コンピュータセキュリティ 19－8
（２００２．１２．２０）

一階述語論理を用いた脆弱性診断ツール

河内 清人† 北澤 繁樹† 中野 初美† 大越 丈弘† 藤井 誠司† 河木 理一‡
†三菱電機(株) 情報技術総合研究所 〒247-8501 神奈川県鎌倉市大船 5-1-1
‡三菱電機(株) 通信機製作所 〒661-8661 兵庫県尼崎市塚口本町 8-1-1
†E-mail: kawauchi@iss.isl.melco.co.jp

あらまし 脆弱性診断で一般的に使用されるセキュリティスキャナは, 脆弱点の列挙を行うのみであるため, それらを組み合わせて利用した場合の脅威や, 盗聴のような受動的な攻撃を受けた場合の脅威についての情報をツールから得ることはできない. そこで本稿では, 攻撃をスクリプトとして表現し, 一階述語論理上での自動推論に従って実行することで, 上記場合も含めて, 可能な限り攻撃者の振る舞いを模擬可能な脆弱性診断ツールを提案し, その有効性について述べる.

キーワード 不正アクセス, 脆弱性診断, 一階述語論理, Prolog

A Vulnerability Assessment Tool Using First-Order Predicate Logic

KAWAUCHI Kiyoto†, KITAZAWA Shigeki†, NAKANO Hatsumi†, OHKOSHI Takehiro†,
FUJII Seiji†, KAWAKI Motokazu‡
†Mitsubishi Electric Corporation, Information Technology R&D Center
‡Mitsubishi Electric Corporation, Communication Systems Center
†5-1-1, Ofuna, Kamakura, Kanagawa, 247-8501, JAPAN
‡8-1-1, Tsukaguchi-Honmachi, Amagasaki, Hyogo, 661-8661, JAPAN
†E-mail: kawauchi@iss.isl.melco.co.jp

Abstract Attackers often intrude their target sites by using one or more vulnerabilities found in victim hosts. Also, they can use viruses to put backdoors into target sites. Security scanners which are commonly used for security assessment cannot detect such risks described above, because they can just enumerate vulnerabilities. In this paper, we propose a vulnerability assessment tool, which can simulate whole possible activities taken by attackers. This is achieved by applying an inference engine based on first-order predicate logic to decide attack scenarios.

Keyword Unauthorized Access, Vulnerability Assessment, First-Order Predicate Logic, Prolog

1. 背景

近年, サイトの大規模, 複雑化にともない, はじめからセキュリティ上不備の無いネットワークを構築することはますます困難となりつつある. そのため, 自サイトにセキュリティ上の脆弱性が無いか調査する脆弱性診断はより重要な意味を持ってきている.

現在, 脆弱性診断において, セキュリティスキャナと呼ばれるツールを用いるのが一般的である. ISS 社の Internet Scanner[1], や nessus.org の公開

している Nessus[2]がその代表的な例である. しかし, これら既存のセキュリティスキャナは, 次の点で不十分と言える.

第一に, 既存のセキュリティスキャナが, 発見されたセキュリティホールとその一般的な深刻度を列挙するだけである, という点である.

ホールがサイトに与える真の脅威は, サイトの構成や, 他のホールとの組み合わせによって変化し得る. つまり, スキャナによって深刻度低と報告されたホールでも, 状況によっては, より深刻な攻撃に結びつく可能性がある訳である.

そのため, 真の脅威が判明しない限り, 検出されたホールを全て修正しなければ安全とは言えない事になる. もしホールの修正が既存のサービスの可用性に影響を与えるならば, 管理者はホールの真の脅威を自ら推測しなければならず, このことは, 管理者に多大な負担をかけることになる.

第二に, ホールの検出は, あくまでも能動的な攻撃によって行われるという点が挙げられる. 攻撃者は能動的攻撃に加え, ウィルスや盗聴, クロスサイトスクリプティング等の受動的な攻撃も利用する. このような攻撃まで含めたサイトの総合的な脆弱性は既存のツールでは評価できない.

これらの問題点の本質は, 既存のセキュリティスキャナが攻撃の一部(個々の脆弱点の発見)しか模擬できていない点にある.

そこで, 今回筆者らはこの問題に着目し, 脆弱点の発見だけでなく, 攻撃者の行為を全て模擬できるような脆弱性診断ツールの実現方式を検討した. さらに, 本ツールが新たなホールや手法に容易に追随できるだけの拡張を備えることについても検討を行った.

本稿は, 以下§2で本ツールの実現方式について説明する. §3でツールの実装について説明し, §4 で考察を行い§5で結論を述べる.

2. 実現方式

2.1. 用語定義

以降の議論で用いる用語について定義する.

- 攻撃. . . 何らかの目的を実現するために攻撃者によって行われる一連の行為
- 効果. . . 攻撃が成功したときに満たされる目的
- 事前条件. . . 攻撃が成功するために必要な前提となる条件

2.2. 攻撃の連鎖

前節の定義から, 以下の命題が成り立つ.

「事前条件」⇒「攻撃が成功」

「攻撃が成功」⇒「効果」

ここで, ある攻撃(A と呼ぶ)の事前条件を満足させるような効果を持つ別の攻撃(B と呼ぶ)が存在したとする. この場合, B が成功すれば, A も成功するはずである. つまり,「B の事前条件」⇒「A の効果」が成り立つ事になる.

同様に,「B の事前条件」を他の攻撃が実現し･･･と攻撃の連鎖が続いていき, 最終的には, ある初期条件さえ満たされていれば, 当初の効果を得ることができる.

2.3. 脆弱性診断

前節の議論を攻撃者の立場から解釈すれば,「攻撃者は初期条件(前提知識)から最終目的を達成するために, 必要な一連の攻撃を推論し実行する」と考えることも可能である.

そこで, 筆者らは, この過程を自動的に実行することで攻撃者の行為を模擬する脆弱性診断ツールを提案する. 以下にその実現方式について説明する.

2.4. 攻撃連鎖の推論の実現方式[4]

§2.1 で示した命題には, 実際には攻撃対象ホストアドレスや, ポート番号等のパラメータが含まれて

いる. そこで, 形式言語の一種であり, 変数を含んだ論理的な関係を表すのに適している一階述語論理を用いて命題を表現することにした.

一階述語論理上での推論を行うエンジンとして Prolog インタプリタを挙げることができる. Prolog インタプリタは, ある目標(Goal 節)を与えると, その目標が実現可能かどうか, 既にある知識(ルール)を用いて推論する機能を持つ.

この機能を利用することで, 最終的な「効果」に通じる一連の攻撃の発見を容易に実現できると考え, 本稿では Prolog を用いて前述の推論を行う.

2.5. 攻撃手順の記述方式

診断の際, 個々の攻撃は現実にパケットを送出したり, ファイルを操作したりしなければならない. そのため, 攻撃の手順の記述には, 手続き型のプログラミング言語である Perl[5]を用いる.

2.6. スクリプト

§1 でも述べたとおり, 本診断ツールを設計する上で, 診断機能の拡張性についても検討を行った.

そこで, 攻撃の 3 要素, すなわち「事前条件」, 「効果」, 「攻撃手順」を攻撃毎に一つのファイルにまとめ, ファイル単位で容易に追加, 変更が行えるようにした. このファイルを本ツールではスクリプトと呼ぶ.

以下の節では, 上記実現方式に基づいたツールの実装について詳しく述べる.

3. 実装

3.1. システム構成

3.1.1. 全体構成

本ツールは以前著者らが発表した脆弱性診断ツール[3]をベースにしており, 図 1 に示したように, 以下のコンポーネントで構成される.

- 検査クライアント
- シナリオ実行サーバ
- 踏み台エージェント

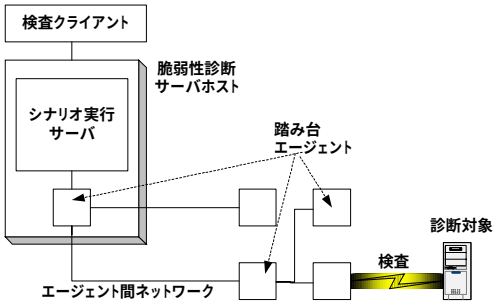

図 1 脆弱性診断ツール全体構成

検査クライアントは検査の実行の指示や, 検査結果の表示等の機能をユーザに提供する.

シナリオ実行サーバは, 本ツールの本体部分である. 踏み台エージェントを操作して, 診断を行う.

踏み台エージェントはパケットの送出等, 診断の際に実際に攻撃を行う. シナリオ実行サーバと異なるホスト上でも動作可能である.

3.1.2. シナリオ実行サーバ

シナリオ実行サーバの内部構成を図 2 に示す. 図に示すように, シナリオ実行サーバは以下の要素で構成される.

- クライアント通信制御
- 推論エンジン及び知識ストア
- スクリプト実行制御及びスクリプトストア
- 踏み台エージェント制御

クライアント通信制御は検査クライアントとのインタフェースを提供する機能である.

推論エンジン及び知識ストアは§2.3で述べたように, 述語論理に基づいて攻撃の連鎖を推論するためのコンポーネントである.

スクリプト実行制御はスクリプトストアに格納されているスクリプトの操作を行う. スクリプトはテキストファイルとしてスクリプトストアに格納されている.

踏み台エージェント制御は踏み台エージェントを管理するためのコンポーネントである. スクリプトに記

述された攻撃手順に従い, 踏み台エージェントの操作を行う.

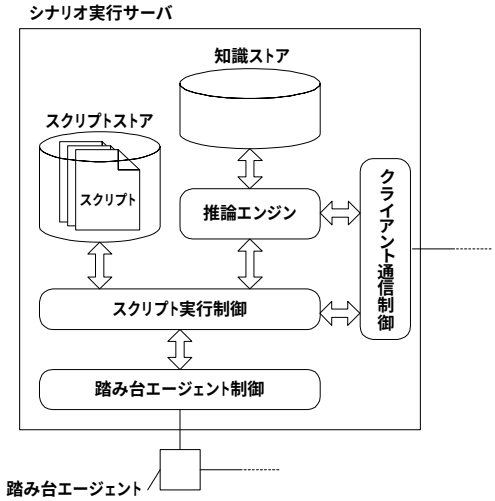

図 2 シナリオ実行サーバ内部構成

3.2. スクリプト

スクリプトはテキストファイルとして実現されており, 内部はプロパティ記述部, 動作記述部の二つの部分で構成されている.

プロパティ記述部には, 推論エンジンが診断のための推論を実行するのに必要な情報が記述され, 動作記述部は実際の攻撃手順が記述される. 以下各部分について説明する.

3.2.1. プロパティ記述部

プロパティ記述部には, 次の 4 種類の情報が記述される.

- 事前条件
- 効果
- 入力パラメータ列
- 出力パラメータ列

3.2.1.1. 事前条件

事前条件は, そのスクリプトの表す攻撃を実行するにあたって, 事前に満たされていなければならない条件を表す.

事前条件は Prolog 言語における Goal 節の形式で表現する.

3.2.1.2. 効果

そのスクリプトに記述された攻撃が成功した際に得られる攻撃効果(例. ターゲット上でコマンドを実行できる), もしくは得られる知識(例. ユーザ名一覧)を記述する. 効果は Prolog 言語における原子文として表現される.

3.2.1.3. 入力パラメータ列

入力パラメータ列は, スクリプトの動作記述部から参照可能な変数群を宣言する領域である. 変数はカンマ(,)で区切られて記述される. 入力パラメータ列に出現する変数は, 事前条件もしくは事後条件中に現れていなければならない.

3.2.1.4. 出力パラメータ列

出力パラメータ列は, スクリプトの動作記述部が実行されることによって返される変数群を宣言する領域である. 入力パラメータ列と同様, 変数はカンマで区切られて記述される.

入力, 及び出力パラメータ列は事前条件と効果からの派生情報である. これらは, 推論には直接影響を与えないが, 動作記述部が実行されるときに推論エンジンとの間でパラメータをやりとりするために導入した.

3.2.1.5. プロパティ記述部の記載例

プロパティ記述部の記述例を 図 3 に示す. この例は, 以下のプロパティを表現するものである.

効果

対象ホスト HOST 上でコマンド COMMAND を実行する. 標準出力を RESULT に返す.

事前条件

- 対象ホスト HOST 上のポート PORT で Web サーバが動いていること.
- Web サーバは Apache であること.
- プラットフォームは Linux であること.
- 踏み台 AGENT から HOST の PORT にアクセスできること.

上記言明中, 全て大文字の英単語(HOST 等)は変数である.

effect, pre-cond, in-param, out-param は, それぞれ効果, 事前条件, 入力パラメータ列, 出力パラメータ列を表している. 今回の実装ではこれらの出現順序は自由である.

```
effect:
        cmd_exec(HOST,COMMAND,RESULT)
pre-cond:
        service(HOST,PORT,'www')
        webserver(HOST,PORT,'Apache')
        os(HOST,'Linux')
        connect_to(AGENT,HOST,PORT)
in-param:
        HOST,COMMAND,PORT
out-param:
        RESULT
```

図 3 スクリプトプロパティ記述例

3.2.2. 動作記述部

動作記述部には, 攻撃を実現するためのロジックを記述する. 推論エンジンが攻撃実行を指示すると, インタプリタが子プロセスとして起動し, 対応するスクリプトの動作記述部を実行する.

攻撃が成功した場合, プロパティ記述部の「効果」が実現されていなければならない.

動作記述部に書かれた攻撃手順が必要とするパラメータは, 実行時に推論エンジンから与えられ, プロパティ記述部の入力パラメータ列で宣言した名前で参照される.

同様に, 検査結果はスクリプトの出力パラメータ列で宣言された変数名を介して, 推論エンジンに返される. 検査結果は, 推論エンジンからも同名の変数として参照される. 実行後, 出力パラメータを複数セット返すことが可能である. これらは, 推論の際に順次使用される.

動作記述部から推論エンジンを呼び出し, 任意の Goal 節を評価させることも可能である. これにより, 取得したデータからある知識を推論したり, 他のスクリプトを呼び出すことが可能となっている.

3.3. 脆弱性診断の動作

本節では, 推論エンジン(Prolog インタプリタ)とスクリプトによって, どのように脆弱性診断が行われるかについて説明する.

3.3.1. サーバ起動時の動作

サーバが起動すると, 推論エンジンはまず脆弱性診断に有用な推論ルール, 例えば「Linux の管理者アカウントは"root"である」をロードし, 知識ストアに格納する.

次に, 推論エンジンは, スクリプトストアに格納されている全スクリプトからプロパティ記述部を読み出し, 図 4 に示されるルールを生成し, 知識ストアに登録する.

```
「効果」:-
        「事前条件」,
        run_script(
                「スクリプトファイル名」,
                [ 「入力パラメータ列」 ],
                __OUTPUT_DATA ),
        member(
                [ 「出力パラメータ列」 ],
                __OUTPUT_DATA ).
```

図 4 プロパティ記述部から生成されるルール

"「"*領域名*"」"は, プロパティ記述部の対応する領域に記載された内容で置き換えられることを表している. このルールの大意は以下の通りである.

(1) 「事後条件」を満たすには
(2) 「事前条件」が満たされ, かつ,

(3) 「スクリプト名」で示されるスクリプトの実行が成功すれば良い. (run_script 述語)

述語 run_script は本ツールのために, Prolog に組み込まれる述語である. 本述語が評価されると, スクリプト実行制御によって, スクリプトが実行される.

述語 member は, スクリプトからの出力が出力パラメータ列を複数セット含んでいた場合に, それらを 1セットずつ取り出す機能を果たす.

3.3.2. 検査実行時の動作

脆弱性診断は, 検査クライアントからシナリオ実行サーバに検査目標を与えることで開始される. 要求を受け取った推論エンジンは Prolog の演算規則に従い登録されているルールを評価していく.

評価が失敗した場合には, Prolog はバックトラックし, 別のパラメータの組み合わせや, 他のルールを用いた脆弱性診断を試みる. 全てのルールが失敗すると, 既存のスクリプトでは攻撃不能として, 処理を終了する.

4. 考察

4.1. 効率化

今回の設計においては, 推論をシンプルに保つことに注力し, 効率については重視していない. 効率化のために, カットオペレータによる推測木の刈り取りや, ルールの選択順序が最適化(スクリプト実行の少ないパスから実行していく)されるようなスクリプトルールの登録方法を実現していく必要があると思われる.

4.2. 無限再帰

推論エンジンである Prolog は, スクリプトから読み込んだルールをもとに推論を行っていくが, 攻撃 A の事前条件を満たすために攻撃 B を選択し, 攻撃 B の事前条件を満たすために攻撃 A を選択すると無限再帰が発生してしまう.

これを避けるために, 述語 run_script 内でスクリプト呼び出し履歴のスタックを用意しておき, 履歴内に登録されているスクリプトが, 同一のパラメータで呼び出された場合には, すぐさま false で終了するような機構の検討が必要である.

5. 結論

本稿では脆弱点の列挙だけでなく, 攻撃者の行う一連の行為を模擬可能な脆弱性診断ツールを提案した.

本ツールでは攻撃に, 効果及び事後条件という属性を与え, それらを述語論理形式で記述することで, Prolog 処理系を用いて攻撃者を模擬した診断を実行可能である.

また, これらを攻撃手順とともに, スクリプトファイルにすることで, 拡張も容易である.

今後は今回行った検討をもとに, 本ツールを実装し, 性能評価等を行う予定である.

参考文献

[1] Internet Security Systems, Inc. http://www.iss.net/

[2] Nessus.org http://www.nessus.org/

[3] 河内, 藤井, 勝山 “ハッキング手順模擬機能を有するセキュリティホール診断ツールの実装と評価”, 情報処理学会第 62 回全国大会 7F-05

[4] I.Brakto 著, 安部憲広 訳 “Prolog への入門” 近代科学社 ISBN4-7649-0165-X

[5] Larry.Wall 他 "Programming Perl", O'Reilly