VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

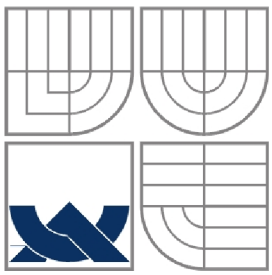
SPECIFIKAČNÍ JAZYKY A NÁSTROJE PRO ANALÝZU A VERIFIKACI BEZPEČNOSTNÍCH PROTOKOLŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

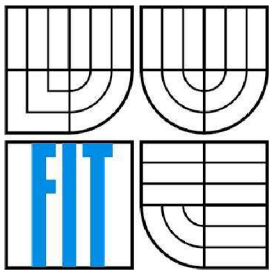
AUTOR PRÁCE
AUTHOR

Bc. MICHAL PTÁČEK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SPECIFIKAČNÍ JAZYKY A NÁSTROJE PRO ANALÝZU A VERIFIKACI BEZPEČNOSTNÍCH PROTOKOLŮ

SPECIFICATION LANGUAGES AND TOOLS FOR ANALYSIS AND VERIFICATION OF SECURITY PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL PTÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL OČENÁŠEK

BRNO 2007

Zadání:

1. Seznamte se s problematikou analýzy a verfikace bezpečnostních protokolů (BP). Detailněji nastudujte specifikační jazyky a nástroje používané na ověřování BP.
2. Popište základní bezpečnostní protokoly, které se u uvedených nástrojů nejčastěji vyskytují. Uveďte jejich vlastnosti, autory a chyby, které u nich byly objeveny.
3. Detailněji popište jednotlivé nástroje a klasifikujte je podle metod, na kterých jsou založeny. Dále uveďte podmínky jejich použití (licence). Popište vývoj jednotlivých nástrojů (autoři, publikování, nové verze atd.) Nástroje také vzájemně porovnejte.
4. U jednotlivých verzí nástrojů uveďte, které BP jimi byly testovány a s jakým výsledkem. Uveďte také, pro jaké typy BP (případně konkrétní BP) příslušné nástroje vhodné nejsou a které útoky nástroje nejsou schopny rozeznat.
5. V eletronické podobě zpracujte materiály k jednotlivým nástrojům (příslušné články, zdrojové texty, prostředí nástrojů, ...). Funkčnost vybraných nástrojů ověřte na vybraných BP.
6. Proveďte souhrn problematiky a nastiňte možnosti dalšího rozvoje práce.

Licenční smlouva

Licenčni smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Tato diplomová práce se zabývá analýzou verifikačních nástrojů, které jsou používané pro automatickou verifikaci převážně bezpečnostních protokolů. Bezpečnostní protokoly se používají pro vytvoření privátních komunikačních kanálů v nezabezpečené síti. Bezpečnost nebude nikdy absolutní, a proto je nezbytné stále hledat chyby v bezpečnostních protokolech a podle nich pak tyto protokoly vylepšovat. V rámci této diplomové práce jsem se zaměřil na hledání různých verifikačních nástrojů na Internetu, následně jsem podle dostupných informací každý nalezený nástroj popsal v této práci tak, aby si čitatel mohl snadno vybrat, který nástroj se hodí zrovna pro ten jeho konkrétní problém. Na konci popisu jednotlivých nástrojů jsem hlavní klady a zápory shrnul do několika bodů.

Klíčová slova

Nástroje, protokol, bezpečnost, Otway-Rees, Yahalom, Needham-Schroeder, verifikace protokolů

Abstract

This diploma thesis is focused on the analysis of security tools, which are commonly used for automatic verification of security protocols mainly. These security protocols are used for creation of private communication channells in insecured networks. Security will never be perfect, so finding of weaknesses in security protocols is always necessary and we need to improve these protocols. In this diploma thesis I have focused on looking for various verification tools on Internet. Subsequently, I have described each tool in this thesis, in way that each reader can easily find out, which tool is for him useful and which is not. I have summarized main benefits and drawbacks of each tool at the end of his description.

Keywords

Tools, protocol, security, Otway-Rees, Yahalom, Needham-Schroeder, protocol verification

Citace

Ptáček Michal: Specifikační jazyky a nástroje pro analýzu a verifikaci bezpečnostních protokolů, diplomová práce, Brno, FIT VUT v Brně, 2007

Specifikační jazyky a nástroje pro analýzu a verifikaci bezpečnostních protokolů

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně. Při práci jsem vycházel z pramenů uvedených v seznamu použitých zdrojů a z konzultací s vedoucím diplomové práce. Tímto dávám Fakultě informačních technologií Vysokého učení technického v Brně právo používat, upravovat, publikovat a distribuovat text této práce, nebo jejich částí při zachování autorských práv.

……
Michal Ptáček
Datum

Poděkování

Děkuji vedoucímu diplomové práce, panu Ing. Pavlu Očenáškovi, za cenné rady a podněty poskytnuté při vytváření této práce. Rovněž bych chtěl poděkovat svým rodičům, kteří mě nejen finančně podporovali po celou dobu mého studia.

© Michal Ptáček, 2007.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Při tvorbě prvních počítačových sítí nehrála bezpečnost téměř žádnou roli. První testovací síť byla instalována počátkem roku 1968 v Národní výzkumné laboratoři ve Velké Británii. Tato síť však neopustila hranice jedné budovy. Požadavek na vybudování podobné sítě a zároveň i potřebné finanční prostředky přišly z resortu obrany, konkrétně od grantové agentury ministerstva obrany *USA* s názvem *ARPA*[1]. Podle této grantové agentury byla experimentální síť, která vznikla v roce 1969 také pojmenována jako *ARPANET* [1]. Ten se stal předchůdcem dnešního Internetu. V této krásné době počítačových průkopníků byl hlavní důraz kladen na spolehlivost a na funkčnost. Vyplývá to i z vojenské podstaty celého projektu. Počítačů připojených na Internet bylo v roce 1984 jen tisíc a lidé spoléhali hlavně na vzájemnou kolegialitu a výpomoc při sdílení informací. Ani v několika dalších letech se počet připojených počítačů příliš nezvýšil, nicméně v roce 1992 bylo již připojených počítačů milión. Nastala nová doba v dějinách lidstva - doba Internetu. A v ní hraje bezpečnost stále významnější roli.

Pryč jsou časy, kdy Internet sloužil pro komunikaci několika nadšenců či vědců, kteří touto cestu rychle a levně mezi sebou komunikovali. Dnes je Internet svět ve světě, kde lidé tráví svůj čas, pracují, relaxují nebo hledají informace. Kdyby byli všichni lidé čestní a spravedliví, tak by zabezpečení nebylo vůbec třeba. Skutečnost dnešních dnů je ale taková, že internetové aplikace na celém světě jsou denně napadány hackery[2], kteří mění internetové stránky, data apod., ať už s cílem získání vlastního užitku nebo jen pro své pobavení. Je na místě otevřeně říci, že žádný server není 100 % bezpečný, přestože napadání serverů a internetových aplikací na nich umístěných se netýká jen malých firem a na zabezpečení jsou vynakládány celosvětově velké částky. Provozovatelé serverů se přirozeně snaží maximálně zabezpečit své počítače proti pirátským útokům a pro hackery je otázkou osobní prestiže i tyto maximálně zabezpečené servery napadnout. Boj proti hackerům se stal každodenní záležitostí a požadavky na zabezpečení elektronické komunikace se neustále zvyšují. Původní sada protokolů není dostačující, protože v různých prostředních se hodí používat odlišné protokoly. Počet prostředí a uplatnění bezpečnostních protokolů roste s rostoucí důvěrou lidí v tento druh komunikace. Vyvíjejí se a implementují nové, bezpečné modely protokolů. Důležitou součástí vývoje bezpečnostních protokolů je jejich formální verifikace. Jedná se vlastně o analýzu protokolu a jeho pravidel po formální stránce, kdy se zjišťuje, zda lze protokol prolomit a napadnout. Pokud by se například podařilo prolomit protokol zabezpečující online transakce na Internetu, mohlo by to vyústit nejen v obrovské finanční ztráty pro oběti takového útoku, ale i ztrátu důvěry v celé elektronické bankovnictví. V této diplomové práci se proto zaměřím na hledání nástrojů, které by měly zautomatizovat a zjednodušit formální analýzu bezpečnostních protokolů.

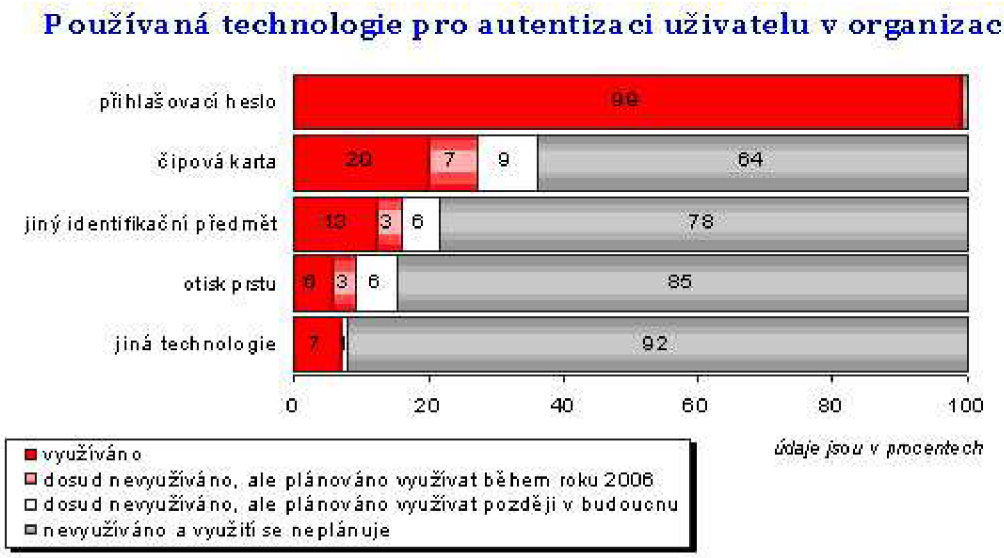
[1] Advanced Research Projects Agency
[2] Počítačový pirát

2 Základní pojmy

Před popisem vlastních protokolů je nutné stručně vymezit pojmy, které se budou v této práci vyskytovat. Pro snazší orientaci jsou pojmy abecedně seřazeny.

2.1 Autentizace

Autentizace je proces, při kterém se ověřuje, zda je uživatel nebo entita opravdu ten, za koho se vydává. Pokud je v komunikačním protokolu zařazen kvalitní autentizační mechanismus, je pro útočníka velmi obtížné podvrhnout zprávu s falešným původem. Druhy autentizace používané v současnosti v prostředí českých firem uvádí následující graf [2].



Obr. 2.1: Používané technologie pro autentizaci

2.2 Bezpečnostní protokol

Bezpečnost v počítačových sítích by se neobešla bez bezpečnostních protokolů. Jedná se o protokoly, které vykonávají některou z bezpečnostních funkcí a používají kryptografické metody. Bezpečnostní protokoly jako takové rozdělujeme do dvou hlavních kategorií na autentizační protokoly a protokoly pro výměnu klíčů. Většina protokolů však plní obě funkce současně.

2.2.1 Autentizační protokol

Autentizační protokol je dohodnutá sada pravidel, podle kterých účastníci komunikace zasílají zprávy tak, aby prokázali svoji identitu.

2.2.2 Protokol pro výměnu klíčů

Pokud některé dva subjekty chtějí navázat zabezpečenou (šifrovanou) komunikaci, je potřeba nějakým bezpečným způsobem zajistit, aby oba subjekty měly šifrovací klíče, pomocí nichž mohou zprávy pro druhý subjekt šifrovat a příchozí zprávy dešifrovat. K tomuto účelu slouží protokoly pro výměnu klíčů.

2.3 Bezpečnostní požadavky

Systém je bezpečný, pokud zachovává tzv. bezpečnostní požadavky. Ty lze rozdělit na čtyři základní typy [3].

2.3.1 Důvěrnost

Služby a informace jsou dostupné pouze oprávněným uživatelům. Nejčastějším prostředkem pro dosažení důvěrnosti je kryptografie[3] (další možností je např. steganografie).

2.3.2 Integrita

Integrita je jednou z nejdůležitějších vlastností pro přenos dat. Jde o stav, kdy přečtená data jsou totožná s daty uloženými, tedy při přenosu dat nedošlo k jejich neočekávaným změnám – poškození, chybám přenosu nebo záměrnému pozměnění. Integritu zajišťujeme pomocí kontrolních součtů, hashovacích funkcí nebo samoopravných kódů.

2.3.3 Dostupnost

Dostupnost znamená, že informace jsou oprávněným uživatelům dostupné vždy v případě potřeby.

2.3.4 Nepopiratelnost

Posledním základním bezpečnostním požadavkem je nepopiratelnost. Ta garantuje, že odesílatel zprávy nemůže později popřít, že zprávu odeslal a příjemce zprávy nemůže popřít, že ji přijal.

[3] Tento pojem bude objasněn na následující straně.

2.4 Kryptografie

Kryptografie neboli šifrování je nauka o metodách utajování smyslu zpráv převodem do podoby, která je čitelná jen se speciální znalostí. Slovo kryptografie pochází z řečtiny – *kryptós* je skrytý a *gráphein* znamená psát [4]. Někdy je tento pojem obecněji používán pro vědu o čemkoliv spojeném se šiframi jako alternativa k pojmu kryptologie. Kryptologie zahrnuje kryptografii a kryptoanalýzu, neboli luštění zašifrovaných zpráv. V praxi se používají 2 typy šifer. Symetrická šifra je taková, která pro šifrování i dešifrování používá tentýž klíč a asymetrická šifra používá veřejný klíč pro šifrování a soukromý klíč pro dešifrování. Asymetrická kryptografie má oproti klasické symetrické velkou výhodu v tom, že není nutné domlouvat tajný sdílený klíč před zahájením utajené komunikace. Na druhou stranu však hrozí podvržení veřejného klíče (lze volně šířit veřejné klíče s nepravou identitou). Tomuto zamezují certifikační autority, které potvrzují pravost veřejných klíčů komunikujících subjektů.

2.5 Nonce

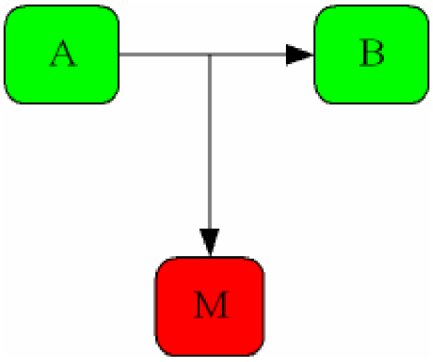
Nonce je náhodně vygenerované číslo, jehož účelem je zahrnout do komunikace informaci, která umožní zabránit útokům založeným na přehrání zprávy tím, že zachovává "čerstvost". Pojem nonce se v češtině nejčastěji překládá jako "keksík".

2.6 Útoky na protokoly

Použití bezpečnostních protokolů samo o sobě nezaručuje absolutní bezpečnost komunikace. Praxe ukázala, že mnoho bezpečnostních protokolů obsahuje chyby, kterých může využít útočník za účelem získání tajných informací. Pokud je při běhu protokolu porušen alespoň jeden výše uvedený požadavek na bezpečnost, pak tuto akci považujeme za útok na bezpečnostní protokol. Útoky je možné klasifikovat do různých kategorií podle různých hledisek [5].

2.6.1 Odposlouchávání

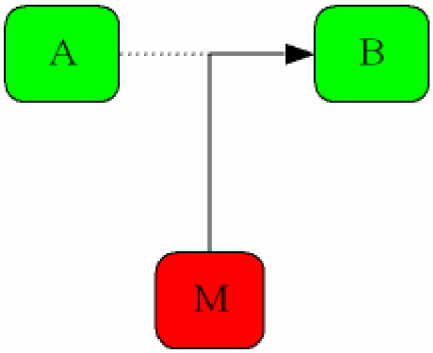
Komunikace mezi legitimními účastníky komunikace *A* a *B* je odposlouchávána pasivním útočníkem *M*.



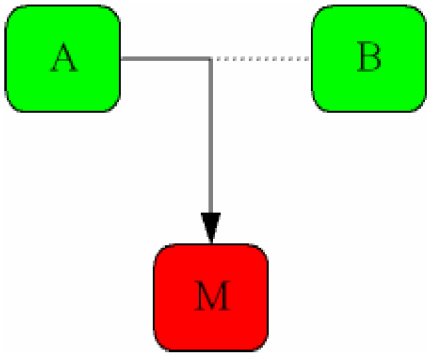
Obr. 2.2: Útok - Odposlouchávání

2.6.2 Podvržení identity

Útočník *M* se může pokusit o vytvoření zprávy s falešnou identitou předstírajíc, že je *A* (Obr. 2.3). Nebo se může pokusit předstírat, že je *B*, který obdržel zprávu od *A* (Obr. 2.4). Zde již hraje útočník během komunikace aktivní roli, což je podstatně nebezpečnější než v předchozím případě.



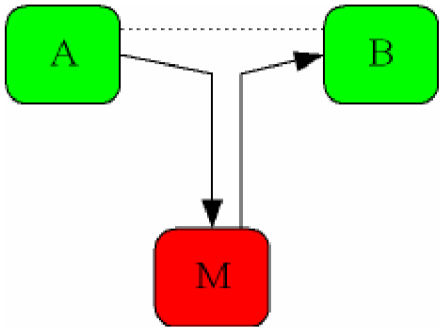
Obr. 2.3: Útok - Podvržení identity (a)



Obr. 2.4: Útok - Podvržení identity (b)

2.6.3 Modifikace zprávy

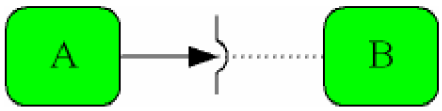
Útočník odposlechne zprávu od *A*, modifikuje ji a přepošle ji *B*. Účastníci komunikace *A* i *B* si myslí, že komunikují přímo jeden s druhým.



Obr. 2.5: Útok – Modifikace zprávy

2.6.4 Přerušení komunikace

Útočníkovi se podaří zničit nebo znepřístupnit přenášenou zprávu.



Obr. 2.6: Útok – Přerušení komunikace

2.6.5 Útok přehráváním

Jedná se asi o nejběžnější typ útoku na bezpečnostní protokoly. Je založen na odposlouchávání a ukládání komunikace. Takto získaná data mohou být použita pro podvrhnutí identity útočníka vůči některému ze subjektů. Vzdálený subjekt totiž nemá žádnou možnost ověřit aktuálnost zprávy (freshness). Proti tomuto se bojuje vkládáním nonce nebo časových razítek[4] do zprávy.

2.6.6 Útok ze středu

Jedná se o útok, kdy se aktivní útočník postaví do středu komunikace mezi dva komunikující subjekty. Následně naváže s oběma komunikaci, přičemž se vždy vydává za regulérního partnera. S využitím informací odposlechnutých od druhé strany se útočník jeví pro *A* jako *B* a naopak.

[4] Časové razítko obsahuje především aktuální datum a čas, sériové číslo razítka a identifikaci autority, která toto razítko vydává. Podrobnosti specifikuje RFC 3161.

3 Bezpečnostní protokoly

V této kapitole budou prezentovány základní bezpečnostní protokoly [6]. Protokoly *Otway-Rees*, *Wide Mouth Frog* a *Yahalom* vznikly aplikací symetrických šifrovacích principů. To znamená, že pro zašifrování a dešifrování zpráv používají stejný klíč. Před odesláním je zpráva zašifrována pomocí klíče, který znají odesilatel i příjemce zprávy. Při uvedené komunikaci se často vyskytuje třetí důvěryhodná strana, která je v obrázcích zaznačena jako *S* (server). Tato strana může sloužit buď jako *KDC*[5] (kap.3.1), která zajišťuje bezpečnou distribuci klíčů mezi subjekty, nebo jako *KTC*[6] (kap.3.3), která klíče nedistribuuje, ale jen mění kódování zpráv.

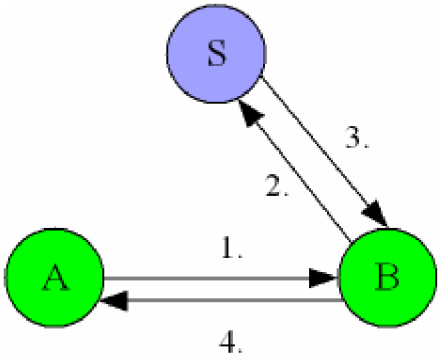
Alternativou k symetrickým protokolům jsou asymetrické protokoly, které jsou založeny na šifrovacích algoritmech používajících dvojici veřejný a privátní klíč. Privátní klíč reprezentuje identitu jeho vlastníka, veřejný klíč je volně k dispozici všem ostatním subjektům, kteří se na komunikaci podílejí. Aplikací principů asymetrické kryptografie, vznikl známý autentizační protokol *Needham-Schroeder* (kap.3.2).

3.1 Otway-Rees protokol

Byl publikován v roce 1987 a jeho autory jsou pánové Otway a Rees. Protokol *Otway-Rees* je autentizační protokol navržený pro použití v nezabezpečených sítích jako je Internet. Umožňuje tak jednotlivcům ověřit si navzájem identitu, zabraňuje odposlechu a některým druhům útoků, jako jsou útoky přehrátím nebo útok modifikací zprávy.

Protokol může být formálně popsán následující sekvencí pravidel, kde *A* se autentizuje vůči *B* s použitím serveru *S*, který figuruje jako *KDC*.

1. $A \rightarrow B : N_A, A, B, \{ N_A, A, B\}_{Kas}$
2. $B \rightarrow S : N_A , A, B, \{ N_A, A, B\}_{Kas} , N_B, \{ N_B, A, B\}_{Kbs}$
3. $S \rightarrow B : N_A , \{ N_A, K_{AB} \}_{Kas}, \{ N_B, K_{AB}\}_{Kbs}$
4. $B \rightarrow A : N_A , \{ N_A, K_{AB} \}_{Kas}$



Obr. 3.1: Otway-Rees protokol

[5] Key Distribution Center
[6] Key Translation Center

Popis jednotlivých kroků:

1. *A* kontaktuje *B* a vytvoří nonce N_A pro identifikaci spojení.

2. *B* kontaktuje *S* a vytvoří nonce N_B. Nemůže ale přečíst zprávu $\{ M, A, B, NA \}_{Kas}$, protože je zašifrována klíčem uživatele *A*, takže ji pouze přepošle *S* spolu se zprávou $\{M, A, B, NB\}_{Kbs}$ zašifrovanou svým klíčem.

3. Server zná všechny klíče zúčastněných stran, proto může obě zprávy dekódovat a vytvoří nový relační klíč K_{AB}, který slouží jako certifikát pro komunikaci mezi *A* a *B*.

4. *B* převezme svůj certifikát a zkontroluje zda přijatá nonce N_B je stejná jako ta, co odeslal v kroku 2. Pokud toto souhlasí, pak přepošle druhý certifikát subjektu *A*, který podobně přijatý certifikát ověří.

3.2 Needham-Schroeder protokol

Termín *Needham-Schroeder protokol* značí dva rozdílné protokoly představené v roce 1978. Jejich autory jsou pánové Roger Needham a Michael Schroeder.

Needham-Schroeder Symmetric Key Protocol je založen na principech symetrické kryptografie. Stal se základem pro dnes široce používaný protokol *Kerberos* a jeho hlavním účelem je vytvoření certifikátu pro vzájemnou komunikaci dvou stran přes nezabezpečenou síť.

Needham-Schroder Public-Key Protocol je založen na principech asymetrické kryptografie a je hlavně oblíbený pro svou jednoduchost. V následujícím textu se zaměřím právě na tuto verzi kvůli srovnání s bezpečností symetrických protokolů.

Protokol může být formálně popsán následující sekvencí pravidel, kde *A* se autentizuje vůči *B* s použitím serveru *S*, který figuruje jako *KDC*.

1. $A \rightarrow S: A, B$

2. $S \rightarrow A: \{ K_{PB}, B\}_{Kss}$

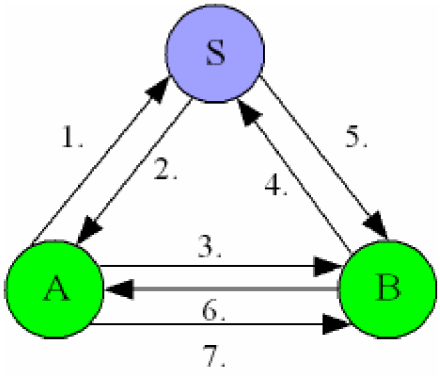
3. $A \rightarrow B: \{ N_A, A\}_{Kpb}$

4. $B \rightarrow S: B, A$

5. $S \rightarrow B: \{ K_{PA}, A \}_{Kss}$

6. $B \rightarrow A: \{ N_A, N_B \}_{Kpa}$

7. $A \rightarrow B: \{ N_B \}_{Kpb}$



Obr. 3.2: Needham-Schroeder protokol

Popis protokolu:

Formální popis je dostatečně intuitivní. Vzhledem k tomu, že se jedná o asymetrický protokol, tak má každá strana k dispozici veřejné klíče všech zúčastněných stran a svůj privátní klíč.

K_{SS} , K_{PS} … privátní klíč *S* a veřejný klíč *S*
K_{AB} , K_{AB} … privátní klíč *A* a veřejný klíč *A*
K_{SB} , K_{PB} … privátní klíč *B* a veřejný klíč *B*
N_A, N_B … nonces vytvořené stranami *A* a *B*

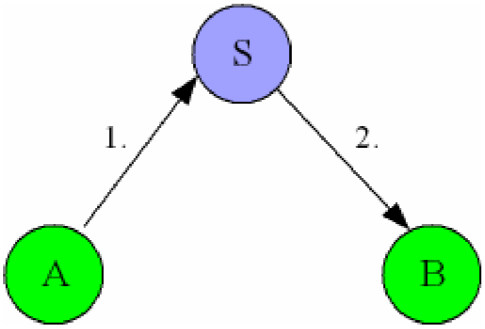
3.3 Wide Mouth Frog protokol

Wide Mouth Frog protokol je asi nejjednodušší autentizační protokol založený na principech symetrické kryptografie používaný pro autentizaci v nezabezpečených sítích. Umožňuje účastníkům komunikace navzájem si ověřit identitu a zároveň poskytuje adekvátní ochranu proti útokům přehráváním a odposlouchávání. Protokol umí rovněž detekovat modifikaci zprávy a zabraňuje neautorizovanému čtení.

1. $A \rightarrow S: A, \{ T_A, B, K_{AB} \}_{Kas}$

2. $S \rightarrow B: \{T_S , A, K_{AB} \}_{Kbs}$

Autentizace *A* vůči *B* proběhla ve dvou krocích pomocí serveru *S*, který funguje jako *KTC*.



Obr. 3.3: Wide Mouth Frog protokol

Protokol má následující slabiny:

- Je nezbytné použít globální synchronizované hodiny.
- Server má k dispozici všechny klíče.
- Hodnota klíče pro komunikaci *A* a *B* je kompletně v režii *A*.

3.4 Yahalom protokol

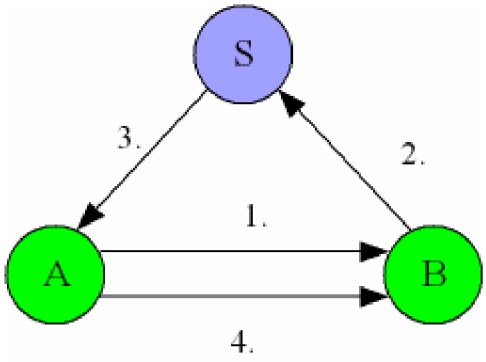
Bezpečná autentizace byla i námětem pro vznik protokolu *Yahalom*, který vznikl v roce 1988. Protokol může být formálně popsán následující sekvencí pravidel, kde *A* se autentizuje vůči *B* s použitím serveru *S*, který figuruje jako *KDC*.

1. $A \rightarrow B : A, N_A$

2. $B \rightarrow S : B, N_B, \{ A, N_A \}_{Kbs}$

3. $S \rightarrow A : \{ B, K_{AB}, N_A , N_B \}_{Kas}, \{ A, K_{AB} \}_{Kbs}$

4. $A \rightarrow B : \{ A, K_{AB} \}_{Kbs} , \{ N_B \}_{Kab}$



Obr. 3.4: Yahalom protokol

Popis:

1. *A* kontaktuje *B* a vytvoří nonce N_A pro identifikaci spojení.

2. *B* kontaktuje *S* a vytvoří nonce N_B. Pošle *S* zprávu $\{ A, N_A \}$ zašifrovanou svým klíčem K_{BS}.

3. Certifikát K_{AB} je vytvořen serverem *S* a poslán přímo účastníkovi *A* (v kroku 3) a nepřímo účastníkovi *B* (v kroku 4). Protokol musí garantovat, že certifikát K_{AB}[7] bude znám jen *A*, *B* a *S*.

4. *A* kontaktuje *B* a pošle mu certifikát K_{AB} a nonce N_B zašifrovaný K_{AB}.

[7] K_{AB} je společný klíč *A* a *B* pro vzájemnou komunikaci většinou v rámci jednoho sezení.

4 Nástroje pro verifikaci protokolů

Tato kapitola tvoří stěžejní část této práce. Poskytuje abecední seznam verifikačních nástrojů, které jsem vyhledal na Internetu. Jedná se převážně o volně dostupné nástroje, které si může potenciální zájemce zdarma stáhnout a vyzkoušet. Na začátku každé kapitoly věnované konkrétnímu nástroji jsem se pokusil strukturovaně popsat základní informace o nástroji a na konci uvádím stručně v bodech nejmarkantnější klady a zápory. Práce je koncipována tak, aby si každý mohl snadno vybrat ten nástroj, který se nejvíce hodí pro jeho konkrétní problém.

4.1 AAPA

Domovská stránka nástroje: nezjištěna
Primární publikace: [8]
Sekundární publikace: [9]
Typ: používá *BGNY* logiku
Dostupnost: nezjištěna
Aktuální verze nástroje: *AAPA2*
Autor: Stephen Brackin

Dr. Stephen Brackin popsal jazyk pro specifikaci rozhraní (*ISL*) i *AAPA*, který se používá k dokazování požadovaných vlastností u protokolů a k popisu jejich případných nedostatků. *AAPA* může automaticky dokázat nebo vyvrátit, zda protokol má požadované vlastnosti resp. přesně najít a identifikovat slabiny. *AAPA* provádí důkazy na základě *BGNY* logiky, která je derivátem logiky *GNY*. Autory *GNY* logiky jsou Gong, Needham a Yahalom [5]. Pro tuto logiku je specifické, že nepočítá s tím, že v zašifrovaných zprávách existuje redundance. Místo toho zavádí nový pojem, „rozpoznatelnost“ (recognizability) pro reprezentaci faktu, že subjekt očekává jistý formát zprávy, kterou přijme. Subjekt zúčastněný v protokolu očekává od zprávy, kterou přijímá, určité vlastnosti. Tato skutečnost je zohledněna v analýze. Pokud tedy krok v protokolu určuje, že *A* přijme nonce N_A a N_B, pak následující dvě hodnoty budou interpretovány jak nonce. Jedním z principů *GNY* logiky jsou pojmy víra[8] (belief) a vlastnictví [9](possession).

[8] Založena na axiomizaci důvěryhodných tvrzení
[9] Tajemství a klíče, které subjekt zná.

V této rozšířené logice má v každém kroku verifikace každý subjekt dvě množiny:

- **Množinu důvěryhodných tvrzení** (belief set), která určuje kterým tvrzením subjekt věří (například, že zakódovaná zpráva nelze dekódovat bez znalosti klíče).
- **Množinu vlastnictví** (possesion set), která určuje klíče a tajemství držené subjektem. Příkladem je analýza *TMN* protokolu pomocí *AAPA*, kterou najdete v dokumentu [8].

Vlastnosti protokolu jsou popsány pokročilým systémem stavů, kde nově přibyla možnost popsat a analyzovat protokoly, které používají vícenásobné kódování dat a heše jako klíče. Pomocí *BGNY* logiky lze popsat i protokoly pro výměnu klíčů.

AAPA může být použit samostatně nebo v kombinaci s nějakým grafickým rozhraním usnadňujícím uživateli práci. *AAPA* byl použit při verifikaci bezpečnostích protokolů *Needham-Schroder*, *Wide Mouth Frog* [69], a dalších.

Výhody:

- ✓ Nespornou výhodou tohoto nástroje je fakt, že časová i prostorová náročnost při analýze protokolů roste kvadraticky s velikostí protokolů. To činí z *AAPA* vhodný nástroj i pro analýzu velkých a robustních protokolů.

Nevýhody:

- ✗ *AAPA* nenajde všechny chyby, nejčastěji nenajde chyby, které nesouvisí se zajištěním důvěrnosti zprávy. Navzdory nenalezení chyb v některých protokolech, byly pomocí *AAPA* nalezeny nekonzistence a nejasnosti v dokumentaci některých protokolů.
- ✗ Další nevýhodou *AAPA* je nemožnost zkonstruovat útoky na základě objevených zranitelností.
- ✗ *AAPA* nenajde chyby, které umožňují útočníkovi vícenásobné používání protokolu nebo použít například útok typu (man in the middle[10]).

[10] Útok ze středu (kap.2.6.6).

4.2 Artifex

Domovská stránka nástroje: http://www.rsoftdesign.com/products/network_modeling/Artifex/
Primární publikace: [10]
Typ: verifikace modelem
Dostupnost: komerční nástroj
Aktuální verze nástroje: Artifex 4.4-2
Autor: *RSoft Design Group, Inc.*

Artifex je skupina nástrojů, které umožňují uživateli modelovat komplexní systémy. Tyto nástroje podporují veškeré aktivity nutné k iterativnímu modelování systémů. *Artifex* používá Petriho[11] sítě ke grafickému modelování systému na libovolné úrovni. Tyto modely mohou být kompilovány do *C/C++* kódu a následně překompilovány na samostatné aplikace. *Artifex* jako plnohodnotná platforma pro modelování komplexních systémů se skládá z následujících nástrojů:

- ***Artifex.Model*** – tento nástroj modeluje komplexní systém jako skupinu komponent, které spolu komunikují zasíláním zpráv. Graficky jsou tyto komponenty vyjádřeny pomocí jazyka Petriho sítí, která umožňuje vnořování *C/C++* kódu do specifikace systému.
- ***Artifex.Validate*** – zajišťuje kontrolu chování systému. Umožňuje vložení ladících bodů do procesu verifikace, odhalení logických zranitelností, export vzorku dat pro budoucí analýzu,…
- ***Artifex.Measure*** – dává přehled o výsledcích chování systému a jeho výkonnosti. Jedná se o poměrně detailní a flexibilní informace.
- ***Artifex.Report*** – umožňuje vložení komentářů do jakékoliv části specifikace systému. Automaticky generuje online *HTML* dokumentaci pro kontrolu chování systému. Poskytuje velmi vhodný doplněk pro práci uživatele během celého procesu.

Pro tvorbu samostatných aplikací a správu datových struktur jsou určeny následující dva nástroje:

- ***Artifex.Deploy*** – implementuje simulátory a distribuované aplikace. Generuje software bez dalšího úsilí uživatele. Používá *IPC*[12] aparát pro komunikaci s *TCP*, *UDP* nebo *DCOM* procesy.

[11] Petriho síť je matematická reprezentace diskrétních distribuovaných systémů.
[12] IPC – Interprocess communication

- ***Artifex.Data*** – popisuje komplexní datové struktury systému a jejich vztahy pomocí *ERD* diagramů. Definuje atributy pomocí *C/C++* a umožňuje dynamickou manipulaci s daty libovolné části modelu.

Výhody:

- ✓ *Artifex* umožňuje grafické modelování systémů s podporou objektově orientovaných přístupů.
- ✓ Část specifikačních kódů lze psát v známém jazyce *C* nebo *C++*.
- ✓ Nástroj je vhodný pro verifikace během počáteční fáze vývoje nového softwaru, kde podstatně urychluje vývoj a zlepšuje kvalitu vyvíjeného produktu.
- ✓ *Artifex* je velmi cennou pomůckou během iterativního vývoje systému nebo nového protokolu.

Nevýhody:

- ✗ *Artifex* je komerční nástroj.
- ✗ Nástroj není vhodný u konečných fází vyvíjení softwaru, kdy je třeba detailnější a formálnější analýza.

4.3 Astral

Domovská stránka nástroje: http://www.cs.ucsb.edu/~rsg/projects/astral/index.html
Primární publikace: [11]
Sekundární publikace: [12]
Typ: verifikace modelem
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: Astral SDE beta verze 1.0 (na CD)
Autoři: Dr. Richard Kemmerer, ...

Astral je formální specifikační jazyk určený speciálně pro nekonečně stavové systémy pracující v reálném čase. Verifikační model v jazyce *Astral* funguje na principu ověřování plnění požadavků na bezpečnost v pevně stanoveném čase. Nástroj pracující v jazyce *Astral* se jmenuje *Astral symbolic model checker*, který je součástí *Astral Software Development Environment* (SDE).

Systém pracující v reálném čase je modelován kolekcí specifikací procesů a jednou globální specifikací. Každá specifikace procesu se skládá z několika úrovní. Každá úroveň nabízí abstraktní pohled na specifikovaný proces. Tato abstrakce způsobuje, že lze *Astral* použít pro verifikaci velkých, robustních systémů. Formální dokazování se skládá z mezi-úrovňových a intra-úrovňových důkazů.

- **Mezi-úrovňové důkazy** (Inter-level proofs) jsou založeny na dokazování konzistence úrovně i+1 s úrovní i.
- **Intra-úrovňové důkazy** (Intra-level proofs) jsou založeny na dokazování korektnosti úrovně i.

Verifikátor generuje *C++* kód pro každý specifikovaný systém, následně simuluje všechny větve prováděcího stromu a kontroluje, zda jsou splněna časová omezení. Teoreticky neomezený nedeterminismus během provádění translační funkce při verifikaci však může vyústit až v přehnaně dlouhý průběh verifikace. Řadu příkladů specifikací systémů pracujících v reálném čase lze nalézt v sekundární dokumentaci.

Tento nástroj byl použit při analýze celé škály protokolů, například u protokolu *Needham-Schroeder*, *TMN protokolu*, kontrolního systému robotů nebo u zabezpečených telefonních datových přenosů. Zranitelnost v *TMN* neobjevil z důvodu neúměrně dlouhého provádění kódu způsobené extrémně velkým počtem rozsáhlých proměnných specifikujících systém. Výsledky u těchto univerzálních protokolů jsou však jen předběžné a věří se, že pravá síla tohoto nástroje souvisí s použitím na verifikaci *real-time protokolů*[13].

Výhody:

- ✓ *Astral* byl vytvořen speciálně pro analýzu protokolů pracujících v reálném čase.
- ✓ *Astral* poskytuje dostatečnou míru abstrakce, takže lze verifikovat i robustní protokoly

Nevýhody:

- ✗ Nástroj neobjeví všechny chyby[14], takže jím nelze vždy prokázat korektnost protokolu. Hodí se proto během počáteční fáze vývoje protokolu, ale v poslední fázi je třeba použít více formální verifikační techniku.

[13] Protokoly pracující v reálném čase. Časový aspekt je pro jejich verifikaci stěžejní.
[14] Chyby nenajde většinou u robustních problémů s neúměrně dlouhou dobou provádění verifikace.

4.4 Athena

Domovská stránka nástroje: nezjištěna
Primární publikace: [15]
Sekundární publikace: [16]
Typ: využívá verifikaci modelem i dokazování teorému
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: integrována v *SyMP beta 0.3* (na CD)
Autoři: D.Song,…

Athena je efektivní nástroj pro automatickou analýzu bezpečnostních protokolů. Používá logiku schopnou vyjádřit bezpečnostní požadavky na autentizaci, důvěrnost a mnohé další používané například v elektronickém bankovnictví. *Athena* může dokázat korektnost u řady protokolů a nemá žádná omezení ohledně velikosti protokolu. Běžné, často používané bezpečnostní protokoly jako *Needham-Schroeder* dokáže verifikovat ve velmi krátkém čase[15]. Athena používá několik sofistikovaných technik k dosažení takové flexibility a výkonnosti. Nástroj je založen na formalismu prostorového modelu vláken (*SSM*[16]), kde "*Strand*" reprezentuje posloupnost akcí prováděných legitimním účastníkem komunikace nebo útočníkem. Bezpečnostní požadavky jsou specifikované jednoduchou logikou a automatická verifikace využívá výhody metod založených na verifikačním modelu (*model checking*) i na dokazování hypotézy (*theorem proving*).

Na počátku verifikace se ověřuje, zda je požadovaná vlastnost splněna v počátečním stavu. Poté se vytvoří stromová struktura stavů, která se prochází. V případě, že se nalezne uzel, kde neplatí požadovaná vlastnost, tak se vygeneruje příklad úspěšného útoku. V opačném případě je dokázána korektnost. Zásadním rozdílem oproti ostatním nástrojům (nástroje založené na *TBM*[17]) je úplně odlišná reprezentace provádění protokolu. *TBM* je založen na modelování protokolu pomocí asynchronních paralelních procesů resp. agentů, které spolu komunikují zasíláním zpráv. *Athena* používá *SSM* model, který obsahuje kompaktní stavovou strukturu balíčků (*semi-bundles*) a závazných cílů (*goal-bindings*).

Závazný cíl je relace, která se týká přesně těch informací, které stály za původem zprávy. Využitím tohoto přístupu se dramaticky redukuje stavový prostor o řadu zbytečných uzlů a cest. Následující tabulka uvádí počet stavů procházených několika nástroji při analýze *NSL protokolu* [6].

[15] Řádově do několika sekund.
[16] Strand Space Model
[17] Trace Based Model

Murphi	Brutus(1)	Brutus(2)	Athena
1706	1208	146	19
514550	-----	186340	19

Pzn: Brutus(1)…Brutus bez symetrické redukce, Brutus(2)...Brutus s mechanismy symetrické redukce

Jednou z technik omezení stavového prostoru je i použití nedosažitelných hypotéz (*unreachability theorems*). Tyto hypotézy označíme predikátem θ, kde $\theta(S)$ platí, pokud je S nedosažitelný stav. S se tak může bezprostředně odstranit z množiny procházených stavů.

Jak již bylo řečeno, nástroj Athena je implementací *SSM* modelu, která automatizuje verifikační principy, které bylo nutné v *SSM* provádět manuálně.

SSM model je založen na těchto termínech:

- **Akce** – množina akcí, které mohou být provedeny procesy během provádění protokolu
- **Události** – dvojice $< akce, a >$, kde a je argumentem akce. Dá se zjednodušit na + *send,* - *receive.*
- **Strand** – sekvence událostí, které jsou prováděny procesem (agentem) během komunikace. Definují tak jeho roli v systému.
- **Bundle** – specifikuje provádění protokolu s určitým nastavením

Dá se říci, že legitimního provádění protokolu je specifikováno konfiguračním balíkem (*bundle*), kde všechny sekvence událostí (*strands*) jsou vykonány přesně podle plánu. Nástroj *Athena* je implementován v jazyce *SML/NJ*, který je dostupný pro *Windows*[18] i pro *Linux*. *Athena* byla použita při verifikaci více než 30 klasických protokolů, z nichž většina (*Otway Rees, Needham Schroeder, Kerberos protokol*) skončila do 1 sekundy. Při testování 1641 automaticky vygenerovaných protokolů trvala verifikace všech protokolů jen 103,8 sekund (na starém PC Pentium).

Autoři tohoto nástroje rovněž zkoušeli verifikovat i autentizační protokoly založené na bezpečné distribuci klíče. Opět pomocí nástroje pro automatické generování protokolů vygenerovali 11,000 kandidátních protokolů s různými bezpečnostními vlastnostmi. Vlastní analýza všech těchto protokolů skončila za méně než 2 hodiny.

Athena lze rozšířit i o kryptografické hešovací funkce nebo *MAC* funkce.

[18] Nutné použít emulátor Cygwin.

Výhody:

- ✓ Verifikace pomocí nástroje *Athena* je velmi rychlá a efektivní.
- ✓ Nástroj velmi slibným způsobem kombinuje výhody verifikačních metod založených na verifikaci modelem a metod založených na dokazování hypotéz. Úspěšně se tak redukuje procházený konečně stavový prostor.
- ✓ V případě nalezení chyby v návrhu protokolu, dokáže *Athena* automaticky vygenerovat popis úspěšného útoku na protokol nebo navrhnout protiopatření.

Nevýhody:

- ✗ Navzdory velmi slibným výsledkům je vývoj metody prezentované nástrojem *Athena* teprve na začátku a autoři chtějí dospět k ještě lepším výsledkům.

4.5 AVISPA

Domovská stránka nástroje: http://www.avispa-project.org/
Primární publikace: [17]
Typ: používá specifikační jazyk *HLPSL*
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Avispa 1.1* (na CD)
Autoři: A.Armando, Y.Chevalier, D.Basin, J.Cuellar,…

AVISPA je projekt založený evropskou komisí v rámci programu *"Information Society Technologies Programme"*, který vznikl 1.1.2003. V rámci projektu *AVISPA*[19] vznikl stejnojmenný nástroj pro tvorbu a analýzu bezpečnostních protokolů.

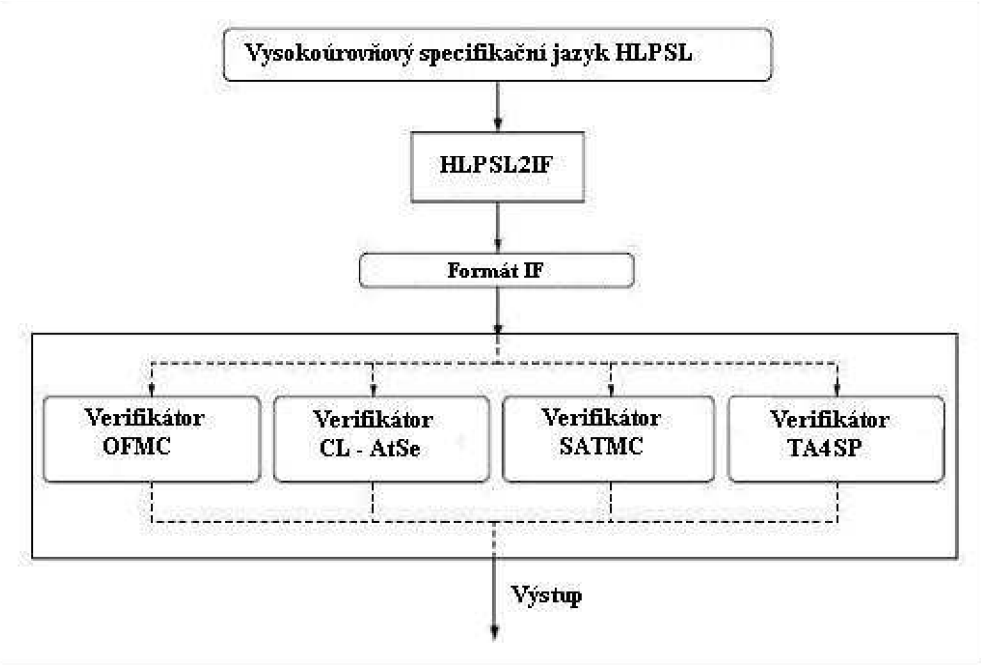
Komunikace uživatele s nástrojem AVISPA funguje na následujícím principu:

Uživatel nejdříve specifikuje analyzovaný protokol v jazyce *HLPSL* a to včetně vlastností, které chce verifikovat. *HLPSL* je formální jazyk s velkou vyjadřovací silou, který umí vyjádřit toky dat, datové struktury, různé kryptografické operátory s jejich algebraickými vlastnostmi, komplexní bezpečnostní požadavky a řadu dalších věcí.

[19] Projekt navázal na úspěchy projektu *AVISS*, který začal v květnu 2001 a trval jeden rok.

Poté, co jsou bezpečnostní požadavky včetně specifikace protokolu napsány v *HLPSL*, tak se spustí vlastní *AVISPA*, která v dalším kroku automaticky spustí *HLPSL2IF* kompilátor, který přetransformuje specifikaci v *HLPSL* jazyce do formálního jazyka *IF* (*intermediate format*).

Specifikace v *IF* je dána na vstup čtyřem různým analyzátorům. Tyto analyzátory pak vypíší výsledek v čitelné formě. Přehledně tuto strukturu popisuje následující obrázek.



Obr. 4.1: Architektura *AVISPA*

AVISPA je díky *HLPSL* silným nástrojem pro specifikaci a verifikaci protokolů. Integrace 4 různých verifikátorů (*back-ends*) dosahuje slibných výsledků.

AVISPA našla nové chyby v protokolech rodiny *ISO-PK*, u protokolů *IKEv2, SET, ASW, H.530* a řadě dalších. Transformace uživatelsky příjemnější *HLPSL* specifikace do nízko-úrovňového *IF* kódu je nutná, protože se díky ní získá nekonečně stavový systém, jehož přechody mezi stavy jsou formulovány prostřednictvím přepisovacích pravidel. Bezpečnostní požadavek je specifikován termínem *"goal"* a dosažené stavy jsou testovány, zda nesplňují tento cíl. V případě, že je takový stav nalezen, tak je označen za stav útoku *(attack state*) a cesta mezi počátečním uzlem a tímto stavem pak popisuje úspěšný útok, který porušil bezpečnostní požadavek specifikovaný v zadání. *AVISPA* byla testována na hledání problémů tří následujících typů:

- Specifický útok s omezeným počtem sezení (*protocol sessions*).
- Předem nespecifikovaný útok s omezeným počtem sezení.
- Specifický útok na s neomezeným počtem sezení[20].

  (výsledky lze nalézt v tabulce [17])

[20] Dosud nebyl dokončen vývoj v této oblasti.

Výhody:

- ✓ Integrace 4 různých analyzátorů způsobuje nalezení většiny chyb. Byli touto cestou nalezeny i dříve neobjevené problémy u známých bezpečnostních protokolů.
- ✓ Většina problémů je nalezena poměrně rychle. Transformace *HLPSL* do *IF* je provedena řádově v milisekundách.

Nevýhody:

- ✗ Používání tohoto nástroje vyžaduje detailní znalost analyzovaných protokolů.
- ✗ Chyby v *HLPSL* specifikaci můžou znemožnit vlastní analýzu nebo způsobit chyby ve výsledcích.
- ✗ Nástroj má celkem náročné ovládání, protože vyžaduje po uživateli, aby se naučil nový programovací jazyk.
- ✗ Analýza výsledků včetně vystopování útoků je někdy časově náročná.

4.6 BRUTUS

Domovská stránka nástroje: nezjištěna
Primární publikace: [18]
Typ: verifikace modelem
Dostupnost: nezjištěna
Aktuální verze nástroje: nezjištěna
Autoři: E.M.Clarke, S.Jha, W. Marrero

BRUTUS je nástroj pro verifikaci vlastností bezpečnostních protokolů. Je plně automatický a pokud objeví během verifikace nějakou zranitelnost, tak nabídne uživateli protiopatření nebo ukáže, jakým způsobem může útočník provést útok. Toto je neocenitelné pro vývojáře opravující protokoly. Modelování systému je opět založeno na agentech[21], kteří spolu komunikují prostřednictvím zpráv.

[21] Typicky účastník komunikace.

Zprávy se skládají z atomických zpráv, které mohou být těchto čtyř typů:

- **Klíče** – používají se pro kódování zpráv
- **Jména** – jména účastníků komunikace
- **Nonces** – náhodně vygenerovaná čísla (každá zpráva s noncem má záruku jisté čerstvosti)
- **Data** – nehrají žádnou roli v protokolech, ale jsou předmětem komunikace mezi účastníky

Se zprávami lze provádět řada operací. Například derivace[22], spojování, projekce, kódování nebo dekódování. Systém se modeluje jako nezabezpečené médium, kde každá zpráva může být modifikována nebo zaslána přímo útočníkem. Kvůli zajištění konečnosti verifikačního modelu je počet pokusů o komunikaci protokolem omezen[23].

V každém sezení hraje agent určitou roli, která je prezentována pětic*í* < *N, S, I, B, P* > :

- ***N***...jméno agenta
- ***S***…unikátní *ID* agenta,
- ***I***…množina zpráv, které jsou známy agentovi
- ***B*** : *vars(N)→M* množina proměnných používaná agentem *N*, které jsou svázány s jeho rolí v systému
- ***P***…popis procesu, tj. posloupnost akcí, které mají být agentem provedeny

Útočník je modelován podobným způsobem, ale nemusí striktně dodržovat protokol, proto pro něj neplatí některá omezení. Komunikace pak probíhá pomocí akcí *"send"* a *"receive"*, které implikují změny stavu s systému. Na následujícím zdrojovém kódu je část specifikace modelu protokolu *Needham-Schroeder* v jazyce *BRUTUS*. Zpráva *"internal"* označuje místa v protokolu, kdy agent začal a kdy skončil komunikaci se subjektem označeným v druhém parametru zprávy.

```
INITIATOR =
choose (b)
internal ("begin-initiate",b)
send (a, b, {na, a}Kb)
receive (a, b, {na, nb}Ka)
send (a, b, {nb}Kb)
internal ("end-initiate",b)
```

```
RESPONDER=
receive (a,b,{na, a}Kb)
internal ("begin-respond",a)
send (a, b, {na, nb}Ka)
receive (a, b, {nb}Kb)
internal ("end-respond",a)
```

[22] Derivovat znamená transformovat podle nějakého derivačního pravidla.
[23] Takové komunikaci říkáme sezení (*session*).

Kromě protokolu *Needham-Schroeder* (kap.3.2) byl nástroj *Brutus* použit při verifikaci *iKP* protokolů[24], protokolu *Wide Mouth Frog* (kap.3.3) a řady dalších.

BRUTUS během verifikace prochází konečně stavový systém pomocí prohledávání do hloubky (*depth-first search*). Pokud narazí na stav, kdy neplatí požadovaná podmínka, tak zastaví prohledávání a začne modelovat protipříklad pomocí deduktivních metod ve speciální logice. Tato kombinace procházení konečně-stavového prostoru a dedukčního přístupu odlišuje *BRUTUS* od ostatních nástrojů, které jsou založeny jen na procházení stavového prostoru.

Výhody a nevýhody nástroje *BRUTUS*:

- ✓ Procházení konečně stavového prostoru a generování protipříkladu probíhá zcela automaticky.
- ✗ Vydedukované zprávy, které používá útočník musí být napevno zakomponovány do systému (u přístupu založeném na dokazování hypotéz stačí přidat pár pravidel do logiky systému).
- ✗ Počet stavů roste exponenciálně s počtem agentů.

Porovnání s *FDR* (kap.4.10):

- ✓ Vzhledem k tomu, že *BRUTUS* popisuje protokol pomocí atomických formulí a přepisovacích pravidel, můžeme takto implicitně reprezentovat potenciálně nekonečnou množinu formulí. Nemusíme tedy uměle omezovat například počet slov, které se útočník může dozvědět. Rovněž přístup k dedukování protipříkladu je podle autorů přirozenější a intuitivnější.
- ✗ *FDR* může na rozdíl od nástroje *BRUTUS* použít některý z existujících modelů a pokud s ním má vývojář zkušenosti, tak může snadno a rychle získat výsledky

Porovnání s *Murphi* (kap.4.19):

- Podobně jako s *FDR*. Díky nástroji *CASPER* je však modelování útočníka snazší u *FDR* než u *Murphi*.

Porovnání s *NPA* (kap.4.20):

- ✓ *NPA* vyžaduje interakci s uživatelem, a protože je založena na přístupu dokazování hypotézy, tak má tato technika problém nalézt protipříklad.
- ✗ Korektnost protokolu je dokazována v *NPA* bez omezení na počet aktérů apod.

[24] Protokoly pro zabezpečení elektronických plateb přes Internet.

Porovnání s *Isabelle* (kap.4.13)

- Podobně jako u *NPA*. *Isabelle* je kompletně založena na verifikaci dokazováním hypotézy.

Porovnání s *Athena* (kap.4.4)

- ✓ Autoři nástroje *BRUTUS* věří, že jejich přístup je více intuitivnější, přirozenější a že má větší vyjadřovací schopnost.
- ✗ V některých případech je modelování času a stavového prostoru v nástroji *Athena* efektivnější. *Athena* má vlastnosti, které nemá žádný z nástrojů popsaných v této práci, proto se některé myšlenky nově implementují i do nástroje *BRUTUS*.

4.7 CAPSL

Domovská stránka nástroje: http://www.csl.sri.com/users/millen/capsl/
Primární publikace: [19]
Sekundární publikace: [20],[21],[22]
Typ: verifikační model stavovými automaty
Dostupnost: nezjištěna
Aktuální verze nástroje: nezjištěna
Autoři: Jonathan Millen, Stephen Brackin, Catherine Meadows

CAPSL je formální jazyk pro vyjádření autentizačních protokolů a protokolů pro bezpečnou výměnu klíčů. Jeho cílem je vyjádřit abstraktní prvky těchto protokolů a podpořit následnou analýzu chyb v protokolech. Lze jej použít i jako rozhraní podporující širokou škálu různých technik pro analýzu bezpečnostních protokolů.

První verze *CAPSL* se objevila v roce 1996 a jeho hlavní myšlenkou je mít jeden společný jazyk pro specifikace, který bude sloužit jako vstup pro jakoukoliv formální analýzu. První koncepce *CAPSL* převzala hodně ze svých principů z jazyka *ISL*. Autorem *CAPSL* je Jonathan Millen, který pracuje jako vědec v *SRI International*[25].

[25] Nezávislá nezisková vědecká organizace, která provádí výzkum a vývoj nových *IT* řešení.

Specifikace CAPSL se skládá ze tří specifických modulů:

- *typespec* – axiomaticky deklaruje kryptografické operátory a funkce. Následující zdrojový kód uvádí příklad specifikace datového typu boolean.

```
TYPESPEC BASIC;
TYPES
    Role, Spec, Agent: Object;
    Tspec, Pspec, Espec: Spec;
END;
TYPESPEC BOOLEAN;
IMPORTS BASIC;
TYPES
    Boolean: Object;
CONSTANTS
    true, false: Boolean;
FUNCTIONS
    and(Boolean, Boolean): Boolean, ASSOC, COMM;
    or(Boolean, Boolean): Boolean, ASSOC, COMM;
    not(Boolean): Boolean;
    if(Boolean, Boolean, Boolean): Boolean;
END;
```

- *protokol* – vlastní popis protokolu. Následuje názorná specifikace protokolu *Needham-Schroeder public key handshake.*

```
PROTOCOL NSPK;        // název protokolu
   VARIABLES          // deklarace proměnných
      A, B: PKUser;
      Na, Nb: Nonce, CRYPTO;
   ASSUMPTIONS        // CRYPTO značí použití kryptografie
      HOLDS A: B;     // předpoklad, že A zná adresu B
   MESSAGES
      A -> B: {A,Na}pk(B);
      B -> A: {Na,Nb}pk(A);
      A -> B: {Nb}pk(B);
   GOALS         // cíle: Na i Nb znají jen Alice a Bob
      SECRET Na; SECRET Nb;
      PRECEDES A: B | Na;        // B musí dostat od A - Na
      PRECEDES B: A | Nb;        // A musí dostat od B - Nb
   END;
```

- *prostředí* (*environment*) – volitelná část, která nastavuje prostředí pro analýzu. Následujícím kódem lze nastavit prostředí pro předcházející implementační příklad.

```
ENVIRONMENT Test1;
   IMPORTS NSPK;
   CONSTANTS
     Alice, Bob: PKUser;
     Mallory: PKUser, EXPOSED;    // Definuje útočníka
   AGENT A1 HOLDS
     A = Alice;
     B = Bob;
   AGENT B1 HOLDS
     B = Bob;
   EXPOSED                        // Popisuje chování útočníka
     {Bob}sk{alice};
END;
```

CAPSL používá jazyk *CIL*, který je blízký konečným automatům a umožňuje jeho použití v různých analytických nástrojích. Například se jazyk *CAPSL* používá jako vstupní jazyk pro *NRL Protocol Analyzer* (kap.4.20).

Výhody:

- ✓ Existuje i verze *MuCAPSL* pro verifikaci vysílacích (*multicast*) protokolů, která umožňuje separaci rolí agentů, tvorbu polí a specifikaci skupin agentů.
- ✓ Výhodou oproti ostatním nástrojům je možnost definovat nové abstraktní datové typy pomocí sekce *typespec*.
- ✓ *CAPSL* je rozhraní podporující širokou škálu různých technik pro analýzu bezpečnostních protokolů

Nevýhody:

- ✗ Jazyk *CAPSL* je stále ve vývoji, ale již teď lze udělat řadu vylepšení jeho efektivnosti. Například lze zavést pojmy *LONGTERM* (deklaruje proměnnou pro více sezení) a *SESSION_SECRET* (od *SECRET* se liší tím, že tajemství nemusí být uchováváno pro příští sezení).
- ✗ Slabá podpora kryptografických operátorů, které jsou nezbytné pro analýzu bezpečnostních protokolů. Lze doplnit uživatelsky pomocí *typespec*.

4.8 CASPER

Domovská stránka nástroje: http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/
Primární publikace: [23]
Typ: kompilátor pro tvorbu *CSP*
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Casper 1.5* (na CD)
Autoři: Tony Hoare, Samson Abramsky, Bill Morton, Nick Trefethen, Richard Brent

V minulých letech byla vyvinuta metoda pro analýzu bezpečnostních protokolů pomocí algebry procesů *CSP*[26] a jejího verifikačního modelu *FDR*. Tato technika byla velmi úspěšná a díky ní byla objevena v protokolech řada nových chyb. Práci stěžovala jen pracnost tvorby *CSP*. To bylo podnětem pro vznik nástroje pro automatickou tvorbu *CSP* popisu protokolů. A tak vznikl *CASPER*, který značně zjednodušuje modelovací a analytické procesy. Nástroj vznikl na *Oxford University* a jedná se o kompilátor, který ze zjednodušeného popisu protokolu vygeneruje *CSP* kód.

Zdrojový kód obsahuje různé sekce potřebné pro popis protokolu – popis rolí agentů, popis výměny zpráv a definici požadavků na protokol, jejichž dodržení je známkou validity protokolu. Překladač pak tento abstraktní zápis převede na *CSP* systém, kde agenti protokolu odpovídají procesům a vzájemná součinnost mezi agenty je reprezentována událostmi zapsanými v *CSP* notaci. Použití *Casperu* má tu výhodu, že se uživatel nemusí zaobírat všemi složitostmi, které popis systému v *CSP* obsahuje, a proto je mnohem více použitelnější a praktičtější.

Vygenerovaný *CSP* kód pak slouží jako vstup pro *FDR*, který ověří, jestli systém popsaný v *CSP* splňuje všechny bezpečnostní požadavky. V případě, že *FDR* nalezne stav, kdy je porušena některá definovaná vlastnost, tak vygeneruje zápis popisující cestu, kterou se k chybě dostal. Touto cestou je v našem případě bezpečnostních protokolů popis, jak provést útok na daný protokol. Zápis cesty k chybě vygenerovaný pomocí *FDR* dokáže *Casper* zpětně transformovat do nám lépe srozumitelného zápisu výměny zpráv mezi agenty. Distribuce *Casper 1.5* se nachází na přiloženém CD[27].

Výhody:

- ✓ Nástroj *Casper* podstatným způsobem usnadní práci úspěšného nástroje *FDR*, protože automaticky vygeneruje *CSP* kód z abstraktního popisu protokolu.

[26] process algebra of Communicating Sequential Processes
[27] Verze nástroje *Casper* na CD obsahuje i nástroj *CasperFDR*, který má příjemnější grafické uživatelské rozhraní.

Nevýhody:

- ✘ Vstupem do nástroje *CASPER* je definice nejen protokolu samotného, ale i celého verifikačního systému. V porovnání například s *CAPSL* je to práce navíc. Tomu stačilo jen definovat protokol. *CASPER* vyžaduje definovat celý systém kvůli vymezení jednotlivých agentů a rolí, které plní. Tato funkcionalita není striktně nevýhodou a navzdory větší pracnosti lze ocenit řadu nově objevených uživatelských chyb, které nástroje typu *CAPSL* nedokáží najít.

4.9 CPN Tools

Domovská stránka nástroje: http://wiki.daimi.au.dk/cpntools/cpntools.wiki
Primární publikace: [24]
Sekundární publikace: [25]
Typ: verifikační model
Dostupnost: existují komerční i nekomerční verze
Aktuální verze nástroje: *CPN Tools 2.2.0*
Autoři: *CPN Group, University of Aarhus*

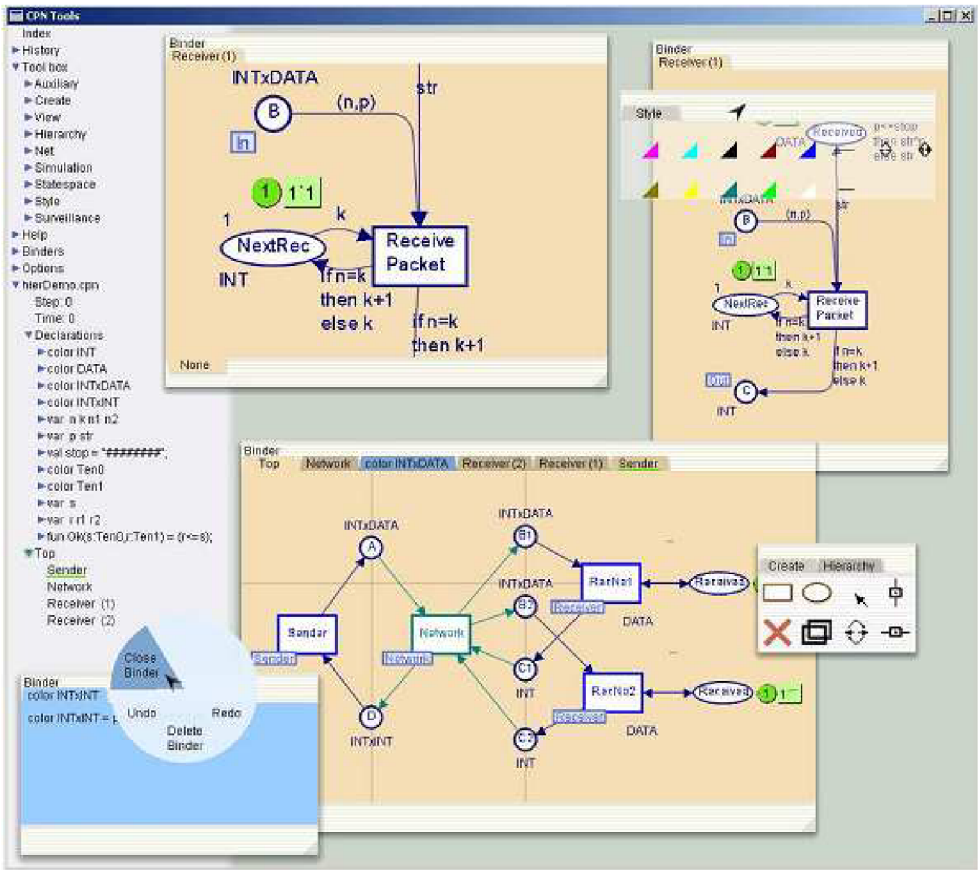
CPN nástroje (*CPN Tools*) jsou nástroje pro úpravu, simulaci a analýzu systémů. Jsou založeny na barevných Petriho sítích (*Coloured Petri Nets*), které jsou vhodným formálním modelovacím jazykem pro popis velkých a komplexních systémů. Jejich klady lze stručně vyjádřit v těchto bodech:

- ✓ mohou tvořit hiearchické modely
- ✓ komplexní informace mohou být reprezentovány tokeny
- ✓ časové informace mohou být obsaženy v modelech
- ✓ existuje celá škála dobře otestovaných *CPN* nástrojů
- ✓ je možné použít stejný (podobný) model pro verifikaci logickou (např. kontrola, zda pakety chodí v daném pořadí), funkcionální (zda protokol dělá co má dělat) i pro analýzu výkonnosti (časové aspekty doručování paketů nebo zda nehrozí zahlcení sítě).

V poslední době byla do nástroje *CPN Tool* přidána následující funkcionalita:

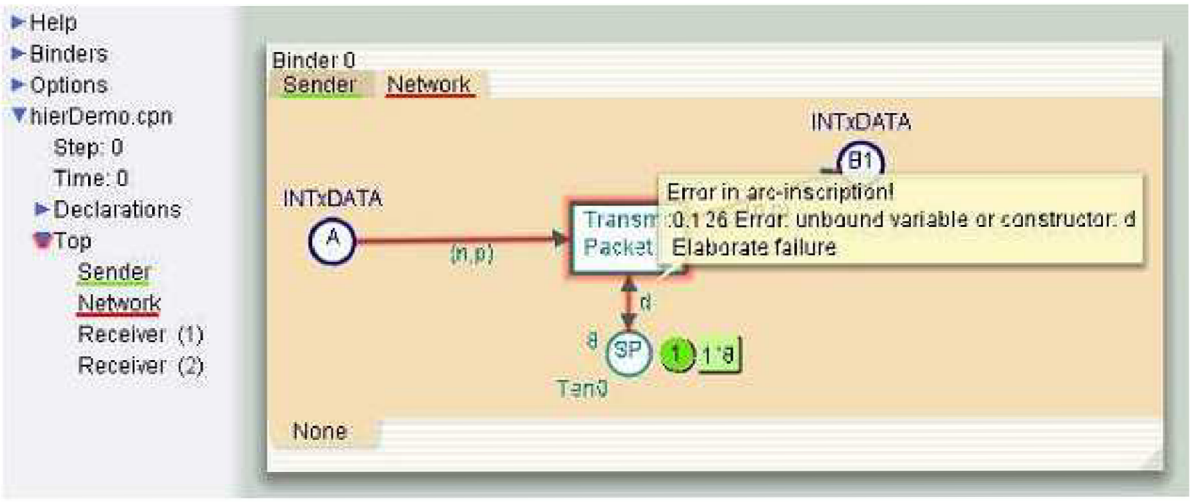
- Data lze měnit s externími procesy přes *TCP/IP*, ale je nezbytné pro přenos upravit model, což může mít závažné důsledky na chování modelu.
- Byly nově přidány monitory, které umožňují monitorovat chování systému, upravovat nebo kontrolovat simulaci bez zásahu do modelu.
- Nově lze vhodně nastavit výstupy (co se má uložit do logu) pro simulaci, které jsou v zápětí uloženy v hierarchické adresářové struktuře

Nástroje *CPN Tools* obsahují kvalitní *GUI*, které je založeno na pokročilých technikách interakce s uživatelem (obr 4.2). Autoři v něm doporučují používat dvě pohybová zařízení současně (myš pro citlivou práci a trackball pro posuny komponent).



Obr. 4.2: Náhled do *CPN Tool GUI*

Systém obsahuje dobrou zpětnou vazbu, která odhaluje kontextové chybové zprávy a indikuje závislostní vztahy mezi elementy sítě. Inkrementální kontrola syntaxe a generování kódu se provádí během sestavování modelu.



Obr. 4.3: CPN Tools - zpětná vazba okamžitě zobrazuje kontextovou bublinu s chybou

Rychlý simulátor efektivně modeluje časově závislé i časově nezávislé sítě. Úplně nebo částečně stavové prostory mohou být vytvořeny a analyzovány pomocí definovaných vlastností či podmínek zajištění živosti.

Výhody:

- ✓ Rychlé a snadno pochopitelné uživatelské rozhraní.
- ✓ Editace a simulace je integrována do jednoho módu.
- ✓ Inkrementální kontrola syntaxe a generování kódu.
- ✓ Rychlý průběh simulace.
- ✓ Možnost vzít zpět jakoukoliv změnu.

Nevýhody:

- ✗ Chybí podpora pro diagramy zpráv, analýzu výkonu, vykreslování stavového prostoru, generování a analýzu redukovaného stavového prostoru, metoda pro mazání stavů během generování stavového prostoru.
- ✗ Pro komerční použití je nutné koupit licenci.

4.10 FDR

Domovská stránka nástroje: http://www.fsel.com/documentation/fdr2/html/index.html
Primární publikace: [26]
Sekundární publikace: [27]
Typ: verifikace pomocí *CSP*
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *FDR2.82* (na CD verze pro Linux)
Autoři: C.A.Hoare, S.D.Brooks, A.W.Roscoe,…

FDR je verifikační nástroj používající model konečných automatů. Základní myšlenka nástroje je založena na principu *CSP (Communicating Sequential Processes)*, kterou vymyslel Hoare v roce 1985. Jeho metoda zkoumá, zda systém splňuje nějakou podmínku pomocí přechodové funkce vhodného automatu. Vstupem do *FDR* jsou dva *CSP* procesy. Jeden proces specifikuje protokol a druhý je jeho implementací. Zkoumá se pak, zda implementace je zjemněním specifikace.

Nová verze nástroje, *FDR2*, umožňuje používat i jiné operátory než je jádro *CSP*, lepší ošetření synchronizace a zkoumání robustních systémů. V publikaci [27] je provedena analýza protokolu *Needham-Schroeder* pomocí *CSP* procesů a následná verifikace s použitím *FDR*. Celá analýza využívá zajímavého poznatku, že pokud existuje útok na libovolně rozsáhlý systém používající určitý protokol, pak lze udělat stejný útok i na podstatně menší systém pouze mezi entitami *A* a *B*.

Výhody *FDR2*:

- ✓ Podpora operátorů mimo *CSP* procesy i pro úplně odlišné jazyky.
- ✓ Vylepšené ošetření synchronizace včetně pravidel pro zapojení komponent během simulace.
- ✓ Zmírnění některých omezení pro *CSP* skripty v porovnání s *FDR*.
- ✓ Vylepšené vyjadřovací schopnosti jazyka pro datové typy a výrazy v porovnání s *FDR*.
- ✓ Potenciál pro pomalou detailní analýzu systému.
- ✓ Iterativní modelování systému, kdy následující iterace by měla být zjednodušeným zápisem předchozí (méně stavů).

Nevýhody *FDR2*:

- ✗ Vyjadřovací síla *CSP* procesů je poměrně malá. Narozdíl od nástrojů typu *SPIN*, kdy lze simulovat i nekonečné chování, jsou výsledky analýzy *CSP* procesů omezeny na existenci konečně dlouhé cesty k výslednému stavu.

- Nutnost vytvoření *CSP* skriptu z definice protokolu, což nemusí být nutně bezchybný proces. Navzdory těmto nevýhodám je však použití *FDR2* stále přínosné a tento nástroj může najít i jinak neobjevené zranitelnosti v protokolech.

4.11 Hytech

Domovská stránka nástroje: http://embedded.eecs.berkeley.edu/research/hytech//

Primární publikace: [28]

Sekundární publikace: [29]

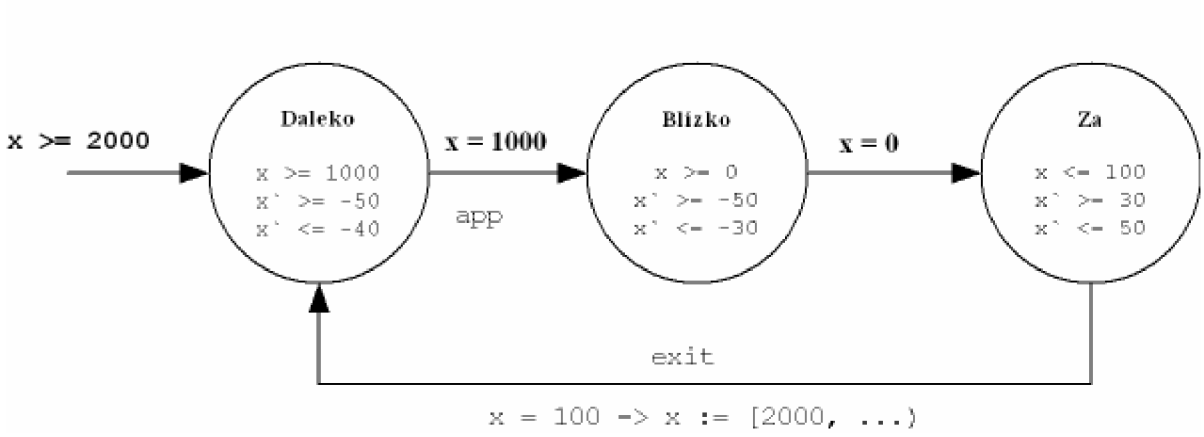
Typ: verifikace hybridními lineárními automaty

Dostupnost: nekomerční nástroj

Aktuální verze nástroje: *Hytech 1.04* (na CD verze pro Linux)

Autoři: Tom Henzinger, Pei-Hsin Ho a Howard Wong-Toi.

HyTech je automatický nástroj pro analýzu *embedded* systémů, který vyčísluje podmínky, za kterých hybridní systém splňuje požadavky na bezpečnost a na čas. Hybridní systém je dynamický systém, jehož chování se přerušovaně i souvisle mění. Hybridní automat je matematickým modelem hybridního systému, jehož přechodová funkce implementuje skokové změny stavů i diferenciální rovnice pro vyjádření spojitých změn stavů. Příklady hybridních automatů si můžete prohlédnout v publikacích [28,29]. Pro ilustraci přikládám obrázek zobrazující hybridní systém pro ovládání závor na železničních přejezdech. Stavy *"daleko"*, *"blízko"* a *"za"* určují polohu vlaku vůči železničnímu přejezdu. Proměnná *x* udává vzdálenost vlaku od železničního přejezdu.



Obr. 4.4: Hybridní automat pro vlaky

Funkcionalita nástroje *HyTech* se dá stručně vyjádřit v těchto bodech:

- *HyTech* vypočítává podmínky, za kterých lineární hybridní systém splňuje specifikované požadavky. Hybridní systém je specifikován množinou hybridních automatů a časové požadavky jsou verifikovány pomocí symbolického modelu.
- *HyTech* je primárně vytvořen pro verifikaci *embedded* systémů, kde vlastní verifikace probíhá na modelu komunikujících konečných hybridních automatů.
- *HyTech* automaticky určí podmínky, za kterých systém splňuje zadané časové a bezpečnostní požadavky. Typickým příkladem je použití *HyTech* při uzavírání závor na železničních přejezdech, kdy se počítá bezpečná vzdálenost od přejezdu, ve které musí vlak poslat signál k uzavření závory (obr. 4.4).
- *HyTech* nabízí diagnostické možnosti analýzy, které usnadňují návrh a debugging protokolů. *HyTech* spočítá nezbytné a dostačující intervaly parametrů tak, aby systém splňoval bezpečnostní požadavky.

Původní verze byla založena na symbolickém algebraickém nástroji *Mathematica*. Nová implementace je napsána v *C++* a používá geometrické algoritmy místo manipulace s formulemi. Nová verze nabízí v porovnání s předcházejícími verzemi větší prostor pro specifikaci vstupu, vyšší přenositelnost a lepší výkonnost.

Vstupem pro verifikaci je textový soubor, který obsahuje specifikaci systému a seznam příkazů pro vlastní analýzu. Sekce příkazů se rozděluje do dvou částí. V první části se deklarují proměnné pro jednotlivé stavy hybridního automatu, zatímco v druhé části je iterativní seznam příkazů pro manipulaci s proměnnými.

HyTech byl použit pro celou řadu kontrolních aplikací včetně distribuovaného ovládání robotů, protokolu pro ovládání audio toku dat firmy *Philips*, ovladačů používaných na železnicích, nelineárního ovladače teploty, přistávacího systému u letadel a dalších. Vždy se však jednalo o aplikace, kde byl hlavní důraz během verifikace kladen na časový aspekt.

Výhody:

- ✓ Hlavní výhodou je možnost zadávat parametry verifikace, kde se poté ověří, zda pro dané parametry splňuje systém časové požadavky.
- ✓ Nástroj je nově podporován skupinou *UPPAAL*, která vyvinula grafické prostředí pro specifikaci systémů.

- ✓ Verifikace systémů pomocí třetí generace nástrojů *HyTech* trvá podstatně kratší dobu, řádově desítky sekund. V případě složitějších parametrických simulací může zpracovávání trvat i několik minut.
- ✓ Nástroj prokázal své dovednosti v řadě komerčních aplikací běžících v reálném čase.

Nevýhody:

- ✗ Nástroj byl vyvinut speciálně pro simulaci protokolů a systémů, které pracují v reálném čase. Požadavky na bezpečnost jsou potlačeny do pozadí.
- ✗ Je nutné specifikovat vstup v co nejjednodušší podobě a s pomocí co nejméně stavů, jinak bude doba provádění simulace podstatně větší.

4.12 Interrogator

Domovská stránka nástroje: nezjištěna
Primární publikace: [30]
Typ: verifikační model na bázi konečných automatů
Dostupnost: nezjištěna
Aktuální verze nástroje: nezjištěna (existují distribuce pro Unix a MacOS)
Autor: Jonathan K. Millen

Jedná se o nástroj napsaný v *Prologu*, který provádí verifikaci na základě simulace protokolu konečným automatem. Nástroj podporuje řadu různých datových transformačních operátorů, které jsou zastoupeny jednoduchým rozhraním. Tento nástroj rovněž podporuje řešení matematických rovnic a řadu aritmetických operátorů jako je například *XOR*, takže je schopen modelovat i kryptografické aspekty u autentizačních protokolů.

První verze nástroje *Interrogator* byla implementována pouze jedním kryptografickým operátorem, který používal podobný popis protokolů jako současné verze. V roce 1987 byl *Interrogator* implementován na stroji *LISP*. Tato verze byla následně přepsána pro platformu *Macintosh*. Všechny tyto verze byly plně automatické a vykazovaly jistou dávku nestability. Jejich úspěch během verifikace závisel na jemných detailech ve specifikaci protokolu. Současná verze má implementace pro *Unix* a pro *Macintosh*. Byla zjednodušena specifikace protokolů a nástroj se stal i se svým novým grafickým uživatelským rozhraním interaktivní.

Účastníci komunikace jsou modelovány složeným konečným automatem, který modeluje komunikaci jednotlivých účastníků pomocí zpráv. Tyto zprávy může útočník modifikovat, zničit nebo jen číst.

Stavy automatu jsou specifikovány trojicí *(N, M, K)* kde:

- ***N:*** $P \rightarrow S$ je funkce zobrazující aktuální stav automatu pro ostatní účastníky komunikace
- ***M:*** přenášená zpráva
- ***K:*** množina datových položek, která je známa útočníkovi

Cílem simulace je najít stav, kdy útočník zjistil nějaké tajemství. *Interrogator* se snaží najít cestu, která vede do takového stavu. Pokud ale cestu nenajde, tak to neznamená, že neexistuje.

Interrogator byl použit při verifikaci mnoha různých protokolů, z nichž většina měla již předem známé slabiny. Efektivně lze použít *Interrogator* například u protokolů *Diffie-Hellman, Needham-Schroeder, Tatebayashi-Matsuzaki-Newman* a dalších.

Výhody:

- ✓ Nespornou výhodou je schopnost nástroje prozkoumat pomocí jednoduchého aparátu široké spektrum možných útoků a v případě provedení úspěšného útoku i podrobněji popsat nalezenou zranitelnost.
- ✓ Automatizace procesu verifikace.

Nevýhody:

- ✗ Nástroj nedokáže namodelovat všechny útoky, lze jím odhalit jen některé. Nástroj vyžaduje znalost chování útočníka k simulování a nalezení případné slabiny v protokolu. Nedá se proto použít při hledání úplně nových zranitelností.
- ✗ Špatné dokazování korektnosti protokolu, pokud se žádný úspěšný útok nenalezne.
- ✗ Je nutné mít precizní specifikaci protokolu na vstupu, jinak jsou výsledky spekulativní a nástroj je nestabilní.

4.13 Isabelle

Domovská stránka nástroje: http://isabelle.in.tum.de/index.html
Primární publikace: [31]
Sekundární publikace: [32]
Typ: induktivní technika
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Isabelle 2005* (na CD je verze pro Linux)
Autoři: Lawrence C. Paulson, Tobias Nipkow

Isabelle je prostředí pro dokazování korektnosti protokolů. Tento nástroj se vyvíjí paralelně na univerzitách v Cambridge a v Mnichově. Za autory projektu *Isabelle* je považována dvojice Prof. Lawrence C. Paulson z *University of Cambridge* a Tobias Nipkow z *Technical University of Munich*.

Isabelle umožňuje matematické formule vyjádřit ve formálním jazyce a následně je dokázat pomocí logického kalkulu. Tento nástroj se používá hlavně při formalizaci matematických důkazů, při semiformální verifikaci korektnosti počítačového hardwaru či softwaru a u dokazování vlastností počítačových jazyků a protokolů. V porovnání s podobnými nástroji je hlavní výhodou *Isabelle* její flexibilita. Většina verifikačních nástrojů je postavena kolem jednoho formálního kalkulu, typicky nějaké logiky vyššího řádu. *Isabelle* má možnost akceptovat různé formální kalkuly. Důkazy mohou být zapsány ve strukturované notaci podle tradičního dokazovacího stylu nebo jako sekvence příkazů. Definice a důkazy mohou být v *TeX*[28] kódu, ze kterého *Isabelle* automaticky vygeneruje dokumentaci.

Isabelle obsahuje velkou knihovnu verifikované matematiky včetně teorie čísel, matematické analýzy, algebry a množinové teorie. Pro snazší orientaci obsahuje i příklady z formální verifikace. *Isabelle* je integrována do uživatelského rozhraní *ProofGeneral*.

V systému Isabelle jsou účastníci komunikace a zprávy modelovány následujícím způsobem:

(i) v systému jsou definovány tři typy účastníků (agentů) : server *S*, běžní účastníci komunikace *(friends)*, kteří jsou indexováni přirozenými čísly a aktivní útočník *(spy)*, který se chová jako běžný účastník komunikace [32].

```
datatype agent = Server | Friend nat | Spy
```

(ii) datový typ *msg* specifikuje typ přenášené zprávy : mohou se přenášet jména agentů, nonces, klíče, složené zprávy a zakódovaná data.

```
datatype msg = Agent agent
| Nonce    nat
| Key      key
| MPair    msg msg
| Crypt    key msg
```

[28] Speciální systém pro sázení textu.

Výhody:

- ✓ Obrovská flexibilita vycházející z podpory široké škály logik.

Nevýhody:

- ✗ Navzdory výhodám indukčního typu dokazování, vyžaduje *Isabelle* poměrně značné úsilí od expertního uživatele pro provedení verifikace. *Isabelle* se snaží snížit úsilí uživatele automatizací některých částí dokazovacího procesu.

4.14 ISL

4.14.1 JML

Domovská stránka nástroje: http://www.cs.iastate.edu/~leavens/JML/

Primární publikace: [33]

Typ: deduktivní

Dostupnost: nekomerční nástroj

Aktuální verze nástroje: *JVM 5.4* (na CD)

Autor: Gary T. Leavens

Projekt *JML* vznikl na *Iowa State Univerzity* pod záštitou oddělení Gary T. Leavense. *JML* je formální jazyk pro specifikaci chování rozhraní (*BISL*) v jazyce *Java*. Umožňuje syntaktickou analýzu nejen zdrojových kódů rozhraní, ale i jejich chování. Kromě vstupních a výstupních podmínek jde do *Java* kódu implementovat výroky, které pomáhají při verifikaci a opravování chyb. *JML* je nástroj speciálně vyvinutý pro *Javu*, jeho ekvivalent pro *C++* je nástroj *Larch*.

4.14.2 Larch/C++

Domovská stránka nástroje: http://www.cs.iastate.edu/~leavens/larchc++.html

Primární publikace: [34]

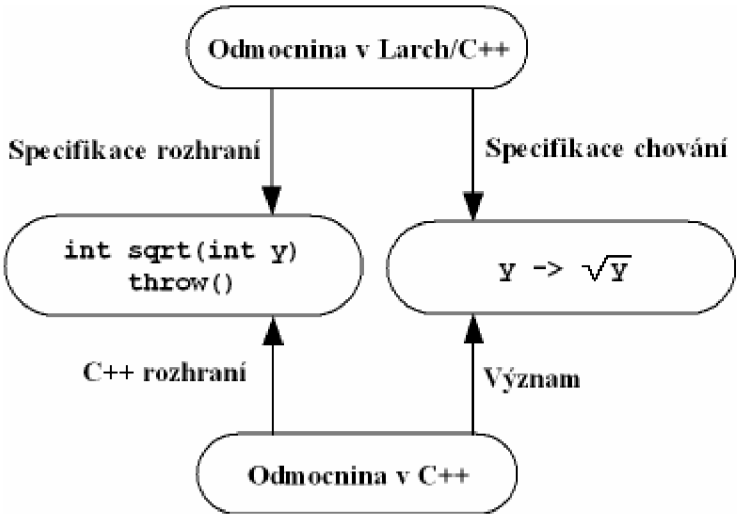
Typ: deduktivní

Dostupnost: nekomerční nástroj

Aktuální verze nástroje: *LCPP 5.14* (na CD)

Autor: Gary T. Leavens

Larch/C++ je obdobný projekt jako *JML*, který vzniká rovněž na *Iowa State Univerzity* pod záštitou oddělení Gary T. Leavense. Je to formální jazyk pro specifikaci chování rozhraní (*BISL*) v jazyce *C++*. Rovněž umožňuje syntaktickou analýzu nejen zdrojových kódů rozhraní, ale i jejich chování. Základní myšlenkou specifikace chování je, že se specifikuje co model dělá a ne jak to dělá. Přehledně to demonstruje obrázek 4.5.



Obr. 4.5: Princip *ISL*

Výhody:

- ✓ Specifikace rozhraní usnadňuje znovupoužívání.
- ✓ Formální specifikace pomáhá analyzovat návrh a může za jistým účelem nahradit zdrojový kód.

Nevýhody:

- ✗ Tyto nástroje neumožňují popsat chování veškerého *Java* resp. *C++* kódu, ale jen jeho částí, které jsou volány z jiných modulů.

4.15 Kronos

Domovská stránka nástroje: http://www-verimag.imag.fr/TEMPORISE/kronos/
Primární publikace: [35]
Sekundární distribuce: [36], [37]
Typ: verifikační model používá *TCTL* časovanou logiku
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Kronos 252* (na CD)
Autoři: Sergio Yovine, Alfredo Olivero, Conrado Daws, Stavros Tripakis

Kronos je nástroj vyvinutý speciálně pro verifikaci robustních systémů pracujících v reálném čase. Komponenty těchto systémů jsou modelovány pomocí tzv. časovaných automatů, požadavky jsou vyjádřeny temporální *TCTL* logikou. Časované automaty jsou automaty, jejichž přechodová funkce je rozšířena o hodiny sloužící k vyjádření časových omezení. Přechod je povolen, jen pokud čas na hodinách odpovídá časovým omezením a provádí se spuštěním nějakého výrazu. Umístění těchto automatů koresponduje s kontrolními body verifikačního procesu. Vlastní systém se skládá z *m* kooperujících procesů, které jsou modelovány paralelní kompozicí *m* časovaných automatů. Podobné hybridní automaty jsou použity u nástroje *HyTech* (kap. 4.11). *Kronos* v sobě implementuje několik různých verifikačních algoritmů, které se dají rozdělit do dvou skupin, na algoritmy prohledávající stavový prostor pozpátku a na algoritmy verifikující systém procházením stromové struktury stavového systému dopředu.

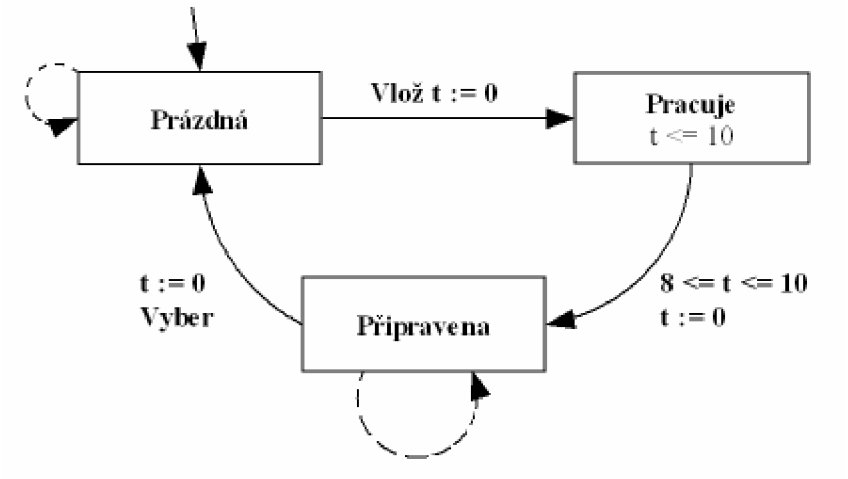
Kronos poskytuje specifikační strukturu, která v sobě integruje logický i behaviorální přístup k verifikaci. V rámci logického přístupu probíhá verifikace dokazováním platnosti formule v *TCTL* logice. V rámci behaviorálního přístupu probíhá verifikace modelováním a zkoumáním chování časovaných automatů.

V případě modelování pomocí behaviorálního přístupu se modeluje časovaný automat, jehož syntaktická podoba je složena z následujících komponent:

- **Stavy** *(locations)* – reprezentují kontrolní body nebo určité stavy systému. Každý stav je specifikován výrazem *loc:n (n...číslo stavu)*
- **Hodiny** *(clocks)* – každý automat obsahuje konečnou množinu proměnných měřících čas. Jedná se o soukromé proměnné každého automatu, které se značí *#clocks<id> (id...identifikátor).*

- **Synchronizační události** *(synchronization events)* – každý automat komunikuje s ostatními automaty pomocí synchronizačních událostí. Synchronizační události jsou označeny *#sync<id>...<id>*.
- **Přechody** *(transitions)* – umožňují změny stavů systému. Přechod představuje označená hrana v grafu, která se specifikuje výrazem:
  <g> → <id>...<id>;<r>,...,<r>; goto n
  <g> je podmínka hlídající přechod. *<id>...<id>* je množina událostí asociovaných s přechodem. *<r>,...<r>* je množina výrazů měnících hodnoty příslušných hodin.
- **Stavový invariant** *(location invariant)* – každý stav je označen výrazem specifikujícím hodiny. Ten je deklarován výrazem *invar: <g>* (*g* má stejný význam jako u přechodů).
- **Booleovské hodnoty** *(boolean proposions)* – každý stav může mít množinu identifikátorů reprezentujících hodnoty booleovských výrazů, které jsou platné v daném stavu.

Na obrázku 4.6 je časovaný automat, který znázorňuje stanici s díly, které robot přesouvá. Na začátku je stanice prázdná a po 8-10 sekundách se dostane do stavu připraveno, což signalizuje přítomnost materiálu k vyzvednutí pro robota. Čas je uložen v proměnné *t*.



Obr. 4.6: Časovaný automat – služební stanice

V případě logického přístupu k verifikaci se ověřuje, zda systém splňuje podmínku specifikovanou v *TCTL* temporální logice.

Ta je specifikována následující gramatikou:

$$\varphi ::= x\#c \mid x - y\#c \mid b \mid enable(l) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists\Diamond_{\#c}\varphi \mid \forall\Diamond_{\#c}\varphi$$

x,y … hodiny, # operátor (<,>,≤,≥,=), c…konstanta, b..booleovský výraz, l…událost

Poté, co je specifikována podmínka v *TCTL* logice, tak je zjišťováno, zda podmínka platí pro všechny stavy některého časovaného automatu.

Nástroj *Kronos* byl použit k verifikaci řady protokolů, z nichž nejznámější jsou například *CSMA/CD*, *FDDI* protokol [36], *Philips audio control protokol* [37], ovládací systém pro roboty a další. Řešení více komplexních systémů vyžaduje použít i pokročilé techniky, které *Kronos* nabízí. Jedná se hlavně o optimalizaci počtu hodin použitých v modelu, použití kompilace za běhu, binární rozhodovací diagramy a další. Všechny tyto techniky mají za cíl hlavně redukovat prostor pro verifikaci a tím snížit velikost potřebné paměti a dobu provádění vlastní verifikace.

Výhody:

- ✓ *Kronos* je rychlejší než ostatní nástroje pracující na principu časovaných automatů (*HyTech*). U *Philips audio control protokolu* se doba verifikace zkrátila z hodin na sekundy.
- ✓ *Kronos* nově obsahuje diagnostickou funkci, která umožní přesně vystopovat, proč systém nesplnil časové požadavky na systém.
- ✓ *Kronos* je vyvinut speciálně pro řešení robustních systémů pracujících v reálném čase

Nevýhody:

- ✗ Stejně jako u nástroje *HyTech*, je verifikace bezpečnostních vlastností potlačena. Důraz je kladen na analýzu a kontrolu požadavků na rychlost.
- ✗ U robustních protokolů je nezbytné použít pokročilé návrhové techniky, které zmenšují procházený stavový prostor.

4.16 Lotos

Domovská stránka nástroje: http://wwwtios.cs.utwente.nl/lotos/
Primární publikace: [38]
Typ: mezinárodní specifikační technika pro specifikaci distribuovaných systémů
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: existuje mnoho různých *LOTOS Tools*
Autoři: vytvořen jako standard společností *ISO* (*International Organization for Standardization*)

Jazyk *LOTOS* je používán pro specifikaci bezpečnostních protokolů a kryptografických operací. Vznikl na principech *CCS*[29] a *CSP*[30]. *LOTOS* je standardizovaný jazyk vhodný pro popis distribuovaných systémů. Pomáhá verifikovat, zda je protokol dostatečně robustní vzhledem k možným útokům na něj. Nástroj dokáže pomocí jednoduchých pravidel definovat chování útočníka a simulovat některé druhy útoků, které nejsou založeny na kryptografii. Tato myšlenka specifikovat chování útočníka byla poprvé úspěšně použita v nástroji *Interrogator*, kde jsou účastníci komunikace modelováni stavovými automaty.

Jazyk *LOTOS* se skládá ze dvou komponent:

- **procesní algebra** se strukturovanou sémantikou operací (inspirovaná *CSP*). Popisuje chování procesů a jejich interakci. V rámci této algebry nabízí *LOTOS* řadu operátorů a abstrakcí.
- **abstraktní jazyk datových typů** (přesně definuje typy zpráv a jiných elementů popisujících chování systému). Abstraktní datový typ je specifikován svým podpisem a rovnicemi, které dávají význam datovým operacím.

Nejedná se tedy o typický nástroj, který provádí verifikaci na nějakém modelu. Nástroje provádějící verifikaci modelem jsou neocenitelnými při hledání zranitelností, ale nejsou vhodnými kandidáty k verifikaci pro dokazování korektnosti, když se žádná zranitelnost nenajde.

Chování účastníků komunikace se modeluje pomocí nezávislých procesů, které přesně definují chování konkrétního účastníka. Tyto procesy spolu komunikují pomocí systému bran. K synchronizaci dvou a více procesů dochází formou interakce na těchto bránách. Procesy probíhají v definovaném prostředí a když se dostanou do nějakého výjimečného stavu, tak informují systém zprávou přes bránu *System_State*.

Existuje novější varianta standardu *LOTOS*, která se jmenuje *E-LOTOS* (*Enhancements to LOTOS*). *E-LOTOS* byl navrhnut s cílem poskytnout formální specifikační jazyk pro popis systémů na různých úrovních abstrakce, který je postaven na stejných principech jako *LOTOS*. *E-LOTOS* však poskytuje řadu vylepšení a zároveň plní veškeré cíle, pro které byl navrhnut původní standard *LOTOS*.

[29] Communication Concurrency System
[30] Communicating Sequential Processes

Mezi základní vylepšení *E-LOTOS* patří:

- **modularita** – již naprogramované komponenty lze použít v jiných projektech
- **datové typy** – přidány metody pro podporu modelování vlastních typů, rozšiřitelných záznamů,..
- **čas** – nově lze modelovat i časový aspekt simulace, který je nezbytný pro modelování systémů pracujících v reálném čase
- **výjimky**, **typové brány**, **částečná synchronizace**,...

Formální jazyk *LOTOS* byl použit k modelování protokolu *Equicrypt* (podmíněný přístup k některým multimediálním službám) a našel některé zajímavé možnosti útoků. *LOTOS* byl použit při verifikaci řady telekomunikačních protokolů. Rovněž se pomocí nástroje *LOTOS* verifikovaly protokoly zaměřené na vzájemnou autentizaci mezi uživatelem a třetí skupinou.

Výhody:

- ✓ *LOTOS* má díky možnosti definování vlastních datových typů obrovskou vyjadřovací schopnost.
- ✓ Výkonnost – schopnost postihnout široké spektrum vlastností služeb a protokolů.
- ✓ Dobrá definovanost – syntaxe a sémantika umožňují jednoduché zacházení a validaci. Lze snadno kombinovat jednotlivé parametry verifikace.
- ✓ Dobrá strukturovanost – přehlednost, jednoduché porozumění specifikace.
- ✓ Abstrakce – umožňuje vyšší abstrakci bez specifikace implementačních detailů.

Nevýhody:

- ✗ Velká vyjadřovací schopnost je i nevýhodou, protože nám umožňuje formulovat věci, které nedávají smysl. Dá se podobným problémům vyhnout, pokud maximálně zjednodušíme specifikaci.
- ✗ Nástroj postrádá kompilátor, který by našel většinu syntaktických chyb.
- ✗ Klasický *LOTOS* postrádá některé funkce, které jsou nezbytné pro moderní vývoj softwaru. Například nelze znovu použít již naprogramovanou komponentu v jiném projektu. *E-Lotos* již tuto funkcionalitu obsahuje.
- ✗ Chybí užitečná redundance při deklaraci událostí, ale naopak se objevuje v nežádoucí podobě u deklarace argumentů funkcí.

4.17 LySa

Domovská stránka nástroje: http://www2.imm.dtu.dk/cs_LySa/
Primární publikace: [39]
Sekundární publikace: [40], [41]
Typ: verifikace zkoumáním chování protokolu
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *LySatool 2.02*
Autoři: Mikael Buchholtz a Hanne Riis Nelson

LySa je procesní kalkul vyvinutý pro analýzu bezpečnostních protokolů. Je založen na $\prod$-*kalkulu* a *Spi-kalkulu*[31], ale liší se od nich nejméně ve dvou aspektech. Prvním je absence kanálů, protože *Lysa* podporuje jen jeden komunikační kanál pro všechny procesy. Druhým je testování vstupu a kódování realizované pomocí porovnávání vzorků.

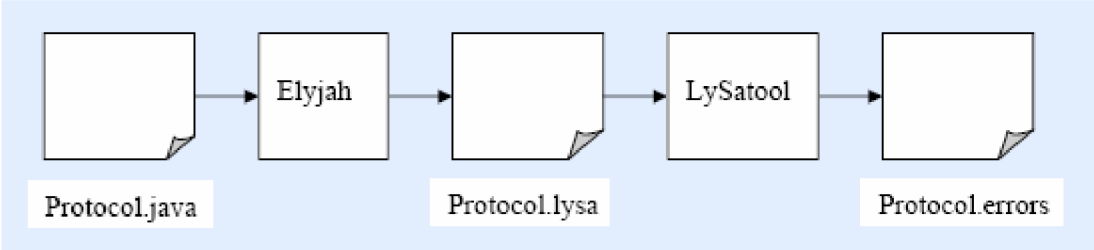
Tento kalkul zahrnuje jazykové primitivy pro modelování síťové komunikace, která je zabezpečena kryptografickými metodami. Analýza se zaměřuje na zajištění důvěrnosti a bezpečné autentizace pro každý *LySa* proces. Platí to i v případě, když je proces cílem libovolného útoku.

Specifikace protokolu pomocí nástroje *LySa* se skládá z následujících prvků:

- **Jména** – používají se pro modelování symetrických klíčů, nonces, zpráv a asymetrických klíčů *m+* a *m-*.
- **Termy** – modelují zprávy, které se skládají ze jmen, proměnných a klíčů.
- **Procesy** – hlavní část specifikace. Proces se skládá ze vstupu, výstupu, generovaných jmen a klíčů. S procesy lze provádět řada úkonů. Mohou se zpravovat paralelně, replikovat nebo navzájem ukončovat.

Implementace jazyka *LySa* je *LySatool*, který je napsán v standardním *SML/NJ* jazyce. *LySatool* má na vstupu *LySa* proces a výsledek analýzy vrací například v *HTML* souboru. *LySa* proces je specifikován v *ASCII* formátu. Pro snazší specifikaci bezpečnostního protokolu v *LySa* kalkulu vznikl nástroj *Elyjah*, který převádí specifikaci protokolu z jazyka *Java* přímo do *LySa* kalkulu. Podrobně to ilustruje následující obrázek.

[31] Podrobný popis naleznete zde *URL*:<http://en.wikipedia.org/wiki/SPI>.



Obr. 4.7: Příklad analýzy protokolu specifikovaného v jazyce *Java*

Současná verze nástroje *Elyjah* neumí transformovat libovolnou specifikaci protokolu, různá syntaktická omezení jsou popsána v [41]. Rovněž jazyk *LySa* podléhá řadě omezení. Například modeluje pouze perfektní kryptografii, takže neuvažuje existenci útoku hrubou silou (jediný způsob zakódování či rozkódování textu závisí na znalosti klíče). Jakmile je protokol úspěšně formulován v *LySa* kalkulu, tak *LySatool* analyzuje zdrojový kód metodou statické analýzy (analýza se provádí aniž by byl vstupní soubor nějakým způsobem prováděn jako série příkazů). *LySatool* poté informuje uživatele o nalezených slabých místech. Například informuje o tom, kde může útočník získat nějaké tajemství nebo kde může podstrčit modifikovanou zprávu. Slabá místa se hledají aproximační technikou, která dokáže garantovat splnění požadavků na důvěrnost nebo na autenticitu. Naopak nevýhodou této techniky je nalezení slabých míst, které nelze v reálném světě úspěšně reprodukovat.

Další zajímavým doplňkem pro *LySatool* je nástroj *For-LySa*, který transformuje specifikaci protokolu v *UML* do procesního kalkulu *LySa*. *LySatool* byl použit například při analýze protokolu *Wide Mouth Frog* (kap. 3.3), *MSR*, *Needham-Schroeder* (kap. 3.1), *Otway-Rees* (kap. 3.2.), *Andrew Secure RPC*, a dalších.

Výhody:

- ✓ Specifikace *LySa* procesů je dostatečně jednoduchá, proto lze snadno transformovat zápis protokolu do *LySa* procesu.
- ✓ Pro tuto koncepci verifikace existuje řada podpůrných nástrojů jako *Elyjah* resp. *For-Lysa*, což umožňuje použít specifikace protokolů například v jazyce *Java* resp. *UML*.
- ✓ *LySatool* je dobře vyzkoušený nástroj použitý u verifikace řady protokolů. Objevil dříve nalezené chyby v návrhu protokolů a dokázal i objevit úplně nový druh útoku během verifikace protokolu *MSR*.
- ✓ Pokud existuje v protokolu chyba, tak ji *LySatool* objeví.
- ✓ Verifikace je provedena v polynomiálním čase[32].

[32] Nástroje pracující na podobných principech mají většinou exponenciální časovou složitost.

Nevýhody:

- Nástroj prezentuje verifikační techniku odlišnou od verifikace modelem i verifikace hypotézou. Analýza protokolu souvisí jen se zkoumáním chování protokolu. To může být problémem kvůli bohatosti jazyka termů.
- Nástroj někdy označí chyby, které ve skutečnosti nepředstavují chybu v návrhu protokolu. To je způsobeno aproximační technikou verifikace, která je odlišná od klasického hledání problému pomocí procházení nějakého konečně stavového prostoru.

4.18 Model-Checking kit

Domovská stránka nástroje: http://www.fmi.uni-stuttgart.de/szs/tools/mckit/
Primární publikace: [42]
Typ: používá různé verifikační modely (Petriho sítě, konečné automaty,…)
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *mckit 1.4.0* (na CD)
Autoři: Javier Esparza, Claus Schroter, Stefan Schwoon

Jedná se o kolekci programů, která umožňuje modelovat konečně stavový systém pomocí různých modelovacích jazyků. Verifikace probíhá pomocí celé škály verifikátorů. Hlavní myšlenkou tohoto nástroje je možnost zvolit si vhodný specifikační jazyk pro verifikaci systému a následně lze použít různé verifikátory na stejný model. Jedná se o otevřený systém, takže se v budoucnosti předpokládá přidání dalších jazyků a nástrojů.

Verifikace probíhá následujícím způsobem:

- nainstalujeme prostředí *Model-Checking kitu*
- zvolíme jazyk pro modelování protokolu - aktuální verze obsahuje ***APNN*** (*Abstract Petri Net Notation*), ***BPN2*** (*Basic Petri Net Notation*), ***CFA*** (*Communicating Finite Automata*), ***IF*** (*Interchange Format*), ***PEP*** (*Low Level PEP notation*) a ***SENIL*** (*Simple Extensible Net Input Language*). Při výběru jazyka je třeba dát pozor, zda je vstup ve tvaru *1-bezpečné Petriho sítě*[33]. U jazyků *APNN, PEP* a *SENIL* by nedodržení této podmínky mohlo vyvolat nesmyslné výsledky.
- uložíme nastavení do souboru *system.txt*

[33] V každém uzlu může být v jednu dobu nejvýše jeden token.

Příklady dotazů:

- pokud chceme zkontrolovat, zda nemůže dojít k uváznutí (*deadlocku*):
  `check ml:dc system.txt`
  (kde ml je zvolený modelovací jazyk a dc je zvolený deadlock checker)
- pokud chceme zkontrolovat, zda existuje stav, který splňuje konkrétní vlastnost:
  `check ml:rc system.txt property.txt`
  (rc je reachability checker, který rozhoduje o dosažitelnosti stavu
- pokud chceme zkontrolovat, zda platí nějaká dočasná vlastnost, tak ji specifikujeme v *CTL* nebo *LTL* jazyce a opět uložíme do souboru *property.txt*:
  `check ml:mc system.txt property.txt`
  (ml je model checker – měl by korespondovat s hledanou vlastností)

Výhody

- ✓ Snadno se nainstaluje a používá.
- ✓ Přenositelnost (všechny programy *MC kitu* jsou napsány v jazyce *C*).
- ✓ Otevřená specifikace (*MC kit* byl vyvíjen jako otevřená knihovna nástrojů, kde lze snadno přidat nové nástroje nebo verifikátory).

Nevýhody

- ✗ Přenositelnost na úkor grafickému rozhraní (*MC kit* komunikuje jen přes textové soubory).
- ✗ Skoro žádná syntaktická kontrola, zda je protokol specifikován dobře.
- ✗ Hodně verifikátorů bylo napsáno pro konkrétní jazyk, proto jejich překlady ztrácejí na rychlosti během provádění verifikace.

4.19 Murphi

Domovská stránka nástroje: http://sprout.stanford.edu/dill/murphi.html#Overview
Primární publikace: [43]
Typ: verifikační model založen na principu podmínka/akce
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Standard Murphi 3.1*
Autoři: David L. Dill, Andreas J. Drexler, Alan J. Hu a C. Han Yang

Nutnost co nejjednodušeji a pokud možnost automaticky verifikovat složité nedeterministické protokoly vedla k vytvoření jazyka *Murphi*, který je údajně nejjednoduším jazykem provádějícím i nedeterministickou analýzu.

Murphi je specifikační jazyk, který je založen na kolekci podmíněných příkazů fungujících na principu podmínka-akce, které jsou prováděny do skončení verifikace. Pravidla jsou v následujícím tvaru:

```
Booleovský výraz  → akce
```

Booleovský výraz obsahuje konstanty, proměnné a operátory, které iniciují provedení akce. *Akce* je sekvence příkazů, které se po splnění podmínky vykonají.

Datové struktury a podmíněné příkazy jsou napsány v jazyce podobném jazyku *C*. Verifikační proces je založen na explicitním vytvoření konečně stavového modelu. Verifikátor provádí prohledávání do hloubky (resp. do šířky) v stavovém grafu podle předepsaných podmíněných přechodů a ukládá navštívené stavy do velké hešovací tabulky. Hešovací tabulka je použita kvůli efektivnímu využití prostoru. Všechny datové typy jsou rovněž kódovány do co nejmenšího počtu bitů kvůli úspoře místa, ta je ale dosažena na úkor rychlosti. Pokud verifikátor navštíví stejný stav dvakrát, neprovádí již následně analýzu jeho následovníků, protože se tak již stalo po první návštěvě stavu (při zapsání do hešovací tabulky). Ke každému stavu se do hešovací tabulky ukládá i předchůdce, takže v případě nějaké chyby lze přejít do předcházejícího stavu. *Murphi* umí kromě chyb v návrhu protokolů detekovat i uváznutí (stavy bez následníků ve stromě provádění).

Verifikace probíhá následujícím způsobem:

- napíše se zdrojový kód protokolu v jazyce *Murphi*
- ten je předán *Murphi* kompilátoru, který vygeneruje *C++* program kompilovaný včetně kódu pro verifikátor a umožní tak kontrolovat platnost invariantů (platnost jisté vlastnosti během celé verifikace), hledat chyby ve výrazech, detekovat uváznutí apod.
- spustí se verifikátor, který vymodeluje všechny možné dosažitelné stavy v systému a snaží se najít stav, kde je platná nějaká chybová podmínka

V dokumentu [43] byl *Murphi* použit při verifikaci dvou velkých hardwarových návrhů, které byly původně určeny pro komerční produkty. Jedná se o *protokol kontroly koherence cache* založený na adresářové struktuře pro multiprocesorový systém a *synchronní protokol pro komunikaci na linkové vrstvě*. *Murphi* byl rovněž použit při demonstraci zranitelností v protokolu *Kerberos, RSA, TMN* a dalších.

V minulosti se objevila řada postupů, jak zefektivnit vlastní verifikaci. Největším problémem byla velikost operační paměti, která znemožňovala provádět analýzy robustních protokolů a omezovala počet paralelních simulací. Bylo navrženo několik postupů, které dokáží s lineární časovou složitostí číst data z magnetických pásků [64].

Rovněž se objevila nová verze nástroje *Murphi* (*Parallel Murphi*), která se specializuje na efektivní zpracovávání paralelních smyček.

Dosažené zrychlení oproti sekvenční verzi demonstruje následující tabulka:

Protokol	**Zrychlení**
SCI	44,2
DASH	27,8
FLASH	29,4

Zrychlení bylo dosaženo díky následujícím prvkům:

- **Aktivní zprávy** – redukují zpoždění během výměny zpráv. V porovnání s klasickými zprávami obsahují adresu na posluchač (*handler*), který bude vyvolán po obdržení aktivní zprávy na uzlu. Aktivní zprávy jsou řešeny asynchronně, takže kontrola nad prováděním není během zasílání zpráv přerušena a přijetí zprávy je signalizována posluchačem. Posluchače na uzlech jsou průběžně kontrolovány.
- **Paralelní vyčíslování stavů** – na začátku paralelní verifikace volá každý uzel metodu `search()`. Hlavní uzel rozešle počáteční stavy pomocí metody `send()`. Uzel přijme posluchačem zprávu a zkontroluje stav se svou tabulkou stavů. Poté stav přidá do své tabulky stavů a zařadí ho k provádění. V dalším kroku metody `search()` se stav na vrcholu fronty vyřadí a jeho následovníci jsou informování aktivní zprávou.

Výhody

- ✓ *Murphi* umožňuje efektivně spustit několik simulací krátkých protokolů a zkoumat zranitelnosti z útoků přehráváním.
- ✓ Nástroj *Parallel Murphi* je slibným kandidátem pro analýzu paralelních systémů a dosahuje podstatných zrychlení oproti sekvenční verzi.

Nevýhody

- U verifikace rozsáhlejších systémů by se hodila lepší kontrola verifikačního procesu. V těchto situacích by práci usnadnily body, kdy by se verifikace mohla zastavit a dala by se ručně nastavit další cesta.

4.20 NRL Protocol Analyzer

Domovská stránka nástroje: nezjištěna
Primární publikace: [44]
Sekundární publikace: [45]
Typ: verifikační model stavovými automaty
Dostupnost: nezjištěna
Aktuální verze nástroje: nezjištěna
Autoři: Catherine Meadows a Paul Syverson

NRL Protocol Analyzer (NPA) je speciální verifikační nástroj napsaný v *Prologu*, který byl vytvořen pro analýzy kryptografických protokolů používaných při autentizaci a distribuci klíčů po síti. Nástroj byl vyvinut v *U.S. Naval Research Laboratory*[34]. *NPA* pravděpodobně převyšuje všechny podobné nástroje širokou škálou útoků, které je schopen objevit a analyzovat. Jedním z důvodů, proč je tento nástroj tak účinný, je indukční přístup k prohledávání stavového prostoru.

Nástroj je založen na principech přepisovacího modelu, který vymysleli pánové Dolev a Yao. Tento model předpokládá, že útočník je schopen číst, měnit i ničit veškeré zprávy procházející systémem a provádět prakticky libovolné kryptografické operace. Dále se předpokládá, že útočník nezná některá tajemství, která se snaží získat.

Uživatel specifikuje protokol pomocí stavových automatů, které spolu komunikují. Poté nastaví limity pro omezení hledaného prostoru pomocí lemmat a specifikuje nezabezpečený stav, kdy útočník získal tajemství (tzv. *seed term*).

NPA následně vygeneruje jazyk obsahující tajemství a induktivní technikou zjišťuje, zda se útočníkovi nemůže podařit toto tajemství získat. Pokud je útočník úspěšný, lze pomocí *NPA* přesně zjistit chybu v návrhu a lze jím i vygenerovat případný útok.

[34] Vědecký institut vytvořený v roce 1992 americkým námořnictvem.

***NPA* generuje jazyk ve třech fázích:**

- **Generování termů** – *NPA* pro všechny termy jazyka najde kompletní množinu cest, kterými se útočníkovi podaří konkrétní term zjistit.
- **Verifikace pravidel** – *NPA* prochází všechny tyto cesty a z dalšího uvažování odstraní ty cesty, ke kterým útočník potřebuje znát nějaký prvek z jazyka, který nezná.
- **Generování nových pravidel** – *NPA* podle zbývajících cest vygeneruje novou skupinu přepisovacích pravidel. Ke generování těchto pravidel používá heuristiky.

Tyto tři fáze se opakují tak dlouho, dokud *NPA* neeliminuje všechny cesty ve fázi verifikace pravidel. V takovém případě se podařilo úspěšně vygenerovat jazyk protokolu. Pokud se v třetí fázi nepodaří vygenerovat žádné nové pravidlo, tak generování jazyka skončilo neúspěšně.

V dokumentu [44] se srovnává *NPA* s verifikační *BAN* logikou. Navzdory tomu, že *BAN* logika v minulosti našla hodně nových chyb v protokolech, tak je často kritizována za problematickou tvorbu vstupních pravidel a za nejasná omezení na typy chyb, které je schopna identifikovat. *NPA* používá odlišný postup, kde na začátku verifikace je poměrně přesně specifikován protokol i schopnosti útočníka. Je tedy snazší určit, které chyby je *NPA* schopen rozpoznat.

Postup verifikace je prováděn přechodovou funkcí, kde každé pravidlo je definováno následujícími výrazy:

- vstup od útočníka před aplikováním přepisovacího pravidla
- hodnoty lokálních proměnných před aplikováním přepisovacího pravidla
- výstup, který získá útočník po aplikování pravidla
- nové hodnoty lokálních proměnných po aplikování přepisovacího pravidla

Podobný princip byl dříve použit u nástroje *Interrogator* (kap. 4.12), ale *NPA* se snaží prokázat, že se simulace nedostane do některého z nechtěných stavů. Můžeme tímto způsobem řešit i nekonečně stavové systémy, ale proces verifikace je méně automatický v porovnání právě s nástrojem *Interrogator*. Další zajímavou vlastností tohoto nástroje je schopnost prokázat, zda je protokol imunní proti útokům, kdy útočník komunikuje ve více sezeních[35] a podvádí při komunikaci tím, že předává zprávy z jiného kontextu.

[35] Varianta útoku přehráváním (kap.2.6.5).

V minulosti byl *NPA* úspěšně použit pro analýzu *Internet Key Exchange Protocol*, *Secure Electronics Transactions Protocol*, *Selective Broadcast Protocol*, *Resource Sharing Protocol*, *Neumann-Stubblebine reauthentication protocol* a dalších.

Výhody

- ✓ Na začátku verifikace je zřejmé, které chyby v návrhu je *NPA* schopen najít a které nikoliv.
- ✓ V případě nalezení chyby v návrhu je *NPA* schopen vykonstruovat příslušný útok.
- ✓ Lze jím řešit i nekonečně stavové prostory (teoreticky). Je to způsobeno indukčními principy, které drasticky omezují procházený prostor.
- ✓ Je schopen odhalit i útoky přehráváním.
- ✓ V minulosti byl použit na verifikaci řady klíčových protokolů zabezpečujících například platební transakce na Internetu.

Nevýhody

- ✗ Techniky používané v *NPA* postrádají nezávislou formální specifikaci. *NPA* nebyl konstruován pro analýzu datových struktur libovolných velikostí.
- ✗ Proces verifikace je méně automatický něž například u nástroje *Interrogator*.

4.21 NS-2 (The network simulator)

Domovská stránka nástroje: http://www.isi.edu/nsnam/ns/
Primární publikace: [46]
Sekundární publikace: [47]
Typ: objektově orientovaný simulátor
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *NS-2.31*
Autoři: M.Sc. Frida Eng, Ph.D. Fredrik Gunnarso,…

NS2 je simulátor událostí zaměřený na výzkum síťových protokolů. Jedná se o nástroj, který vznikl jako následovník projektu *REAL network simulator* z roku 1989. *NS2* je určen pro simulaci *IP* protokolů jako *TCP, UDP, RTP, SRM* a dalších směrovacích a komunikačních protokolů u drátových i bezdrátových sítích. Je naprogramován v jazyce *C++* a jeho rozhraní pro zpracování vstupu vyžaduje specifikaci protokolů v jazyce *OTCL* (objektová verze *TCL*). *NS2* je nadějný projekt, jehož vývoj nebyl dosud ukončen. Uživatelé stále nacházejí nové chyby v návrhu, které jsou postupně opravovány.

Nástroj je volně šiřitelný, objektově-orientovaný a poskytuje prostředky pro vytvoření modelu sítě, specifikaci vstupních dat i pro analýzu a prezentaci výsledků. K dispozici je i zdrojový kód simulátoru, což uživatelům umožňuje přidávat podporu pro nové komunikační protokoly a monitorovací nástroje.

Instalace[36] *NS2* není tak snadná, jak jsme zvyklí u *GNU* softwaru, tj. není ve stylu *"configure, make, make install"*. Po instalaci můžeme začít vytvářet simulační skripty. Každý skript je napsán v jazyce *TCL* a kompletně popisuje simulační úlohu. Skript rovněž musí zahrnovat popis síťové topologie, komunikačních protokolů a událostí, tj. kdy mají být posílána jaká vstupní data. Například u protokolu *TCP* je možné specifikovat velikosti front směrovačů a limity pro *TCP* okénka. Vytváření simulačních skriptů je složitá úloha vyžadující znalost vytvářeného modelu sítě, tříd simulátoru *NS2* a programování v *TCL*.

NS2 potřebuje operační paměť pro evidenci všech paketů, které jsou právě v síti. V rozlehlých vysokorychlostních sítích mohou být v kterémkoliv okamžiku v síti řádově desítky až stovky tisíc paketů. Potřebná operační paměť pak může dosáhnout řádu jednotek gigabajtů.

Pro snížení nároků na objem operační paměti lze doporučit zařadit na začátek simulačního skriptu příkazy, kterými omezíme počet hlaviček, jež budou pro každý paket evidovány:

```
remove-all-packet-headers
add-packet-header TCP IP
```

V případě verifikace nového protokolu v *NS2* je zapotřebí přidat *C++* kód a příslušný *OTCL* kód do datové základny simulátoru. Přidaný *C++* kód specifikuje parametry a metody, které jsou nově dostupné pro *OTCL* skripty. Architektura *NS2* je vytvořena podle *OSI*[37] modelu, což usnadňuje verifikaci *IP* protokolů, které mohou být simulovány na různých úrovních *OSI*. Zajímavé výsledky verifikace u bezdrátových protokolů a srovnání nástrojů *NS2*, *OPNET* a *GloMoSim* v tomto prostředí představuje dokument [47].

NS2 používá v současné době *CESNET* pro výzkum mechanismů pro řízení zahlcení v rozlehlých vysokorychlostních sítích. Při simulacích s *NS2* se setkali s následujícími problémy:

- *TCP* vysílač někdy přestane vysílat na dobu několika sekund.
- *TCP Reno* přejde do fáze *slow start* po naplnění fronty směšovače.
- Diagramy dosažené propustnosti s malou časovou granularitou vykazují fluktuace.
- Ve výstupním souboru se objevují nenumerické artefakty.

[36] Postup k instalaci lze nalézt na *URL*<http://www.isi.edu/nsnam/ns/ns-win32-build.html>.
[37] Jedná se o abstraktní popis síťové komunikace a protokolů použitých pro komunikaci mezi počítači, který je dělen do sedmi vrstev.

Závěrem se dá říci, že *NS2* je dobrý nástroj pro analýzu celého spektra protokolů na různých úrovních *OSI* modelu. Nástroj je vyvíjen pod *GNU* licencí což je jeho kladem i záporem. Ve srovnání s komerčním nástrojem *OPNET Modeler* však trochu zaostává.

Výhody

- ✓ Nástroj je volně dostupný a vyvíjený pod *GNU* licencí.
- ✓ *NS2* je vhodný pro hlavní *IP* protokoly jako *TCP, UDP,…*

Nevýhody

- ✗ Většinou neaktuální dokumentace. Zdrojové kódy jsou volně k dispozici a často se někomu podaří nalézt nějaké zlepšení či chybu, která je následně opravena, ale nestačí se zanést do dokumentace.
- ✗ Obrovské paměťové nároky. Každý paket musí být během simulace uložen v operační paměti, často tak vznikají až několika gigabajtové požadavky.
- ✗ Špatná rozšiřitelnost u větších simulací.

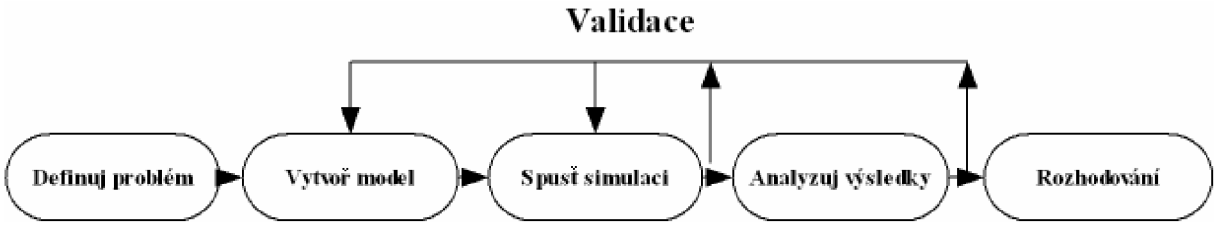
4.22 OPNET Modeler

Domovská stránka nástroje: http://www.opnet.com/products/modeler/home-2.html
Primární publikace: [48]
Sekundární publikace: [49],[50]
Typ: vytváří objektový model
Dostupnost: komerční nástroj
Aktuální verze nástroje: nezjištěna
Autor: *OPNET Technologies*

OPNET Modeler je softwarové vývojové prostředí vhodné pro návrh a analýzu komunikačních sítí. Vlastní verifikátor je součástí softwarového balíku *OPNET* od americké firmy *OPNET Technologies*. Usnadňuje návrh komunikačních sítí a lze ho použít pro verifikaci celé škály protokolů. Nástroj umožňuje velkou flexibilitu během testování. *OPNET Modeler* je hierarchicky a objektově orientován, grafické prostředí odráží reálné rozmístnění jednotlivých síťových komponent a na nejnižší úrovni je chování jednotlivých komponent zapsáno v jazyce *C/C++*. Nástroj poskytuje široké možnosti v oblasti simulace a analýzy výsledků.

Klíčové prvky nástroje *OPNET Modeler*:

- **modelační a simulační kruh** – robustní validační nástroje provázející uživatele po třech z pěti etap návrhového kruhu (obr. 4.8).
- **hierarchické modelování** – každá úroveň hierarchického modelu modeluje jiné aspekty
- **specializace na komunikační sítě** – nástroj obsahuje rozsáhlou knihovnu existujících protokolů a umožňuje uživateli použít pozměněné modely nebo vytvořit své vlastní automatické generování simulačních *OPNET* modelů, které mohou být kompilovány do spustitelných kódů.



Obr. 4.8: Modelační a simulační kruh

OPNET Modeler již v sobě obsahuje řadu knihoven jednotlivých síťových komponent převážně pro *Ethernet, FDDI, TCP, ATM, HTTP* atd. Tento nástroj je používán vedoucími firmami v oblasti vývoje síťových technologií. Nástroj je rozšířen i v akademické sféře, například se používá v *Centru Aplikované Kybernetiky* na *ČVUT*.

Výhody

- ✓ *OPNET Modeler* má atraktivní *GUI* rozhraní.
- ✓ Obrovskou výhodou je možnost editovat zdrojové kódy knihoven, což umožňuje zkoumat možnosti některých základních prvků v systému.
- ✓ Nástroj má dobrou a rozsáhlou dokumentaci.
- ✓ Jedná se o komerční nástroj, ale existuje řada editorů usnadňující modelování na různých úrovních *OSI*.

Nevýhody

- ✗ Nevýhodou tohoto nástroje je potřeba vytvořit konečný automat pro každou úroveň protokolu. To může být problém, pokud je protokol definován jen pseudokódem.

- Pro simulaci lze použít řadu již existujících komponent (*MAC* vrstvy, transceivery, linky,..), ale je nutné každý nový prvek definovat konečným automatem, který se obtížně rozšiřuje a spravuje.
- Standardní verze nástroje neobsahuje utility *ACE* (*Application Characterization Environment*) a *SPG package*, což omezuje možnosti analýzy toku dat

SROVNÁNÍ *OPNET Modeler* a *NS-2*

Vzhledem k tomu, že *OPNET Modeler* a nástroj *NS-2* jsou určeny prakticky pro analýzu stejných *IP* protokolů, zaměřím se nyní stručně na srovnání právě těchto nástrojů. Výsledky vycházejí z dokumentu [50].

- V případě analýzy toků dat s konstantní datovou šířkou (*CBR*) se při použití standardních knihoven jeví efektivnější nástroj *NS-2*.
- Při větším toku dat (5-6Mb/s) dává přesnější výsledky *OPNET Modeler*.
- V případě analýzy *FTP* toků dat dává přesnější výsledky opět *OPNET Modeler*.
- Doba provádění simulace je u obou nástrojů podobná.
- Open source původ *NS-2* je atraktivnější pro vývojáře, kteří nemusí platit licenční poplatky jako u nástroje *OPNET Modeler*.
- K vytvoření modelu v *NS-2* je zapotřebí napsat skript v jazyce *OTCL*, což může být hlavně pro začátečníky problém. Naproti tomu nabízí nástroj *OPNET Modeler* kvalitní *GUI* rozhraní.

Oba nástroje prokazují zhruba stejnou funkcionalitu a při srovnávacích testech dávají podobné výsledky. Volně dostupný nástroj *NS-2* však bude jistě rozšířenější mezi vývojáři, zatímco více různých funkcí a pěkné *GUI* přitáhne k nástroji *OPNET Modeler* pozornost spíše síťových operátorů.

4.23 PDL

Domovská stránka nástroje: nezjištěna
Primární publikace: [51]
Sekundární publikace: [52]
Typ: deduktivní
Dostupnost: nezjištěna
Aktuální verze nástroje: nezjištěna
Autor: Stephen Brackin

Tento nástroj pro automatickou verifikaci protokolů, který je založen na *HOL* logice, představil Brackin. *HOL* logika je kolekce nástrojů pro formální verifikační dokazování. Současná verze je založena na modelování stavů, čímž zvýšila svou jednoduchost a vyjadřovací sílu v porovnáním s přecházejícími verzemi.

Verifikace protokolů pomocí *PDL* spočívá v popisu detailů akcí prováděných procesy v rámci protokolu pomocí nízko-úrovňové logiky. *PDL* definuje procesy pomocí jednoduchého imperativního programovacího jazyka, který je schopen modelovat i kryptografické protokoly a přitom dokáže vyjádřit libovolný jazyk realizovatelný na Turingovu stroji. *PDL* modeluje účastníky komunikace jako agenty, kde každý agent může plnit několik rolí. Role je funkce, která určuje dostupná data pro agenta.

PDL se skládá ze tří rekurzivních jazyků:

- **jazyk pro popis akcí** *(action language)* – definuje akce prováděné individuálními procesy
- **jazyk pro popis stavů** *(state language)* – popisuje stavy sítě včetně chování útočníka
- **jazyk pro popis báze představ** *(belief language)* – popisuje informace, které zná nebo může znát účastník komunikace

Pro popis stavů, ve kterých se protokol nachází, používá PDL kombinaci slotů a akcí:

- **Slot** je omezené datové úložiště pro hodnoty libovolných datových typů. Neexistují žádné globální proměnné, každý proces může přistupovat jen ke svým slotům. Proces je jednoznačně identifikován jménem agenta, který ho vytvořil, jménem role, kterou agent během komunikace hraje a číslem sezení.
- **Agenti** jsou převzati z nástroje *CAPSL* a mohou modifikovat data ve slotech jen pomocí akcí "`send`" a "`receive`". Časový aspekt verifikace je oficiálně nástrojem podporován, nikde však není explicitně vyjádřen.

Na *HOL* logice je postaveno hned několik nástrojů pro formální dokazování. Jedna z její klíčových myšlenek je definovat hypotézu jako datový typ nad metajazykem, který dokáže vyjádřit veškeré jazyky. Taková hypotéza pak umožní provádět verifikaci během kompilování v metajazyku a následně automaticky dokázat nebo vyvrátit jeho platnost.

Pro lepší ilustraci nyní uvedu přehled datových typů, které *PDL* nabízí:

- **Primitivní datové typy**
  - **agent** – plní určitou roli v systému
  - **data** – libovolná data, klíče, nonces,…
  - **funkce** – kódovací funkce, hešovací funkce,….
  - **lokace** – jméno konkrétního slotu
  - **role** – jméno role protokolu
- **Term** – dodává sémantiku pro řetězce bitů
- **Action** – pojmenovává akce prováděné účastníky komunikace
- **Proc** – mapuje funkcím sloty pro datové hodnoty
- **State** – definuje stav protokolu
- **Belief** – definuje podmínky na síti, kterým může verifikátor důvěřovat

Kromě běžných útoků umí PDL modelovat i následující:

- ✓ útočník použije neadekvátní typy nebo špatně ošetřenou kontrolu rovnosti výrazů *(inadequate type and equality checking)*.
- ✓ konkurenční sezení *(concurrent sessions)* – útočník komunikuje ve více sezeních a posílá zprávy vytržené z kontextu jiného sezení.
- ✓ nedorozumění *(misinterpretations)* – akce prováděné legitimními účastníky komunikace mají vlivem útočníka jiný význam než bylo zamýšleno.
- ✓ odhalení *(accidental disclosure)* – útočník získá nějaké tajemství.
- ✓ útoky založené na algebraických vlastnostech *(algebraic properties attacks)* – útočník může využít algebraických vlastností kódovacích funkcí a pozměnit tak zakódovaný text bez znalosti otevřeného textu.

Výhody

- ✓ Hlavní výhoda tohoto přístupu je ta, že se dílčí důkazy nemusí znovu dokazovat v širším kontextu. To z *PDL* dělá stejně důvěryhodnou verifikační metodu jako jsou metody založené na konstrukci útoků s rychlostí podobnou deduktivním metodám.
- ✓ *PDL* má obrovskou vyjadřovací schopnost, dokáže proto odhalit více slabin v návrhu protokolů než většina podobných nástrojů.
- ✓ Kromě zranitelností umožňuje *PDL* detekovat i širokou škálu útoků.

Nevýhody

- Vývojář se musí naučit celou *HOL* logiku, aby mohl provádět verifikaci protokolů.
- *PDL* zatím postrádá *GUI* rozhraní, což by zpřehlednilo a usnadnilo práci.

4.24 PRISM

Domovská stránka nástroje: http://www.cs.bham.ac.uk/~dxp/prism/
Primární publikace: [53]
Sekundární publikace: [54],[55]
Typ: verifikace pravděpodobnostním modelem
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Prism 3.1*
Autoři: Marta Kwiatkowska, Gethin Norman, Dave Parker a Mark Kattenbelt

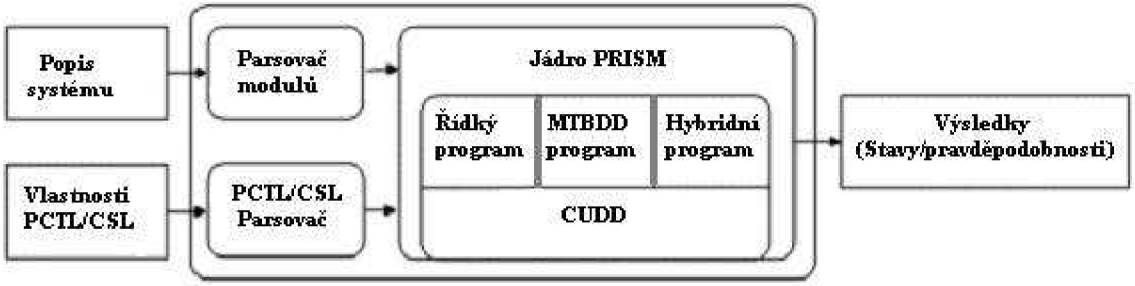
PRISM je pravděpodobnostní verifikační nástroj, který umožňuje analýzu a modelování systémů pomocí verifikačního modelu. Verifikace probíhá pomocí pravděpodobnostní kontroly modelu. Pravděpodobnostní kontrola modelu je formální verifikační technika, která je založena na konstrukci precizního matematického modelu systému, který má být analyzován. Vlastnosti takového systému jsou formálně vyjádřeny pomocí *PCTL* temporální logiky[38] a následně automaticky analyzovány na vytvořeném modelu. Vstup může být vyjádřen i pomocí *CSL*.

***PRISM* podporuje tři typy pravděpodobnostních modelů:**

- **Markovovy řetězce diskrétního času** *(DTMC - Discrete-Time Markov Chains)*
- **Markovovy řetězce spojitého času** *(CTMC - Continuous-Time Markov Chains)*
- **Markovovy rozhodovací procesy** *(MDP - Markov Decision Processes)*

Verifikace probíhá ve třech nezávislých verifikátorech. První verifikátor je založen na řídkých maticích *(angl. sparse matrix)*. Druhý je založen na binárních rozhodovacích diagramech (*MTBDD*) a třetí v sobě kombinuje výhody obou přístupů. Tuto architekturu demonstruje obrázek 4.9.

[38] *PCTL* logika je podrobně popsána na následujícím *URL*< http://en.wikipedia.org/wiki/Probabilistic_CTL>.



Obr. 4.9: Architektura *PRISM*

PRISM je implementován v jazycích *Java* a *C++*. Všechny části systému na vyšší úrovni jako parsery a uživatelská rozhraní jsou napsány v *Javě*. Nízko-úrovňový kód jako knihovny a programové balíky jsou napsány v *C++*. Celý *PRISM* používá *CUDD* balík[39], který je naprogramován v klasickém *C*.

Základními komponentami modelu nástroje *PRISM* jsou moduly a proměnné. Model je složen ze skupiny modulů, které spolu komunikují. Každý modul se skládá ze skupiny lokálních proměnných. Hodnoty těchto proměnných definují stav modulu. Globální stav celého systému je určen lokálními stavy všech modulů.

Chování modulu je popsáno příkazy, které mohou být následujícího tvaru:

```
[] guard -> prob_1 : update_1 + ... + prob_n : update_n;
```

"`Guard`" je predikát určený proměnnými modelu (včetně proměnných z jiných modulů). Každý "`update`" popisuje transakci, kterou může modul provést, pokud platí predikát. Transakce přidává proměnným modulu nové hodnoty. Každý "`update`" se provede jen s určitou pravděpodobností "`prob`". Množství demonstračních příkladů nalezneze přímo v manuálu *PRISM* [65].

V minulosti byl *PRISM* použit u celé řady studií zkoumajících chování různých systémů:

- komunikační protokoly pracující v reálném čase – *Firewire, Bluetooth, Zeroconf, CSMA/CD, WLAN,...*
- pravděpodobnostní bezpečnostní protokoly zaručující anonymitu, bezpečnou výměnu dat, nepopiratelnost,…
- náhodnostní distribuované algoritmy
- dynamické napájecí systémy
- a mnohé další…

[39] *CUDD* balík poskytuje funkcionalitu pro manipulaci s binárními rozhodovacími diagramy *BDD*.

Výhody

- ✓ Nástroj *PRISM* může být ovládán z příkazové řádky nebo pomocí grafického *GUI*.
- ✓ *PRISM* byl v minulosti úspěšně použit při analýze různých systémů.
- ✓ Díky implementaci v *Javě* je nástroj použitelný prakticky na všech běžně používaných platformách.
- ✓ V porovnání s klasickými nástroji fungujícími na verifikaci modelem, dává nástroj *PRISM* přesnější výsledky. Při srovnání s analytickými nástroji umožňuje *PRISM* detailnější analýzu systému.

Nevýhody

- ✗ Vývoj nástroje nebyl dosud ukončen. Plánují se změny zefektivňující celý proces verifikace. Pravděpodobně se koncept nástroje přesune do oblasti hybridních systémů.
- ✗ Nevýhodou je problém s velikostí procházeného stavového prostoru, která je způsobena aproximačními výsledky použitých statistických metod.

4.25 SPIN

Domovská stránka nástroje: http://spinroot.com
Primární publikace: [56]
Sekundární publikace: [57]
Typ: verifikační model
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *Spin 4.2.9*
Autor: Dr. Gepard J. Holzmann

SPIN je jedním z nejznámějších nástrojů pro analýzu logické konzistence souběžných systémů a komunikačních protokolů. Byl vytvořen v *Bellových laboratořích* Dr. Gepard J. Holzmannem kolem roku 1980. V roce 1991 byl zveřejněn. Vlastním modelovacím jazykem je jazyk *PROMELA (PROcess MEta LAnguage)*, která modeluje paralelní kompozici konečně mnoha procesů. Ty mezi sebou komunikují zásadně prostřednictvím synchronních či asynchronních zpráv nebo sdílených proměnných [56].

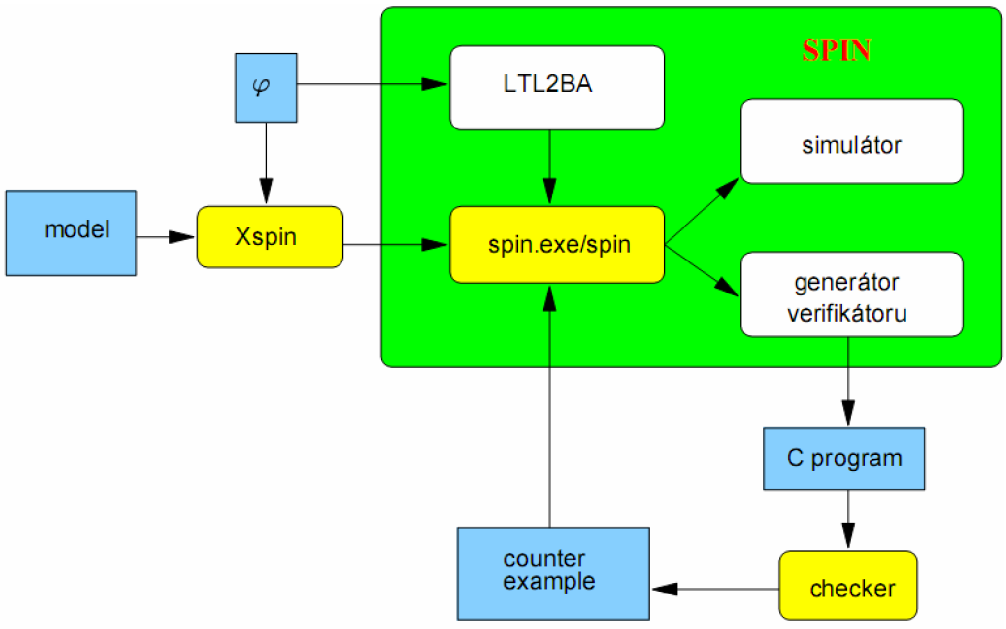
***PROMELA* model se skládá z následujících prvků:**

- Deklarace typů
- Deklarace komunikačních kanálů
- Deklarace globálních proměnných
- Deklarace procesů
- Inicializační proces

***SPIN* jako formální verifikační nástroj poskytuje následující:**

- Intuitivní notaci pro jednoznačné specifikování návrhu bez implementace detailů.
- Notaci pro vyjádření všeobecných požadavků na korektnost.
- Metodologii pro vytvoření logické konzistence mezi návrhem protokolu a protokolem splňujícím požadavky na bezpečnost.

Za zmíňku stojí, že existuje hodně verifikačních nástrojů poskytujících první dvě vlastnosti, ale tu třetí splňují jen některé. *SPIN* akceptuje specifikaci protokolu napsanou ve verifikačním jazyce *PROMELA* a požadavky na korektnost jsou specifikovány v *LTL (Linear Temporal Logic)* logice[40].



Obr. 4.10: Architektura *SPIN*

[40] Tato logika je podrobně vysvětlena na následujícím *URL*:
< http://en.wikipedia.org/wiki/Linear_temporal_logic>

Na obrázku 4.10 je vidět základní schéma architektury nástroje *SPIN*. Vstupem bývá většinou specifikace protokolu z grafického rozhraní *XSPIN*. Následně jsou ze vstupu odstraněny syntaktické chyby. Třetím krokem verifikace je dynamické vytvoření *(tzv. "on the fly")* optimalizovaného verifikačního programu. Tento program je následně zkompilován a spuštěn. Pokud jsou detekovány nějaká porušení pravidel, tak jsou pomocí interaktivního simulátoru vráceny uživateli kvůli posouzení.

SPIN provádí náhodné simulace provádění systému nebo může generovat program v jazyce *C*, který provede efektivní verifikaci bezpečnosti systému. Během verifikace *SPIN* kontroluje, zda systém neobsahuje uváznutí *(tzv. deadlock)*, nespustitelný kód či neošetřené výjimky. *SPIN* může být rovněž využit k verifikaci systémových invariantů. Nástroj rovněž dokáže rozpoznat nekonečné smyčky a vyjádřit verifikaci bezpečnostních formulí ve formulích lineární časové logiky.

Výhody

- ✓ Nástroj má minimální paměťové nároky a dokáže s matematickou jistotou prohlásit, zda zkoumaný systém je bez chyb či nikoliv.
- ✓ *SPIN* je vytvořen na základě více než desetiletého úsilí o vytvoření tzv. *"on-the-fly"* automatického verifikačního nástroje. Ten dokáže lépe komunikovat s uživatelem a iterativním způsobem zlepšovat verifikační proces.
- ✓ *SPIN* detekuje komplexní množinu všech chyb v návrhu včetně uváznutí procesů, neošetřených výjimek, porušení požadavků na bezpečnost a mnoha dalších.
- ✓ Počet protokolů a systémů, které jsou verifikovány nástrojem *SPIN* stále roste. *SPIN* je rozšířen hlavně v akademické sféře. V průmyslové sféře zatím poněkud pokulhává.
- ✓ *SPIN* je efektivně naprogramován pomocí jazyka *C*.
- ✓ *SPIN* obsahuje pěkné *GUI* rozhraní *XSPIN*, které usnadní verifikaci

Nevýhody

- ✗ Jazyk *PROMELA* není programovacím jazykem, ale pouze jazykem pro modelování abstrakce implementace protokolů. Tato abstrakce umožní studovat jednotlivé vlastnosti systému izolovaně a zároveň potlačí implementační a programové detaily. Což, jak se ukázalo, nemusí být nutně nevýhodou. U robustních systémů se tento postup ukazuje jako jediný možný.
- ✗ Největší nevýhodou všech *"on-the-fly"* algoritmů je nutnost ukládat do paměti záznamy o již navštívených uzlech.
- ✗ V případě verifikace *ATMR* protokolu pomocí nástrojů *SPIN* a *VIS* se ukázalo, že v řadě případů je hardwarový verifikátor *VIS* rychlejší než *SPIN* a využívá paměť efektivněji.

4.26 SyMP

Domovská stránka nástroje: http://www.cs.cmu.edu/~modelcheck/symp.html
Primární publikace: [58]
Sekundární publikace: [59]
Typ: kombinuje výhody verifikačního modelu a verifikace založené na dokazování hypotézy
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *SyMP beta 0.3* (na CD)
Autoři: Sergey Berezin, Alex Groce, ...

SyMP je všeobecně použitelný specifikační jazyk napsaný v jazyce *SML/NJ*. Hlavním cílem tohoto jazyka je poskytnout spolehlivé prostředí pro rychlé a efektivní prototypování nových převážně hardwarových specifikací. Kombinuje v sobě výhody přístupů založených na verifikaci modelem i dokazování hypotézy. Základní myšlenka nástroje opět spočívá v automatické verifikaci modelem. S rostoucím počtem stavů je tento druh verifikace více paměťově i časově náročný. *SyMP* proto využívá i techniku známou jako dokazování hypotézy *(theorem proving)*, která na úkor automatizace verifikace podstatně zefektivní celý proces.

***SyMP* používá dva verifikační systémy:**

- ***Standardní systém*** (framework kombinující vlastnosti verifikace modelem i dokazování hypotézy)
- ***Systém Athena*** (specializovaný systém pro verifikaci bezpečnostních protokolů)

Verifikační systém *Athena* je v této práci již popsán podrobněji, nyní se proto zaměřím na popis standardního systému.

Vstup do nástroje *SyMP* je stejnojmenný jazyk, který je otevřený pro přidání nových verifikačních metod, které mohou být jednoduše dodatečně implementovány. Vlastní model verifikovaného systému se skládá z několika deklarací modulů (mohou být vnořené) a vlastností deklarovaných jako hypotézy. Příklad deklarace modulu naleznete na následující straně.

Deklarace modulu:

```
// definuje stat. a dyn. vl.
module <id> [ '['<static parms>']' ] [ <pattern> ] = <module expr>
// sekvence deklarací mezi výrazy begin a end
<module expr> ::= <begin-end clause>
// jméno modulu s aktuálními stat. a dyn. vl.
             |   <module instance>
// synchronní vs. Asynchronní kompozice
             | <parallel composition>
```

Každý modul má statické a dynamické parametry. Statické parametry umožňují uživateli popsat celou třídu podobných zařízení v jednom modulu. Dynamické parametry jsou použity pro propojení modulů s komunikačními kanály a umožňují tak měnit parametry modulů během provádění verifikace.

Výhody

- ✓ Specifikace v *ML* je elegantní a dramaticky redukuje velikost vstupu.
- ✓ *SyMP* je typově bezpečný, tj. při překladu se neobjeví žádné běhové *(run-time)* výjimky.
- ✓ Existuje několik nástrojů, které podporují *SyMP* jako možný vstup pro vlastní verifikaci.

Nevýhody

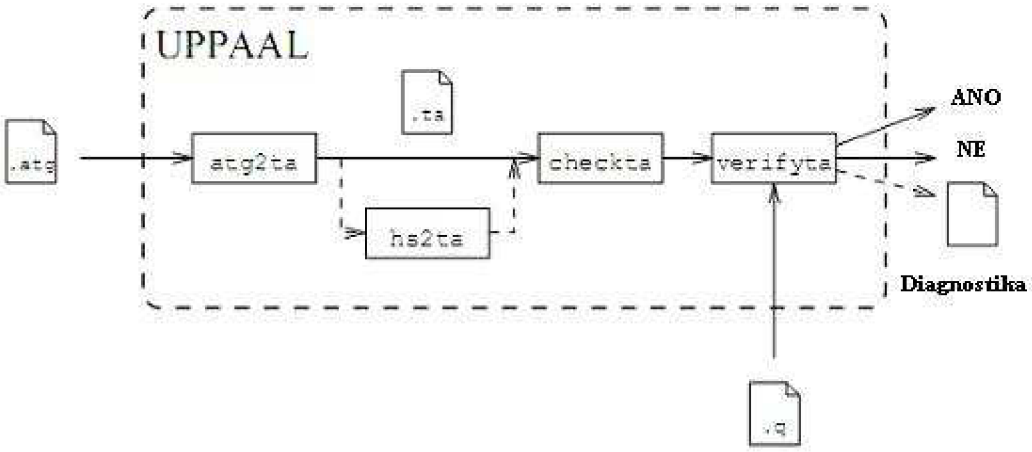
- ✗ *SyMP* není kompletní verifikační nástroj, poskytuje jen jazyk pro snadnou specifikaci vstupu.

4.27 UPPAAL

Domovská stránka nástroje: http://www.uppaal.com/
Primární publikace: [60]
Sekundární publikace: [61]
Typ: verifikace modelem
Dostupnost: komerční nástroj
Aktuální verze nástroje: *UPPAAL 4.0.5* (na CD)
Autoři: vyvíjen univerzitami v Uppsale a Aalborgu.

UPPAAL je integrační prostředí pro modelování, validaci a verifikaci systémů pracujících v reálném čase jako jsou sítě časovaných automatů s rozšířenými datovými typy *(ohraničený integer, pole, atd.)*.

Principy, kolem kterých je nástroj vystavěn, jsou efektivita a snadné používání. Nástroj nabízí konzolovou aplikaci pro verifikaci modelů *(verifyta)* i grafické uživatelské prostředí, ve kterém lze grafickou formou modelovat systém, provádět jeho simulaci i spouštět verifikaci požadavků. Přehledně proces verifikace popisuje následující obrázek.



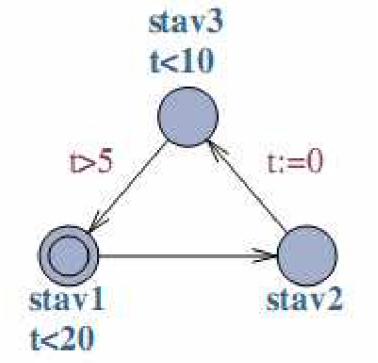
Obr. 4.11: Přehled verifikace pomocí nástroje *UPPAAL*

Kromě specifikace systému pomocí textového popisu časovaných automatů (přípona souborů je `.ta`) je možné importovat vstup v grafické podobě jako síť časovaných automatů modelovanou nástrojem *Autograph*[41] (přípona souborů je `.atg`). V případě grafické podoby vstupu přeloží kompilátor `atg2ta` vstup do souboru `.ta`. V určitých případech lze časové požadavky definovat v intervalech, pak hovoříme o hybridních automatech, které se musí kompilátorem `hs2ta` kompilovat na běžné časované automaty. Program `checkta` zkontroluje vstupní soubor po syntaktické stránce. Vlastní verifikace je pak řešena pomocí programu `verifyta`, který automaticky vygeneruje výsledek verifikace a zprávu o verifikaci (diagnostika).

Modelování

Model systému v nástroji *UPPAAL* se skládá z jednoho nebo více procesů. Popis procesu vychází z časovaného automatu (obr. 4.12), ale umožňuje ho rozšířit i o další prvky. Popis celého modelu je tedy rozšířením sítě časovaných automatů.

[41] Komerční nástroj, jehož domovská stránka je na *URL*:< http://www.autograph-math.com/>.



Obr. 4.12: Časovaný automat v *UPPAAL*

Základní prvky tohoto modelu jsou následující komponenty:

- **proměnné a konstanty** – model v *UPPAAL* může pracovat i s jinými proměnnými než hodinami. Výrazy s proměnnými se mohou využít jako podmínky uvolnění hran nebo invarianty míst. Konstanty jsou prvky modelu, které mají jméno, ale jejich hodnota je na rozdíl od proměnných konstantní.
- **šablony** – lze je použít jako instance u více různých procesů
- **kanály** – zprostředkovávají synchronizaci hran v grafech

Existuje více typů kanálů:

- **binární synchronizační kanály** – Značí se *'`chan k;`'*. Zprostředkovávají vzájemnou synchronizaci mezi hranou s návěštím *'`k!`'* a hranou s návěštím *'`k?`'* z jiného procesu.
- **broadcast kanály** – Značí se *'`broadcast chan k;`'*. Synchronizuje se jedna hrana *'`k!`'* se všemi hranami s návěštím *'`k?`'* u kterých může dojít k přechodu. Pokud není žádná hrana volná, pak může dojít k přechodu aspoň u hrany s návěštím *'`k!`'*.
- **urgentní kanály** - Značí se *'`urgent chan k;`'*. Hrany se synchronizovanými přechody se provedou ihned, jakmile jsou splněny podmínky přechodu.

Modelování v *UPPAAL* má i další specifika jako deklarace *"urgentních míst"*, *"comitted míst"* apod. Před spuštěním verifikace lze nastavit některé její parametry. Patří mezi ně nastavení způsobu reprezentace stavového prostoru nebo to, zda se má při nesplnění formule generovat protipříklad a jestli má jít o nejkratší možný protipříklad (ať už z časového hlediska nebo počtem stavů na cestě).

Aktuální verze nástroje je implementována v jazyce *C++*. V nástrojích tohoto typu (verifikace modelem) je hlavní problém ve velikosti prohledávaného prostoru během verifikace. V *UPPAAL* je tento problém řešen kombinací průběžné verifikace a nové symbolické techniky redukující verifikační problém na řešení lineárního systému.

Výhody

- ✓ V porovnání s jinými nástroji pro verifikaci systémů pracujících v reálném čase využívá *UPPAAL* efektivněji dobu verifikace i paměťový prostor.
- ✓ *UPPAAL* byl aplikován na řadu protokolů a byl úspěšný při hledání i opravování chyb.
- ✓ Nástroj má pěkné grafické rozhraní a umožňuje i specifikovat grafickou cestou vstup.
- ✓ Používání nástroje pro nekomerční účely je bezplatné.

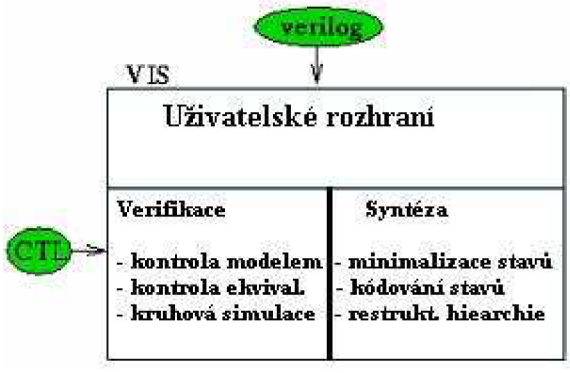
Nevýhody

- ✗ *UPPAAL* nevyčísluje pravděpodobnost, jednoduše prochází všechny možnosti a pokud objeví chybu, tak nedokáže určit pravděpodobnost jejího výskytu.

4.28 VIS

Domovská stránka nástroje: http://vlsi.colorado.edu/~vis/
Primární publikace: [62]
Sekundární publikace: [63]
Typ: verifikační model
Dostupnost: nekomerční nástroj
Aktuální verze nástroje: *VIS 2.1* (na CD)
Autoři: *Vyvíjen na University* of California at Berkeley, *University of Colorado at Boulder* a na *University of Texas, Austin.*

VIS je systém pro formální verifikaci, kompozici a simulaci konečně stavových hardwarových systémů. *VIS* provádí verifikaci systému pomocí kolekce vzájemně komunikujících konečných automatů. Ovládání aplikace se provádí pomocí jazyka *Verilog* a *CTL* na jejichž základě se vytvoří verifikační model. Hlavním cílem *VIS* je maximalizovat výkonnost verifikace.



Obr. 4.13: Blokový diagram *VIS*

Jazyk *Verilog*

Verilog umožňuje popsat hardware pomocí statických struktur a dynamického chování. Chování je specifikováno v konstruktech jazyka vysoké úrovně (fork/join u procesů, kontrolní smyčky,...). Specifikace systému ve *Verilogu* se skládá z hierarchické struktury modulů. Kořenový modul představuje uzavřený systém hardwarových modelů a testovacích dat. Ostatní moduly představují jednoduché časti hardwaru jako třeba *AND* brány nebo celé systémy jako *CPU*. Moduly spolu komunikují přes vstupní a výstupní porty.

CTL

CTL je logika, která se používá pro vyjádření bezpečnostních požadavků kladených na hw systém. Verifikovaná vlastnost se vyjádří v *CTL* formuli a předá se spolu s definicí systému v jazyce *Verilog* nástroji. *VIS* poté pomocí kompilátoru `VL2MV` transformuje specifikaci systému z vysoko-úrovňového *Verilogu* do nízko-úrovňové specifikace v jazyce `BLIF-MV`. Jedná se tedy o podobný kompilátor, jakým je *Casper* pro *CSP* procesy.

Výhody

- ✓ Jedná se o specializovaný systém, který umožňuje formální verifikaci u hardwarových systémů.
- ✓ *VIS* je implementován v jazyce *C*, což zaručuje jeho nezávislost na platformách.

Nevýhody

- ✗ Nástroj se nedá použít při verifikaci softwarových protokolů, jeho doménou je hardware.
- ✗ *VIS* je stále ve vývoji, v budoucnu se plánuje přidání možnosti verifikace i asynchronních systémů.

5 Praktická aplikace

V této kapitole se pokusím shrnout některé praktické poznatky, které jsem nabyl při zkoušení vybraných verifikačních nástrojů z předcházející kapitoly. Hlavním účelem bylo vyzkoušet si práci s těmito nástroji na konkrétních problémech a popsat jejich použitelnost. Byly vybrány poměrně jednoduché příklady, které snadno umožní čitateli rychle se zorientovat v možnostech jednotlivých nástrojů. Při výběru demonstračních příkladů jsem se vždy snažil zvolit problematiku verifikace korektnosti bezpečnostních protokolů, což se mi nepodařilo u nástroje *OPNET Modeler*, který byl primárně vytvořen pro analýzu toků dat. U nástroje *CPN Tools* jsem zvolil příklad jednoduchého protokolu pro posílání paketů, protože analýza bezpečnostního protokolu by si vyžádala podstatně více prostoru, který není vzhledem k obecné srovnávací náplni práce dostupný.

5.1 Implementace v nástroji SPIN

SPIN jsem si zvolil pro jeho rozšířenost při verifikaci protokolů v akademické sféře a pro jeho bezplatnou podstatu. Nástroj naleznete na CD ve verzi 4.2.9 pro Linuxové platformy i pro emulátor *Cygwin* v prostředí Windows. Na CD mimo jiné naleznete i množství demonstračních příkladů. Vzhledem k zaměření této práce na bezpečnostní protokoly jsem zvolil implementaci protokolu *Needham-Schroeder* (Dodatek B).

Program modeluje chování Alice, Boba a útočníka (*Intruder*). Na začátku má každý z nich svůj nezkompromitovaný pár klíčů a nonce. Cílem komunikace je pro Alici a Boba přesvědčit se navzájem o své identitě. Útočník může jednat jako běžný účastník komunikace, takže může padělat zprávy a používat odposlechnuté zprávy. Ale ani on nedokáže bez znalosti klíče dekódovat zprávu. Zkráceně lze tuto zjednodušenou verzi protokolu napsat v následujících třech bodech:

A posílá *B* nonce a svou identitu zakódovanou veřejným klíčem *B*
1: A -> B: $\{N_A, A\}_{Kb}$

B posílá *A* nonce *A* a nonce *B* zakódované veřejným klíčem *A*
2: B -> A: $\{N_A, N_B\}_{Ka}$

A posílá *B* nonce *B* zakódovaný veřejným klíčem *B*
3: A -> B: $\{N_B\}_{Kb}$

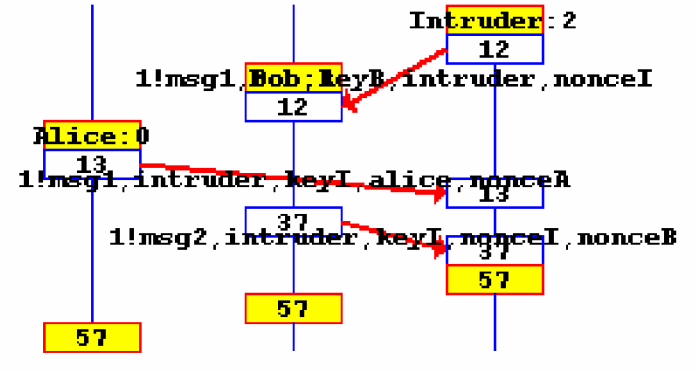
Pro verifikaci jsem zvolil grafické rozhraní *XSpin*, které jsem nainstaloval do emulátoru *Cygwin*. S nástrojem jsem se poměrně snadno naučil pracovat a až na pár detailů (např. během vykreslování stavového automatu se počítač vždy zacyklil), které jsem nebyl schopen vzhledem k časové tísni

v průběhu této práce objasnit, jsem provedl řadu simulací a nalezl i případný útok ze středu (kap. 2.6.6). Nebudu zde komentovat celý zdrojový kód použitého algoritmu, ten je již bohatě okomentován v příloze. Zaměřím se proto jen na pár klíčových proměnných.

`Boolean knowNA, knowNB` - Indikuje, zda se útočníkovi podařilo zjistit nonces N_A a N_B.

`partnerA, partnerB` - Indikuje, kdo je partnerem pro komunikaci s *A* a kdo pro *B*.

Na následujícím obrázku je výsledek simulace provedené v prostředí *XSpin*.



Obr. 5.1: Výsledek simulace v prostředí *XSpin*

V 57 krocích dospěje simulace k následujícímu naplnění proměnných:

```
knowNA   =  1
knowNB   =  1
partnerA  =  intruder
partnerB  =  intruder
```

Jedná se tedy o klasický útok ze středu, kdy si *A* myslí, že komunikuje s *B* a *B* si myslí, že komunikuje s *A*. Útočník během něj získá nonce N_A i N_B.

Závěr:

Během verifikace protokolu se s nástrojem pracovalo relativně dobře. Grafické rozhraní *XSpin* ale dost zaostává po grafické stránce v porovnání s podobnými nástroji. Největší výhodou je zřejmě škála odchycených chyb a rychlost s jakou proběhla verifikace. Nevýhodou je zcela jistě grafické prostředí, které je jen o málo lepší než čistě textová verze *Spin*. Při zpracovávání automatu zobrazující stavy procesu se *XSpin* téměř vždy zacyklil.

5.2 Implementace v nástroji OPNET Modeler

V této části shrnu své praktické zkušenosti s tímto nástrojem a představím aspoň základní vlastnosti, které tento rozsáhlý a komplexní nástroj má. *OPNET Modeler* disponuje hierarchicky uspořádaným grafickým rozhraním, které umožňuje velice efektivní práci s jednotlivými částmi daného projektu.

Vlastní simulační jádro je postaveno na jazyce *C*, takže všechny modely jsou přenositelné mezi systémy *Windows* a *Unix*. Důležitou vlastností je objektové chování všech částí modelu. *OPNET Modeler* umožňuje generovat široké spektrum graficky znázorněných statistik simulace, výsledné zprávy umí ukládat do formátu *XML* nebo *HTML*. Simulaci lze spustit s velkým zrychlením, což umožňuje modelovat i několikaměsíční činnost protokolu.

Modelování lze provádět na těchto úrovních:

- **Globální síťový model** – umožňuje zahrnout do simulace i geografickou polohu prvků sítě.
- **Síťový model** – popisuje jednotlivé lokální sítě. Je velice jednoduché vytvořit model i komplikovanější sítě.
- **Model uzlu** – umožňuje modelovat jednotlivé prvky sítě.
- **Procesní model** – nejnižší úroveň modelování. Model je reprezentován konečným automatem a popisuje jednotlivé stavy procesu a přechody mezi nimi.

Správa projektů je asi nejslabším článkem celého modelování [66]. Každý prvek sítě je uložen v samostatném souboru a *OPNET Modeler* nenabízí žádný nástroj pro správu těchto souborů. Naopak knihovna komponent je dobře zpracovaná a rozsáhlá, takže lze snadno vkládat do projektu prakticky libovolný komunikační nebo síťový prvek. Dokonce lze měnit i poměrně rozsáhlý seznam atributů těchto komponent, u serverů například hardwarovou konfiguraci.

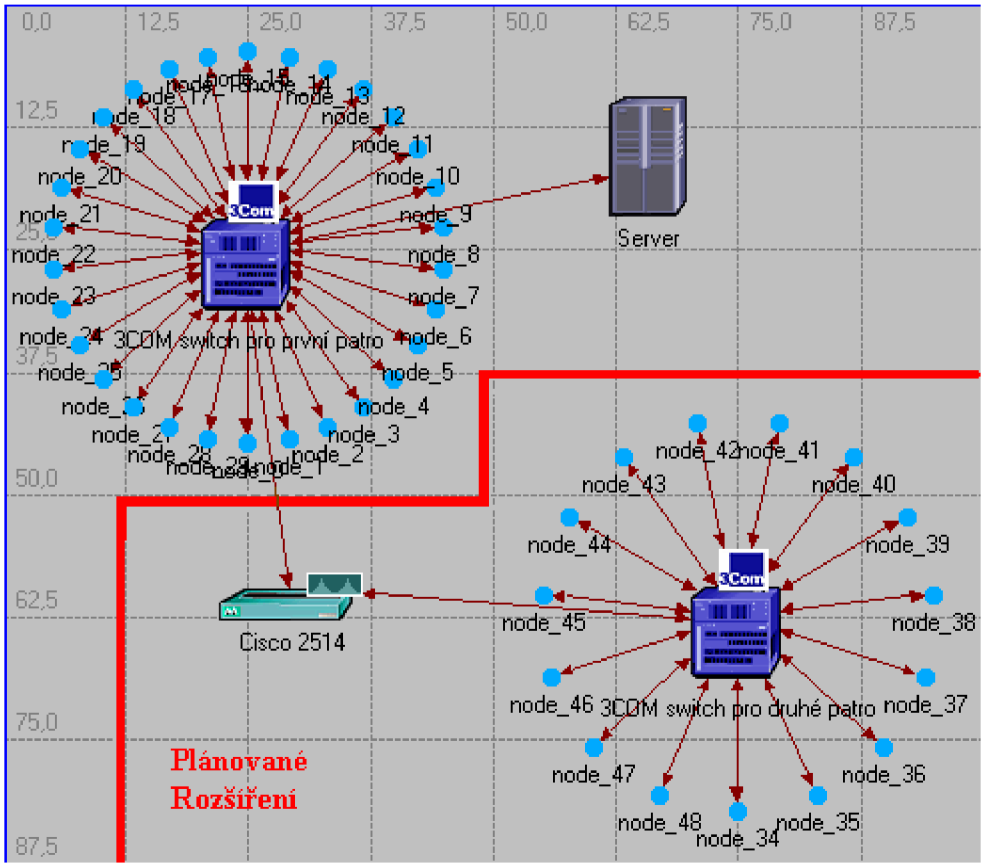
Příjemně mě překvapily možnosti nastavení simulace, které jsou poměrně rozsáhlé. Z těch základních lze zmínit simulovaný čas (řádově i desítky let) a hodnotu pro generování statistik, která udává kolik hodnot se bude během simulace sbírat.

Další užitečnou funkcí je možnost volby simulace mezi statickou a dynamickou. Dynamická simulace spočívá v jednorázovém spojení všech souborů projektu do jednoho, který je následně prováděn jádrem *OPNET Modeleru*. Statická simulace umožňuje předpřipravit simulační soubor pomocí nástroje `op_mksim`. Toto řešení je vhodné, pokud chceme rozsáhlý projekt řešit v různých scénářích a nechceme ručně provádět změny parametrů a znovu celý projekt pro každý scénář kompilovat.

Možnosti ladění a trasování prováděné simulace jsou opět velice rozsáhlé. Lze například nastavit části kódu, které se vykonávají pouze v režimu ladění. Můžeme tímto způsobem pro část kódu nastavit vypisování obsahu paketů. Ladící režim lze ve zdrojovém kódu vyvolat i nastavením speciálních ladících bodů.

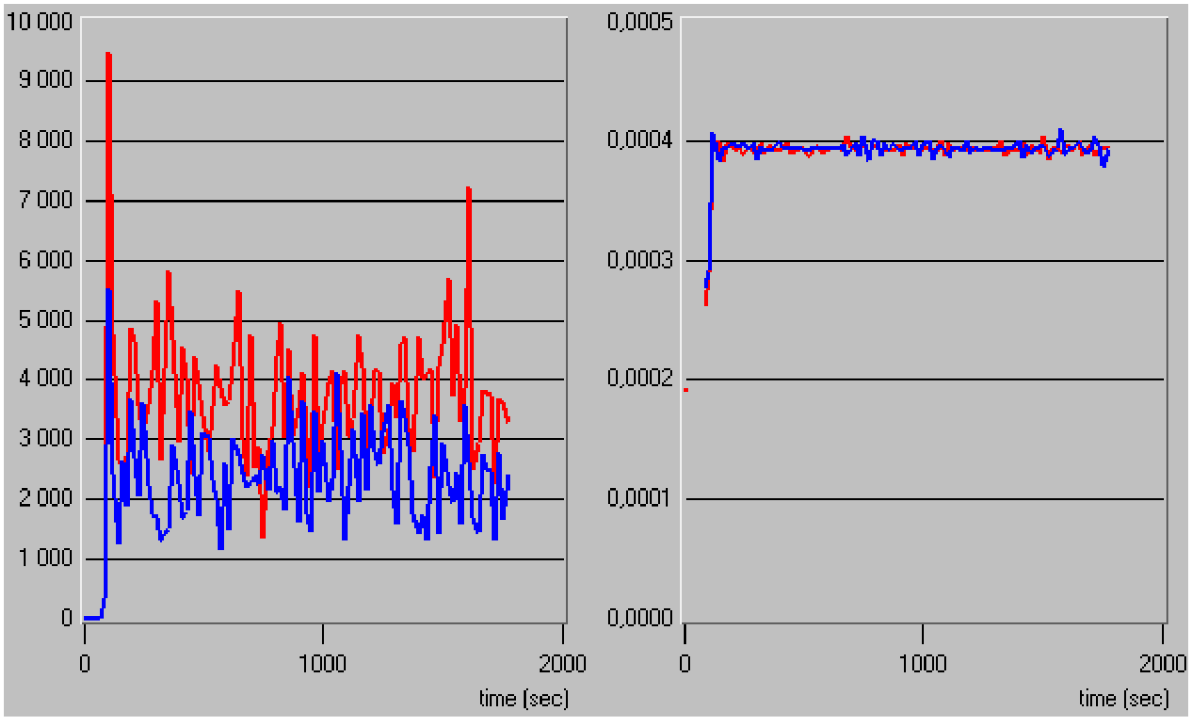
Analýza výsledků může být prováděna v prohlížeči animací *(Animation viewer)* nebo v prohlížeči výsledků *(rozhraní View Result)*. Pro analýzu výsledků je třeba nejdřívě nastavit, která data se mají zaznamenávat. *OPNET Modeler* má mnoho různých předdefinovaných voleb pro záznam nejrůznějších údajů. A to od záznamu aktivity nějakého procesu, až po počet zasílaných bytů.

Jak nástroj funguje jsem si vyzkoušel na modelu podnikové sítě, která používá hvězdicovou topologii pro připojení pracovních stanic, jeden server a přepínač od firmy *3COM*. Tento příklad demonstruje primární účel tohoto nástroje, analýzu *IP* protokolů. Podnik připravuje expanzi sítě o další hvězdicovou topologii, která by bylo spojena s předcházející sítí routerem *Cisco 2514*. Firma chce zjistit, jakým způsobem se změní zatížení serveru a zpoždění sítě.



Obr. 5.2: Schéma plánovaného rozšíření firemní sítě

Síť jsem snadno namodeloval a zjistil jsem následující výsledky, které jsem zanesl do grafů (obr. 5.3). Červeně je znázorněna firemní síť po rozšíření a modře je znázorněna firemní síť před rozšířením.



Obr. 5.3: Graf zatížení serveru a graf doby reakce sítě na požadavek

Vlevo je znázorněno zvýšení zatížení serveru. Vpravo je znázorněna doba odpovědi, která se prakticky nezvětšila. Znamená to, že plánované rozšíření sítě nezpůsobí ztrátu výkonnosti a lze ji proto doporučit.

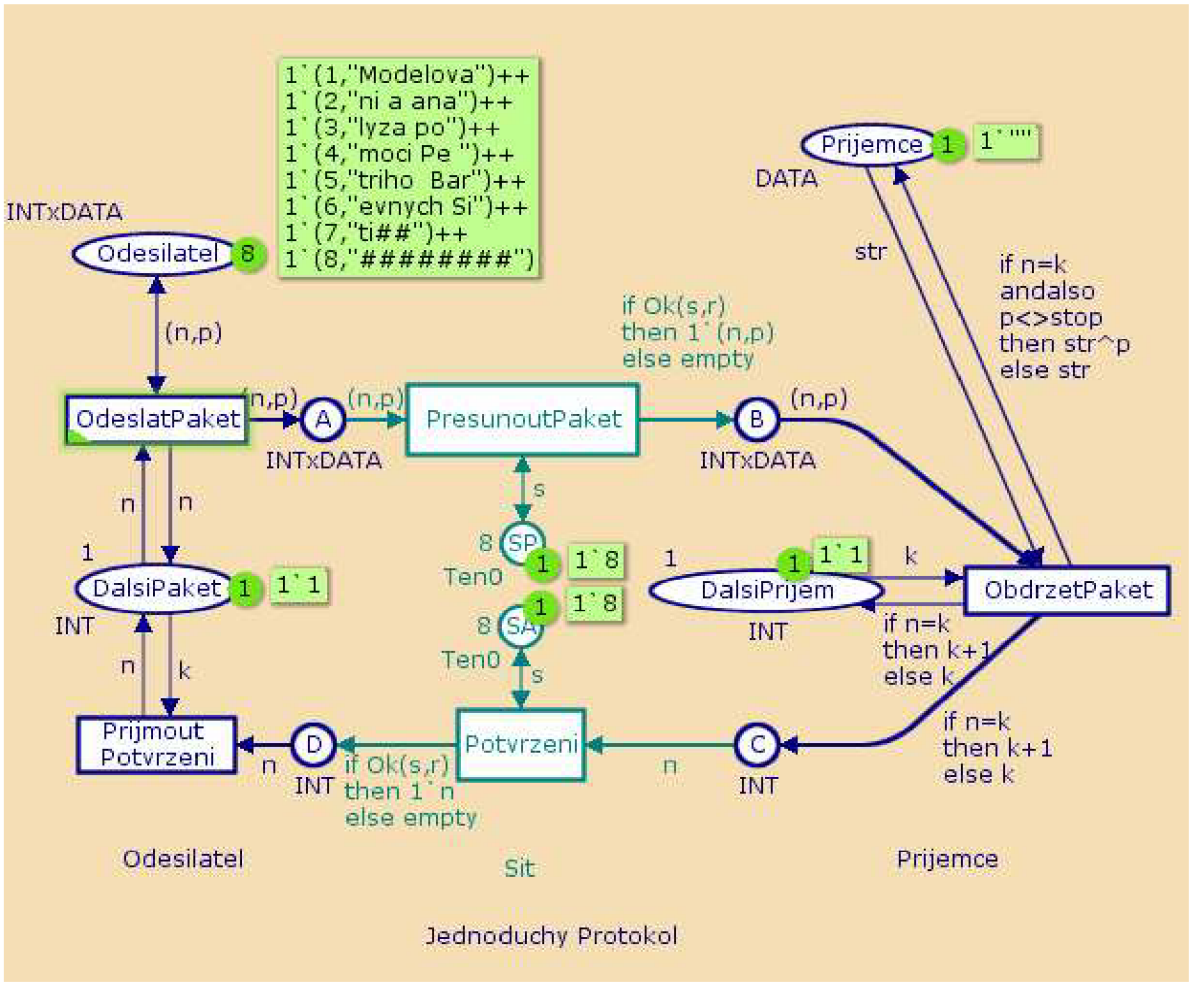
Závěr:

OPNET Modeler je mocný nástroj pro analýzu simulací z oblasti sítí s rozsáhlými možnostmi simulace, analýzy a prostředků pro návrh systémů. Ovládání nástroje je poměrně intuitivní, ale vzhledem k rozsáhlým možnostem nástroje, bude pro uživatele zpočátku zřejmě problematické, orientovat se v nesmírném množství tabulek a nastavení, které nástroj podporuje. Nástroj doporučuji pro řešení komplexních problémů, jako je zatížení velkých sítí apod. Slabší je to s analýzou bezpečnostních protokolů, které se dají také analyzovat, ale není to hlavním cílem tohoto nástroje. Názorným příkladem takové analýzy je analýza *VISA*[42] protokolu [70], ale i zde není výsledkem korektnost protokolu, ale jen měření zatížení serverů a doba reakce na autentizační výzvu při rozšíření sítě.

[42] adVanced Inter-System Authentication

5.3 Implementace v nástroji CPN Tools

Dalším zajímavým nástrojem je nástroj *CPN Tools 2.2.0*, který jsem si stáhl a vyzkoušel v rámci akademické licence. Nástroj pracuje na modelování systémů pomocí Petriho sítí. Pro modelování jsem si vybral jednoduchý protokol, pomocí kterého odesilatel posílá pakety k příjemci. Komunikační médium může ztratit pakety nebo může způsobit, že pakety přijdou ve špatném pořadí. Pakety připravené k odeslání přes uzel *"OdeslatPaket"* jsou umístěny v horním levém rohu grafu v uzlu *"Odesilatel"*. Každý paket obsahuje číslo paketu a datový obsah paketu. Uzel *"DalsiPaket"* představuje číslo paketu, který má být odeslán. Toto číslo je iniciováno na 1 a inkrementuje se každým dalším potvrzením o přijetí paketu od příjemce. Přijaté pakety jsou umístěny na uzlu *"ObdrzetPaket"* v pravém horním rohu grafu. Uzel *"PresunoutPaket"* transportuje přesune paket po síti z uzlu *A* do uzlu *B*. Uzel *"Potvrzeni"* přesouvá potvrzující paket z uzlu *C* do uzlu *D*.



Obr. 5.4: Jednoduchý protokol v nástroji *CPN Tools*

Cílem analýzy je přijmout textový řetězec *"Modelovani a analyza pomoci Petriho Barevnych Siti"*, který vznikne konkatenací obsahu jednotlivých paketů.

Provedl jsem 4 simulace nad stejnými daty a dospěl jsem k různým délkám simulace, které ukazuje následující tabulka.

Číslo simulace	**Počet kroků simulace**
1.	231
2.	204
3.	146
4.	494

Při analýze je neocenitelnou pomůckou ukládání logu o simulaci do textového souboru, který může být následně analyzován. Ukládají se však jen strohé informace o tom, ve kterém uzlu se zrovna simulace nachází. Největším přínosem je zřejmě grafické rozhraní, které je velmi praktické a usnadňuje modelování. Vytknout se mu dá jen to, že po otevření více rolovacích seznamů v levé části obrazovky, nelze přistupovat k položkám mimo obrazovku, dokud se některé rozvinuté seznamy opět neuzavřou.

Závěr:

V nástroji *CPN Tools* lze v přehledném a praktickém uživatelském rozhraním modelovat různé systémy. Nástroj je vhodný i pro analýzu bezpečnostních protokolů. *CPN Tools* našel slabinu v *IKE protokolu*, *SSL protokolu* a *SET protokolu*. Popis a analýza těchto protokolů v nástroji *CPN Tools* by vyžadovala detailnější přípravu a více prostoru, která není vzhledem k podstatě práce možná. Ovládací panel obsahuje hodně prvků, které usnadňují a zpřehledňují simulaci. Lze například snadno nastavit délku simulace a časovou prodlevu mezi jednotlivými kroky. Průběh simulace se přehledně zobrazuje v CPN grafu a výpis simulace lze přesměrovat do textového souboru pro pozdější analýzu. V porovnání s jinými nástroji jsou však tyto informace poměrně strohé a prakticky nepoužitelné.

5.4 Implementace v nástroji UPPAAL

Zajímavou volbou je i použití nástroje *UPPAAL 4.0.5*, který opět naleznete na přiloženém CD. Jedná se o volně dostupný nástroj pro nekomerční používání. Nástroj vyžaduje nainstalované *JRE 2.0* na cílové platformě. Pro ověření funkčnosti a použitelnosti nástroje jsem si vybral dva modely. Prvním z nich je zjednodušená verze protokolu *Needham-Schroeder* a druhým demonstračním modelem je ovládání semaforu na železničních přejezdech, které je pro využití nástroje *UPPAAL* typičtější.

Nástroj *UPPAAL* se skládá ze 3 různých rozhraní:

- **Editor** – obsahuje nástroje pro definici proměnných, editaci šablon, přiřazení šablon modelům a definici výsledného systému z dílčích modelů pro simulaci a verifikaci.
- **Simulátor** – umožňuje krokovat vývoj stavů systému, graficky zobrazovat stavy systému a vypisovat hodnoty systémových proměnných příslušných stavů systému.
- **Verifikátor** – umožňuje verifikovat vlastnosti systému pomocí temporální logiky, zaznamenat a poslat nalezený stav do simulátoru,…

5.4.1 *Needham-Schroeder* v nástroji *UPPAAL*

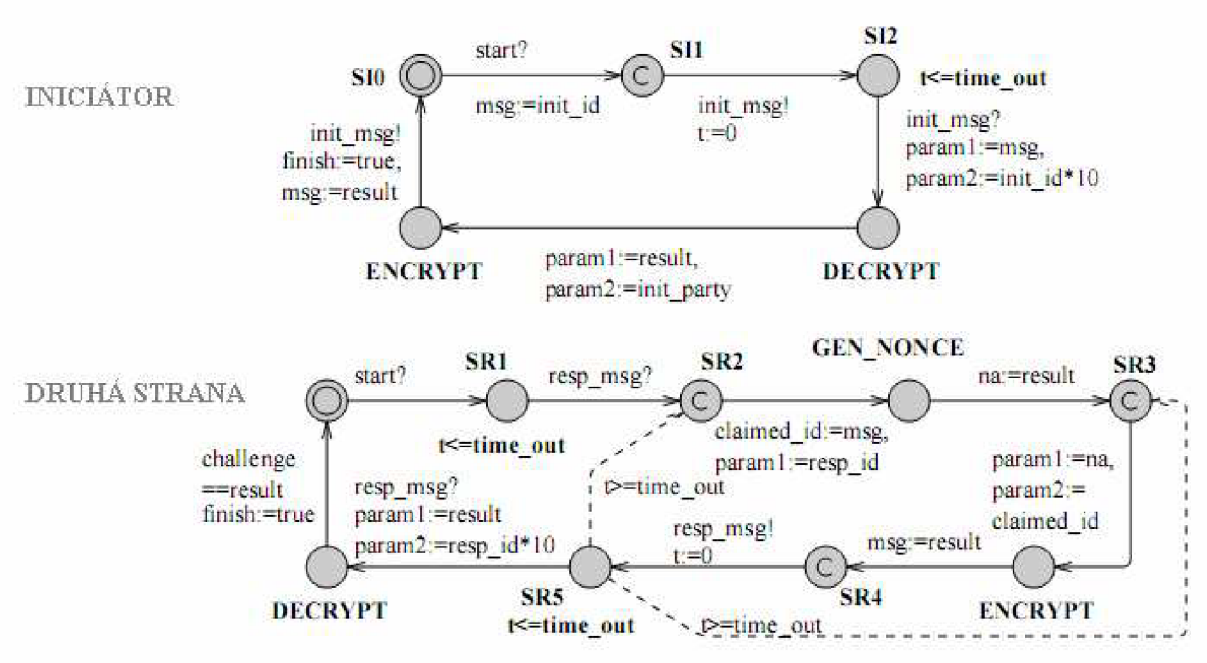
Tato zjednodušená verze protokolu slouží pouze k autentizaci subjektu *A* vůči *B*. Zjednodušeně ji lze popsat ve třech bodech [67]:

A posílá *B* svou identitu
1: A -> B: A

B posílá *A* nonce *B* zakódované veřejným klíčem *A*
2: B -> A: $\{N_B\}_{Ka}$

A posílá *B* nonce *B* zakódovaný veřejným klíčem *B*
3: A -> B: $\{N_B\}_{Kb}$

Nejprve vytvoříme modely časovaných automatů pro *A* a druhou stranu *(B resp. útočník)*. Tyto modely jsou na následujícím obrázku:



Obr. 5.5: Modely časovaných automatů pro *A* a pro *B (resp. M)*

Nyní specifikujeme kontrolní vlastnost pomocí temporální logiky:

```
// Tato vlastnost je platná, pokud B dokončí provádění, ale nezískal
// správnou představu o identitě A
AUT = E <> Responder.finish and Responder.claimed_id != resp_party
```

UPPAAL objevil dva možné útoky na tento protokol. Prvním je útok ze středu, kdy *A* i *B* nevědomky komunikují s útočníkem. Stejný útok je popsaný již v části 5.1. Útok je specifikován následující skupinou pravidel:

$$A \rightarrow I: A$$

$$I(A) \rightarrow B: A$$

$$B \rightarrow I(A): \{N_B\}_{Ka}$$

$$I \rightarrow A: \{N_B\}_{Ka}$$

$$A \rightarrow I: \{N_B\}_{Ki}$$

$$I(A) \rightarrow B: \{N_B\}_{Kb}$$

Řešení: Možným řešením je zpřísnit požadavky na dobu odezvy. Poté je útok tohoto typu neproveditelný.

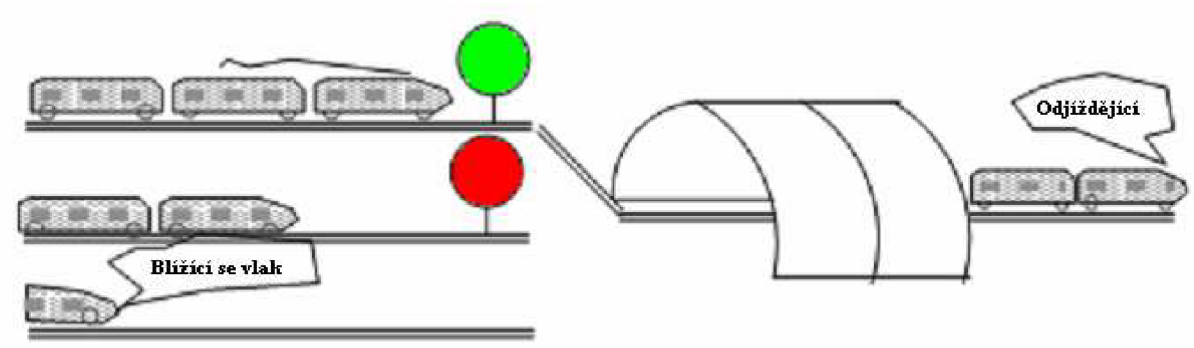
Druhému útoku však časová omezení nezabrání navzdory tomu, že je podstatně jednodušší. Jedná se o útok přehráváním, který je popsán následující trojicí pravidel:

$$I(B) \rightarrow B: B$$
$$B \rightarrow I(B): \{N_B\}_{Kb}$$
$$I(B) \rightarrow B: \{N_B\}_{Ki}$$

Při tomto útoku přesvědčí útočník subjekt *B*, že komunikuje sám se sebou, což ale není pravda. Komplikovanější kryptografické protokoly vyžadují podstatně více prostoru pro analýzu. Stručně proto nyní představím typičtější příklad využití nástroje *UPPAAL*, kterým je řízení systému pracujícím v reálném čase pro ovládání semaforů v železniční dopravě.

5.4.2 Ovládací systém pro semafory v nástroji *UPPAAL*

Problematiku popisuje následující obrázek:



Obr. 5.6: Ovládací systém semaforu před vjezdem do tunelu

Obrázek představuje situaci, kdy čtyři různé vlaky řízené semaforem vjíždějí postupně do tunelu. Tunelem může projíždět jen jeden vlak a další nemůže dostat zelenou, dokud předcházející vlak tunel neopustí. Vlaky informaci o své poloze zasílají do ovladače semaforu. Cílem mojí simulace je otestovat funkčnost navrženého semaforu. Tento model je implementován pomocí dvou typů automatů, ovladače semaforu a automatů vlaků [68].

Provedl jsem následující verifikace vlastností pomocí temporální logiky a výsledky jsem zapsal do tabulky:

Výraz v temporální logice	Význam výrazu	Výsledek
`A[] not deadlock`	V systému nemůže dojít k uváznutí.	✓
`E<> Gate.Occ`	Brána korektně přijímá a ukládá zprávy od přicházejících vlaků do front.	✓
`A[] forall (i : id_t) forall (j : id_t) Train(i).Cross && Train(j).Cross imply i == j`	Do tunelu může vjet v jeden okamžik je jeden vlak.	✓
`E<> Train(0).Cross and Train(1).Stop`	Vlak 0 vjíždí do tunelu a vlak 1 stojí na semaforu.	✓
`E<> Train(0).Cross and (forall (i : id_t) i != 0 imply Train(i).Stop)`	Vlak 0 vjíždí do tunelu a ostatní vlaky stojí na semaforu.	✓
`A[] Gate.list[N] == 0`	Nemůže nikdy být *n* vlaků čekajících na semafor, tj. pole s vlaky nemůže přetéci.	✓
Train[0].Appr → Train[0].Cross	Nedochází ke stárnutí během čekání na semaforu.	✓

Zdá se, že systém funguje bezchybně, ale i tak mě napadlo několik případů, kdy může výsledek skončit katastrofou:

- ✗ Pokud se nedodrží časová omezení pro průjezd vlaku tunelem nebo před vjetím do tunelu.
- ✗ Chyba lidského faktoru, kdy strojvedoucí vlaku ignoruje nebo přehlédne znamení semaforu
- ✗ Chyba v synchronizaci mezi modely jednotlivých vlaků a řídícím systémem semaforu

Závěr:

Implementace celé technologie v jazyce *Java* umožňuje platformovou nezávislost na úkor rychlosti, protože verifikace se musí provádět v *byte kódu* na *JVM*[43]. Simulace je pěkně graficky znázorněna a usnadňuje pochopení celé problematiky. Lze měnit i rychlost provádění simulace. Verifikace je poměrně jednoduchá a intuitivní. Existuje řada nastavení a omezení na procházený stavový prostor, který umožňuje optimální provedení verifikace.

[43] *JVM* – virtuální stroj pro provádění *byte kódu*.

6 Závěr

Cílem této diplomové práce bylo nalezení, analýza, porovnávání a praktická implementace co největšího počtu verifikačních nástrojů, relevantních pro analýzu převážně bezpečnostních protokolů. Toto kritérium opodstatňuje poměrně větší velikost práce.

V úvodních třech kapitolách byla představena problematika bezpečnostních protokolů a popsány některé známé protokoly. Ve stěžejní části práce jsem popsal verifikační nástroje, které jsem v rámci této práce nastudoval a analyzoval. U každého z těchto nástrojů jsem se snažil o zachování jisté štábní kultury, která se projevila hlavně v počáteční hlavičce popisující základní atributy příslušného nástroje a bodového seznamu výhod a nevýhod, které jsem vyzdvihl na konci popisu každého nástroje.

Hlavním myšlenkou práce bylo vytvořit poměrně rozsáhlý seznam nástrojů, ze kterého by si čitatel mohl vybrat nástroj, který je pro něj nejvhodnější. Vzhledem k tomu, že každý nástroj je postaven na trochu jiném principu, nedá se jednoznačně říci, který je nejlepší a který nejhorší. Rovněž se mi u všech nástrojů nepodařilo získat některé informace, což svědčí o tom, že nejsou volně přístupné, nebo že tyto nástroje existují jen na bázi konceptů ve vědeckých dokumentech. Většina nástrojů je však pro nekomerční účely volně dostupná a to s poměrně rozsáhlou dokumentací. Tento fakt přispěl k vytvoření databáze dokumentů související s problematikou jednotlivých nástrojů, které jsou po dohodě s vedoucím práce umístěny na přiloženém CD a to včetně volně dostupných distribucí nástrojů.

Nezanedbatelným přínosem práce je tedy usnadnění orientace ve verifikačních nástrojích a možnost nalézt veškeré potřebné informace přímo na CD.

K podpoření názornosti problému verifikace bezpečnostních protokolů slouží pátá kapitola, ve které jsem u vybraných nástrojů implementoval konkrétní verifikační problémy. Snažil jsem se vždy o verifikaci bezpečnostních protokolů, ale u některých nástrojů to nebylo možné kvůli velké složitosti implementace bezpečnostních protokolů v konkrétním nástroji nebo faktu, že verifikace bezpečnostních protokolů není pro tyto jinak velmi robustní nástroje stěžejní.

Na práci lze navázat například provedením detailnější analýzy vybrané skupiny nástrojů (například jen těch nástrojů, které jsou určeny pro komerční účely) a tyto nástroje poté aplikovat na společný verifikační problém.

Literatura

[1] *Historie internetu*, URL: <http://www.webdesign.paysoft.cz/clanky/2006/historie-internetu/> [cit. 2006-12-22].

[2] *Autentizace uživatelů v Česku*,URL: <http://www.computerworld.cz/cw.nsf/id/trendy_zpusoby_autentizace_uzivatelu_v_cr?OpenDocument&cast=1> [cit. 2006-12-23].

[3] *Bezpečnost dat v praxi*, URL: <http://casopis.systemonline.cz/2004-bezpecnost-dat-v-praxi.htm> [cit. 2006-12-26].

[4] *Kryptografie*, URL: <http://cs.wikipedia.org/wiki/Kryptografie> [cit. 2006-12-26].

[5] OČENÁŠEK, P.: *Verifikace bezpečnostních protokolů*. Brno 2003: Diplomová práce na FIT VUT, 55 s. Vedoucí diplomové práce Doc. Daniel Cvrček.

[6] OČENÁŠEK, P.: *Využití stromových automatů při verifikaci bezpečnostních komunikačních protokolů*. Brno 2004: Studie do předmětu PID na FIT VUT.

[7] STEFANOS G. - DIOMIDIS S. - PANAGIOTIS G. *Security Protocols over Open Networks and Distributed Systems: Formal Methods for their Analysis, Design, and Verification*, [cit. 2006-12-17]. Dostupné z URL:<http://www.spinellis.gr/pubs/jrnl/1997-CompComm-Formal/html/formal.htm>.

[8] BRACKIN S.: *An Interface Specification Language For Automatically Analyzing Cryptographic Protocols.* sndss, p. 40, 1997 Symposium on Network and Distributed System Security, 1997.

[9] BRACKIN S.: *Automatically Detecting Authentication Limitations in Commercial Security Protocols.* Dostupné z URL: <http://csrc.nist.gov/nissc/1999/proceeding/papers/p26.pdf>. [cit. 2007-01-13].

[10] BOGGIO. G - CAI. J. - CARENA A.:*NetworkDesigner-Artifex-OptSim:a suite of integrated software tools for synthesi and analysis of high speed network*, Září 2001. Dostupné po registraci z URL: <http://www.rsoftdesign.com/products/network_modeling/Artifex/pub_form.cfm?Author=Boggio,Burzio,Portinaro,Cai,Cerutti,Fumagalli,Tacca,Valcarenghi,Carena,Gaudino&Description=ArtifexOptSimIntegration>, [cit. 2007-03-13].

[11] GHEZZI C. - KEMMERER R.A. "*ASTRAL: an assertion language for specifying realtime systems". Proceedings of the 3rd European Software Engineering Conference*, Milan, Italy, 21-24 Oct. 1991. Dostupné z URL: <http://citeseer.ist.psu.edu/cache/papers/cs/21218/http:zSzzSzwww.cs.ucsb.eduzSz~rsgzSzASTRALzSzpubzSzesec91.pdf/ghezzi91astral.pdf>,[cit. 2007-05-13].

[12] KOLANO P.Z. - DANG Z. - KEMMERER R.A. *"The Design and Analysis of Real-Time Systems Using the ASTRAL Software Development Environment". Annals of Software Engineering, volume 7*, 1999. Bussum, The Netherlands: Baltzer Science Publishers. Dostupné z URL: <http://citeseer.ist.psu.edu/cache/papers/cs/11336/http:zSzzSzwww.cs.ucsb.eduzSz~kolanozSzpubzSzase99.pdf/kolano99design.pdf>,[cit. 2007-05-13].

[15] SONG D. - BEREZIN S. - PERRIG A. *Athena - a novel approach to efficient automatic security protocol analysis.* Dostupné z URL: <http://www.ece.cmu.edu/~dawnsong/papers/athena-jcs.pdf>, [cit. 2007-05-13].

[16] SONG D. *Athena: a New Efficient Automatic Checker for Security Protocol Analysis*, Computer Science Department, Carnegie Mellon University. Dostupné z URL: <http://citeseer.ist.psu.edu/cache/papers/cs/9287/http:zSzzSzwww.cs.cmu.eduzSz~skyxdzSzpaperszSzAthena.pdf/athena-a-new-efficient.pdf>, [cit. 2007-05-13].

[17] VIGANO L. *Automated Security Protocol Analysis with the AVISPA Tool*. Proceedings of the XXI Mathematical Foundations of Programming Semantics (MFPS'05), ENTCS 155:61--86, Elsevier. Dostupné z URL:<http://www.avispa-project.org/papers/avispa-mfps21.pdf>, [cit. 2007-05-13].

[18] CLARKE E.M. - JHA S. - MARRERO W. *Verifying Security Protocols with Brutus.* Dostupné z URL: <http://www.aladdin.cs.cmu.edu/papers/pdfs/y2000/brutus.pdf>, [cit. 2007-05-20].

[19] BRACKIN S. – MEADOWS C. – MILLEN J. *CAPSL interface for the NRL Protocol Analyzer.* Dostupné z URL: <http://citeseer.ist.psu.edu/cache/papers/cs/1763/http:zSzzSzwww.itd.nrl.navy.milzSzITDzSz5540zSzpublicationszSzCHACSzSz1999zSz1999meadows-ASSET99.pdf/brackin99capsl.pdf>, [cit. 2007-05-20].

[20] MILLEN J. – RUESS – DENKER. *The CAPSL Integrated Protocol Environment by Denker, Millen, and Ruess*; TIPE project final report and language reference document. Dostupné z URL: <http://citeseer.ist.psu.edu/cache/papers/cs/13015/http:zSzzSzwww.csl.sri.comzSz~denkerzSzpublzSzdiscexCAPSL99.pdf/denker00capsl.pdf>,[cit. 2007-05-20].

[21] MILLEN J. – DENKER. CAPSL *Integrated Protocol Environment by Denker and Millen*, for DARPA Information Survivability Conference, January 2000. 15-page overview. Dostupné z URL: <www.csl.sri.com/papers/denmil00/denmil00.ps.gz>, [cit. 2007-05-20].

[22] MILLEN J. – DENKER. CAPSL *Intermediate Language by Denker and Millen*, presented at FMSP '99. A ten-page summary of CIL concepts and issues. Dostupné z URL: <http://citeseer.ist.psu.edu/denker99capsl.html>, [cit. 2007-05-20].

[23] LOWE G. *Casper - A Compiler for the Analysis of Security Protocols*. Dostupné z URL: <http://citeseer.ist.psu.edu/lowe98casper.html>, [cit. 2007-05-20].

[24] RATZER – WELLS – LASSEN. *CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets.* Dostupné z URL: <http://wiki.daimi.au.dk/cpntools/_files/cpntools_pn03.pdf>, [cit. 2007-05-20].

[25] WELLS L. *Performance Analysis using CPN tools*. Dostupné po registraci z URL: <http://portal.acm.org/citation.cfm?id=1190171>,[cit. 2007-05-20].

[26] LAWRENCE J. *Practical Application of CSP and FDR to Software Design*, ISBN: 978-3-540-25813-1, vydalo nakladatelství Springer Berlin

[27] LOWE G. *Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR*. Dostupné z URL: <web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Papers/NSFDR.ps>, [cit. 2007-05-20].

[28] HENZINGER – PEI-HSIN HO – HOWARD WONG-TOI. *HyTech - A Model Checker for Hybrid Systém*. Dostupné z URL: <http://citeseer.ist.psu.edu/henzinger97hytech.html>, [cit. 2007-05-20].

[29] HENZINGER – PEI-HSIN HO – HOWARD WONG-TOI. *A User guide to HyTech* Dostupné z URL:<http://citeseer.ist.psu.edu/henzinger95user.html>, [cit. 2007-05-20].

[30] MILLEN J.K. *The Interrogator Model*, Dostupné po registraci z URL: <http://portal.acm.org/citation.cfm?id=882491.884227>, [cit. 2007-05-20].

[31] PAULSON L.C. *The Foundation of a Generic Theorem Prover*, ISSN:0168-7433

[32] NIPKOW T. – PAULSON L.C. – WENZEL M. *A Proof Assistant for Higher-Order Logic.* Dostupné z URL:<http://www4.in.tum.de/~nipkow/LNCS2283/tutorial.pdf>, [cit. 2007-05-20].

[33] LEAVENS G.T. – BAKER A.L. – RUBY C. *Preliminary design of JML: a behavioral interface specification language for java.* Dostupné z URL:<http://citeseer.ist.psu.edu/245976.html>, [cit. 2007-05-20].

[34] LEAVENS G.T. *Preliminary design of JML - a behavioral interface specification language for java.* Dostupné z URL:<http://citeseer.ist.psu.edu/245976.html>, [cit. 2007-05-20].

[35] YOVINE S. *KRONOS – A Verification Tool for Real Time Systems.* Dostupné z URL:<http://citeseer.ist.psu.edu/260450.html>, [cit. 2007-05-20].

[36] DAWS – OLIVERO – TRIPAKIS. *Verification of the FDDI protocol with Kronos*. Dostupné z URL: <http://www-verimag.imag.fr/TEMPORISE/kronos/dist/FDDI.tar.gz>, [cit. 2007-05-20].

[37] DAWS – YOVINE. *Verification of the Philips audio control protocol with KRONOS.* Dostupné z URL: <http://www-verimag.imag.fr/TEMPORISE/kronos/dist/Philips.tar.gz>, [cit. 2007-05-20].

[38] LEDUC G. – GERMEAU F. *Verification of security protocols using LOTOS-method and application*. Dostupné z URL:<http://citeseer.ist.psu.edu/leduc00verification.html>, [cit. 2007-05-20].

[39] BODEI C. – BUCHHOLTZ M. – DEGANO P. *Control Flow Analysis Cand Find News Flaw Too* In Proc. WITS'04, Barcelona, 2004.

[40] BUCHHOLTZ M. *User's Guide for the LySatool version 2.01.* Dostupné z URL: <http://www2.imm.dtu.dk/cs_LySa/lysatool/lysatool-2.01.pdf>, [cit. 2007-05-20].

[41] University of Edinburgh, Division of Informatics. *Elyjah: A security analyzer for Java implementations of communications protocols.* Dostupné z URL: <http://homepages.inf.ed.ac.uk/stg/software/elyjah/elyjah.pdf>, [cit. 2007-05-20].

[42] ESPARZA J. – SCHROTER C. – SCHWOON S. *Web-based documentation for Model-Checking Kit.* Dostupné z URL:<http://www.fmi.uni-stuttgart.de/szs/tools/mckit/mckit-web-030623.tar.gz>, [cit. 2007-05-20].

[43] DILL. D. – DREXLER A. – HU A. – YANG C. *Protocol Verification as a Hardware Design Aid* Dostupné z URL:<verify.stanford.edu/dill/PAPERS/verification/DDHY92.ps>, [cit. 2007-05-20].

[44] MEADOWS C. *A Model of Computation for the NRL Protocol Analyzer.* Dostupné z URL: <http://chacs.nrl.navy.mil/publications/CHACS/1994/1994meadows-foundations.pdf>, [cit. 2007-05-20].

[45] MEADOWS C. *The NRL Protocol Analyzer: An Overview.* Dostupné z URL: <chacs.nrl.navy.mil/publications/CHACS/1994/1994meadows-pap.ps>, [cit. 2007-05-20].

[46] KESHAV S. *REAL: A Network Simulator.* Technická zpráva CSD-88-472. Publikováno v roce 1988.

[47] CAVIN D. – SASSON Y. – SCHIPER A. *On the Accuracy of MANET Simulators.* Dostupné z URL: <http://infoscience.epfl.ch/getfile.py?mode=best&recid=49940>, [cit. 2007-05-20].

[48] CHANG X. *Network simulations with opnet.* ISBN:0-7803-5780-9. Publikováno v roce 1999.

[49] SALAH K. *An OPNET-based simulation approach for deploying VoIP.* ISSN:1099-1190. Publikováno v roce 2006.

[50] LUCIO G. – JAMMEH E. *OPNET Modeler and Ns-2: Comparing the Accuracy Of Network Simulators for Packet-Level Analysis using a Network Testbed.* Dostupné z URL: <http://privatewww.essex.ac.uk/~fleum/weas.pdf>, [cit. 2007-05-20].

[51] BRACKIN S. *A HOL Formalization of CAPSL Sémantice.* Dostupné z URL: <http://csrc.nist.gov/nissc/1998/proceedings/paperF21.pdf>, [cit. 2007-05-20].

[52] BRACKIN S. *A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols.* Dostupné z URL: <http://citeseer.ist.psu.edu/brackin96hol.html>, [cit. 2007-05-20].

[53] KWIATKOWSKA M. – NORMAN G. – PARKER D. *PRISM: Probabilistic Symbolic Model Checker.* ISSN 0302-9743.

[54] KWIATKOWSKA M. – NORMAN G. – SPROSTON J. – WANG F. *Symbolic Model Checking for Probabilistic Timed Automata.* Dostupné z URL:<http://citeseer.ist.psu.edu/642174.html>, [cit. 2007-05-20].

[55] HINTON A. – KWIATKOWSKA M. – NORMAN G. – PARKER D. *PRISM: A Tool for Automatic Verification of Probabilistic Systems.* Dostupné z URL: <http://www.cs.bham.ac.uk/~dxp/talks/dave-tacas06.pdf>, [cit. 2007-05-20].

[56] HOLZMANN G.J. *Design and Validation of Computer Protocols*, 1990. Dostupné z URL: <http://www.spinroot.com/spin/Doc/Book91_PDF/Design_and_Validation_1991.pdf>, [cit. 2007-05-20].

[57] HOLZMANN G.J. *The Model Checker SPIN.* Dostupné z URL: <http://www.spinroot.com/spin/Doc/ieee97.pdf>, [cit. 2007-05-20].

[58] BEREZIN S. – GROCE A.*SyMP - The User's Guide.* Dostupné z URL: <www.cs.cmu.edu/~modelcheck/symp/userguide.ps.gz>, [cit. 2007-05-20].

[59] BEREZIN S. – GROCE A. *SyMP – The Hacker's Manual.* Dostupné z URL: <http://citeseer.ist.psu.edu/495729.html>, [cit. 2007-05-20].

[60] BENGTSSON J. – LARSEN K. – LARSSON F. – PETTERSSON P. – YI W.*UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems.* Dostupné z URL: < http://www.brics.dk/RS/96/58/BRICS-RS-96-58.pdf>, [cit. 2007-05-20].

[61] BEHRMANN G. – DAVID A. – LARSEN K. *A Tutorial on Uppsal.* Dostupné z URL: <http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf>, [cit. 2007-05-20].

[62] BRAYTON R. – HATCHEL G. – AZIZ A. *VIS: A System for Verification and Synthesis.* Dostupné z URL:<http://citeseer.ist.psu.edu/30782.html>, [cit. 2007-05-20].

[63] BRAYTON R. – HATCHEL G. – AZIZ A. *VIS.* ISBN:3-540-61937-2. Publikováno v roce 1996.

[64] STERN U. – DILL D.L.*Using Magnetic Disk instead of Main Memory in the Murφ Verifier*, Computer Science Department, Stanford University. Dostupné z URL: <verify.stanford.edu/PAPERS/SD98.ps>, [cit. 2007-05-20].

[65] *PRISM Manual – Version 3.1.* Dostupné z URL: <http://www.cs.bham.ac.uk/~dxp/prism/doc/manual.pdf>, [cit. 2007-05-20].

[66] CIGÁNEK T.: *Simulace real-time komunikace pomocí OPNET Modeler. 2004*: Diplomová práce na ČVUT, 70 s. Vedoucí diplomové práce O.Dolejš.

[67] CORIN R. – ETALLE S. *Timed Analysis of Security Protocols*. 2005. Dostupné z URL: <http://www.msr-inria.inria.fr/~corin/pubs/cor05s.pdf>, [cit. 2007-05-20].

[68] MIKUCIONIS M. – SASNAUSKAITE E.*On-the-fly Testing Using UPPAAL*. Jaro 2003: Diplomová práce na Aalborg University, 87 s. Vedoucí diplomové práce B.Nielsen.

[69] KRIVOŠ-BELLUŠ R.*Formálna analýza bezpečnosti kryptografických protokolov*. Košice 2002: Diplomová práce na Přírodovědecké fakultě Univerzity Pavla Jozefa Šafárika.. Vedoucí diplomové práce J. Jirásek.

[70] BHARATHAN A. – McNAIR J.*An OPNET Modeler Simulation Study of the VISA Protocol for Multi-Network Authentication.* Dostupné z URL:<http://www.wam.ece.ufl.edu/visa/paper.pdf>, [cit. 2007-05-20].

Seznam zkratek

AAPA	Automatic Authentication Protocol Analyzer
ATM	Asynchronous Transfer Mode
AVISS	Automated Validation of Infinite-State Systems
BAN	Burrows Abadi Needham
BGNY	Brackin Gong Needham Yahalom
BISL	Behavioral Interface Specification Language
CCS	Communication Concurrency System
CIL	CAPSL Intermediate Language
CPN	Coloured Petri Nets
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSP	Communicating Sequential Processes
CTL	Computational Tree Logic
ERD	Entity Relationship Diagram
FDDI	Fiber Distributed Data Interface
FDR	Failures-Divergence Refinement
GNU	GNU's is not Unix
GNY	Gong, Needham, Yahalom
GUI	Graphics User Interface
HOL	High Order Logic
HLPSL	High Level Protocol Specification Language
HTML	Hyper Text Mark-up Language
HTTP	Hyper Text Transfer Protocol
IF	Intermediate Format
IKE	Internet Key Exchange
ISL	Interface Specification Language
JML	Java Modeling Language
LOTOS	Language Of Temporal Ordering Specification
OPNET	Optimum Network Performance
PCTL	Probabilistic Computation Tree Logic
PDL	Protocol Description Logic
NRL	Naval Research Laboratory
NPA	NRL Protocol Analyzer
RSA	Rivest Shamir Adleman
RTP	Real-time Transport Protocol
SKKE	Symmetric-Key Key Establishment
SPIN	Simple Promela INterpreter
SRM	Scalable Reliable Multicast
SyMP	Symbolic Model Prover
TCTL	Timed Computation Tree Logic
TCP	Transmission Control Protocol
TMN	Telecommunications Management Network
VIS	Verification Interacting with Synthesis

A – Dostupnost nástrojů

Název nástroje	Typ licence	Cena	Platformy
	Nekomerční, Akademická, Komerční		
AAPA2		**Nepodařilo se zjistit**	
Artifex 4.4-2	**-, - , +**	Jedna licence 5000€, každá další pro výukové účely 1000€. Lze stáhnout 30-denní plně funkční zkušební verzi.	Windows 2000/XP, Unix, Linux
Astral SDE beta verze 1.0	**+, +, -**	Produkt je o začátku vyvíjen pod *GNU* licencí. Přístup k webu nástroje musí být domluven – lze z *IP* adres *FIT VUT*.	Linux, Unix,SunOS
Athena - integrována v SyMP beta 0.3	**+, +, -**	Nástroj je zcela zdarma a lze stáhnout v rámci nástroje *SyMP beta 0.3*.	Libovolná platforma podporující gcc a SML/NJ
Avispa 1.1	**+, +, -**	Každý zaregistrovaný uživatel si může stáhnout *AVISPU* pro jím specifikované účely.	Linux, Mac OS
Brutus		**Nepodařilo se zjistit**	
CAPSL		**Nepodařilo se zjistit**	Linux, Solaris, SunOS a FreeBSD
Casper 1.5	**+, +, -**	Nástroj je možné zcela zdarma používat pro komerční i nekomerční účely. Na CD je verze pro Linux a rozhraní CasperFDR.	Linux, Solaris, SunOS a FreeBSD
CPN Tools 2.2.0	**+, +, +**	Nástroj je možné zcela zdarma používat pro komerční i nekomerční účely. K instalaci musíte získat *CPN* licenci. V rámci akademické licence jsem stáhnul verzi pod Windows i Linux (obojí na CD).	Windows, Linux
FDR2.82	**+, +, -**	Pro plnou aktivaci balíků je nutné stáhnout licenci. Na CD je verze pro Linux. Nástroj je zcela zdarma.	FreeBSD, MacOS, Linux, Solaris
Hytech 1.04	**+, +, -**	Na CD je verze pro Linux.	Linux, Solaris, SunOS, Windows (cygwin)
Interrogator		**Nepodařilo se zjistit**	Unix, MacOS
Isabelle 2005	**+, +, -**	*Isabelle* je volně dostupná ke stažení pod *BSD* licencí. Na CD je verze pro Linux.	Linux, Solaris, MacOS
JVM 5.4	**+, +, -**	Nástroj je vyvíjený pod *GPL* licencí, tj. zcela zdarma.	Nezávislý na platformě

LCPP 5.14	**+, +, -**	Nástroj je zcela zdarma. Distribuce třetím stranám však vyžaduje speciální povolení. Na CD je verze pro Linux.	Linux, Windows
Kronos 252	**+, +, -**	Distribuce jsou určeny pro nekomerční účely. V případě komerčního využití je třeba kontaktovat autory.	Linux, Solaris 5.7
Lotos	**+, +, -**	Existuje mnoho různých nástrojů využívajících principů *Lotos*. Jsou většinou dostupné z akademických webů.	Linux, Solaris
LySatool 2.02	**+, +, -**	Produkt je zdarma pro nekomerční použití. Pro komerční je třeba kontaktovat autory. Na CD je verze pro Linux.	Linux, MacOS, Windows
mckit 1.4.0	**+, +, -**	Produkt je zdarma pro nekomerční použití. Na CD je verze pro Linux. Autoři doporučují používat společně s nástrojem *SPIN*.	Linux, Solaris
Murphi 3.1	**+, +, -**	Produkt je zcela zdarma použitelný pro komerční i nekomerční účely.	Linux, Windows (cygwin)
NPA		**Nepodařilo se zjistit**	
NS-2.31	**+, +, -**	Nové verze jsou vyvíjeny pod *GPL* licencí, tj. zcela zdarma.	FreeBSD, Linux, SunOS, Solaris
OPNET Modeler	**-, +, +**	Komerční verze s 30 licencemi stojí zhruba 2 938€.	Windows, Unix
PDL		**Nepodařilo se zjistit**	
Prism 3.1	**+, +, -**	Nyní pod *GPL* licencí, tj. zcela zdarma. Na CD je verze pro Windows.	Linux, Solaris, MacOS, Windows
SPIN 4.2.9	**+, +, -**	*SPIN* je volně dostupný pro vědecké a studijní účely. V případě komerčního užití je nutné splnit licenční podminky. Na CD je verze pro Windows i pro Linux.	Linux, Windows
SyMP beta 0.3	**+, +, -**	Produkt je zdarma pro nekomerční použití. Pro komerční je třeba kontaktovat autory.	Libovolná platforma podporující gcc a SML/NJ
UPPAAL 4.0.5	**-, +, +**	Komerční licence a cena lze domluvit přes mail. Zdarma je pouze akademická verze pro nekomerční účely.	UPPAAL je vyvinut v jazyce Java, čili je nezávislý na platformách.
VIS 2.1	**+, +, -**	Produkt je zdarma pro nekomerční použití. Pro komerční je třeba kontaktovat autory.	Linux, Windows, Solaris

B – Needham Schroeder v jazyce Promela

```
/* Needham-Schroeder public-key protokol v jazyce Promela */

mtype = {msg1, msg2, msg3, alice, bob, intruder,
         nonceA, nonceB, nonceI, keyA, keyB, keyI, ok};

typedef Crypt { /* the encrypted part of a message */
  mtype key, d1, d2;
}

/* Zprava je modelovana jako n-tice
     msg# ( adresat, zakodovana_data )
*/
chan network = [0] of {mtype, /* msg# */
                       mtype, /* receiver */
                       Crypt};

/* Partneri udavaji agenta, s kterym je navazana komunikace*/
mtype partnerA, partnerB;
mtype statusA, statusB;

/* .Udava zda se utocnik dostal k nonces */
bool knowNA, knowNB;

active proctype Alice() { /* Alice pro jeden beh programu */
  mtype  partner_key, partner_nonce;
  Crypt  data;

  if /* Nedeterministicky zvoli partnera komunikace */
  :: partnerA = bob; partner_key = keyB;
  :: partnerA = intruder; partner_key = keyI;
  fi;

  d_step {
    /* Vytvori zpravu msg a posle ji partnerovi */
    data.key = partner_key;
    data.d1 = alice;
    data.d2 = nonceA;
  }
  network ! msg1(partnerA, data);

  /* ceka na odpoved */
  network ? msg2(alice, data);
  end_errA:

/* ujisti se, zda partner rozkodoval zpravu a precetl si
nonce*/
```

```
  (data.key == keyA) && (data.d1 == nonceA);
  partner_nonce = data.d2;

  d_step {
    /* odpovi treti zpravou a uspesne skonci */
    data.key = partner_key;
    data.d1 = partner_nonce;
    data.d2 = 0;
  }
  network ! msg3(partnerA, data);
  statusA = ok;
} /* proctype Alice() */

active proctype Bob() { /* Bob pro jeden beh programu*/
  mtype partner_key, partner_nonce;
  Crypt data;

  /* ceka na zpravu msg1 od Alice */
  network ? msg1(bob, data);
  /* pokusi se zpravu rozlustit */
  end_errB1:
  (data.key == keyB);
  partnerB = data.d1;

  d_step {
    partner_nonce = data.d2;
    /* najde verejny klic partnera */
    if
    :: (partnerB == alice) -> partner_key = keyA;
    :: (partnerB == bob) -> partner_key = keyB;
    /* nemelo by se to stat */
    :: (partnerB == intruder) -> partner_key = keyI;
    fi;
    /* odpovi zpravou msg2 */
    data.key = partner_key;
    data.d1 = partner_nonce;
    data.d2 = nonceB;
  }
  network ! msg2(partnerB, data);

  /* ceka na msg3, zkontroluje klic a nonce a uspesne skonci*/
  network ? msg3(bob, data);
  end_errB2:
  (data.key == keyB) && (data.d1 == nonceB);
  statusB = ok;
}
```

```
active proctype Intruder() {
  /* Utocnik muze jednat nezavisle na protokolu, nasleduje
seznam akci, ktere muze provest
  */
  mtype msg;
  Crypt data, intercepted;
  mtype icp_type;   /* typ odposlechnute zpravy */

  do
  :: /* Posle msg1 pro B (jiny krok by nedaval smysl.
     */
     if /* prehraje odposlechnutou zpravu nebo posle vlastni*/
     :: icp_type == msg1 -> network ! msg1(bob, intercepted);
     :: data.key = keyB;
     if
     :: data.d1 = alice;
     :: data.d1 = intruder;
     fi;
     if
     :: knowNA -> data.d2 = nonceA;
     :: knowNB -> data.d2 = nonceB;
     :: data.d2 = nonceI;
     fi;
        network ! msg1(bob, data);
     fi;
  :: /* Posle msg2 to A. */
     if
     :: icp_type == msg2 -> network ! msg2(alice,
intercepted);
     :: data.key = keyA;
        if
     :: knowNA -> data.d1 = nonceA;
     :: knowNB -> data.d1 = nonceB;
     :: data.d1 = nonceI;
     fi;
     if
     :: knowNA -> data.d2 = nonceA;
     :: knowNB -> data.d2 = nonceB;
     :: data.d2 = nonceI;
     fi;
        network ! msg2(alice, data);
     fi;
  :: /* Posli msg3 pro B. */
     if
     :: icp_type == msg2 -> network ! msg3(bob, intercepted);
     :: data.key = keyB;
     if
     :: knowNA -> data.d1 = nonceA;
     :: knowNB -> data.d1 = nonceB;
     :: data.d1 = nonceI;
     fi;
```

```
        data.d2 = 0;
        network ! msg3(bob, data);
     fi;
  :: /* Prijme nebo odposlechne zpravu od A nebo B. Zkusi
rozbalit nonces. */
     network ? msg (_, data) ->
     if /* Uloz pro pozdejsi pouziti */
     :: d_step {
       intercepted.key = data.key;
       intercepted.d1 = data.d1;
       intercepted.d2 = data.d2;
          icp_type = msg;
     }
     :: skip;
     fi;
     d_step {
     if /* Pokusi se rozkodovat zpravu */
     :: (data.key == keyI) -> /* Mame novy nonce? */
        if
        :: (data.d1 == nonceA || data.d2 == nonceA) -> knowNA =
true;
           :: else -> skip;
        fi;
        if
        :: (data.d1 == nonceB || data.d2 == nonceB) -> knowNB =
true;
        :: else -> skip;
        fi;
     :: else -> skip;
     fi;
     }
  od;
}
```