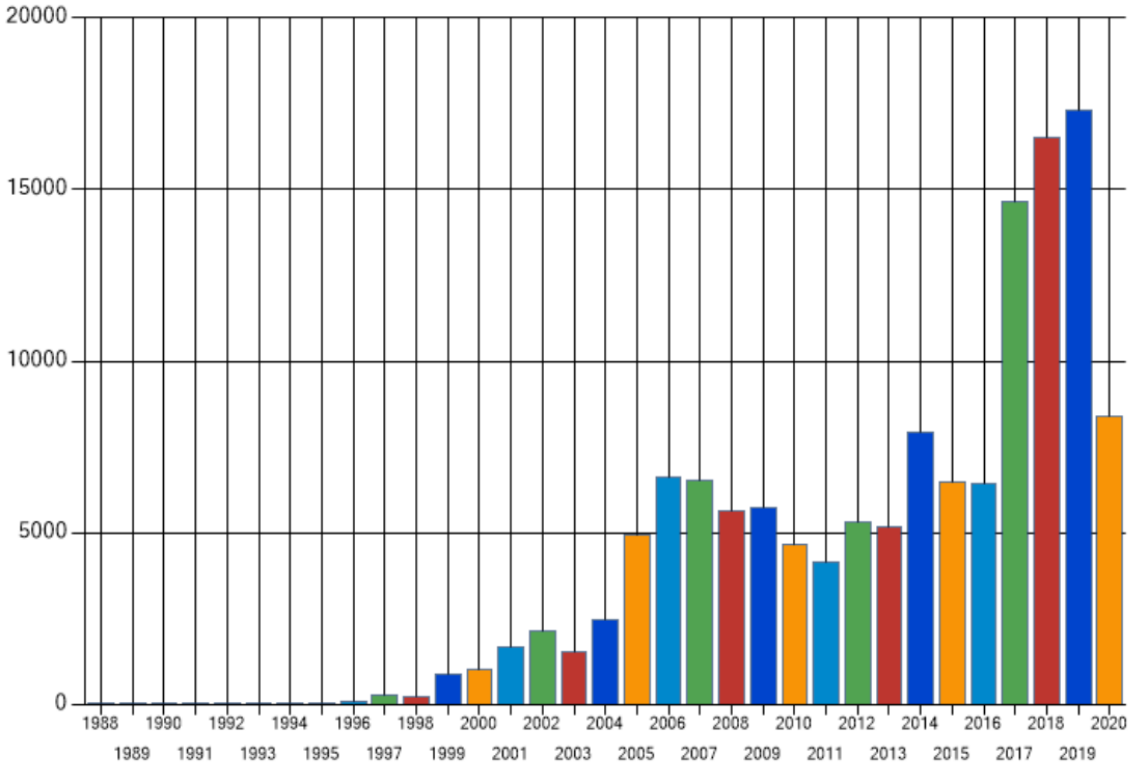
Nezisková společnost MITRE[1], jež je sponzorována vládou Spojených států Amerických, založila seznam častých softwarových a hardwarových slabin *Common Weakness Enumeration (CWE)*[2], které mají bezpečnostní následky. Slabiny mohou být chyby, zranitelnosti a jiné problémy v implementaci, návrhu nebo architektuře systémů, sítí a hardwaru, které následně mohou umožnit zneužití. Jakmile je softwarová chyba považována za zranitelnost, je zaregistrována společností MITRE jako CVE.

Standardizace názvů bezpečnostních chyb

Common Vulnerabilities and Exposures (CVE)[3] je seznam standardizovaných názvů pro zranitelnosti a další ohrožení informační bezpečnosti. CVE se snaží o standardizaci názvů všech veřejně známých bezpečnostních chyb. CVE teda není databáze zranitelností, ale pouze jejich názvů [14]. CVE je navrženo tak, aby bylo možné spojit více databází zranitelností dohromady a umožnit tak srovnání bezpečnostních nástrojů a služeb. CVE tedy neobsahuje informace o risku, ty je možné najít až v databázích zranitelností.

[1] https://www.mitre.org/
[2] https://cwe.mitre.org/
[3] https://cve.mitre.org/

Jen v roce 2019 bylo zveřejněno 17306 nových zranitelností. Největší skok v nárustů zranitelností byl v roce 2017. Bylo zaznamenáno 14645 nových zranitelností oproti pouze 6447 v roce 2016 [10].



Obrázek 2.1: Přehled nově zveřejněných zranitelností podle roku. Převzato z https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&search_type=all

Hodnocení závažnosti zranitelností

Common Vulnerability Scoring System (CVSS) je otevřený standard pro hodnocení závažnosti zranitelností. Jedná se o nejběžněji používaný standard. Za jeho vznikem stojí nezisková americká organizace Forum of Incident Response and Security Teams. Samotné hodnocení závažnosti zranitelností se podle CVSS skládá ze tří skupin metrik. První skupinou je *základní (base)*, druhou *časová (temporal)* a třetí *okolnostní (environmental)* [15].

Základní metrika se týká charakteristiky zranitelnosti, jež je neměnná v čase a nezávisí na jiných okolnostech. Časová metrika upravuje závažnost zranitelností až z pohledu času. Rozděluje zranitelnosti například podle toho, zda je daná zranitelnost popsána pouze teoreticky, nebo byla již někdy zneužita. Dále hodnotí to, jestli jsou dostupné oficiální záplaty či jiné možné řešení. Třetí skupina zohledňuje význam dané zranitelnosti v systému či infrastruktuře [3].

Základní metrika

Základní metrika reprezentuje charakteristiku zranitelnosti, jež není závislá na čase a na rozdílných uživatelských prostředích. Skládá se ze dvou dílčích metrik - *zneužitelností* a *dopadem*. Tohle rozdělení umožňuje rozdělit komponenty obsahující zranitelnost a komponenty, které mohou být dotčeny příslušnou zranitelností. Zneužitelnost reflektuje jednoduchost využití zranitelnosti a její technické metody. Skládá se z následujících charakteristik [16]:

- *Attack Vector (AV)* - vektor útoku vyjadřující, jak je možné zranitelnost využít
- *Attack Complexity (AC)* - vyjadřuje složitost podmínek, které musí být splněny pro provedení útoku, a jejich spolehlivost (některé využití zranitelností může například záviset na náhodě)
- *Privileges Required (PR)* - nutná oprávnění k provedení útoku na zranitelný systém
- *User Interaction (UI)* - nutnost uživatelské akce k provedení útoku (otevření přílohy e-mailu, spuštění programu, navštívení webové stránky)

Druhou základní metrikou je dopad. Ten je vyhodnocován podle následujících pravidel:

- *Confidentiality (C)* - značí, jak moc důvěrná data lze získat zneužitím zranitelnosti (to znamená například, zda útočník získá oprávnění čtení dat)
- *Integrity (I)* - označuje, zda útočník může poškodit či upravit data (zda může získat oprávnění zápisu a úpravy)
- *Availability (A)* - týká se dopadu na dostupnost, útok může například vyřadit komponentu z provozu

Do základní metriky patří i její vlastnost *Scope (S)*. Scope zde znamená autorizační prostor, který je souborem uživatelských práv definovaných nějakou autoritou. Autoritou můžeme myslet například aplikaci, operační systém nebo firmware. Tato vlastnost vyjadřuje, zda lze zneužitím zranitelnosti ovlivnit zdroje, které jsou v jiném autorizačním prostoru. Příkladem je to, že aplikace může smazat data jiné aplikaci, počítač může číst data ve druhém počítači [3].

Tím jsme uzavřeli základní metriku závažnosti zranitelností, jejímž výsledkem jsou *skóre a vektorový řetězec*. Skóre je určeno číslem na stupnici od 0 do 10. Závažnost lze vyjádřit také slovně v intervalech uvedených v tabulce 2.1 [3]. Vektorový řetězec je zkrácený slovní popis jednotlivých dílčích metrik, jak znázorňuje obrázek 2.2.

Tabulka 2.1: Slovní popis závažnosti zranitelností

Skóre základní metriky	Závažnost
0	žádná
0,1-3,9	nízká
4-6,9	střední
7-8,9	vysoká
9-10	kritická

Obrázek 2.2: Příklad Vector String. Vygenerováno na https://www.first.org/cvss/calculator/3.1

Časová metrika

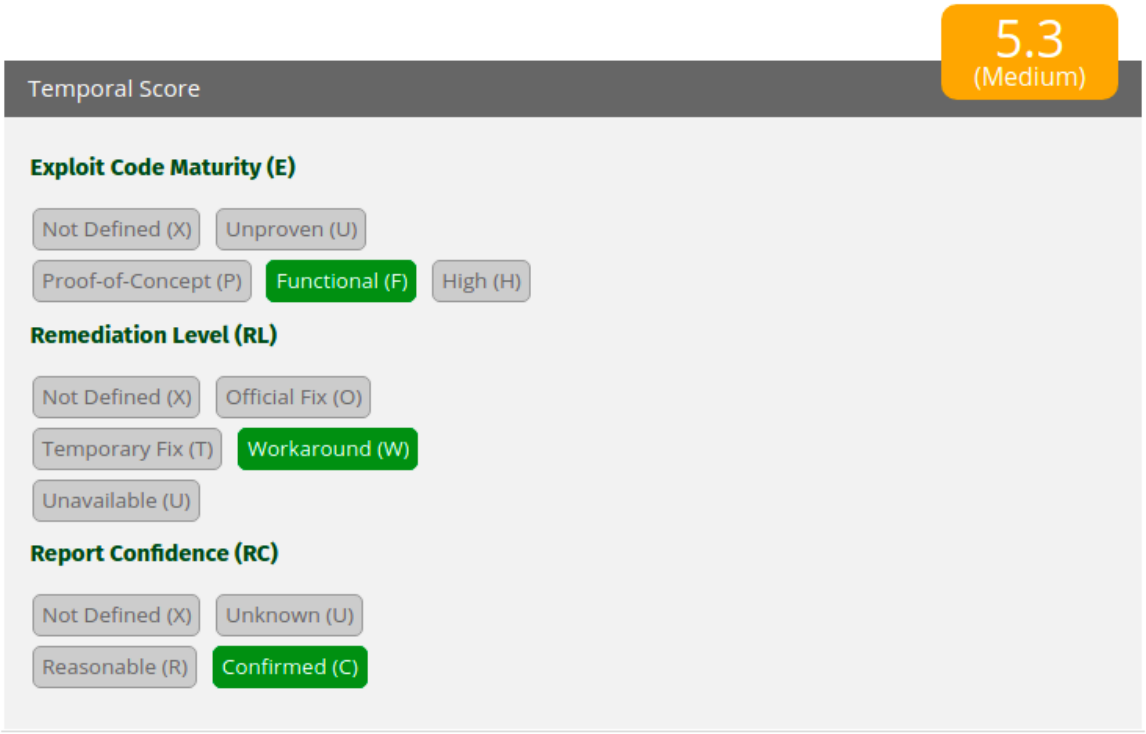
Po základní metrice zde máme metriku časovou, do které spadají charakteristiky, jež se mohou v čase měnit. Patří sem následující vlastnosti:

- *Exploit Code Maturity (E)* - určuje pravděpodobnost útoku na danou zranitelnost, pro nově zveřejněné zranitelnosti je například zneužití jen teoretické, nicméně postupem času mohou vzniknout volně dostupná řešení pro zneužití, což pravděpodobnost zvýší
- *Remediation Level (RL)* - stav protiopatření určuje, jestli a jaké jsou dostupná řešení zranitelností (oficiální patch, workaround)
- *Report Confidence (RC)* - definuje míru důvěryhodnosti zranitelností, například nově objevené zranitelnosti bez bližších podrobností ji mají menší než zranitelnosti oficiálně potvrzené výrobcem

Okolnostní metrika

Okolnostní metrika upravuje výsledné skóre na základě míry rizika vzhledem ke konkrétnímu prostředí zasažené komponenty. Zranitelnost má jinou roli v systému, jenž obsahuje data kriticky důležité pro vaši organizaci narozdíl od systému, který taková data neobsahuje. Okolnostní metrika je vyjádřena následujícími body [16]:

- *Confidentiality Requirement (CR)* - nárok na důvěrnost dat v systému, který může být zranitelností zasažen



Obrázek 2.3: Příklad časové metriky. Vygenerováno na https://www.first.org/cvss/calculator/3.1

- *Integrity Requirement (IR)* - nárok na integritu dat v systému, který může být zranitelností zasažen
- *Availability Requirement (AR)* - nárok na dostupnost dat v systému, který může být zranitelností zasažen

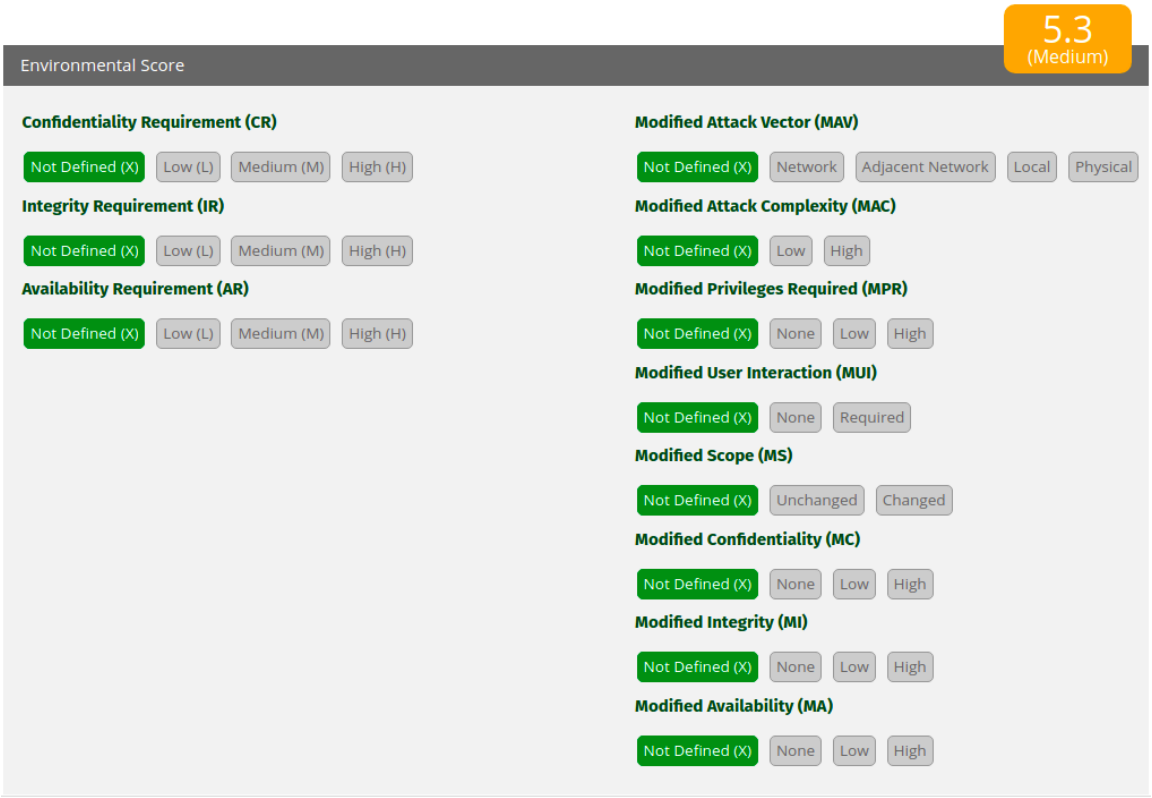
Databáze zranitelností

Databáze zranitelností je platforma, která sbírá, spravuje a sdílí informace o nalezených zranitelnostech.

National Vulnerability Database

National Vulnerability Database (NVD)[4] je databáze zranitelností spravovaná vládou Spojených států Amerických. Webové rozhraní NVD umožňuje hledání informací o konkrétních zranitelnostech i stažení celé databáze ve formátech XML nebo JSON. Databáze je pravidelně aktualizována a poskytuje zranitelnosti ve striktně daném formátu, který zjednodušuje další zpracování dat. Databáze je spravována jinou organizací, nicméně položky v NVD jsou pravidelně synchronizovány se změnami v CVE [5].

[4] https://nvd.nist.gov/



Obrázek 2.4: Okolnostní metrika a její hodnoty. Vygenerováno na https://www.first.org/cvss/calculator/3.1

Snyk Vulnerability Database

Druhou kvalitní databází zranitelností je Snyk Vulnerability Database[5]. Tahle databáze je využívána především samotným řešením pro hledání zranitelností Snyk. Zranitelnosti lze procházet ve webovém rozhraní jako u NVD, nicméně není poskytnuta možnost stáhnout všechny zranitelnosti pro další práci se zveřejněnými zranitelnostmi. V databázi lze hledat jak podle názvu zranitelnosti, tak podle CVE. Zvláštností téhle databáze je, že zranitelnosti jsou třízeny podle toho, jakým systémem jsou spravovány. Například knihovny programovacího jazyka Python jsou zařazeny do složky pip, což systém správy balíčků pro moduly Pythonu [9].

Příklady zranitelností

Existuje mnoho kritérií, podle kterých lze zranitelnosti třídit.

Příklady zranitelností podle OWASP [7]:

- *Injekce* - Chyby injekce (např. SQL, NoSQL, OS, LDAP injekce) nastávají, když aplikace interpretuje dotaz či příkaz, které obsahují neověřená data. Data vložená útočníkem můžou zmást aplikaci a ta provede příkazy nebo přistoupí k datům bez řádné autentizace.

[5] https://snyk.io/vuln

- *Nefunkční autentizace* - Funkce aplikace, které zajišťují autentizaci a pracují se session, jsou často špatně implementovány a umožňují útočníkovi přístup k heslům, klíčům nebo ke zneužití jiných chyb implementace a k převzetí identit uživatelů.

- *Externí entity XML (XXE)* - Starší nebo špatně nastavené parsery XML můžou zpracovat data od uživatele s externími entitami XML a následně je zpracovat. Tahle chyba může dovolit útočníkovi vyvolání odepření služeb, oskenování vnitřní sítě, spuštění příkazů a případně čtení místních souborů.

- *Nefunkční oprávnění přístupu* - Přihlášení uživatelé můžou přistupovat na místa, kam by mít oprávnění neměli. Útočníci můžou tyhle chyby využít a například zobrazit soubory s citlivými informacemi, změnit oprávnění přístupu, číst nebo měnit data jiných uživatelů.

- *Špatná konfigurace zabezpečení* - Jedná se o nejčastější problém. Tenhle problém často pramení v použití výchozí konfigurace, neúplné konfigurace nebo zbytečně podrobného chybového hlášení obsahující citlivé údaje. Nejen že musí být všechny operační systémy, aplikace, frameworky a knihovny správně nakonfigurovány, ale také musí být s nainstalovanými aktualizacemi a bezpečnostními záplatami.

- *Cross-site Scripting (XSS)* - XSS jsou druhem injekce, ve které jsou do jinak důvěryhodných webových stránek vloženy škodlivé skripty. Škodlivé skripty se posílají pomocí webových aplikací na stranu klienta k uživateli. Tyhle chyby jsou docela rozšířené a objevují se všude, kde uživatel zadává neošetřená data do webové aplikace. Útočník pak může libovolně pozměnit webovou stránku na straně klienta a využít to ke krádeži informací umístěných ve stránce nebo jako počáteční krok k dalším útokům.

- *Nezabezpečená deserializace dat* - Ta často vede ke vzdálenému spuštění kódu nebo jiným útokům.

- *Použití komponent se známými zranitelnostmi* - Komponenty jako knihovny, frameworky a jiné softwarové moduly běží se stejnými oprávněními jako samotná aplikace. Pokud je zranitelná komponenta využita, může způsobit ztrátu dat nebo ovládnutí serveru. Aplikace či API (rozhraní pro programování aplikací) používající zranitelné komponenty můžou také vytvořit bezpečnostní díru a umožnit další útoky.

- *Nedostatečné logování a monitoring* - Nedostatečné logování a monitoring spojený s chybějící či pozdní reakce na případný incident dovoluje útočníkům pokračovat v útoku či pokračovat v dalších. Podle studií je doba detekce asi 200 dní a typicky je útok detekován třetí stranou a ne monitoringem interních procesů.

Tohle samozřejmě není jediné dělení zranitelností. OWASP zahrnuje především zranitelnosti související s webovými aplikacemi.

2.2 Skenování zranitelností a OpenVAS

Skener zranitelností je automatizovaný nástroj navržený tak, aby hledal zranitelnosti v počítačích, sítích a aplikacích. Používají se především ke hledání zranitelností ve webových aplikacích a počítačových systémech. Skenery zranitelností jsou označovány jako nástroje

dynamického testování bezpečnosti aplikací. Dynamická analýza hledá zranitelnosti v již spuštěných aplikacích a nasazených systémech narozdíl od statické, která vyhodnocuje zdrojový kód. Skenování zranitelností se dělí na autentizované a neautentizované.

- *Autentizované skenování* - Umožňuje skeneru se vzdáleně přes síť přihlásit na počítač. Vzdálený přístup umožňují protokoly vzdáleného přístupu jako Secure Shell (SSH) nebo Remote Desktop Protocol (RDP). Stačí jim poskytnout přihlašovací údaje. Přihlášení umožní skeneru přístup k datům jako jsou spuštěné služby, konfigurační detaily, instalovaný software, či chybící aktualizace operačního systému.
- *Neautentizované skenování* - Spočívá ve vyhodnocení zranitelností přístupných na otevřených portech.

Mezi nástroje umožňující skenování zranitelností patří kromě OpenVAS například komerční skener Nessus[6], Acunetix WVS[7], Burp Suite[8], w3af[9] a N-Stealth[10] [11].

OpenVAS

Open Vulnerability Assessment Scanner je plnohodnotný skener zranitelností. Mezi jeho schopnosti patří jak neautentizované, tak autentizované skenování. Testuje internetové i komerční protokoly a je možné skenování optimalizovat. Následně byl OpenVAS registrován jako projekt v neziskové organizaci Software in the Public Interest, Inc., aby si chránil svou doménu openvas.org [6].

Historie OpenVASu

V roce 2005 se vývojáři skeneru zranitelností Nessus[11] rozhodli, že skončí jeho vývoj s otevřeným zdrojovým kódem. OpenVAS vznikl jako fork (alternativní větev vyvíjeného programu) skeneru Nessus a pokračovalo se v jeho vývoji s otevřeným zdrojovým kódem. Následně byla v Německu založena firma Greenbone Networks GmbH[12], aby se starala o další vývoj nástroje OpenVAS. Firma si stanovila tři hlavní úkoly:

- Pokročit ze skenování zranitelností na kompletní řešení správy zranitelností.
- Vytvořit hotové řešení pro firmy.
- Pokračovat v konceptu otevřeného softwaru.

Toho roku se staly aktivními další dvě firmy - indický Secpod a Security Space z Kanady. Obě měly zájem se podílet na rozšíření testů zranitelností a spolu s firmou Greenbone začaly tvořit spolehlivou a aktuální databázi testů. Nejprve vyčistili zdrojové soubory a testy zranitelností, kde nebyla jasná či kompatibilní licence. Databáze s testy se začala rychle rozšiřovat. Greenbone přidal moduly pro řešení správy zranitelností. Jednalo se o webové rozhraní a centrální službu pro správu zranitelností. V roce 2010 se dostalo řešení Greenbone

[6] https://www.tenable.com/products/tenable-io/web-application-scanning/
[7] https://www.acunetix.com/
[8] http://www.portswigger.net/
[9] http://www.w3af.org/
[10] http://www.nstalker.com/
[11] https://www.tenable.com/products/nessus
[12] https://www.greenbone.net/

Security Manager poprvé na trh. Během let 2010 až 2016 byl komerční produkt spolu s moduly s otevřeným zdrojovým kódem postupně vylepšován a rozšiřován. Řešení správy zranitelností bylo zdokonaleno a začalo získávat každodenní aktualizace bezpečnostních doporučení a podpořili svobodný software licencí GNU GPL[13], jenž byla kompatibilní s německými skupinami DNF-CERT[14] a CERT-Bund[15] [6].

V roce 2017 začala nová éra. Firma Greenbone přejmenovala framework (struktura podpůrných programů a knihoven) obsahující skener OpenVAS na Greenbone Vulnerability Management (GVM). K žádným změnám licencí nedošlo a po verzi OpenVAS 9 následoval GVM 10. Druhá velká změna se týkala služby databáze testů zranitelností. Spousta firem zneužívala koncepci otevřeného zdrojového kódu a nepoužívalo správně licenci GPL. To donutilo firmu Greenbone vytvořit zvláštní komerční databázi Greenbone Security Feed (GSF) a veřejnou přejmenovat na Greenbone Community Feed (GCF). Z veřejné databáze byly vyřazeny komerční testy zranitelností. Třetí velkou změnou bylo přesunutí vývoje na modernější infrastrukturu - GitHub[16] a komunitní fórum[17]. V roce 2019 byl rebranding dokončen a OpenVAS nyní označuje pouze skener zranitelností jako takový.

V roce 2019 byla vydána zatím poslední verze frameworku GVM s pořadovým číslem 11. Byla aktualizována celá architektura řešení. Původní služba `openvassd` se změnila na konzolovou aplikaci `openvas`. Ta je nyní obsluhována službou `ospd-openvas`. Tenhle koncept nahradil původní stavový protokol OpenVAS Transfer Protocol novým bezestavovým Open Scanner Protocol, který je založen na konceptu požadavků a odpovědí v XML formátu.

Nyní firma Greenbone poskytuje zdrojový kód frameworku GVM na GitHub. Své komerční řešení dodává spolu s nastaveným operačním systémem Greenbone OS. Řešení je poskytováno jako virtuální nebo fyzický stroj. Tohle kompletní řešení umožňuje firma vyzkoušet jako Greenbone Community Edition, což je omezená verze Greenbone Security Manageru One. Operační systém Greenbone OS je založen na linuxové distribuci Debian[18], který je distribuován také pod licencí GPL [6].

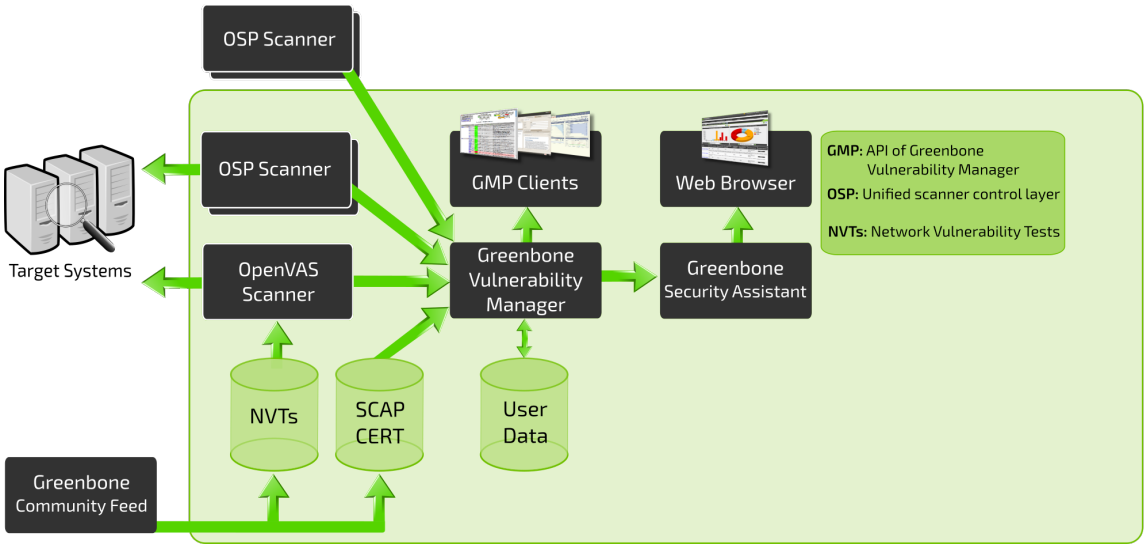
Architektura frameworku GVM 11

Greenbone Vulnerability Management je framework skládající se z několika modulů, jenž jsou vyvíjeny jako součást řady komerčního produktu Greenbone Security Manager [19].

Greenbone Vulnerability Manager

Greenbone Vulnerability Manager (GVMd) je centrální služba, která z jednotlivých skenování zranitelností dělá kompletní řešení správy zranitelností. Služba komunikuje se skenerem OpenVAS pomocí protokolu Open Scanner Protocol (OSP), který je bezestavový a založený na XML. GVMd služba jako taková poskytuje také protokol Greenbone Management Protocol (GMP), jímž jde služba ovládat. GVMd používá databázi PostgreSQL[19], kam si ukládá veškerou konfiguraci a data obsahující výsledky skenování. Dále služba obsahuje systém správy oprávnění, rolí a uživatelů. Je možné ji napojit například na LDAP server. GVMd může také skenování a ostatní rutiny plánovat. Důležitými částmi návrhu řešení

[13] https://www.gnu.org/licenses/licenses.en.html#GPL
[14] https://www.dfn-cert.de/
[15] https://www.bsi.bund.de/CERT-Bund_en)
[16] https://github.com/
[17] https://community.greenbone.net/
[18] https://www.debian.org/
[19] https://www.postgresql.org/



Obrázek 2.5: Architektura frameworku GVM 11. Převzato z https://community.greenbone.net/t/about-gvm-architecture/1231

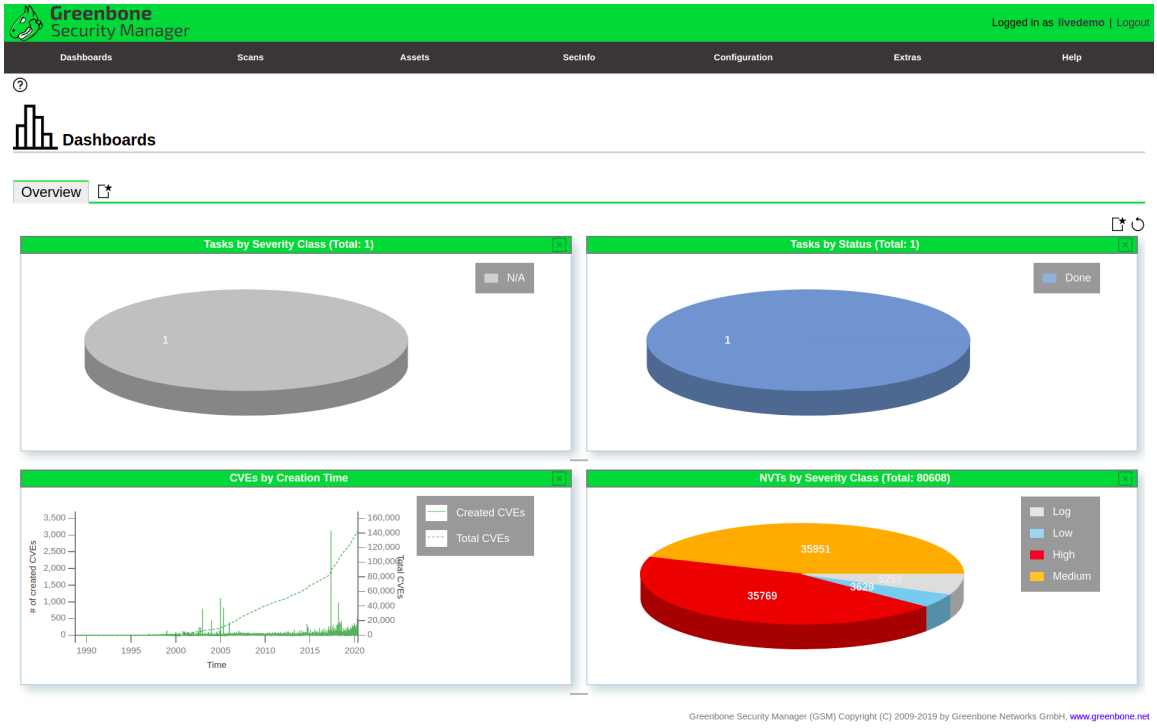
správy zranitelností jsou (ponecháno anglické názvosloví, aplikace není dostupná v českém jazyce) [19]:

- *Alert* - akce, jež může být vyvolána různými událostmi (například notifikace o nových zranitelnostech poslána přes e-mail)
- *Asset* - počítače a operační systémy objevené při skenování sítě
- *Filter* - filtr pro výběr určitých prvků ze skupiny stejných
- *Group* - skupina uživatelů
- *Host* - počítač připojený do sítě, který může být oskenován
- *Note* - poznámka, která může být připojena k NVT
- *Network Vulnerability Test (NVT)* - rutina, která hledá známé, či potencionální chyby v systému, jsou seskupeny podle typu do skupin (tzv. NVT Family)
- *Override* - pravidlo, které upravuje závažnost (severity) položek na jednom, či více reportech
- *Permission* - oprávnění, které umožňuje uživateli vykonat určitou akci
- *Port List* - seznam portů, který má každý target, určuje skenované porty systému
- *Quality of Detection (QoD)* - určuje spolehlivost detekce zranitelností nebo aplikací, číslo od 0 do 100 specifikující procento spolehlivosti
- *Report* - výsledek skenu obsahující souhrn toho, co odhalily použité NVT
- *Report Format* - formát reportu, do kterého je ho možné exportovat
- *Result* - výsledek vygenerovaný skenerem, jenž je součástí reportu

- *Role* - role, která specifikuje množinu oprávnění skupiny nebo uživatele
- *Scan* - sken je spuštěný task, výsledek skenu je report
- *Scanner* - OpenVAS nebo jiný skener podporující Open Scanner Protocol
- *Scan Configuration* - konfigurace skenu umožňující výběr NVT a dalších parametrů
- *Schedule* - čas, kdy se spustí task, umožňuje plánování skenování
- *Severity* - závažnost vycházející z CVSS
- *Solution Type* - druh možného vyřešení zranitelnosti
- *Target* - definuje množinu systémů (IP adresy nebo CIDR notace), které budou oskenovány
- *Task* - target s konfigurací skenu, spuštění tasku inicializuje sken

Greenbone Security Assistant

Greenbone Security Assistant (GSA) je webové rozhraní frameworku GVM. Zprostředkovává plnohodnotné uživatelské rozhraní pro ovládání GVMd. GSA se skládá z webové aplikace napsané v React[20] a HTTP serveru komunikujícím s GVMd pomocí GMP [19].



Obrázek 2.6: Greenbone Security Assistant

[20] https://reactjs.org/

OpenVAS

Skener OpenVAS je plnohodnotný skener, který provádí průběžně aktualizované testy zranitelností. Existují dvě databáze testů, první obsahující i komerční testy Greenbone Security Feed (GSF) a druhý zdarma dostupný Community Feed (GCF) [19].

GVM-Tools

Greenbone Vulnerability Management Tools jsou kolekce nástrojů, které umožňují ovládání GVMd. Tyhle nástroje zjednodušují ovládání pomocí protokolů OSP a GMP a zapouzdřují je do příkazů na vyšší úrovni, čímž umožňují například skriptování. Obsahuje interaktivní a neinteraktivní klienty [19].

Python-GVM

Greenbone Vulnerability Management Python API knihovna (python-gvm) je kolekce API, které pomáhají ovládat GVM. Jedná se o abstrakci přístupu k GMP a OSP. Knihovna umožňuje skriptování v programovacím jazyku Python 3 [19].

2.3 Průběžná integrace a Atlassian Bamboo

Software se v poslední době stává čím dál komplexnější. Už dávno neplatí, že programátor dostane zakázku, navrhne řešení, naimplementuje jej a nasadí. Na vývoji daného řešení spolupracují obrovské týmy. Softwarové řešení se často skládá z několika modulů, které je potřeba spojit do sebe. Pak přichází na řadu testy, zda naimplementované řešení odpovídá zadání. Celý tenhle koloběh vývoje obsahuje spoustu rutin, které se často opakují, a právě s těmi nám pomáhají systémy pro průběžnou integraci, průběžnou dodávku a průběžné nasazení. Průběžná integrace je tedy souhrn různých vývojářských nástrojů a metod k urychlení vývoje softwaru a spolupráce týmů. Jedná se o součást metodik extrémního programování. Slouží mimo jiné k urychlení nalezení nedostatků a chyb u softwarových projektů ve fázi vývoje. Pro spojení těchto metod a nástrojů se používají servery pro průběžnou integraci a nasazení (CI/CD servery) [17]. Pod slovem integrace je v tomto článku myšlena systémová integrace.

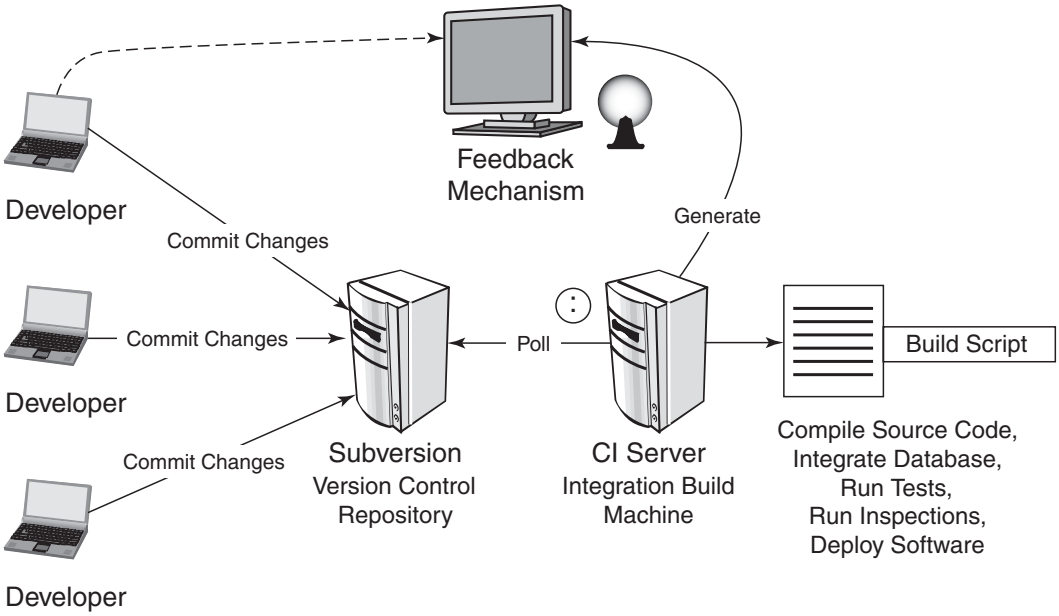
Build

Build neboli sestavení je dnes mnohem více než jen překlad zdrojového kódu do spustitelné binární podoby, nebo alternativní variace interpretovaných jazyků. Build se může skládat z překladu, testů, statické analýzy zdrojového kódu a například nasazení dané aplikace. Build v tomhle ohledu zastává spojení všech zdrojových kódů dohromady a kontrolu, zda vše spolu spolupracuje [12].

Prvky průběžné integrace

Vývojář

Vývojář provádí změny ve své lokální kopii repozitáře verzovacího systému. Jakmile je hotov s úpravami, přeloží si projekt na svém stroji a spustí případné testy. Pokud je s výsledkem spokojen, tak provedené změny pošle do centrálního repozitáře a tím zinicializuje průběžnou integraci [12].



Obrázek 2.7: Prvky průběžné integrace. Převzato z [12].

Repozitář verzovacího systému

Bez verzovacích systémů si lze dnešní vývoj aplikací jen těžko představit. Umožňují nejen správu verzí, ale i možnost přehledné spolupráce. Repozitář obsahuje databázi všech commitů (potvrzení editačních změn a jejich uložení do repozitáře verzovacího systému) a je možné se k jakémukoliv vrátit. Využití verzovacího systému by měla být pro vývoj softwaru samozřejmost i bez průběžné integrace. K uskutečnění průběžné integrace je verzovací systém nezbytný. Integrační server zpravidla kontroluje hlavní větev repozitáře, která obsahuje hotové funkce. Existují různé typy verzovacích systémů jako například Git[21] nebo Subversion[22].

Integrační server

CI server spustí integrační build vždy, když dojde ke změnám ve sledované větvi repozitáře verzovacího systému. CI server je srdcem průběžné integrace, je v něm implementována logika průběžné integrace a uložena všechna nastavení. Stará se o samotné spouštění build skriptu a dále umožňuje vyhodnocení výsledků buildu a specifikaci dalších kroků. Spouštění buildovacích skriptů může být prováděno na různých strojích, které jsou tak nastaveny a CI server k ním má vzdálený přístup. To umožňuje využití odlišných prostředí, například různých operačních systémů [12].

Buildovací skript

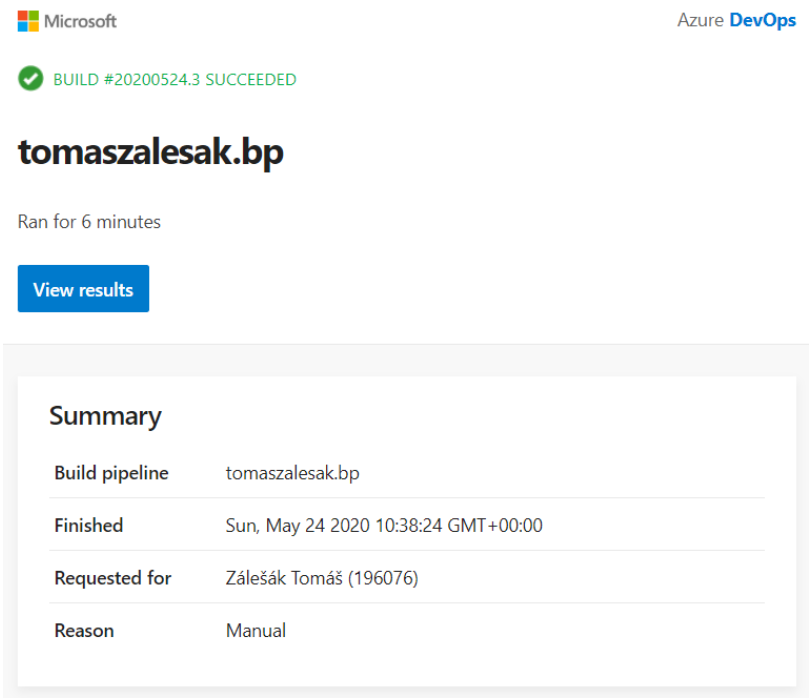
Build skript je jeden či několik skriptů, jenž jsou používány k překladu, spuštění testů, statické či dynamické analýze, nebo k nasazení programu. Build skripty se využívají i mimo

[21] https://git-scm.com/
[22] https://subversion.apache.org/

CI, kdy automatizují sestavení programů na lokální stanici vývojáře. Existují nástroje, které sestavení programu automatizují jako make[23], Ant[24], MSBuild[25]. Automatizaci překladu obsahují zpravidla také IDE (integrované vývojové prostředí).

Mechanizmus zpětné vazby

Jedna z klíčových vlastností průběžné integrace je poskytnutí zpětné vazby z integračního buildu. Pokud by integrační build selhal, tak je důležité zaznamenat chyby a předat je zpět odpovědné osobě. Běžně se využívají e-maily, SMS, či RSS.



Obrázek 2.8: Ukázka e-mailové zprávy informující o úspěšném buildu

Typický scénář integračního procesu

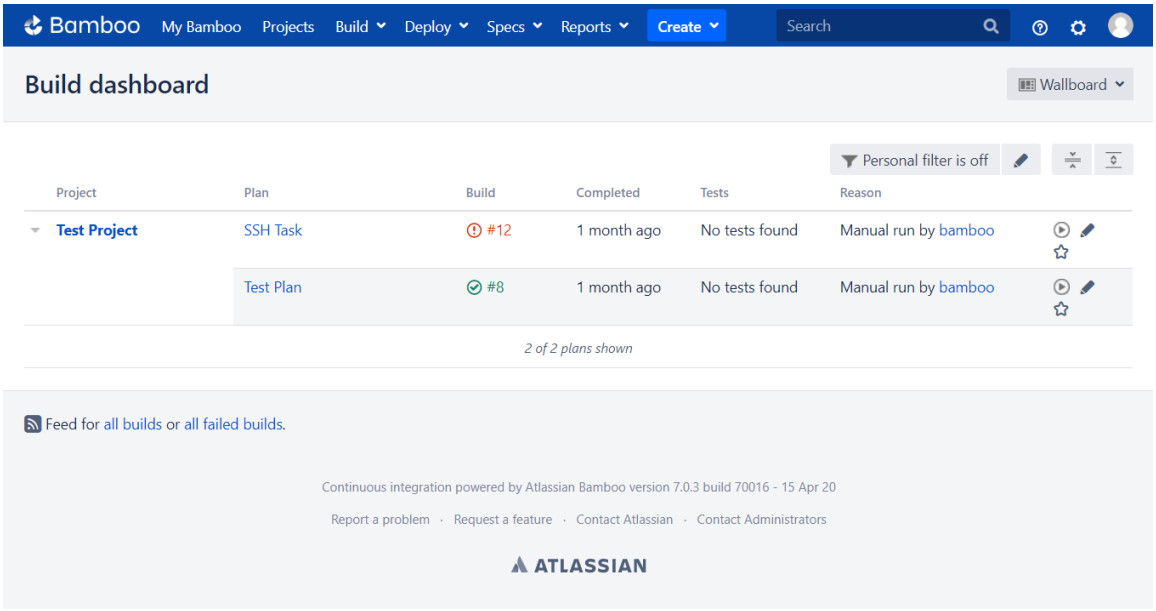
Scénář dnešního moderního vývoje softwaru typicky obnáší [12]:

1. Programátor provede zamýšlené změny ve zdrojové kódu a uloží změny do repozitáře verzovacího systému. CI server provádí pravidelnou kontrolu změn v repozitáři, například každých několik minut.
2. Brzy po provedení změn v repozitáři, CI server změny detekuje a stáhne si aktuální kopii repozitáře.
3. CI server spustí nadefinovaný build skript a například pošle odpovědnému programátorovi e-mail s informacemi o průběhu integrace či testů.
4. CI server pokračuje v čekání na změny v repozitáři.

[23] https://www.gnu.org/software/make/
[24] https://ant.apache.org/
[25] https://docs.microsoft.com/en-us/visualstudio/msbuild/

Atlassian Bamboo

Bamboo je software od firmy Atlassian, který umožňuje průběžnou integraci, nasazení a spojení s automatizovanými buildy, testy a vydáváním nových verzí v rámci jednoho workflow. Je napsán v programovacím jazyce Java a je dostupný pro Microsoft Windows, Linux, Solaris i MacOS X. Bamboo je komerční software, ale pro open-source projekty je zdarma. Firma Atlassian poskytuje i verzovací nástroj BitBUcket, nástroj ke správě projektů pro agilní týmy Jira[26] a wiki pro sdílení informací Confluence[27]. K ovládání Bamboo slouží webové rozhraní. Na obrázku 2.9 je domovská stránka s projekty a plány.



Obrázek 2.9: Webový klient Atlassian Bamboo.

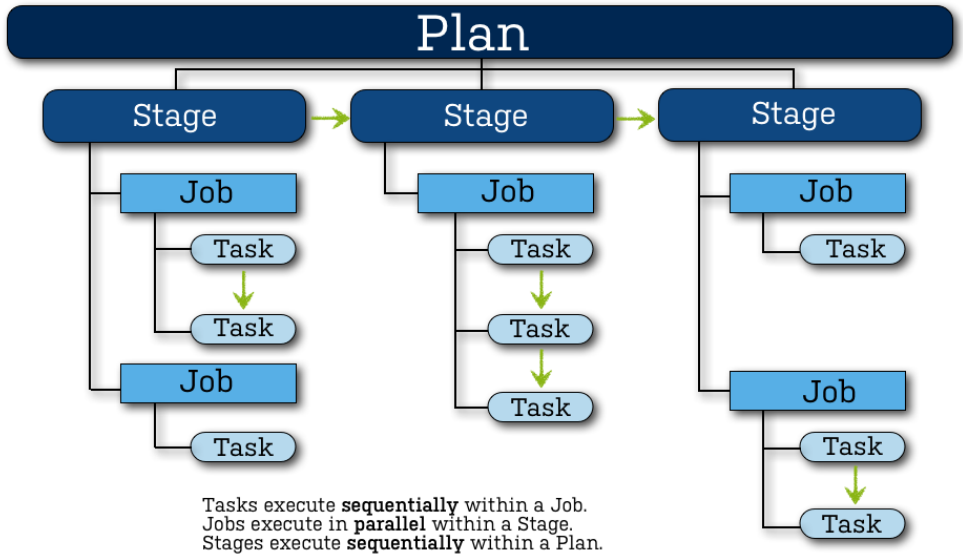
Na svých webových stránkách uvádí tyhle důvody, proč zvolit Bamboo [2]:

- hluboká integrace s vývojářskými nástroji (například Jira Software, Bitbucket Server[28])
- vestavěná podpora pro doručení softwaru zákazníkům
- snadné rozšíření o nové agenty (stroje, na kterých běží buildy) například z Amazon Web Services[29]
- automatické spojování větví verzovacích systémů
- vestavěné automatické detekování, build, testování a spojení větví do jedné pro průběžné nasazení

Bamboo používá koncept projektů (Project), plánů (Plan), fází (Stage), prací (Job) a úkolů (Task). Vše je organizováno následovně:

[26] https://www.atlassian.com/software/jira
[27] https://www.atlassian.com/software/confluence
[28] https://www.atlassian.com/software/bitbucket
[29] https://aws.amazon.com/

- *Projekt* - Může obsahovat několik plánů. Poskytuje informace o plánech v projektu a odkazy do jiných aplikací. Umožňuje nastavení oprávnění pro všechny plány, které obsahuje.
- *Plán* - Ve výchozím stavu má jednu fázi, ale může jich být přidáno víc, aby seskupil práce do zvláštních fází. Fáze jsou v rámci plánu prováděny sekvenčně. Plán také specifikuje výchozí repozitář, určuje způsob spuštění buildu, kdo má k plánu přístup, notifikace výsledků buildu a proměnné společné pro celý plán.
- *Fáze* - Seskupuje práce, které se provádí paralelně v rámci dané fáze, pokud je dostupných více zprostředkovatelů (Agent). Zprostředkovateli jsou míněni počítače připojené do Bamboo, na nichž se spouští buildy. Pokud neskončí úspěšně všechny práce, nepokračuje se do další fáze.
- *Práce* - Vykonává obsahující úkoly sekvenčně za sebou na stejném zprostředkovateli. Určuje pořadí úkolů, v jakém jsou vykonány.
- *Úkol* - Malá jednotka obsahující například stáhnutí zdrojových souborů, spuštění skriptu nebo testu. Je vykonán sekvenčně v rámci práce.



Obrázek 2.10: Koncepce Bamboo plánu. Převzato z https://confluence.atlassian.com/bamboo/files/289277285/319619101/1/1359074967455/BambooPlanAnatomy.png

Bamboo umožňuje manuální spuštění plánu pomocí tlačítka `Run` nebo tzv. `triggerů`. Trigger je funkce, která pomáhá s automatickým spouštěním a plánováním buildů. Ve webovém rozhraní je možné nastavit konkrétní čas a dny v týdnu, kdy se build spustí. Build může být také automaticky spuštěn, když doběhne jiný plán nebo se objeví změny ve sledovaném repozitáři. Plán má také své proměnné, které ve skriptech využívá. Proměnné lze využít jako parametry plánu a upravit je pro konkrétní běh buildu.

2.4 Dostupná řešení implementace OpenVAS s nástroji průběžné integrace

Tato podkapitola shrnuje dostupná řešení implementace OpenVAS s nástroji pro průběžnou integraci. Uvádí také volně dostupné konkurenční řešení pro automatizaci dynamické analýzy bezpečnosti aplikací s využitím průběžné integrace.

Při automatizaci dynamické analýzy bezpečnosti aplikací je nutné propojit nástroj pro průběžnou integraci se skenerem OpenVAS. Nástroje pro průběžnou integraci disponují různými pluginy, nicméně žádný plugin neumožnuje integraci skeneru OpenVAS či frameworku Greenbone Vulnerability Management (GVM). Integraci skeneru OpenVAS je možné provést pomocí CLI (Command Line Interface) příkazů využívající rozhraní pro programování aplikací (API) daného modulu. API je poskytováno jak skenerem OpenVAS, tak službou pro správu zranitelností GVM. Je tedy k implementaci možné použít jak samotný skener, který pouze poskytne report s výsledky, tak kompletní řešení správy zranitelností GVM. Spuštění skenu pomocí skriptu je možné dosáhnout následujícími způsoby:

- *gvm-cli* - Jedná se o CLI nástroj z modulu `gvm-tools`, jenž umožňuje se připojit přes SSH, TLS nebo Unix Socket ke skeneru a zadat příkazy v Open Scanner Protocolu (OSP), nebo k GVM a zadat příkazy v Greenbone Management Protocolu (GMP).
- *gvm-script* - Je to také CLI nástroj z modulu `gvm-tools`. Umožňuje napsání vlastních OSP/GMP skriptů v jazyce Python využívající knihovnu `python-gvm`, kterou ve skriptu není nutné importovat.
- *gvm-pyshell* - Poslední nástroj dostupný v modulu `gvm-tools`. Je stejný jako *gvm-script*, nicméně je dostupný ve formě interaktivního shellu Pythonu.
- *python-gvm* - Knihovna napsaná v jazyce Python umožňující ovládání skeneru OpenVAS a GVM modulu. Při jejím importu je možné psát vlastní skripty v jazyce Python.

Existují podobné řešení pro automatizaci dynamické analýzy bezpečnosti aplikací, které využívají jiné skenery. Mezi volně dostupné řešení například patří:

- *Gitlab CI/CD se skenerem OWASP ZAP* - Nástroj pro průběžnou integraci a nasazení Gitlab CI/CD obsahuje možnost nasazení aplikace do Docker kontejneru a oskenovat webovou aplikaci pomocí nástroje OWASP ZAP, což je jeden z nejpoužívanějších skenerů webových aplikací [4].
- *Jenkins se skenerem OWASP ZAP* - Pro CI/CD server Jenkins existuje taky řešení, jak využít skener OWASP ZAP. Stačí si nainstalovat do Jenkins Official OWASP ZAP plugin. Plugin umožní spojení průběžného nasazení se skenováním webové aplikace. Návod na použití pluginu je zdokumentován na webové stránce [13].
- *Jenkins se skenerem Probely* - Jenkins je také možné pomocí pluginu propojit s řešením od firmy Probely, která poskytuje stejnojmenný skener webových aplikací [1].
- *Circle CI se skenerem Probely* - Skener webových aplikací Proberly se dobře hodí k použití do průběžného nasazení. To dokazuje i jeho další rozšíření, jenž umožňuje propojení s Circle CI [8].

Podobné řešení obsahují pouze nástroje pro ulehčení implementace a nejsou hotovým řešením pro nasazení, používají jiné skenery a tím pádem mají i jiné případy užití.

Kapitola 3

Zhodnocení současného stavu a plán práce

Následující kapitola obsahuje informace o dostupných řešeních, zhodnocení současného stavu a plán práce pro automatizaci dynamické analýzy bezpečnosti aplikací pomocí nástroje OpenVAS.

3.1 Zhodnocení dosavadního stavu

Pro automatizaci dynamické analýzy bezpečnosti neexistuje jedno univerzální řešení. Různé vývojářské týmy používají různé platformy, různé nástroje pro průběžnou integraci a nasazení. Pro skenování aplikace je třeba nejprve skenovanou aplikaci nasadit, ujistit se, zda vše běží tak, jak má, a až pak lze daný systém oskenovat. Celý proces průběžné integrace a nasazení je třeba mít zautomatizovaný a potom lze přidat dynamickou analýzu bezpečnosti. Pokud by například neexistoval způsob, jak automatizovaně nainstalovat a případně nastavit aplikaci, tak by ani nešlo zautomatizovat celý proces dynamické analýzy bezpečnosti dané aplikace. V současnosti není dostupné řešení či návod, jak implementovat skener OpenVAS s nástroji pro průběžnou integraci tak, aby bylo možné sken konfigurovat a použít pro automatizovanou analýzu bezpečnosti.

3.2 Specifikace požadavků

Cílem je integrovat OpenVAS s Atlassian Bamboo tak, že se skenování zranitelností bude spouštět pravidelně. Je potřeba umožnit konfiguraci skenu, aby bylo možné potlačit falešné detekce. Řešení má být použitelné pro regresní testování bezpečnosti. Úkolem je také zvážit použití Greenbone Security Manager, nástroje pro správu zranitelností, který zahrnuje OpenVAS.

3.3 Plán práce

Je potřeba navrhnout infrastrukturu, ve které bude dostupný OpenVAS, Atlassian Bamboo a skenovaný systém. To zahrnuje výběr technologií jako jsou virtualizační nástroje a operační systémy. Bude vybráno, zda použít samostatně skener OpenVAS, nebo kompletní řešení pro správu zranitelností Greenbone Vulnerability Manager. V dané infrastruktuře je třeba navrhnout postup, jak bude automatizace skenování probíhat. Také je třeba zvolit

způsob konfigurace skenování pro potlačení nechtěných detekcí a navrhnout, jak rozlišit již známé a nově objevené zranitelnosti. Posledním krokem bude implementace navrženého řešení a otestování na vybrané aplikaci s otevřeným kódem.

Kapitola 4

Popis vlastní práce

Následující kapitola obsahuje výběr technologií a návrh architektury řešení, popis nasazení, konfiguraci skenování zranitelností s OpenVAS, automatizaci spouštění skenování s Bamboo a ověření funkčnosti a interpretaci výsledků.

4.1 Výběr technologií a návrh architektury řešení

Nejprve bylo potřeba vybrat, zda použít samostatný OpenVAS nebo framework Greenbone Vulnerability Manager (GVM), který OpenVAS obsahuje. GVM k OpenVAS přidává centrální službu pro správu zranitelností (GVMd), webové rozhraní Greenbone Security Assistant (GSA), kolekci nástrojů pro vzdálené ovládání GVM-Tools a Python knihovnu Python-GVM. Jelikož je potřeba konfigurovat skenování, prohlížet a porovnávat výsledky, plánovat spuštění a vzdáleně ovládat nástroj OpenVAS, zvolil jsem použití frameworku GVM. Framework GVM je dostupný zdarma předinstalovaný v rámci virtuálního stroje Greenbone Community Edition (GCE) nebo jako komerční produkt Greenbone Security Manager (GSM). GCE je ovšem omezeno tím, že neumožnuje plánování skenování a notifikace o výsledcích, čímž své použití v tomhle případě znemožňuje. Kód frameworku GVM je otevřený a jednotlivé repozitáře jeho modulů jsou dostupné na GitHub. Vybral jsem instalaci frameworku GVM z veřejně dostupných zdrojových souborů. Pro instalaci frameworku GVM byl zvolen operační systém Ubuntu 18.04[1]. Firma Greenbone dodává GSM a GCE s operačním systémem Greenbone OS, jenž je založen na linuxové distribuci Debian.

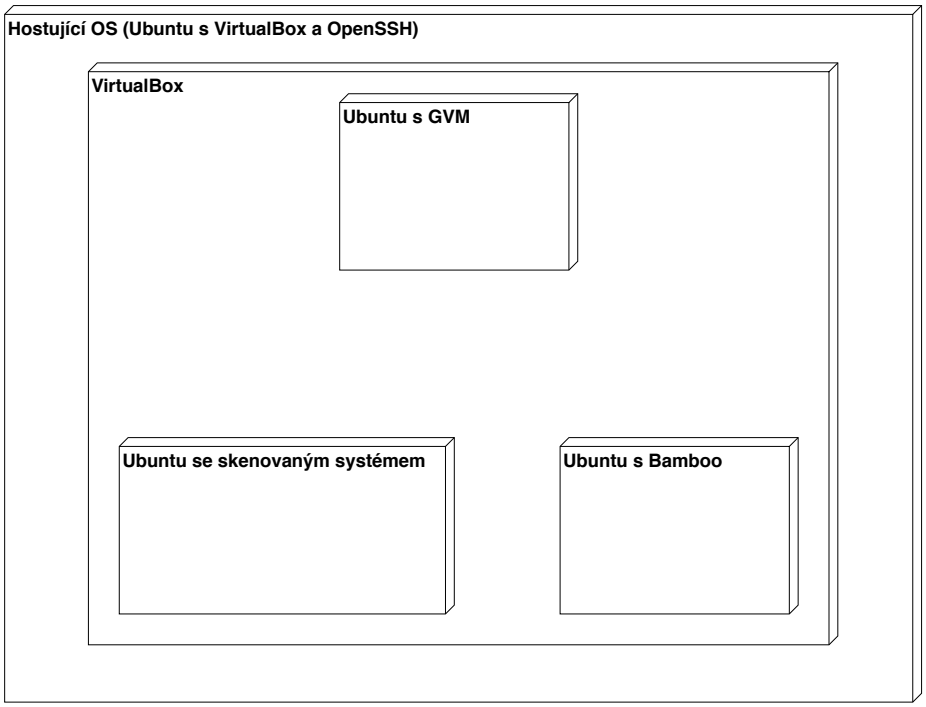
Druhou částí je využití nástroje pro průběžnou integraci Atlassian Bamboo. Ten je dostupný pro všechny hlavní platformy. Pro jeho nasazení byl zvolen operační systém Ubuntu.

Třetí částí je skenovaný systém. Platforma skenovaného systému závísí na tom, co do něj instalujeme. Pro ukázku byla vybrána aplikace s otevřeným kódem Rocket.Chat[2]. Rocket.Chat je webová služba umožňující týmovou komunikaci. Je vyvinuta v programovacím jazyce JavaScript s využitím frameworku Meteor. Pro její nasazení byl zvolen také operační systém Ubuntu.

Skenovaný systém je potřeba automatizovaně vytvářet nový či vracet do stavu před instalací aplikace, což efektivně umožňují virtualizační nástroje. Pro celou infrastrukturu tří systémů byl zvolen virtualizační nástroj VirtualBox. VirtualBox je zdarma, má také otevřený kód a především umožňuje vzdálené ovládání virtuálních strojů CLI příkazy. VirtualBox je dostupný na většině platforem. VirtualBox byl nainstalován do hostujícího operač-

[1] https://ubuntu.com/
[2] https://rocket.chat/

ního systému Ubuntu. Aby bylo možné ovládat vzdáleně VirtualBox, byl také nainstalován do hostujícího operačního systému SSH server OpenSSH[3]. Na obrázku 4.1 je znázorněna architektura řešení.



Obrázek 4.1: Návrh architektury řešení.

4.2 Nasazení vybraného řešení

Navržené řešení se skládá ze čtyř hlavních částí. První částí je hostující operační systém s VirtualBoxem a OpenSSH serverem. Instalaci VirtualBoxu a OpenSSH serveru lze v Ubuntu dosáhnout CLI příkazem `sudo apt install openssh-server virtualbox`. Aby mohly virtuální stroje komunikovat mezi sebou a zároveň mít přístup na internet, bylo třeba přidat ve VirtualBoxu nový *Host-only Adapter* a pak při vytváření virtuálních strojů daný adaptér přiřadit jako druhý k *Network Address Translation* adaptéru. To vytvořilo novou privátní síť mezi všemi čtyřmi systémy. Pokud by byl aktivní firewall, tak je nutné povolit komunikaci mezi hostujícím systémem a virtuálními stroji. Na všechny tři virtuální stroje byl nainstalován operační systém Ubuntu. Každému virtuálnímu stroji bylo přiřazeno úložiště o velikosti 20GB a 4GB operační paměti. Parametry skenovaného stroje bylo potřeba zvolit podle toho, co vyžaduje nasazovaná aplikace. U všech virtuálních strojů byla nastavena statická IP adresa na nové virtuální síťové rozhraní podle tabulky 4.1. Zde je ukázka konfiguračního souboru `/etc/netplan/50-cloud-init.yaml`:

```
network:
    ethernets:
        enp0s3:
```

[3] https://www.openssh.com/

```
            dhcp4: true
        enp0s8:
            dhcp4: no
            addresses:
              - 192.168.56.3/24
    version: 2
```

Tabulka 4.1: IP adresy

Systém	IP adresa
Hostující OS	192.168.56.1
GVM	192.168.56.2
Bamboo	192.168.56.3
Skenovaný systém	192.168.56.4

GVM

Druhou částí je virtuální stroj s frameworkem GVM. Zdrojové soubory jednotlivých modulů a pomocných knihoven a nástrojů jsou dostupné v repozitářích firmy Greenbone na GitHub.

Tabulka 4.2: Repozitáře modulů GVM

Název repozitáře	URL repozitáře
gvm-libs	https://github.com/greenbone/gvm-libs
OpenVAS	https://github.com/greenbone/openvas
GVMd	https://github.com/greenbone/gvmd
openvas-smb	https://github.com/greenbone/openvas-smb
GSA	https://github.com/greenbone/gsa
ospd-openvas	https://github.com/greenbone/ospd-openvas
ospd	https://github.com/greenbone/ospd

Byly staženy nejnovější vydání uvedených modulů a nainstalovány podle návodů uvedených v souborech README nebo INSTALL, které se nachází v kořenové složce repozitářů. Při instalaci modulu GVMd byla vytvořena databáze PostgreSQL, která ukládá data a nastavení centrální služby GVMd. Webové rozhraní GSA běží na portu 443. Pro automatický start služeb po spuštění virtuálního stroje byly vytvořeny spouštěcí skripty pomocí démonu systemd[4].

Tabulka 4.3: Spouštěcí skripty

Popis skriptu	Umístění
Spuštění služby GVMd	`/etc/systemd/system/gvmd.service`
Spuštění webového rozhraní GSA	`/etc/systemd/system/gsad.service`
Spuštění OSPd	`/etc/systemd/system/ospd-openvas.service`

Po vytvoření byly skripty načteny. Služby byly povoleny a spuštěny pomocí příkazu `systemctl`.

[4] https://www.freedesktop.org/wiki/Software/systemd/

Pro vzdálené ovládání GVM byla doinstalována Python knihovna `python-gvm`. OpenVAS využívá databázi Greenbone Community Feed (GCF), kterou je potřeba pravidelně aktualizovat. Pro tento účel byl vybrán démon `cron`, jenž umožňuje plánování spuštění příkazů. Je nutné nastavit `cron` uživatele, pod kterým GVM běží. Pomocí příkazu `crontab -e` a přidání následujích příkazů byla zajištěna každodenní aktualizace GCF. Uvedený skript spustí příkaz o půlnoci, v 1:00 a ve 2:00.

```
SHELL=/bin/bash
PATH=/opt/gvm/bin:/opt/gvm/sbin
0 0 * * * /opt/gvm/sbin/greenbone-scapdata-sync
0 1 * * * /opt/gvm/bin/greenbone-nvt-sync
0 2 * * * /opt/gvm/sbin/greenbone-certdata-sync
```

Služba GVMd využívá pro posílání notifikací program `sendmail`. Program byl nainstalován a nakonfigurován se službou Gmail od firmy Google.

Bamboo

Třetí částí architektury je virtuální stroj s Bamboo. Bamboo bylo nainstalováno podle oficiálního návodu dostupného na webových stránkách. Pro automatické spuštění byl vytvořen spouštěcí skript pomocí démonu `systemd`. Po dokončení instalace běží webové rozhraní Bamboo na portu 8085.

Skenovaný systém

Skenovaný systém byl aktualizován příkazem `sudo apt update && sudo apt upgrade` a poté byl vytvořen snapshot (obraz systému v konkrétním čase) virtuálního stroje. Snapshot umožní jednoduchý návrat virtuálního stroje do stavu před nasazením skenované aplikace.

4.3 Konfigurace skenování zranitelností

Pro konfiguraci skenování zranitelností bylo využito webové rozhraní GSA. Ke konfiguraci bylo třeba vytvořit nový Port List, Credential, Target a Task. Port List byl vytvořen následujícím postupem:

1. Vytvoření nového Port List na `192.168.56.2/portlists` přes tlačítko New Port List v levém horním rohu.
2. Pojmenování Port Listu a specifikace portů. Specifikace portů `T:22,U:1-3,5` například znamená, že na skenovaném systému budou oskenovány TCP port 22 a UDP porty 1, 2, 3, 5. Pokud bude využito autentizované skenování, je důležité přidat i port, přes který se OpenVAS do systému přihlašuje. V případě využití protokolu SSH je nutné přidat TCP port 22.
3. Uložení Port List tlačítkem Save.

Přihlašovací údaje do skenovaného systému byly přidány následujícím postupem:

1. Vytvoření nového Credential na `192.168.56.2/credentials` přes tlačítko New Credential v levém horním rohu.

2. Pojmenování Credential, výběr typu (Username + Password) a zadání uživatelského jména a hesla skenovaného systému.

3. Uložení Credential tlačítkem Save.

Pro specifikaci IP adresy skenovaného systému byl přidán nový Target:

1. Vytvoření nového Target na 192.168.56.2/targets přes tlačítko New Target v levém horním rohu.

2. Pojmenování nového Target, specifikace IP adresy skenovaného systému v poli Hosts a Port Listu, který byl vytvořen v předcházejícím postupu. Pro autentizované skenování je třeba vybrat protokol a port přes, který se bude OpenVAS přihlašovat do skenovaného systému.

3. Uložení Target tlačítkem Save.

Ke spuštění skenu s určitou konfigurací slouží Task. Task byl vytvořen následujícím způsobem:

1. Vytvoření nového Task na 192.168.56.2/tasks přes tlačítko New Task v levém horním rohu.

2. Pojmenování nového Task, výběr Target, zaškrtnutí Alterable Task, aby bylo možné Task upravovat a zvolení Full and very deep ultimate u Scan Config. Ostatní položky byly ponechány ve výchozím stavu.

3. Uložení Task tlačítkem Save.

Aby bylo možné OpenVAS vzdáleně spouštět, byl vytvořen skript v jazyce Python. Skript využívá knihovnu python-gvm a spouší se třemi argumenty. Prvním argumetem je jméno uživatele služby GVMd, druhým je jeho heslo a třetí je identifikační číslo Task, který spustí.

```
#!/usr/bin/env python3

import sys
from gvm.connections import UnixSocketConnection
from gvm.errors import GvmError
from gvm.protocols.latest import Gmp
from gvm.transforms import EtreeCheckCommandTransform
from gvm.xml import pretty_print

def main():

    if len(sys.argv) != 4:
        print('Usage: run.py <user_name> <user_password> <task_id>')
        exit()

    path = '/opt/gvm/var/run/gvmd.sock'
    connection = UnixSocketConnection(path=path)
```

```
transform = EtreeCheckCommandTransform()
gmp = Gmp(connection=connection, transform=transform)

username = sys.argv[1]
password = sys.argv[2]
task_id = sys.argv[3]

try:

    with gmp:
        gmp.authenticate(username, password)
        gmp.start_task(task_id)

except GvmError as e:
    print('An error occurred', e, file=sys.stderr)

if __name__ == "__main__":
    main()
```

4.4 Automatizace skenování

Pro automatizaci nasazení skenovaného systému a následné spuštění skenování zranitelností je využit CI/CD server Atlassian Bamboo. Celá automatizace byla rozdělena do tří hlavních částí. První částí je vrácení skenovaného systému do stavu čisté instalace virtuálního stroje. Druhou částí je nasazení aplikace do skenovaného systému. Třetí částí je spuštění skeneru OpenVAS.

Ve webovém rozhraní Bamboo byl vytvořen nový projekt a v něm nový plán. Do plánu byly přidány následující fáze (Stage), práce (Job) a úkoly (Task):

- Stage *Create clean VM* - vrácení skenovaného systému do stavu čisté instalace
  - Job *Create clean VM*
    - SSH Task *Restore Clean Ubuntu Snapshot*
    - Script *Wait for SSH Server To Start*
- Stage *Deploy application* - nasazení aplikace do skenovaného systému
  - Job *Deploy application*
    - SSH Task *Install application*
    - Script *Wait until app is ready*
- Stage *Run OpenVAS* - spuštění skenování
  - Job *Run OpenVAS*
    - SSH Task *Run OpenVAS*

Fáze a práce rozdělují plán do logických částí. Úkoly obsahují skripty, které budou při spuštění plánu provedeny.

SSH Task je v Bamboo druh úkolu, ve kterém se specifikuje IP adresa, přihlašovací údaje a skript. Při spuštění se Bamboo přihlásí na počítač se zadanou IP adresou pomocí poskytnutých přihlašovacích údajů a spustí skript. *Script* je v Bamboo druh úkolu, ve kterém dojde ke spuštění zadaného skriptu.

Stage Create clean VM

SSH Task *Restore Clean Ubuntu Snapshot* se přihlásí do hostujícího systému s běžícím VirtualBox (IP adresa 192.168.56.1) a spustí skript, který zastaví běžící skenovaný systém (pojmenován ScannedVM v aplikaci VirtualBox), vrátí systém na snapshot *snapshotname*.

```
vboxmanage controlvm ScannedVM poweroff
vboxmanage snapshot ScannedVM restore snapshotname
vboxmanage startvm ScannedVM --type headless
```

Script *Wait for SSH Server To Start* vyčká, až je možné se do skenovaného systému přihlásit. Jedná se o bash skript:

```
CODE=‘nmap 192.168.56.4 -PN -p ssh | grep -o open‘
echo $CODE
while [ "$CODE" != "open" ]
do
   sleep 2
   echo "Trying again..."
   CODE=‘nmap 192.168.56.4 -PN -p ssh | grep -o open‘
   echo $CODE
done
sleep 2m
```

Stage Deploy application

SSH Task *Install application* se přihlásí do skenovaného systému a spustí skript pro nasazení skenované aplikace. IP adresa skenovaného systému je 192.168.56.4. Dále je nutné se ujistit, zda je vše nainstalováno a spuštěno. K tomu slouží Script *Wait until app is ready*. U vybrané testovací aplikace jde například o to, aby byla spuštěna webová služba na portu 3000. Následující skript vyčká, až je port otevřen.

```
CODE=‘nmap 192.168.56.4 -PN -p 3000 | grep -o open‘
echo $CODE
while [ "$CODE" != "open" ]
do
   sleep 2
   echo "Trying again..."
   CODE=‘nmap 192.168.56.4 -PN -p 3000 | grep -o open‘
   echo $CODE
done
sleep 5m
```

Stage Run OpenVAS

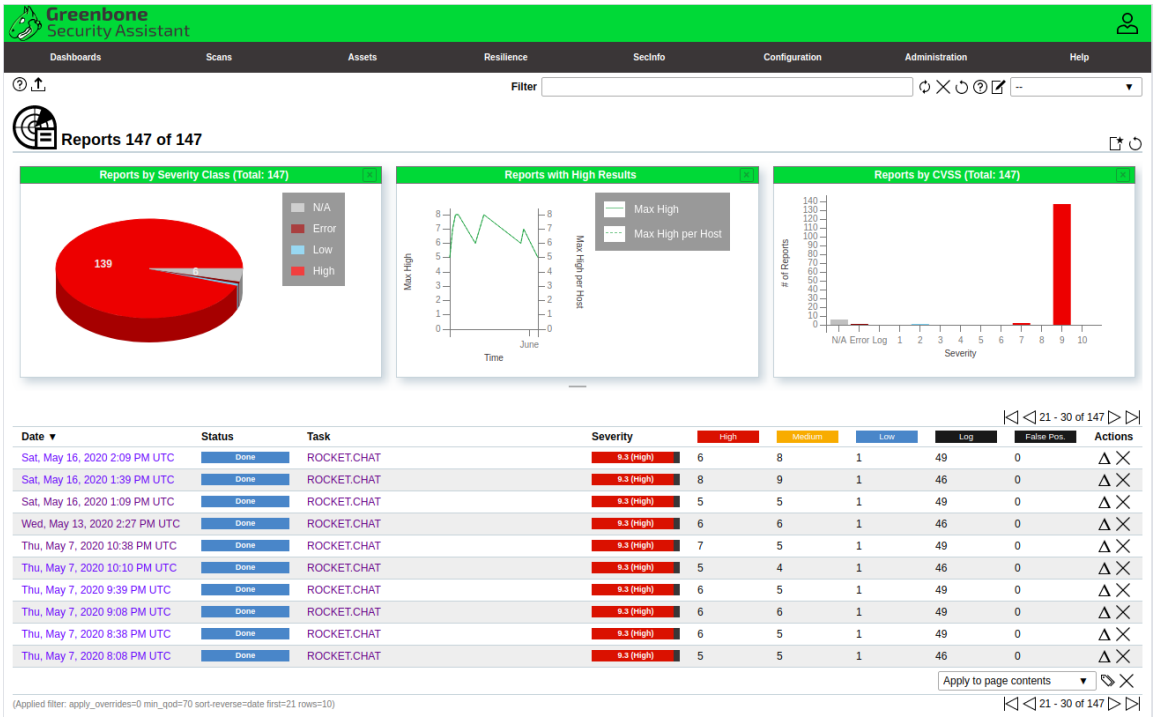
Poslední fáze obsahuje jeden SSH Task, který se přihlásí do virtuálního systému s GVM (IP adresa 192.168.56.2) a spustí skript `/home/tomas/run.py`. Tím se spustí skenování zranitelností.

Spuštění plánu

Bamboo umožňuje jak manuální, tak plánované spuštění plánu. Pro manuální spuštění obsahuje webové rozhraní tlačítko Run. Plánované spuštění je řešeno pomocí tzv. triggerů (spouštěčů). Plány můžou být spuštěny při změnách v repozitářích, při skončení jiných plánů nebo v určený čas a den v týdnu.

4.5 Ověření funkčnosti a interpretace výsledků

Pro ověření funkčnosti byl systém otestován s aplikací Rocket.Chat. Byly vybrány následující testy:



Obrázek 4.2: Automatické spouštění skenování aplikace Rocket.Chat po časových intervalech.

Test 1

Bude provedeno manuální spuštění Bamboo plánu. Až plán doběhne, bude nastaven Bamboo trigger tak, aby se plán spouštěl pravidelně v časových intervalech dostatečně vzdálených, aby stačil plán doběhnout. Bude ověřeno, zda skeny proběhly ve webovém rozhraní GVM. Bude také zkontrolováno, zda byly detekovány nějaké zranitelnosti v systému Ubuntu 18.04 s nasazenou aplikací Rocket.Chat.

Vulnerability	Severity ▾	QoD	Host IP	Host Name	Location
LiteServe URL Decoding DoS	9.3 (High)	99 %	192.168.56.4		3000/tcp
Mongoose Webserver Content-Length Denial of Service Vulnerability	7.8 (High)	99 %	192.168.56.4		3000/tcp
HTTP User-Agent overflow	7.5 (High)	99 %	192.168.56.4		3000/tcp
HTTP version number overflow	7.5 (High)	99 %	192.168.56.4		3000/tcp
HTTP Cookie overflow	7.5 (High)	99 %	192.168.56.4		3000/tcp
HTTP 1.0 header overflow	7.5 (High)	99 %	192.168.56.4		3000/tcp
Jigsaw webserver MS/DOS device DoS	5.0 (Medium)	99 %	192.168.56.4		3000/tcp
HTTP header overflow	5.0 (Medium)	99 %	192.168.56.4		3000/tcp
HttpBlitz Server HTTP Request Remote Denial of Service Vulnerability	5.0 (Medium)	70 %	192.168.56.4		3000/tcp
Xitami '/AUX' Request Remote Denial Of Service Vulnerability	5.0 (Medium)	99 %	192.168.56.4		3000/tcp
Crash SMC AP	5.0 (Medium)	99 %	192.168.56.4		3000/tcp
HTTP 1.1 header overflow	5.0 (Medium)	99 %	192.168.56.4		3000/tcp
TCP timestamps	2.6 (Low)	80 %	192.168.56.4		general/tcp

Obrázek 4.3: Výsledky skenování aplikace Rocket.Chat

Vyhodnocení testu 1

Jeden build trval asi 14 minut. Byl nastaven interval spouštění 30 minut. Systém byl nechán běžet a skenování proběhlo celkem 147 krát a vždy úspěšně, viz obrázek 4.2. Ve vybraném výsledku bylo detekováno 13 zranitelností. Výsledky jsou zachyceny na obrázku 4.3. Byly úspěšně splněny podmínky testu.

Test 2

Po proběhnutí skenu bude ověřeno, zda lze výsledky zobrazit a případně třídit a označovat jako vyřešené.

Vyhodnocení testu 2

Po proběhnutí skenu bylo úspěšně ověřeno, že lze výsledky zobrazit, třídit a označovat jako vyřešené. Výsledky byly zobrazeny ve webovém rozhraní GVM, třízení jde uskutečnit pomocí filtru a označení výsledků jako vyřešené pomocí overrides.

Test 3

Bude vyzkoušena konfigurace skenování ve webovém rozhraní a bude ověřeno, zda se změna konfigurace projevuje ve výsledcích skenování.

Vyhodnocení testu 3

Byla vyzkoušena konfigurace skenování a změny se projevily ve výsledcích. Při testu byla zvolena konfigurace, aby se oskenoval jen port 3000, na kterém běží webové rozhraní aplikace Rocket.Chat. V reportu byly pouze výsledky s portem 3000. Test proběhl úspěšně.

Test 4

Pro ověření automatizace řešení bude nastaven trigger, aby se skenování pravidelně spouštělo. Budou restartovány virtuální stroje s řešením a poté bude zkontrolováno, zda bylo vše uloženo a zda se vše zase automaticky spustí.

Vyhodnocení testu 4

Virtuální stroje s řešením byly restartovány a poté se skenování zase znovu automaticky rozběhlo po půl hodině, jak bylo nastaveno. Test proběhl úspěšně.

Vyhodnocení testů

Všechny testy proběhly úspěšně a byla ověřena základní funkcionalita definovaná zadáním. Pro ověření funkčnosti byl tedy vytvořený systém otestován. Bylo zvoleno spouštění skenu v pravidelných časových intervalech pomocí Bamboo triggeru. Skenování se pravidelně spouštělo a uložilo výsledky do GVM. Výsledky je možné prohlížet, třídit, porovnávat přes webové rozhraní GSA. Webové rozhraní také disponuje možností konfigurace skenování a potlačení falešných detekcí. U aplikace Rocket.Chat bylo odhaleno celkem 13 zranitelností.

Kapitola 5

Závěr práce

Cílem práce bylo implementovat skener OpenVAS s nástrojem pro průběžnou integraci Atlassian Bamboo tak, aby analýza zranitelností byla prováděna automatizovaně, byla konfigurovatelná a bylo možné potlačit falešné detekce. Tento cíl byl splněn.

Po prostudování dostupných materiálů týkajících se zranitelností, skenování zranitelností a průběžné integrace byly hlavní poznatky popsány ve shrnutí současného stavu. Více rozvedeny byly nástroje OpenVAS a Atlassian Bamboo. Byl implementován Atlassian Bamboo s open-source frameworkem Greenbone Vulnerability Management, který obsahuje OpenVAS a umožňuje konfiguraci skenování, porovnání a filtrování výsledků a potlačení falešných detekcí. Pro pravidelné spouštění skenování byly použity triggery v rámci Bamboo plánu. Implementace byla ověřena na aplikaci Rocket.Chat s otevřeným kódem. Systém byl také v obdobné podobě nasazen ve společnosti Y Soft.

Práce objevila 13 zranitelností ve skenovaném systému Ubuntu 18.04 s nasazenou aplikací Rocket.Chat a naučila mě nejen základy dynamické analýzy bezpečnosti systémů, ale také konfiguraci operačních systémů, použití virtualizačních nástrojů a skriptování pomocí jazyka Bash a Python.

Dalším pokračováním práce by mohlo být rozšíření skenovaného systému o další virtuální stroje a testování nasazené aplikace na více platformách. Usnadnila by také další práci implementace automatického zálohování při přechodech na nové verze frameworku Greenbone Vulnerability Management.

Literatura

[1] *Probely Security Scanner Plugin* [online]. Santa Clara, California, USA: Jenkins, 2019 [cit. 2020-05-08]. Dostupné z: https://github.com/jenkinsci/probely-security-plugin.

[2] *Bamboo Server vs. Jenkins* [online]. Sydney, Austrálie: Atlassian, 2020 [cit. 2020-05-08]. Dostupné z: https://www.atlassian.com/software/bamboo/comparison/bamboo-vs-jenkins.

[3] *Common Vulnerability Scoring System version 3.1: Specification Document: CVSS Version 3.1 Release* [online]. Cary, North Carolina, USA: FIRST, 2020 [cit. 2020-05-08]. Dostupné z: https://www.first.org/cvss/specification-document.

[4] *Dynamic Application Security Testing (DAST)* [online]. San Francisco, California, USA: GitLab, 2020 [cit. 2020-05-08]. Dostupné z: https://docs.gitlab.com/ee/user/application_security/dast/.

[5] *National Vulnerability Database: General Information* [online]. Gaithersburg, Maryland, USA: NIST, 2020 [cit. 2020-05-08]. Dostupné z: https://nvd.nist.gov/general.

[6] *OpenVAS - Open Vulnerability Assessment Scanner* [online]. Osnabrück, Německo: Greenbone, 2020 [cit. 2020-05-08]. Dostupné z: https://www.openvas.org/.

[7] *OWASP Top Ten: Top 10 Web Application Security Risks* [online]. Bel Air, Maryland, USA: OWASP, 2020 [cit. 2020-05-08]. Dostupné z: https://owasp.org/www-project-top-ten/.

[8] *Probely Security Scanner Orb* [online]. Lisabon, Portugalsko: Probely, 2020 [cit. 2020-05-08]. Dostupné z: https://github.com/Probely/probely-orb.

[9] *Snyk Intel Vulnerability DB Access* [online]. Boston, Massachusetts, USA: Snyk, 2020 [cit. 2020-05-08]. Dostupné z: https://snyk.io/product/vulnerability-database/.

[10] *Statistics Results: Total Matches By Year* [online]. Gaithersburg, Maryland, USA: NIST, 2020 [cit. 2020-05-08]. Dostupné z: https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&search_type=all.

[11] Doupé, A., Cova, M. a Vigna, G. Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In:. 2010, sv. 6201, s. 111–131. ISBN 3642142141.

[12] Duvall, P., Matyas, S. M. a Glover, A. *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*. Boston, Massachusetts, USA: Addison-Wesley Professional, 2007. ISBN 0321336380.

[13] Farook, U. *A Step by step guide to integrate zap with jenkins* [online]. Bengaluru, Indie: We45, 2018 [cit. 2020-05-08]. Dostupné z: https://www.we45.com/blog/how-to-integrate-zap-into-jenkins-ci-pipeline-we45-blog.

[14] Krátký, R. CVE. In: *AbcLinuxu* [online]. Praha: Nitemedia, 2020 [cit. 2020-05-01]. ISSN 1214-1267. Dostupné z: https://www.abclinuxu.cz/slovnik/cve.

[15] Mell, P., Scarfone, K. a Romanosky, S. Common Vulnerability Scoring System. *IEEE Security & Privacy*. IEEE. 2006, sv. 4, č. 6, s. 85–89. ISSN 1540-7993.

[16] Ou, X. a Singhal, A. The Common Vulnerability Scoring System (CVSS). In: *Quantitative Security Risk Assessment of Enterprise Networks*. New York, NY: Springer New York, 2011, s. 9–12. SpringerBriefs in Computer Science. ISBN 9781461418597.

[17] Sacolick, I. What is CI/CD? Continuous integration and continuous delivery explained. *InfoWorld.com*. San Mateo: Infoworld Media Group. 2020. Dostupné z: http://search.proquest.com/docview/2340149003/.

[18] Shirey, R. W. *Internet Security Glossary* [online]. verze 2.0. RFC Editor, 2007 [cit. 2020-05-01]. DOI: 10.17487/RFC4949. Dostupné z: https://rfc-editor.org/rfc/rfc4949.txt.

[19] Wagner, J.-O. *About GVM Architecture: GVM Overview*. Osnabrück, Německo: Greenbone, 2019 [cit. 2020-05-08]. Dostupné z: https://community.greenbone.net/t/about-gvm-architecture/1231.