elektrorevue
ISSN 1213 - 1539

Rok / Year:	Svazek / Volume:	Číslo / Number:	Jazyk / Language
2022	24	3	CZ

Testování průmyslových protokolů pomocí fuzz techniky

Fuzz Testing of Industrial Protocols

Antonín Bohačík, Petr Ilgner, Radek Fujdiak

Antonin.Bohacik@vut.cz

Fakulta elektrotechniky a komunikačních technologií VUT v Brně

DOI: -

Abstract: Industrial control systems have traditionally been very isolated and worked with very consistent data traffic. Along with the Industry 4.0 trend, their isolation is decreasing and industrial systems are much more connected to other networks. In these systems, great emphasis is placed on their reliability, but also recently on cyber security. The article discusses the fuzz testing of two widely used industrial protocols IEC 104 and Modbus, including their security aspects and structures. Moreover, its fuzz tool is presented together with architecture and capabilities to detect possible vulnerabilities.

elektrorevue
ISSN 1213 - 1539

Testování průmyslových protokolů pomocí fuzz techniky

Antonín Bohačík, Petr Ilgner, Radek Fujdiak

Fakulta elektrotechniky a komunikačních technologií VUT v Brně
Email: Antonin.Bohacik@vut.cz, ilgnerp@vut.cz, fujdiak@vut.cz

Abstrakt *– Průmyslové řídicí systémy byly vždy tradičně velmi izolované a pracovaly s velmi konzistentním datovým provozem. Spolu s trendem Průmyslu 4.0 se jejich izolace snižuje a průmyslové systémy jsou mnohem více propojeny s ostatními sítěmi. Velký důraz je v těchto systémech kladen na jejich spolehlivost, ale také na kybernetickou bezpečnost. Článek pojednává o možnostech testování fuzzingu průmyslových protokolů IEC 104 a široce používaném protokolu Modbus, jejich bezpečnostních aspektech a strukturách. Je představen nástroj pro fuzzingové testování těchto protokolů s cílem odhalit možné zranitelnosti. Podrobně je popsána architektura navrhovaného fuzzeru. Schopnosti fuzzeru jsou pak demonstrovány na testovacím zařízení.*

1 Úvod

Průmyslové řídicí systémy (ICS – Industrial Control Systems) se v dnešní době používají v celé řadě průmyslových odvětví. Používají se k plánování a řízení výrobních linek, k řízení prvků inteligentních měst, ale také v řídicích systémech obytných budov. Samozřejmě se používají i v rámci prvků kritické infrastruktury, jako jsou energetické sítě. V minulosti tato zařízení obvykle fungovala v izolovaných, oddělených sítích. V souvislosti s trendem revoluce průmyslu 4.0 a internetu věcí (IoT – Internet of Things) se tyto systémy rychle propojují se systémy vyšší úrovně. V současnosti je běžné řídící systémy či počítače propojovat prostřednictvím různých řídicích aplikací, cloudových služeb do internetu, jako jsou například systémy SCADA (Supervisory Control and Data Acuistion). Tyto systémy slouží převážně k řízení energetických a průmyslových sítí. Při návrhu těchto systémů je kladen obrovský důraz na jejich spolehlivost [1, 2, 3, 4, 5].

Protokol Modbus je jedním z nejpoužívanějších průmyslových protokolů a je velmi rozšířený, zejména díky značnému rozsahu podporovaných systémů a snadné implementaci. Při samotném návrhu protokolu Modbus bylo běžné, že průmyslové aplikace fungovaly na zcela oddělených sítích a zařízeních. To byl jeden z důvodů proč při návrhu protokolu nebyl kladen důraz na možnost zabezpečení, kontrolu zpráv nebo vzájemného ověřování mezi zařízeními. Mezi další hojně využívané protokoly pro energetiku a průmysl patří protokol IEC 60870-5-104 (zkráceně IEC 104) ze standardu IEC 60870. Tento protokol na rozdíl od protokolu Modbus ve svém základu obsahuje techniky pro zajištění komunikační bezpečnosti neboť byl vytvořen pro komunikaci po TCP/IP sítích [6, 7].

Důležitým tématem dnešní doby je kybernetická bezpečnost. Kybernetický útok může v extrémních případech zapříčinit rozsáhlé výpadky elektřiny, které působí značné ekonomické škody a také může dojít k ohrožení samotného obyvatelstva. Z těchto důvodů je článek zaměřen na vytvořený framework pro fuzzing protokolu Modbus a IEC 104. Vytvořený fuzzer umožňuje snadné provádění fuzzingové testování, které dokáže odhalit nedostatky či zranitelnosti implementacích zmíněných protokolů na konkrétních zařízeních [8, 9, 10].

1.1 Současný stav

Náš výzkum se zaměřuje na fuzzing průmyslových protokolů a zejména SCADA protokolů, a proto jsme provedli průzkum dostupných nástrojů pro fuzzing protokolu Modbus a protokolu IEC 104. V této oblasti existuje jen málo prací, které se zaměřují na automatizované testování těchto specifických protokolů.

Winter a Rademacher [11] uvádějí fuzzer založený na Boofuzz, který se zaměřuje na testování standardu IEC 61850 používaného v elektroenergetických systémech. Jejich fuzzer byl použit k testování známé open-source implementace standardu. Dantas a kol. [12] vytvořili fuzzer založený na fuzzeru Peach pro komunikaci protokolu DLSM/COSEM, běžně vysvytující se u komunikace inteligentních měřičů. V současné době je k dispozici pod komerční licencí, komunitní verze není vyvíjena.

Katsigiannis a kol. představují návrh fuzzeru MTF-Storm [13], který je založen na frameworku MFT. Jedná se o přístup, kdy jsou testovací požadavky generovány na základě znalosti struktury protokolu. Jejich řešení podporuje jak komunikaci Modbus TCP, tak i sériovou komunikaci Modbus RTU. Využívá optimalizaci, kdy jsou generované hodnoty rozděleny do tří podmnožin (i) neplatné hodnoty, (ii) zajímavé hodnoty a (iii) nezajímavé hodnoty. Optimalizace pak spočívá v generování testů tak, aby byly testovány všechny „zajímavé hodnoty" a pouze náhodně vybrané „nezajímavé hodnoty".

Sabraoui a kol. [7] rovněž představují fuzzer založený na generování testovacích zpráv podle znalosti struktury protokolu. Pro generování paketů se používá knihovna `Scapy`. Dále jsou použity části knihovny modlib pro údržbu a správu spojení Modbus. Podporována je pouze komunikace přes spojení TCP.

elektrorevue
ISSN 1213 - 1539

Na druhou stranu některé fuzzery neznají strukturu rámců protokolu a pracují na principu mutace zachycené platné komunikace. To je výhodné pro testování proprietárních protokolů, které nejsou dobře popsány. Jedním z takových přístupů je PropFuzz, který autoři přímo prezentují jako vhodný pro testování proprietárních protokolů [14].

Vzhledem k tomu, že protokoly Modbus a IEC 104 jsou v průmyslu široce používány, existuje také několik komerčních nástrojů pro provádění jejich fuzzingového testování. Tabulka 1 uvádí přehled tří nástrojů, které jsou v současné době k dispozici. Všechny tyto fuzzery pracují na základě znalosti protokolů.

Tabulka 1: Srovnání komerečních nástrojů pro fuzzing protokolů Modbus a IEC 104.

Název	**Modbus TCP/RTU**	**IEC 104**
Aegis [15]	Pouze TCP	Ano
BeSTORM [16]	TCP a RTU	Ano
Defensics [17]	TCP a RTU	Ano

2 Specifikace protokolu Modbus

Modbus je jedním z nejpoužívanějších protokolů ICS pro řízení a automatizaci. Díky své otevřenosti, bezplatnému používání a snadné implementaci se stal de facto průmyslovým standardem. Modbus podporuje komunikaci po sériové lince, typicky po sběrnici RS485, nebo po Ethernetu pomocí transportního protokolu TCP. Původní specifikaci protokolu zveřejnila společnost Modicon (nyní Schneider Electric) v roce 1979 [18].

Architektura protokolu je založena na modelu klient-server. Obvykle se používá v systémech SCADA, které vystupují jako klient. Jednotlivé další prvky, jako jsou programovatelné logické automaty (PLC – Programmable Logic Controller) nebo jiné senzory, pak představují servery a odpovídají na požadavky systému SCADA [19, 20].

Z bezpečnostního hlediska se protokol Modbus potýká se 3 typy možných útoků na zařízení. Lze rozdělit na (i) zachycení, (ii) přerušení, (iii) modifikace, (iv) tvorba falešných zpráv. Vzhledem k absenci bezpečnostních mechanismů, jako je ověřování zařízení, pořadová čísla zpráv a podobně, je zřejmé, že komunikace protokolu Modbus, ať už po sériové lince nebo TCP/IP, musí být velmi pečlivě chráněna před vnějším přístupem neoprávněných osob [21].

2.1 Struktura protokolu

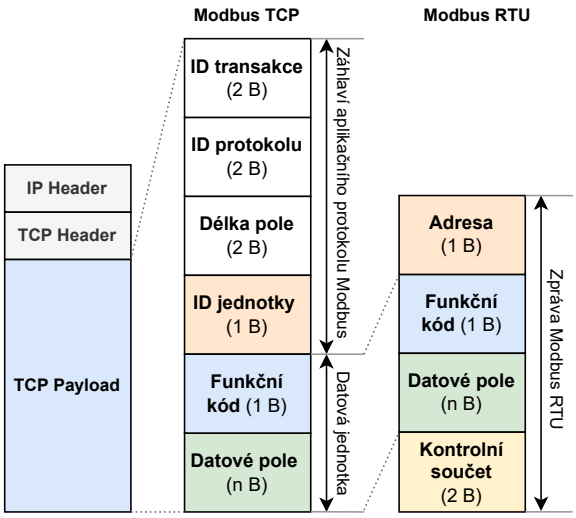
Jednou z výhod protokolu Modbus je jeho otevřenost a relativní jednoduchost struktury zpráv a funkčních příkazů. To je výhodné pro samotnou implementaci fuzzingu tohoto protokolu, protože je možné snadno fuzzovat množinu podporovaných zpráv celého protokolu. Pro možnost generování testovacích zpráv pomocí fuzzeru podle znalosti protokolu je důležité popsat jeho strukturu. V následujícím textu je podrobně popsána struktura zpráv protokolu Modbus včetně jeho základních funkčních kódů. Uvažujeme dvě nejběžnější varianty protokolu Modbus podle použitého přenosového média:

1. **Modbus RTU** – přenosovým médiem je sériová komunikační sběrnice, obvykle RS485, preferovaný režim je 19 200 baudů, 8 datových bitů a sudá parita. V režimu RTU je přenos zprávy realizován jako přenos jednoho znaku (1 znak představuje 8 bitů), začátek a konec přenosu jsou realizovány pomlčkou na sběrnici delší než 3,5 znaku. Celistvost přenášené zprávy je kontrolována pomocí kontrolního součtu CRC (Cyclic Redundancy Check) a paritním bitem. Tento princip umožňuje velmi jednoduchou implementaci protokolu a rychlou a spolehlivou komunikaci [22].

2. **Modbus TCP** – Jako přenosové médium se používá Ethernet. Zprávy jsou přenášeny přes sítě TCP/IP, standardně s použitím portu 502. Protokol Modbus TCP nezajišťuje kontrolu integrity zpráv a spoléhá se na výpočet kontrolního součtu spolehlivého přenosového protokolu TCP [23].

Základní částí zprávy protokolu Modbus je datová jednotka protokolu (PDU – Protocol Data Unit), která je vždy stejná nezávisle na použité variantě protokolu. Obsahuje číslo kódu funkce, které určuje typ akce, která má být s daty provedena. Následuje datová část PDU, jejíž struktura je odvozena od použité funkce.

Aplikační datová jednotka (ADU – Application Data Unit) představuje celý rámec protokolu Modbus a obsahuje PDU a pole podle typu použité sběrnice. K variantě Modbus RTU se přidává adresa a kontrolní součet. U varianty Modbus TCP se přidává identifikátor transakce, pevný identifikátor protokolu a délka rámce. Struktura a obsah ADU pro oba typy je uveden na obrázku 1.



Obrázek 1: Struktura zprávy Modbus [23].

elektrorevue
ISSN 1213 - 1539

Funkce protokolu jsou specifikovány číslem kódu funkce o délce jednoho bajtu. Hodnoty nad 128 jsou vyhrazeny pro přenos informace o výskytu výjimky. Účel funkcí lze rozdělit na (i) funkce přístupu k datům, (ii) diagnostické funkce a (iii) ostatní funkce. Tabulka 2 uvádí význam základních primitivních kódů funkcí pro přístup k datům.

Tabulka 2: Základní funkční kódy protokolu Modbus [23].

Kód	Název funkce	Popis
01	Read Coils	Čtení jednoho nebo více (zapisovatelných) bitů.
02	Read Discrete Inputs	Čtení jednoho nebo více bitů (pouze pro čtení).
03	Read Holding Registers	Čtení jednoho nebo více 16b (zapisovatelných) registrů.
04	Read Input Registers	Čtení jednoho nebo více 16b registrů (pouze pro čtení).
05	Write Single Coil	Zápis jednoho bitu.
06	Write Single Register	Zápis jednoho 16b registru.
15	Write Multiple Coils	Zápis více bitů.
16	Write Multiple Registers	Zápis více 16b registrů. registrů.

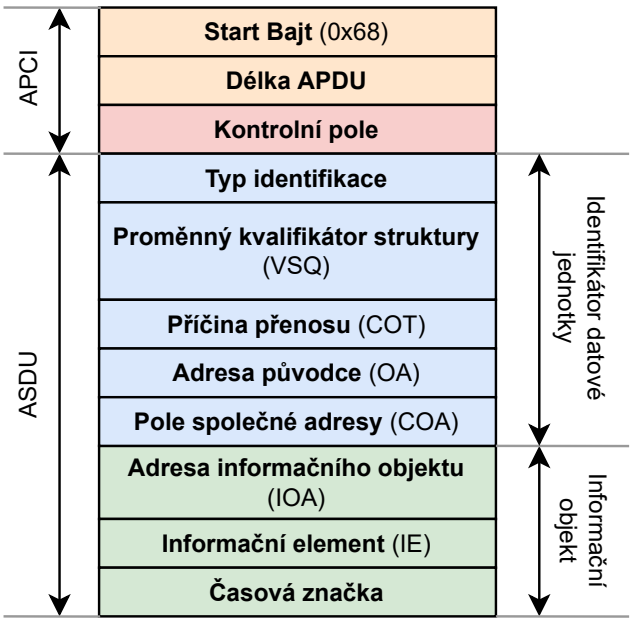
3 Specifikace protokolu IEC 60870-5-104

Protokol IEC 104 patří do skupiny protokolů standardu IEC 60870 používaných pro dálkové ovládání ve SCADA systémech, zejména pro řízení v energetických sítích. Protokol je založen na zprávách STA (Station Object) definovaných v IEC 60870-5-101 (IEC 101). Významnou změnou je, že pro přenos dat je používán protokol TCP/IP [24]. Způsob zapouzdření datových jednotek v rámci protokolu je pro konstrukci fuzzingu zásadní. V této části je podrobněji popsána struktura protokolu [25].

3.1 Formát datového jednotky

Protokol definuje přenášená data v objektech stanice, které jsou identické s objekty používanými protokolem IEC 101. Základním přenosovým rámcem je ACPI (Application Protocol Control Information). ACPI může volitelně obsahovat datovou jednotku aplikační služby (ASDU – Application Service Data Unit), která může obsahovat až 253 informačních objektů [25].

Struktura paketu tzv. APDU (Application Protocol Data Unit) obsahujícího jeden informační objekt je znázorněna na obrázku 2. Protokol IEC 104 definuje tři různé komunikační směry (i) směr monitoru – přenos z řízené stanice na řídicí stanici, (ii) směr řízení – směr přenosu z řídicí stanice a (iii) obrácený směr – směr, kdy monitorovaná stanice vysílá příkazy a řídicí stanice vysílá data ve směru monitoru [24].



Obrázek 2: Struktura APDU protokolu IEC 104 [24].

4 Fuzzing protokolů

Fuzzing je široce používaná technika pro hledání zranitelností v softwaru. Předkládáním neočekávaných vstupů testovanému programu a sledováním jeho činnosti lze detekovat potenciální zranitelnosti, které by bylo obtížné odhalit jinými technikami. V oblasti zabezpečení softwaru se jedná o jednu z nejúčinnějších technik automatizovaného odhalování zranitelností. Cílem fuzzingu je dosáhnout maximálního pokrytí, jinými slovy otestovat co největší část zdrojového kódu testovaného programu.

S rozvojem počítačových sítí se výzkum v oblasti fuzzingu zaměřil také na fuzzing síťových protokolů. Konkrétně se jedná o testování implementací daného protokolu. Chyby v implementacích protokolů mohou útočníkům otevřít cestu k nalezení exploitů, které jim v extrémním případě umožní získat kontrolu nad systémem. Případně mohou tyto chyby způsobit neočekávané selhání zařízení [26].

V případě průmyslových protokolů je kladen velký důraz na spolehlivost, proto je fuzzing implementací průmyslových protokolů vhodným nástrojem, který lze použít ke zvýšení spolehlivosti a bezpečnosti systémů.

4.1 Princip fuzzingu pro průmyslové protokoly

Tento článek se zaměřuje na fuzzingové testování softwaru nebo zařízení komunikujících prostřednictvím protokolu Modbus a IEC 104. Základním principem fuzzingu komunikačních protokolů je konstrukce chybných paketů pomocí technik generování a mutace s následným odesláním testovaného zařízení, obvykle nazývaného Device Under Test (DUT). Toto zařízení je během procesu fuzzingu monitorováno a všechny zjištěné výjimky a neočekávané chování

elektrorevue
ISSN 1213 - 1539

nebo pád zařízení jsou zaznamenány. Je tak zřejmé, která vstupní data způsobila toto neočekávané chování, a vývojář může problém odstranit [27, 28]. Na základě detekce těchto neočekávaných stavů lze odhalit i anomálie, jako je přetečení vyrovnávací paměti v implementaci komunikačního protokolu. Celý proces fuzzingu zařízení implementujícího síťový protokol lze rozdělit do několika kroků [29]:

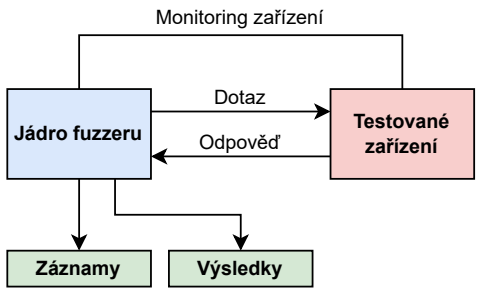
1. **Identifikace DUT** – spočívá v identifikaci vlastností testovaného zařízení, jako je typ síťového protokolu, komunikační rozhraní, síťové porty atd.
2. **Určení vstupů** – na základě identifikace určete, jak jsou generovány deformované pakety.
3. **Generování testovacích dat** – v tomto kroku fuzzer generuje testovací data, nazývaná také kroky fuzzerování, která jsou dále odesílána do DUT.
4. **Spuštění relace fuzzingu** – v tomto kroku běží relace fuzzingu, takže vygenerovaná testovací data jsou odesílána do DUT a jeho stav je průběžně monitorován. Je důležité, aby byly zaznamenány všechny podrobnosti testovacích kroků, zejména těch, které skončily výjimkou, aby bylo možné tyto kroky replikovat.
5. **Analýza záznamů** – Posledním krokem je vyhodnocení výsledků fuzzingového testování, přičemž je důležité určit důvod případných výjimek.

Podle znalosti zdrojového kódu testovaného softwaru, který je využíván pro vytváření testovacích vstupů, lze fuzzingové testování rozdělit na tři základní typy [30]:

1. **Black-box** – testovací vstupy jsou vytvářeny zcela bez znalosti zdrojového kódu testovaného zařízení. To je výhodné zejména u vysoce strukturovaných vstupů, jako jsou datové jednotky komunikačních protokolů, protože specifikace protokolu je obvykle známa.
2. **Gray-box** – pro ovládání fuzzeru se používá znalost zdrojového kódu programu. Obvykle se používají vstupy typu seed, které se dále mutují a generují nová vstupní data pro testování.
3. **White-box** – vytváření testovacích vstupů je založeno výhradně na znalosti a analýze zdrojového kódu testovaného softwaru.

Jelikož je struktura protokolů Modbus i IEC 104 všeobecně dobře známa, náš přístup používá testování typu black-box. Jádro představeného fuzzingového frameworku je založeno na nástroji Boofuzz, do kterého jsme přidali podporu fuzzingování zmíněných protokolů. Boofuzz je fuzzer pro testování síťových protokolů využívajících ethernetovou nebo sériovou linkovou komunikaci. Je naprogramován v jazyce Python a je modulární, což umožňuje specifikovat formát paketu testovaného protokolu. Boofuzz je k dispozici pod licencí GPLv2 [31, 32].

Proces testování pomocí tohoto fuzzeru spočívá v tom, že na základě znalosti protokolu jsou ve vygenerovaném paketu provedeny mutace. Popis specifikace formátu definuje místa, která se nemutují, například oddělovače a další pevné části protokolu. Je možné definovat různé typy podporovaných zpráv skládajících se z jednotlivých bloků, které jsou specificky definovány a následně připojeny ke grafu gramatiky protokolu. Fuzzer generuje jednotlivé testovací případy na základě grafu znalostí testovacího protokolu [31]. Základní koncept fuzzingu typu black-box a zpracování odezvy je znázorněn na obrázku 3.



Obrázek 3: Postup fuzzeru při testování vstupů.

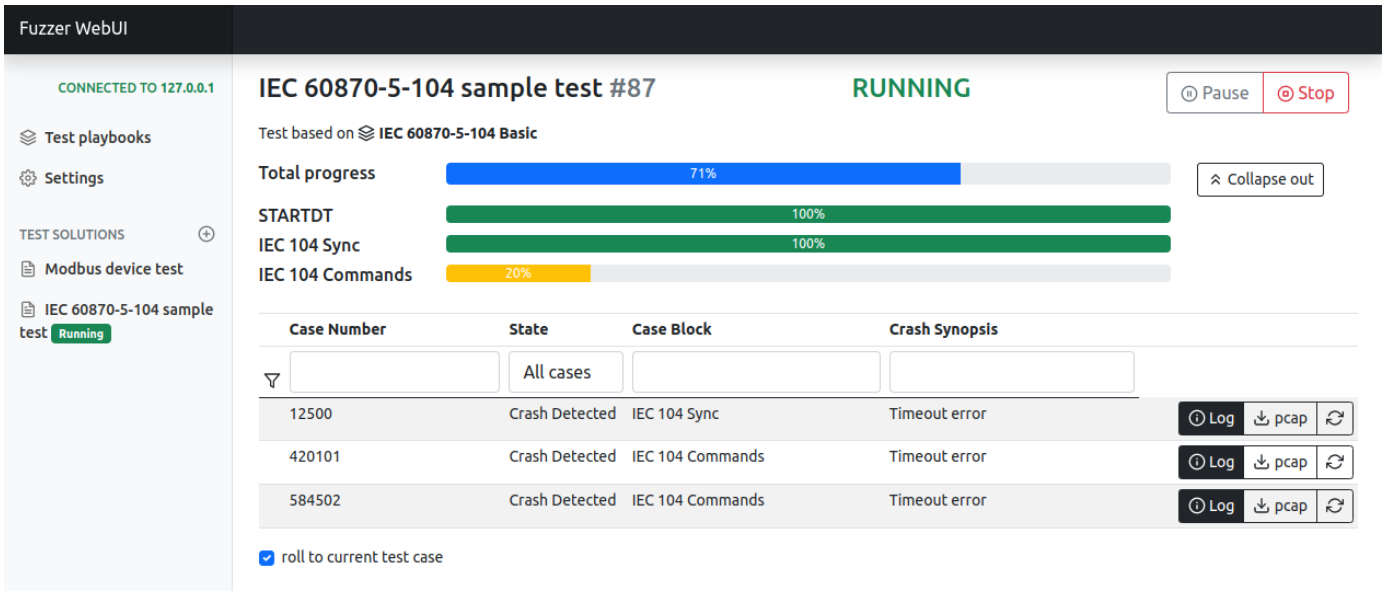
Při návrhu fuzzingového frameworku byl kladen důraz na poskytnutí jednoduchého ovládání bez nutnosti připravovat testovací scénáře přímo v kódu programu. Rovněž na snadné použití, aby jej bylo možné využít při praktickém ověřování nasazovaného softwaru nebo zařízení. Navržený fuzzingový framework je navržen tak, aby podporoval více komunikačních protokolů. Vytvořili jsme (webové) grafické rozhraní, ke kterému lze přistupovat z jiného počítače. Navržený framework tak může běžet na serveru a stav testování lze sledovat a ovládat na dálku. Podoba grafického rozhraní je znázorněna na obrázku 4.

Fuzzingový framework umožňuje vytvářet definice jednotlivých podporovaných protokolů ve formě playbooků, která obsahuje jednak definici struktury protokolu, ale také definici různých konfiguračních možností. Tyto možnosti může uživatel zadat při vytváření jednotlivého testu a spustit je přímo pomocí uživatelského rozhraní. Celková architektura systému je znázorněna na obrázku 5.

Přidanou hodnotou vytvořeného frameworku je možnost paralelního testování, které vychází z předpokladu, že pokud může být k dispozici více instancí testovaného zařízení, které jsou identické, mělo by být identické i chování zařízení při zaslání identických testovacích dat. To umožňuje výrazně urychlit celý proces testování.

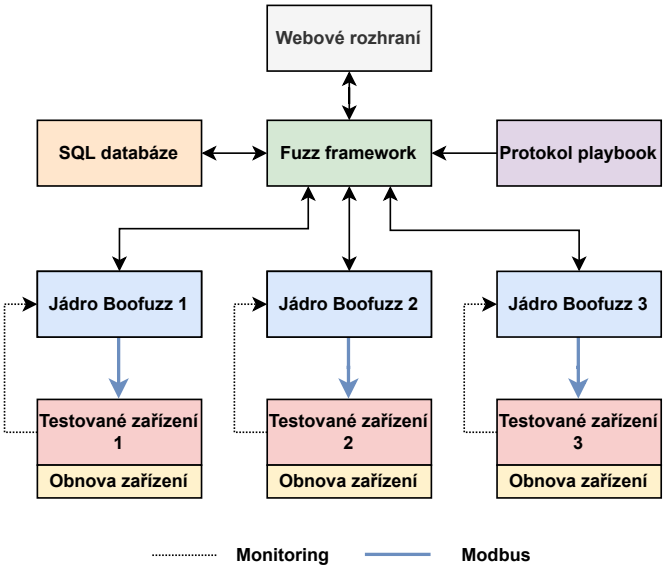
Ačkoli Boofuzz umožňuje zadat více testovacích cílů, na základě provedených výkonnostních experimentů, framework vytvoří pro každou instanci testovaného zařízení samostatnou instanci jádra Boofuzz. Výsledky z těchto instancí jsou předávány řídicí aplikaci frameworku, která poskytuje protokolování informací o provedených testovacích případech a možnost řídit celkové testování. Vytvořený framework je zodpovědný za distribuci jednotlivých testovacích případů do instancí jádra Boofuzz.

elektrorevue
ISSN 1213 - 1539



Obrázek 4: Webové rozhraní vytvořeného Fuzzer frameworku.

Jednotlivé instance testované služby jsou spuštěny v kontejnerech Docker. Ten byl zvolen neboť umožňuje automatizované vytvoření testovacího prostředí a neklade velké nároky na systémové prostředky [33, 34]. V případě, že jednotlivé testovací instance nelze spustit pomocí Dockeru nebo je třeba testovat fyzická zařízení, lze je spustit ručně a poté v konfiguraci fuzzingového frameworku nastavit IP adresy jednotlivých testovaných zařízení.



Obrázek 5: Komunikace v rámci fuzzingu se třemi paralelně testovanými zařízeními komunikující protokolem Modbus.

4.1.1 Protokol playbook

Struktura zpráv protokolu Modbus a IEC 104 je v rámci frameworku definována v souborech playbook. Ty lze snadno přidat jako rozšíření, aniž by bylo nutné měnit zdrojový kód fuzzeru. Soubory playbook obsahují jednak definici protokolu Modbus, protokolu IEC 104, základní logiku testování fuzzingu, či seznam datových polí proměnných, které může uživatel vyplnit při přípravě testovacích scénářů. Operátor fuzzeru vytváří testovací scénáře, ve kterých se nastavují jednotlivé cíle testu a další parametry související s testem.

4.1.2 Monitoring a obnova testovaného zařízení

Testované zařízení lze sledovat třemi možnými způsoby. Nejjednodušší *Síťový monitoring* je založen na principu, že pokud se testované zařízení zhroutí, přestane reagovat na příchozí zprávy. V případě protokolu TCP to obvykle způsobí, že cíl přestane naslouchat na definovaném portu. Druhou možností je použití *Procesní Monitoring*, který lze použít, pokud je testované zařízení aplikací běžící přímo na počítači fuzzeru. V tomto případě může fuzzer spouštět jednotlivé cíle a na základě identifikátoru procesu provádět kontrolu stavu tohoto procesu po každém z kroků testu. Poslední zjednodušenou verzí je *Jednoduchý monitoring*. Jedná se o přístup, kdy v případě, že není možné navázat spojení s cílem, je předchozí testovací krok spuštěný na tomto cíli označen jako neúspěšný, a proto je předchozí krok považován za ten, který způsobil pád.

Pokud je zjištěn pád DUT, je možné zařízení restartovat, takže testování tohoto zařízení může pokračovat. Vytvořený framework podporuje dvě možnosti obnovy zařízení. V případě, že je testovací cíl spuštěn na stejném počítači, dojde při zjištění havárie k automatickému restartu cíle. Druhou možností je definovat skript, který se spustí

elektrorevue
ISSN 1213 - 1539

a zajistí restart. Identifikátor cíle je skriptu předán jako parametr. To je výhodné pro testování skutečných zařízení, u nichž lze jejich restart zajistit např. pomocí zásuvky PDU (Power Distribution Unit) s možností řízení napájecího cyklu, včetně definice časové prodlevy po restartu. V případě, že funkce obnovení není nastavena, fuzzer periodicky opakuje testovací krok, tj. čeká, až obsluha provede restart manuálně.

4.1.3 Záznam výsledků

K ukládání výsledků lze použít buď úložiště souborů SQLite, které používá projekt Boofuzz a které obsahuje (i) protokol testovacího kroku, (ii) popis sestaveného paketu, (iii) výsledek kroku a (iv) časové značky. Druhou možností je použití databázového úložiště frameworku, které využívá databázi MySQL a poskytuje podrobnější protokolování, včetně informací o použitých cílech v jednotlivých krocích, ale také záznam kompletního síťového provozu ve formátu `pcap`. Protokolování síťového provozu je založeno na knihovně `pyshark` [35]. Aby nedocházelo k blokování hlavního vlákna fuzzeru, probíhá zpracování zaznamenaných dat v samostatném vlákně.

4.2 Ověření fuzzingu protokolu Modbus

Pro testování fuzzingového frameworku jsme vytvořili experimentální Modbus TCP server založený na `pymodbus` pro implementující kompletní sadu funkcí pro fungováním Modbus TCP i Modbus RTU. Kromě knihovny `pyserial` nevyžaduje v režimu RTU žádné externí závislosti. Všechny testy byly provedeny v operačním systému Ubuntu 22.04 LTS. Tento vytvořený experimentální server byl upraven odstraněním několika obsluh výjimečných stavů při přístupu na neexistující adresu. Server se pak pokusil přistoupit k neplatné oblasti paměti, což zapříčinilo pád zařízení. V tomto testovacím scénáři byly jednotlivé instance serveru (cíle) spouštěny dynamicky pomocí frameworku, viz topologie na obrázku 5. Podle očekávání byly tyto chyby nalezeny a instance testovacího serveru byla při jejich výskytu automaticky restartována.

Další testovací scénář využíval dva identické komerčně dostupné programovatelné logické automaty (PLC). Tyto PLC jsou založeny na průmyslovém PC se čtyřjádrovým procesorem ARM Allwinner-H5 a jsou vybaveny serverem Modbus TCP při přístupu přes Ethernet a Modbus RTU při použití integrované sběrnice RS485. Jejich restartování bylo realizováno pomocí jednoduchého přípravku sestaveného z vývojové desky Arduino Nano a dvoukanálového reléového modulu přes rozhraní USB.

Při testování Modbus TCP docházelo k poměrně pravidelným detekovaným pádům a restartům PLC. Pozdější analýza odhalila, že u PLC docházelo k chybě *Server accept(): Příliš mnoho otevřených souborů*. Chyba zřejmě souvisí s nesprávným ukončením soketů v rámci serveru Modbus. Dále jsme testovali Modbus RTU pomocí převodníku USB na sběrnici RS485. Tento převodník používá čip CH341 a je kompatibilní s operačními systémy Windows, Linux a MacOS [36]. Na identickém PLC používajícím RS485 se výše uvedená chyba nevyskytla a nebyla zjištěna žádná zranitelnost.

4.3 Ověření fuzzingu protokolu IEC 60870-5-104

Dále jsme k testování vytvořeného frameworku využili knihovnu `j60870` dostupnou pod licencí GPLv3 [37]. Jedná se o knihovnu, která implementuje komunikační protokol IEC 104 napsaný v programovacím jazyce Java. Pomocí knihovny byl vytvořen testovací server a byla do něj vložena zranitelnost přetečení bufferu. V případě, kdy metoda zpracovávající přijatou zprávu *SC_NA_1*, obdrží adresu IOA (Information Object Address), pro kterou není v poli *ioMap* žádný záznam, je vyvolána výjimka. Protože tato výjimka není nijak ošetřena, celá aplikace se po přijetí zprávy neočekávaně ukončí.

Jedná se o simulaci typické programátorské chyby, která může způsobit neočekávané chování systému, případně pád celého programu. Taková chyba je obtížně zjistitelná a může ovlivnit další paměťový prostor, pokud není provedena kontrola přístupu do paměti. V extrémním případě, pokud útočník dokáže využít zranitelnosti a vytvořit exploit, může získat kontrolu nad celým zařízením [38].

Instance testovaného zařízení se zhroutila při provádění fuzzing testování. Sledování testovaného zařízení je realizováno tak, že po odeslání testovacích dat se zjišťuje, zda server naslouchá na daném portu a odpovídá na požadavek o připojení k soketu. Pokud se testované zařízení neočekávaně ukončí nebo přestane odpovídat, požadavek o připojení nebude v časovém limitu vyřízen, a bude tak odhalena chyba implementace. Ukázka fuzzerem vytvořené ASDU, která zapříčinila pád testovaného IEC 104 zařízení, je zobrazena ve výpisu 1.

Výpis 1: ASDU způsobující pád testovaného zařízení.

```
IEC 60870-5-101/104 ASDU: ASDU=1 C_SC_NA_1 Act IOA=5000
"single command"
     TypeId: C_SC_NA_1 (45)
     0... .... = SQ: False
     .000 0001 = NumIx: 1
     ..00 0110 = CauseTX: Act (6)
     .0.. .... = Negative: False
     0... .... = Test: False
     OA: 0
     Addr: 1
     IOA: 500
          IOA:500
          SCO: 0x81
               .... ...1 = ON/OFF: On
               .000 00.. = QU: No pulse defined (0)
               1... .... = S/E: Select
```

5 Závěr

Kybernetická bezpečnost systémů SCADA, které se používají v elektrických inteligentních sítích, je v éře Průmyslu 4.0 velmi důležitá. Systémy jsou nyní propojeny více než kdy dříve a mohou být vystaveny kybernetickým útokům. Sítě inteligentních sítí jsou často součástí kritické infrastruktury a v případě úspěšného útoku nebo selhání

elektrorevue
ISSN 1213 - 1539

systému může dojít ke značným škodám na majetku i životech. Fuzzing je jedním ze způsobů, jak tyto systémy automatizovaně testovat, a díky tomuto typu testování odhalit i jinak obtížně odhalitelné zranitelnosti.

Tento článek se zabýval možností testování protokolů Modbus a IEC 104, které jsou široce používány v průmyslovým řídicích systémech a energetice. Představili jsme vytvořený framework pro fuzzingové testování těchto komunikačních protokolů založeného na znalosti struktury protokolů a formátu jednotlivých zpráv. Vytvořeným fuzzerem bylo otestován PLC zařízení komunikující protokolem Modbus, kde byly nalezeny zranitelnosti, které budou předány výrobci. Dále byl otestován speciálně vytvořený IEC 104 server s implementovanou zranitelností, která byla fuzzerem odhalena. Architektura vytvořeného fuzzeru velmi usnadňuje provádění testování díky přehlednému grafickému rozhraní a možnosti definovat tzv. playbooky pro opakované spuštění testu. Vytvořený fuzzer umožňuje rozložit testování na více testovacích instancí a zkrátit tak dobu testování zařízení.

V budoucnosti se chceme zaměřit na implementaci dalších průmyslových protokolů do našeho frameworku, včetně dalších pokročilých metod pro optimalizaci generování testovacích dat a upřednostňování vstupů, u nichž je větší pravděpodobnost způsobení neočekávané chování.

Poděkování

Prezentovaný výzkum byl podpořen z rozpočtu Technologické agentury České republiky (TAČR) v rámci projektu reg. č. FW02020084.

Reference

[1] Alotaibi, I.; Abido, M.; Khalid, M.; aj. A Comprehensive Review of Recent Advances in Smart Grids: A Sustainable Future with Renewable Energy Resources. *Energies*. ročník 13. 11 2020: str. 6269. doi: 10.3390/en13236269.

[2] COMMISSION RECOMMENDATION of 3.4.2019 on cybersecurity in the energy sector. 2019. Dostupné z: <https://ec.europa.eu/energy/sites/ener/files/commission_recommendation_on_cybersecurity_in_the_energy_sector_c2019_2400_final.pdf>

[3] Miller, A. Trends in process control systems security. *IEEE Security & Privacy*. ročník 3, č. 5. 2005: s. 57–60.

[4] Ghosh, S.; Sampalli, S. A Survey of Security in SCADA Networks: Current Issues and Future Challenges. *IEEE Access*. ročník 7. 2019: s. 135812–135831. doi:10.1109/ACCESS.2019.2926441.

[5] Babu, B.; Ijyas, T.; Muneer, P.; aj. Security issues in SCADA based industrial control systems. In *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*. IEEE. 2017. s. 47–51.

[6] Igure, V. M.; Laughter, S. A.; Williams, R. D. Security issues in SCADA networks. *computers & security*. ročník 25, č. 7. 2006: s. 498–506.

[7] Sabraoui, M.; Hieb, J. L.; Graham, J. H. MODBUS Protocol Fuzzing for Cyber Security and Hardening of Industrial Control Systems.

[8] Ghosh, S.; Sampalli, S. A survey of security in SCADA networks: Current issues and future challenges. *IEEE Access*. ročník 7. 2019: s. 135812–135831.

[9] Maglaras, L. A.; Kim, K.-H.; Janicke, H.; aj. Cyber security of critical infrastructures. *Ict Express*. ročník 4, č. 1. 2018: s. 42–45.

[10] Lee, R. M.; Assante, M. J.; Conway, T. Analysis of the Cyber Attack on the Ukrainian Power Grid. *E-ISAC*. 2016. Dostupné z: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2016/05/20081514/E-ISAC_SANS_Ukraine_DUC_5.pdf>

[11] Winter, E.; Rademacher, M. Fuzzing of SCADA Protocols used in Smart Grids. *Energy Inform*. ročník 3. 2020: s. 1–3.

[12] Dantas, H.; Erkin, Z.; Doerr, C.; aj. EFuzz: A Fuzzer for DLMS/COSEM Electricity Meters. In *Proceedings of the 2nd Workshop on Smart Energy Grid Security*. SEGS '14. New York, NY, USA: Association for Computing Machinery. 2014. str. 31–38. ISBN 9781450331548. doi:10.1145/2667190.2667194. Dostupné z: <https://doi.org/10.1145/2667190.2667194>

[13] Katsigiannis, K.; Serpanos, D. MTF-Storm: a high performance fuzzer for Modbus/TCP. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, ročník 1. IEEE. 2018. s. 926–931.

[14] Niedermaier, M.; Fischer, F.; von Bodisco, A. PropFuzz—An IT-security fuzzing framework for proprietary ICS protocols. In *2017 International conference on applied electronics (AE)*. IEEE. 2017. s. 1–4.

[15] Aegis ICS smart fuzzing framework. Dostupné z: <https://docs.stepfunc.io/aegis/>

[16] beSTORM Fuzzer Datasheet. Dostupné z: <https://www.beyondsecurity.com/wp-content/uploads/2021/08/beSTORM-Datasheet.pdf>

[17] Defensics Fuzz Testing Datasheet. Synopsys. Dostupné z: <https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/ds-defensics-fuzztesting.pdf>

elektrorevue
ISSN 1213 - 1539

[18] Modbus Specifications and Implementation Guidess. Dostupné z: <https://modbus.org/specs.php>

[19] Modbus Organization Replaces Master-Slave with Client-Server. Dostupné z: <https://modbus.org/docs/Client-ServerPR-07-2020-final.docx.pdf>

[20] Clarke, G.; Reynders, D.; Wright, E. *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes. 2004.

[21] Huitsing, P.; Chandia, R.; Papa, M.; aj. Attack taxonomies for the Modbus protocols. *International Journal of Critical Infrastructure Protection*. ročník 1. 2008: s. 37–44.

[22] Drury, B. *Control Techniques Drives and Controls Handbook (2nd Edition)*. Institution of Engineering and Technology (The IET). 2009. ISBN 978-1-84919-013-8. Dostupné z: <https://app.knovel.com/hotlink/toc/id:kpCTDCHE08/control-techniques-drives/control-techniques-drives>

[23] MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b. Dostupné z: <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf>

[24] Matousek, P. Description and analysis of IEC 104 Protocol. Technická zpráva. Faculty of Information Technology, Brno University o Technology. 2017. Dostupné z: <https://www.fit.vut.cz/research/publication-file/11570/TR-IEC104.pdf>

[25] *IEC 60870-5-104:2006: Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101*. Druhé vydání. 2006.

[26] Mouzarani, M.; Sadeghiyan, B.; Zolfaghari, M. Smart fuzzing method for detecting stack-based buffer overflow in binary codes. *IET software*. ročník 10, č. 4. 2016: s. 96–107. ISSN 1751-8806.

[27] Godefroid, P. Fuzzing: Hack, art, and science. *Communications of the ACM*. ročník 63, č. 2. 2020: s. 70–76.

[28] Li, J.; Zhao, B.; Zhang, C. Fuzzing: a survey. *Cybersecurity*. ročník 1, č. 1. 2018: s. 1–13.

[29] Hu, Z.; Pan, Z. A Systematic Review of Network Protocol Fuzzing Techniques. In *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, ročník 4. IEEE. 2021. s. 1000–1005.

[30] Liang, H.; Pei, X.; Jia, X.; aj. Fuzzing: State of the art. *IEEE Transactions on Reliability*. ročník 67, č. 3. 2018: s. 1199–1218.

[31] Boofuzz: Network Protocol Fuzzing for Humans. Dostupné z: <https://github.com/jtpereyda/boofuzz>

[32] Boofuzz: User Guide. Dostupné z: <https://boofuzz.readthedocs.io/en/stable/index.html>

[33] Pathirathna, P. P. W.; Ayesha, V. A. I.; Imihira, W. A. T.; aj. Security testing as a service with docker containerization. In *2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*. 2017. s. 1–7. doi: 10.1109/SKIMA.2017.8294109.

[34] Chae, M.; Lee, H.; Lee, K. A performance comparison of linux containers and virtual machines using Docker and KVM. *Cluster Computing*. ročník 22, č. 1. 2019: s. 1765–1775.

[35] pyshark: Python wrapper for tshark. Dostupné z: <https://github.com/KimiNewt/pyshark>

[36] Nanjing Qinheng Microelectronics: CH341 datasheet. Dostupné z: <http://www.wch-ic.com/downloads/CH341DS1_PDF.html>

[37] Zimmermann, D.; Schall, A. J60870. Dostupné z: <https://www.openmuc.org/iec-60870-5-104/>

[38] Black, P. E.; Bojanova, I. Defeating Buffer Overflow: A Trivial but Dangerous Bug. *IT professional*. ročník 18, č. 6. 2016: s. 58–61. ISSN 1520-9202.