ビルディングブロック型模擬環境構築システム

安田 真悟† 三浦 良介† 太田 悟史† 高野 祐輝† 宮地 利幸†‡

情報通信研究機構
†サイバー攻撃検証研究室 / ‡テストベッド研究開発室

Shingo Yasuda† *Ryosuke Miura*† *Satoshi Ohta*† *Yuuki Takano*† *Toshiyuki Miyachi*†‡

†CYBER RANGE LABORATORY / ‡NETWORK TESTBED RESEARCH AND DEVELOPMENT LABORATORY
NATIONAL INSTITUTE OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

概 要

マルウェアの挙動解析や，サイバー攻撃の演習を行うためには，それぞれの目的に合わせた模擬環境を構築する必要がある．それらの模擬環境は，解析や演習のポリシに基づきネットワークトポロジやネットワークノードが異なるが，利用する OS の種類や基本的なネットワークサービスの種類は重複が多く，それらの設定のみ異なる場合が多い．そこで，我々は主な重複物となるノードのディスクイメージと，それらを特徴付ける設定ファイルや検体などのファイルを個別に管理し，必要に応じて組み合わせる事でビルディングブロック式に環境構築を行うシステムを提案する．本稿では，このビルディングブロック型模擬環境構築システムの設計と実装，及び実際に利用例に基づく知見を述べる．

1 はじめに

インターネットが社会基盤として浸透し，情報セキュリティを取り巻く環境は急速に変化している．特に，感染することでコンピュータ利用者の意図しない様々な動作を引き起こすマルウェアは，標的型攻撃と呼ばれる特定の相手の攻撃に使われるマルウェアも出現する等，多様化・高度化しており [1]，対策技術の研究開発が急務である [2]．マルウェアの解析を行う場合，初期の感染活動において，マルウェアが Command and Control Server（C&C）サーバ等から追加のマルウェアを導入する場合等，静的解析だけでは全体の挙動が把握出来ない場合も多い．また，ソーシャルエンジニアリングなどによりあらかじめ入手した攻撃対象の環境情報がマルウェアの作成に利用されている場合もある．この様なマルウェアの解析には，攻撃対象となる個人利用の PC やオフィスの周辺機器などの攻撃対象環境を模倣した模擬環境を用意し，動的解析を行う必要がある．

また，日本国内でセキュリティに従事する人材について，潜在的に 8 万人のセキュリティ人材不足，現従事者のうち 16 万人がスキル不足であると指摘されるなど，セキュリティ人材の育成が課題となっている [2, 3]．これまで我々は主にネットワークシステムの検証の為に，ネットワーク実証検証環境の自動構築ツールの研究開発を行ってきた [4, 5]．そして，これらの技術を応用し，ITkeys [6], SecCap [7], Hardening Project [8] 等，様々なセキュリティ人材育成プログラム，イベントに協賛し演習・競技環境の構築・運営を担当してきた．

動的解析の為の攻撃対象模擬環境，演習の為の模擬環境共に，恒常的に環境を運用するにはコストがかかるため，我々が有する大規模ネットワークエミュレーション基盤である StarBED [9] 上に，その都度環境構築と解体を行っている．これまで，StarBED ではネットワーク設定以外は均一な実験ノード群を用いた実証検証環境の構築支援を重点に運用していた為，アプリケーションの設定，コンテンツや利用履歴が異なる等，多様なノードの作り込みは環境構築担当者が独自にスクリプトを書いたり，手作業で設定する必要があり人的コストが多く必要としていた．しかし，動的解析・演習・競技のいずれの環境も実際の企業などの組織ネットワークを縮退・模擬した環境である事から，基本となる OS の種類，模擬環境内で提供されるネットワークサービスには重複が多い．

ノード設定や OS インストールを自動化する事で，環境構築を省力化することを目的としたツールとして Vagrant [10], Ansible [11] や Chef [12] などが存在する．

これらを利用する事で，テンプレートとなるノードイメージの複製からノードを自動生成する事も可能であるが，手続き型の処理手順を記述する必要があるなど煩雑な面もある．実際の模擬環境の個々のノードの差異は設定ファイルや検体，模擬ドキュメントなどのファイルの差異だけである場合が多い．その為，よりシンプルにファイルの差し換えにフォーカスしたノード生成手法を用いた方が効率的である．

そこで我々は，テンプレートとなるOSディスクイメージに個々のノードの差分となる実行バイナリや設定ファイル，コンテンツといったファイルを挿入する事で，ビルディングブロック式にノードを生成し，模擬環境を構築するシステム「Alfons」を提案する．本システムによって多様なノードにより構成される模擬環境の簡便な構築と運用が期待出来る．本稿では，Alfonsの設計と実装，そして実際の利用事例に基づいた評価と課題について述べる．

2 関連研究

2.1 標的型攻撃

Mandiant社は2013年に，標的型攻撃の長期観測レポートを公開した[1]．このレポートでは，標的型攻撃により2006年以降141の組織から数百テラバイトものデータを盗まれたとしている．Seth Hardyらは[13]，標的型攻撃の調査を行い，標的型攻撃の脅威レベルをTargeted Threat Index（TTI）と呼ぶ指標でランク付けを行うことを提案している．TTIでは，Eメールの添付ファイルや，ソーシャルエンジニアリングの危険度などを指標として，包括的にランク付けを行っている．Stevens Le Blond [14]らも同様の標的型攻撃について調査を行い，既知の脆弱性を利用した悪意のあるドキュメントが，最も主要な攻撃方法であることが明らかにしている．

2.2 マルウェアの動的解析システム

Cuckoo [15]は，マルウェアの自動解析システムであり，サンドボックス内でマルウェアを実行し解析を行う．Cuckooでは，サンドボックス環境としてKVM [16]などの仮想マシンを利用し，Windowsのexeファイルなどを実行する．Christopher Krugel [17]も，同様に，マルウェアの自動解析環境の提案を行っている．Cuckooとは違い，Krugelの提案では，完全仮想化された仮想マシンを利用して動的解析を行っている．Krugelの主張では，完全仮想化のほうが，インストラクションレベルでの観測可能性や，Windowsとの親和性などを理由に，KVMなどの準仮想化手法よりも適していると

している．三輪[18]らは，仮想環境としてXen [19]を利用して，サンドボックス環境を構築し，マルウェアの動的解析を行っている．しかし，これらの技術はいずれも小規模な解析実行環境や均一なテンプレートに基づく動的解析環境を用いているため，既存のマルウェアには有効であるが，特定の環境に特化したマルウェアでは適切な解析が行えない．

2.3 クラウド技術

OpenStack [20]，VMWare vSphere [21]はクラウド環境構築のためのシステム・ソフトウェア群である．VMWare vSphereは仮想環境，OpenStackは仮想環境とベアメタル環境の双方をサポートしてノードの作成・複製等の制御が行える．StarBEDでは，SpringOS [4, 5]を提供し，大規模実証検証環境であるStarBEDのクラスタコンピューティング環境で実験環境を構築の支援を行っている．SpringOSはOpenStackやVMWare vSphereなどとは違い，ノードの作成・複製などの制御の他，実験シナリオ実行も行えるネットワーク実験支援システムである．しかし，これらの環境構築技術は利用者環境の効率的な運用や，均一な環境でのネットワークシステム検証を目的としており，個々のノードへのサービスアプリケーションの導入や設定は，利用者に任されており，多様なノードを用いた環境の模倣環境を効率的に構築する事は困難である．

2.4 環境構成管理自動化ツール

OSのセットアップやサービスアプリケーションの導入，設置を自動で行うツールとして，Vagrant [10]，Ansible [11]やchef [12]などがある．これらの設定自動化ツールは同一の構成を複数環境作成する事が主な目的である．そのため，設定ファイルは手続き型となっていて，記述が煩雑で行数も多くなる．従ってテンプレートとなるノードの自動生成には有効であるが，多様なノードを用いた模擬環境構築では個々のノードの設定手続きの記述が膨大になり，有効な手段とは言い難い．

2.5 標的型攻撃のシナリオ再現環境の構築

著者らは攻撃環境と被害環境を柔軟に構築できる標的型攻撃シナリオ再現環境を提案してきた[22] [23]．この解析環境では一般的な企業ネットワークをデフォルメした小規模なネットワーク環境を設計し，各種ユーザアカウントやコンテンツを導入した環境構築を行っている．しかし，環境構築では既存のクラウドコントローラや運用者らによる作り込みを行っている．標的型攻撃における攻撃者の活動は多様であり，同時並行

的に動的解析を繰り返し行うためには環境構築の効率化が求められている．

2.6 サイバー攻撃演習環境

セキュリティ人材の育成の重要性が増す中で様々な大学や研究機関，企業によって人材育成プログラムが行われている．これらは机上トレーニングやワークショップ形式の他に，実機や仮想環境を用いた実践的な演習も行われるようになってきた．ENCS [24] や ICS-CERT [25] では，参加者に防御すべき環境を渡し，運営側による攻撃をおこなう Red TEAM - Blue TEAM 型の演習を行っている．我々が環境構築を行っている Hardening Project もこの Red TEAM - Blue TEAM 型の演習プログラムに分類できる．また，CYDER [26] では，事例に基づいて作成された模擬被害環境を用いてインシデントの発見，解析，被害状況の確認など静的な環境での演習を行っている．

これらの演習環境は，演習の規模や想定シナリオによってネットワークトポロジやノードの構成が異なり，加えて同じ演習プロジェクトでも模擬環境の構成は逐次変更されている．環境構築にはコストがかかる為，人材育成の裾野を広げ，多種多様なプログラムを作成・運用する為には，環境構築の効率化が課題となっている．

3 模擬環境構築手法

StarBED ではネットワークトポロジを VLAN を用いて構成するネットワークスイッチの設定機構を提供している．VMWare vSphare 等のクラウドコントローラも仮想スイッチ等の機能を利用し同様のネットワーク設定機構を提供している．その結果，現在模擬環境を作る上で最も人的・時間的コストを要している箇所は，模擬環境で利用するクライアントノードやネットワークサービスを行うサーバノード等の，ネットワークインスタンスの作成・設定を行うインスタンスの作り込みである．本章では，一般的なインスタンスの作成手順を整理し，既存技術の適用範囲と課題を述べ，模擬環境構築に適したインスタンスの作成手法について議論する．

3.1 インスタンス作成手順

模擬環境作成は，その模擬環境内で利用するノードの作成作業を基準に図 1 に示す以下の３つのフェーズに定義する事が出来る．

3.1.1 テンプレート作成

まず，テンプレートとなる OS を物理サーバに直接，または仮想ノードとしてハイパーバイザ上にインストールし，必要となるアプリケーションをインストールするなど，共通の設定を行う．模擬環境で利用する OS が複数必要である場合は複数のテンプレートを用意する．同一 OS でもインスタンス毎にインストールされるアプリケーションが異なる場合は，それぞれ異なるテンプレートを用意するか，全てをインストールしたテンプレートを作成する．しかし，多数のアプリケーションをインストールし，テンプレートの汎用性を上げると，テンプレートイメージの管理は簡便になるが，テンプレートイメージが肥大化し弊害が発生する場合もある．

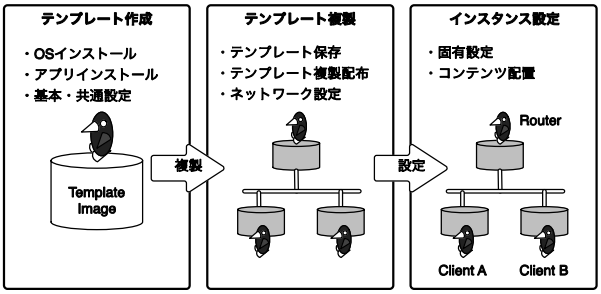

図 1: インスタンス作成手順

このテンプレート作成フェーズの作業を自動化する為に，virt-install 等の OS インストール自動化ツールがある．また，Chef や Ansible でも OS インストール後の設定，アプリケーションの導入と設定などを自動化する事が可能である．しかし，テンプレート作成時の作業は目的毎のテンプレートにつき一度の作業となる．したがって，Chef や Ansible で自動化する事は，自動化により削減する出来る時間に対して手続き処理の記述の負荷が高い．結果的に，この工程の多くは手作業になる．

3.1.2 テンプレート複製

テンプレートとなるディスクイメージを作成後，それを他の物理インスタンスや仮想インスタンスとして複製する．クラウドコントローラや SpringOS では，これらの複製作業を支援する機能を有しており，同時にネットワーク設定まで行える．ただし，複製に伴い MAC アドレス，IP アドレスやホスト名などインスタンス固有の値が変更となり，同時にアプリケーションの設定と不整合が発生する場合もあるため，複製したノードはそのままでは利用出来ない場合が多い．

SpringOS ではインスタンス内のネットワーク設定の変更はサポートしていないが，物理ノードとしての展開を前提として，各物理ノードに管理用のネットワークインタフェースを定義し，MAC アドレスに基づき

DHCP でアドレスを固定的に配布する事で複製後の OS を管理出来る様にしている．しかし，クラウドコントローラ同様にアプリケーションの設定変更はこのフェーズではサポートしていない．

3.1.3 インスタンス設定

作成されたインスタンスにテンプレートとの差分となる，各インスタンス固有の設定や実行ファイル，ドキュメント等のコンテンツの配置を行う．Xenebula, Anybed ではアプリケーションが利用する設定ファイルを NFS のマウント領域に配置し，各インスタンスが起動後その領域をマウントする事で個々のインスタンスの設定ファイルの配置を集約している．しかし，当該領域への IO アクセスがボトルネックになる等の弊害がある．

Chef や Ansible はデーモンやレシピを用意する事で，コンテンツの各インスタンス内への配置を自動化できる．SpringOS でも K 言語で設定作業を記述することで自動化可能である．しかし，これらの手法はインスタンスに管理専用のユーザを設定したり，デーモンを常駐させる必要が有るため，模擬環境の用途によっては不要な痕跡が残るなどの弊害がある．

3.2 ファイルの種類と導入フェーズ

インスタンスに作り込むコンテンツには，そのコンテンツの依存関係の強弱によって大きく分けて 2 種類ある．1 つは主に OS が保存領域を管理し，アプリケーション，ライブラリ等が相互にファイル書き換えを行う等，依存関係があるファイルである．この種のファイルの作成には OS やアプリケーション標準のスクリプト処理が必要である．そしてそれらのコンテンツは，サーバ，クライアント，ルータ，ソフトウェアスイッチ等そのテンプレートの利用目的を決める際に生成される事が多い．従って，この種類のコンテンツはテンプレート作成フェーズで生成される事が多い．また，この種類のコンテンツは IP アドレスなどのインスタンス固有の設定には影響されない．

もう 1 つのファイルはアプリケーション特有の設定やアプリケーションデータ等である．これらのファイルは他の OS やアプリケーションとの依存関係が薄く，変更はファイルの設置・置き換えで済む場合が多い．そして，特定のテンプレートに対してコンテンツファイルの配置・置き換えを行う事で個々のインスタンスを特徴付ける事が可能である．これらの置き換え可能なコンテンツはインスタンス設定フェーズで行うが，これまでの模擬環境構築では，テンプレートのディスクイメージを各物理ノードやハイパーバイザ上に複製・展開後に実施していた．しかし，複製・展開後の作業はリモートからの作業となる為，管理用のデーモンが必要であったり，リモートログインを用いたスクリプト処理が必要である．セキュリティ演習など隔離環境を指向する模擬環境では管理用のデーモンの常駐や，リモートログインによる処理は，不要な痕跡が残るなどの弊害があるため，作り込みの最後にこれらを削除するなどの作業が必要であった．

三輪 [18] らはこの弊害を無くす為に，テンプレートから複製されたインスタンスのディスクイメージを，環境構築システムがマウントし直接編集する事で，インスタンスを起動せずにマルウェアを挿入している．この手法では，インスタンスの OS は起動しない為，マルウェア配置時にインスタンスのディスクイメージに不要な痕跡が残り難い．この手法を応用し，設定ファイルやアプリケーションデータも同様の手法で挿入しインスタンスを作成する手法が一番効率的である．

3.3 ディスクイメージの共有・再利用

環境構築において，テンプレートとなる素の OS がインストールされたクリーンなディスクイメージの作成は時間がかかる作業の為，なるべく共有化・再利用をする事が望ましい．その為，クラウドコントローラを利用した場合には，インストール直後のインスタンスイメージをテンプレートとして登録しておく事が多い．StarBED では iSCSI 接続でのノード起動において，標準的な OS のテンプレートが提供されており，それを複製してノードの起動が行える．

これらのテンプレートは素の OS が入っただけの物や，特定のアプリケーションをインストールした状態の物等，様々なスナップが考えられる．いたずらにテンプレートの種類を増やすことは保存用のストレージ領域を消費すると共に再利用される頻度が少なくなるが，目的，ネットワークアプリケーション毎にインスタンスを分離する場合は，作り込みを行ったインスタンスも複製を保存，再利用することで効率的な環境構築が行える．

4 ビルディングブロック型環境構築システム

本章では 3 章で述べたインスタンス作成手順に基づき，図 2 に示す様にテンプレートイメージに各種ファイルを挿入する事で，インスタンスを作成するビルディングブロック型模擬環境構築システム「Alfons」の設計と実装を述べる．

Alfons は StarBED での模擬環境構築を前提に，主要

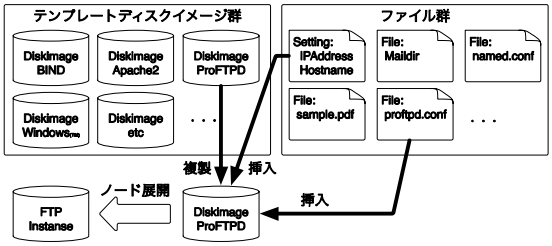

図 2: ビルディングブロック型のインスタンス作成

```
nbs:
  nb1:
  nb2:
vlans:
  vl1: {nb1: {eth0: ut}, nb2: {eth0: ut}}
  vl2: {nb2: {eth1: tg}}
```

図 3: 論理構成ファイル (例)

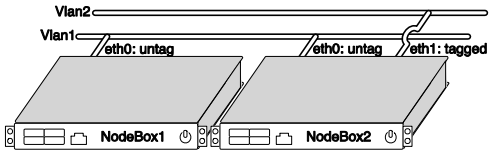

図 4: 図 3 の論理構成ファイルが示す構成

部を Ruby で実装した．大規模エミュレーションテストベッドである StarBED では，管理している 1400 台余りのサーバクラスタを用いたネットワーク実験環境構築を支援する各種機構を提供している．これらの支援機構は外部プログラムのから呼び出すための API モジュールも提供している．Alfons は必要な機能のうち StarBED が有している各種支援機構は当該 API を利用して実装している．

4.1 構成記述フォーマット

模倣環境を構築支援する為には，攻撃対象となったユーザ・組織のネットワーク情報をシステムが解釈可能な記述フォーマットが必要である．

Alfons は目的別に 2 種類のフォーマットに基づいた構成記述ファイルを用いて環境構築を行う．1 つ目は模擬環境の物理資源上での論理構成を YAML 形式で記述する論理構成ファイルである．2 つ目は論理構成ファイルで記述された論理資源上に実際に構築された模擬環境の構成を XML 形式で記述する環境構成ファイルである．論理構成と割当られた物理資源を紐付けると共に，順次環境構築を行った結果を記述する設定ファイルで，この設定ファイルにより構成の保存・再利用を可能にする．

4.1.1 論理構成ファイル

Alfons では模擬環境を構築する物理ノード資源と論理的なネットワークの結合を図 3 の様な YAML 形式の論理構成ファイルで定義する．模擬環境の構築に利用可能な物理ノードを示すリソースを NodeBox と呼び，この論理構成ファイルの例では，図 4 に示す通り，2 台の NodeBox(nb1,nb2) と論理トポロジーを構成する 2 つの vlan(vl1,vl2) があり，nb1 と nb2 の 1 つ目のネットワークインタフェイスに vl1 を untag で，nb2 の 2 つ目のネットワークインタフェイスに vl2 が tagged で接続されていることを示している．

Alfons では演習環境毎に模擬環境 ID を割り当て，模擬環境 ID 毎に論理構成ファイルを登録し管理する．マルウェアの解析や演習では，作成した環境を他の環境から隔離する必要がある．環境構築を行う初期段階で，論理構成ファイルに記載された論理 ID に物理ノード ID,VLAN ID を割り当てる．この時，この演習環境が解体されるまで（論理構成ファイルの登録が削除されるまで）は他の演習環境では利用出来ないように排他制御を行う．これにより作成する模擬環境を他の模擬環境から明示的に隔離する．同時に，物理資源の管理 ID と模擬環境の資源 ID を分離する事で模擬環境の構築・移設・再利用の利便性を向上させている．

4.1.2 環境構成ファイル

模擬環境の構成記述には図 5 に示す XML 形式の記述ファイルを用いる．この例では図 6 の示す，NodeBox 1 台 (nb1) に Ubuntu Linux のインスタンス (ubuntu01) が仮想ノードとして導入されている環境を表している．nodetype='actant' で n1 が物理資源である事を表しており，hostname パラメータで StarBED 上の k001 ノードを設定している．ネットワーク接続では eth0 に vlan 731 を設定している．Ubuntu のインスタンスは，nodetype='emulant' で仮想ノードであると示されており，parentID でどの NodeBox nb1 上に導入されている事を表している．ハイパーバイザに関する記述が無いが，これは Alfons ではハイパーバイザは特別インスタンスとして隠蔽され，emulant が示す NodeNox に自動的に導入される為である．この時，論理構成ファイルとの整合性はシステムで担保する．

4.2 物理リソースの制御

模擬環境をサーバクラスタ上に構築する為には，物理ノードの電源制御，OS イメージの導入やネットワークの設定等，物理資源の各種設定や制御が必要となる．SpringOS では電源の制御，VLAN によるネットワークの設定を行うデーモン群をプログラムから呼び出すための API モジュール群を提供している．Alfons は StarBED の利用を想定している為，これらの物理リ

```
<?xml version='1.0' encoding='UTF-8'?>
<NEED ID='Sample-2015-02-23' MiID='SAMPLE01' >
 <node ID='root' nodetype='root'/>
 <node ID='nb1' parentID='root' nodetype='actant'>
  <parameters>
   <network>
    <hostname> k001 </hostname>
    <interface name='eth0' bindvlan='731'\
    vlanmode='untagged'/>
   </network>
  </parameters>
 </node>
 <node ID='ubuntu1404' componentID='ubuntu1404'\
 parentID='nb1' nodetype='emulant'>
  <parameters>
   <hardware name='kvm'/>
   <network type='emulated'>
    <hostname> ubuntu01 </hostname>
    <interface name='eth0' bindvlan='eth0.731'\
    vlanmode='untagged' ipaddr='192.168.1.1'\
    netmask='255.255.255.0' ipaddr6='fd00::1'\
    prefixlen='64' macaddr='AA:BB:CC:DD:EE:FF'\
    media='e1000' />
   </network>
  </parameters>
 </node>
</node>
```

図 5: 環境構成ファイル (例)

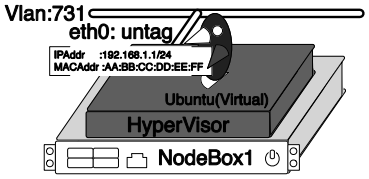

図 6: 図 5 の環境構成ファイルが示す構成

ソースの制御にはこの API モジュールを利用している．一般的なクラウドコントローラでも同様の API を提供している為，他の環境への移植も可能である．

4.3 仮想リソースの制御

SpringOS では実験設備として提供している物理リソースの制御用 API しか提供していないため，仮想化ノードを用いたインスタンスを作成した場合の Hypervisor の設定・制御に関しては別途実装を行った．これは仮想インスタンスの電源制御及び，物理ネットワーク IF に Tagged VLAN で設定された仮想ネットワークを VLAN 毎に個別の Bridge を作成する機能である．仮想インスタンスは Tagged VLAN は利用せず，VLAN ID を L2 の識別子として，それぞれの Bridge にネットワークインタフェースをアタッチする設計となっている．現在の実装では，ハイパーバイザとして Linux KVM と VMware vSphere Hypervisor に対応している．

4.4 コンポーネント DB とコンテンツ DB

テンプレートディスクイメージとインスタンスディスクイメージ，インスタンス作成の為のファイルの保管の為に，Alfons ではコンポーネント DB とコンテンツ DB の 2 種類の DB を有している．Alfons ではテンプレートディスクイメージ，インスタンスディスクイメージをコンポーネントと呼びコンポーネント ID で管理している．展開時にディスクイメージ内に挿入すべきファイルをコンテンツ DB に保管し，インスタンス ID を紐付けて管理している．これにより単一のディスクイメージに異なるファイル群を割り当て，異なるインスタンスとして作成出来る様にしている．コンポーネントの OS イメージは Qcow2 形式またはディスクの Raw イメージで保存されており，必要に応じて Qcow2 と Raw, vmdk 間で自動変換される．これにより，コンポーネント DB の利用量を削減すると共に，Hypervisor が異なる場合でも保存されている単一のコンポーネントを元にインスタンスを作成可能にしている．

```
ComponentID
    |-ComponentID.xml
    |-ComponentID.yml
    |-conf
    |   |-/(Root Dir)からのパスに基づくファイル群
    |-script
        |-設定を生成する為の補助コマンド群
```

図 7: Component ディレクトリ構造

論理 ID に物理資源が割り当てられた後，実験者は NodeBox 上にインスタンスを導入する．この時，インスタンスはコンポーネント DB に保存してある各種 OS のテンプレートイメージと，各インスタンスの差分となるファイルをコンテンツ DB から指定してコンポーネントを作成する．そして，コンポーネントを NodeBox 上に物理インスタンスとして導入するか，仮想ノードとして導入するかを指定してインスタンスの導入を行う．仮想ノードとしての導入が指示された場合 Alfons は必要なハイパーバイザを導入する．

コンポーネントは，DB 上で図 7 に示すディレクトリ構造を持ち保存される．

ComponentID.xml ファイルは図 8 に示す一般的な KVM の設定ファイルフォーマットである．ただし，ホスト名などノード毎に差異が発生する箇所は “__PARAMATER__” で記述されている．この箇所は後述する ComponentID.yml ファイルまたは，インスタンス生成時のオプションで指定可能である．また，物理マシンのメモリ量に基づく計算値や，IP アドレスに基づく MAC アドレス生成等何らかの処理に基づく結果を埋め込む場合には，“_%_SCRIPT_%_” で記述した物は，script ディレクトリ配下にある同名シェルスクリプトの処理結果を埋める事が可能である．

ComponentID.yml ファイルは図 9 に示す，ComponentID.xml ファイルの穴埋めに必要になる情報を記述

```
<domain type='kvm'>
<name>__name__</name>
<memory>__memory__</memory>
<vcpu>__cpu__</vcpu>
<os>
 <type arch='x86_64'>hvm</type>
 <boot dev='hd'/>
</os>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<devices>
 <emulator>/usr/bin/kvm</emulator>
 <disk type='file' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source file='__os_image_path__'/>
  </disk>
 <input type='mouse' bus='ps2'/>
 <graphics type='vnc' port='-1' autoport='yes'/>
 <video>
   <model type='cirrus' vram='9216' heads='1'/>
 </video>
</devices>
</domain>
```

図 8: ComponentID.xml 例

```
memory: 2048000
cpu: 2
osname: Windows 7 Pro
```

図 9: ComponentID.yml 例

する．

また，ComponentID.yml とは別にインスタンス生成時にコマンドラインから図 10 にネットワークインタフェースに関する情報を指定する事で各種設定を渡す事も可能である．

4.5 Alfons の環境構築フロー

Alfons での環境構築時のシステム操作の手順を図 11 に示す．

Alfons では，リソースの論理構成ファイルを基準に，物理リソースの割当から，実験ノードの配置までを CLI により対話的に行う逐次構築と，環境全体を記述した環境構成ファイルによる環境の一括構築と，両方をサポートしている．対話的な環境構築フローでは論理構成ファイルを作成し，これに実際に割り当てる物理ノードと VLAN 番号を指定し，逐次インスタンスの導入を行う．

Alfons は指定されたコンポーネントと，インスタンス ID に紐付くコンテンツ DB のファイルを基に，物理ノードに実ノードまたは仮想ノードの指定された種別でインスタンスを作成導入する．そして，対話的に作成・変更した環境の構成を，環境構成ファイルとして出力する機能を有している．環境構成ファイルには物理資源の ID と論理資源の ID を対応づける情報も含まれており，Alfons はこの環境構成ファイルを元に，全

```
name: client1
component: client1_kvm
machine: kvm
hostname: ubuntu01
description: Ubuntu Client
interfaces:
  eth0.vl1:
    macaddr: AA:BB:CC:DD:EE:FF
    ipaddr: 192.168.1.1/24
    ipaddr6: fd00::2/64
    model: virtio
```

図 10: Custom.yml 例

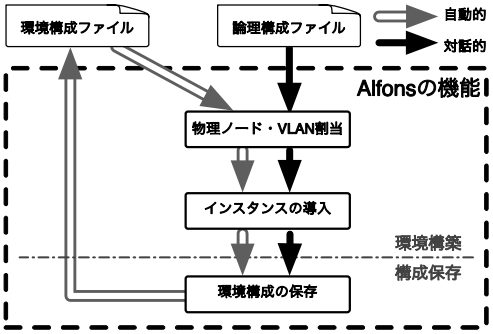

図 11: Alfons の操作フロー

自動的な環境構築も可能である．この環境構成ファイルは，XML に準拠したファイル形式であり，論理資源の ID に紐付けられている物理資源 ID 等を変更する事で，同一または一部異なる環境を容易に定義可能である．

検証実験・演習環境では，再実験の為に解体後に同じ環境を再構築，以前の環境を改変した物を環境の利用等，環境構成を再利用する場合が多い．Alfons では環境構成ファイルに基づき，一括構築する機能により環境の再構築・複製等の実験環境構成を自動化し，環境構成の容易な再利用を可能としている．

5 利用事例に基づく評価

Alfons の動作検証，設計の妥当性を確認する為に，大規模なセキュリティ・イベントである Hardening 10 MarketPlace [27] の競技環境を，Alfons を用いて構築した．本大会は参加チーム毎に模擬 EC サイト運営組織環境を提供し，大会運営側の攻撃チームによる攻撃を受けながら，クローラによる商品購入を守るコンペティション形式の大会である．本章では，この利用事例に基づきシステムの評価を行う．

5.1 環境構築概要

今回環境構築では Alfons の運用の為に専用のサーバを用意し，競技環境は StarBED の Group J のノードと Group N のノードを利用し，Alfons を用いて模擬環境を構築した．利用した Alfons サーバ，StarBED Group

表 1: 環境構築に用いたハードウェアの仕様

Server	CPU	Memory	DISK	Network IF
Alfons	Intel Xeon®E5-2620 v3 x2	64GB	FusionIO SX300 x 2	10Gbps
Group J	Intel Xeon®X5670 x2	48GB	SATA HDD x2	1Gbps
Group N	Intel Xeon®E5-2650 x2	128GB	SATA HDD x1/SSD x4	1Gbps

表 2: 環境構築時間

処理対象	処理段階	所要時間 [s]
ネットワーク	設定	66
ハイパーバイザ	ネットワーク転送	436
	設定	13
インスタンス	複製・ファイル挿入	146
	ネットワーク転送	62
合計		723

J/N のハードウェア仕様を表 1 に示す．Network IF はディスクイメージの転送，設定制御などを行う管理用の Network IF の性能である．

5.2 性能評価

今回 Alfons で構築した環境は物理ノード 80 台を用い，KVM による仮想ノードで 335 インスタンスを構築し，100 個の VLAN ID を用いてトポロジを構成した環境となった．環境の構築では，まずは競技のシナリオを含めた作り込みのために，1 チーム分の EC サイト運営環境及び攻撃チーム用の環境を作り，作った EC サイト運営環境をテンプレートして，最後に予備を含めた 15 チーム分の EC サイト運営組織環境を展開した．

ハイパーバイザとして Ubuntu 12.04 LTS，仮想化技術として Linux KVM を利用した．ハイパーバイザのディスクイメージのサイズは RAW 形式で約 13GB である．インスタンスのテンプレートイメージとして最も多く利用したサーバイメージは Linux のサーバイメージで，ディスクイメージのサイズは Qcow2 形式で約 2.2GB であった．環境構築に要する時間の参考値をこの Linux インスタンスを例に表 2 示す．

1 インスタンスの作成にはハイパーバイザの導入を伴う場合で約 723[s] の時間を要した．このうちハイパーバイザのディスクイメージの転送に要する時間が 436[s] と支配的であった．同一のハイパーバイザに追加のインスタンスを導入する場合は，これらのハイパーバイザの展開に要する時間が不要となる為，これらを除いた 200[s] 程度で作成が可能である．

次に時間を要したインスタンスの複製・ファイル挿入の処理では，テンプレートディスクイメージの複製とファイルの挿入を行うが，テンプレートのディスクイメージが大きくなる程，複製に要する時間が延びる．また，この速度は Alfons サーバのローカルストレージの速度に依存する．今回利用した Alfons 用サーバはディスクイメージの複製に要する時間を削減するためフラッシュストレージを利用した．

5.3 システム評価

Alfons と他の環境構築ツールとの比較を表 3 に示す．SpringOS は物理ノードを用いた環境構築を基本としており，仮想ノードの管理を行えない．また，各種設定は DHCP サーバ等の外部システムで動的に設定するか，環境構築後に K 言語を用いた実験シナリオを作成し設定・導入する必要があり，その仕組みは UNIX OS しかサポートしていない．VMWare vSphere 等のクラウドコントローラは，IP アドレスや MAC アドレスなどのネットワーク識別子の設定はサポートしているが，内部のアプリケーションの設定やファイルの導入には非対応である．また，物理ノードのインスタンスの生成にも非対応である．これに対して，Alfons では/(Root) ディレクトリからのディレクトリ構造に基づくファイル指定で，各種設定やアプリケーションの導入が可能であり，この手法は Windows OS にも適用可能である．そして，仮想ノードによるインスタンス，物理ノードのよるインスタンスを混在させた環境構築も可能であり，必要に応じた模擬環境構築を実現している．

既存の Vagrant や Ansible や chef と言った環境構成管理自動化ツールは，原則的に同一手順によるノードの設定処理を自動化するツールである．その為，インスタンス毎に手順書を作成する必要が有る．また，Vagrant では専用の SSH ユーザを利用して初期設定を行うため，模擬環境では不要なユーザを作成してしまう．加えて，これらツールはインスタンスを起動して設定を行うため，各種ログファイルの削除などを同時に行わないと，不要な作業痕跡を残してしまう．Alfons では，インスタンスの作成時にディスクイメージへのファイルの挿入によって各種設定を行う．インスタンスを起動せずこれらの作業を行う為，インスタンス内に不要な作業痕跡が残らない．また，ディレクトリパスとファイルの対を登録することで，テンプレートや他の環境で作成したファイルを直接配置出来る．今回の環境構築では，Alfons でテンプレートとなるインスタンスを作成後，攻撃シナリオ作成チームが脆弱性の確認やパッチの適用などの作り込みを行った．これらの作業は手作業で行われており，これらの作業痕跡の削除は手作業で行った．環境の作り込み作業時に，逐次自動化ツール

表 3: Alfons と他の環境構築ツールの比較

	設定・ファイル挿入	物理・仮想ノード混在	環境規模
Alfons	○	○	○
SpringOS	×	× (物理ノードのみ)	○
Cloud Controller	△ (設定等限定的)	× (仮想ノードのみ)	○

の手順書を作成する事は作業者の負荷が高い．今回の環境構築では，複製時に変更が必要となるアプリケーション設定ファイル，ネットワーク設定ファイルはコンテンツ DB へ保存し，変更箇所を置換しながらインスタンスの複製・展開を行った．従って，変更が必要なファイル，変更箇所の把握は必要であったが，手続き書の作成は行っていない．その結果 15 チーム分の環境の多様な設定を低コストで行えた．

Alfons を用いて対話的に環境構築を行った場合，最終的な模擬環境構成が環境構成ファイルに保存される．この環境構成ファイルから同一の環境を再構築することが可能である為，環境構成の検討，調整，再利用を一連の作業として支援可能である．この様な環境構築支援は，期間を区切ってノードを利用するクラスタ型ネットワークテストベッドを利用した模擬環境構築では重要な機能である．今回の競技環境も作成後に環境構成ファイルを取得した．

6 まとめ

本稿では，様々な模擬環境をビルディングブロック式に環境構築を行う，ビルディングブロック型環境構築システム「Alfons」の設計と実装について述べた．そして，実際の利用事例として Hardening 10 MarketPlace での利用を紹介し，構築過程で計測したインスタンス作成時間に基づく性能評価を行った．

Alfons では，インスタンスの作成時にディスクイメージへのファイルの挿入によって各種設定を行う．挿入するファイルは，ディレクトリパスとファイルの対を登録することで，テンプレートや他の環境で作成したファイルを直接配置出来る．また，ファイル内容に対する単純な置換機構を提供することで，多様な設定を低コストで行える．そして，インスタンスを起動せずこれらの作業を行う為，インスタンス内に不要な作業痕跡が残らない．Alfons を用いて対話的に環境構築を行った場合，最終的な模擬環境構成が環境構成ファイルに保存される．この環境構成ファイルから同一の環境を再構築することが可能であり，環境構成の検討，調整，再利用を一連の作業として支援可能である．

Hardening 10 MarketPlace の競技環境の構築では，全体で物理ノード 80 台を用いて 335 台の仮想ノードを作成し，VLAN100 個を用いてトポロジを作成した．1 チーム分のインスタンステンプレートを元に予備を含めた 15 チーム分の環境を構築することが可能であった．また，作成した環境構成をファイルとして取得し環境構成を保存した．この結果は，提案する環境構築支援手法が，想定する模擬環境構築を支援できることを示している．

6.1 今後の課題

Hardening 10 MarketPlace での運用では，作成した環境の死活監視や負荷状況把握が必要となった．Alfons では監視の為のネットワークの設定や監視ノードの設定は可能であるが，ハイパーバイザのネットワークインタフェースのタップ設定など監視系の仕組みを作る機能は考慮されていなかった．競技・演習環境の利活用には基本的な計測監視機構の自動構築が不可欠であると考えられる事から，これらの機能を今後設計に取り込み研究開発を行う予定である．本提案の有効性の確認には，今回の環境構成及び，その過程で作成されたインスタンステンプレートを今後積極的に利活用する必要が有る．今後の継続的な利活用を通して，本提案が有効性を検証していく．

謝辞

本論文を書くにあたって有益な助言と協力を頂いた，当機構 テストベッド研究開発室 三輪信介 博士，当機構 サイバー攻撃検証研究室ならびにサイバー防御戦術研究室の諸氏，北陸先端科学技術大学院大学 高信頼ネットワークイノベーションセンター 井上朋哉 特任助教，同情報社会基盤研究センター 篠田研究室 明石邦夫氏，システムの実践の場を提供頂いた Hardening Project 実行委員会，および関係者各位に感謝致します．

参考文献

[1] Mandiant APT1: Exposing One of China's Cyber Espionage Units. http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf, 2013.

[2] 内閣官房情報セキュリティセンター. サイバーセキュリティ戦略. http://www.nisc.go.jp/active/kihon/pdf/cyber-security-senryaku-set.pdf, 2013.

[3] 独立行政法人情報処理推進機構. 情報セキュリティ人材の育成に関する基礎調査. http://www.ipa.go.jp/files/000014184.pdf, 2012.

[4] 宮地利幸, 中田潤也, 知念賢一, ラズバン・ベウラン, 三輪信介, 岡田崇, 三角真, 宇多仁, 芳炭将, 丹康雄, 中川晋一, 篠田陽一. StarBED 大規模ネットワーク実験環境. Vol. 49, No. 1, pp. 1–14, 2008.

[5] Toshiyuki Miyachi, Takeshi Nakagawa, Ken ichi Chinen, Shinsuke Miwa, and Yoichi Shinoda. StarBED and SpringOS architectures and their performance. In *TRIDENTCOM*, Vol. 90, pp. 43–58, 2011.

[6] ITKeys. 先導的 IT スペシャリスト育成推進プログラム. http://it-keys.naist.jp, 2015.

[7] SecCap. 分野・地域を越えた実践的情報教育協働 nw -セキュリティ分野-. https://www.seccap.jp, 2015.

[8] Hardening Project. 「守る技術」の価値を最大化することを目指す、全く新しいセキュリティ・イベント. http://wasforum.jp, 2015.

[9] StarBED Project. http://www.starbed.org.

[10] Vagrant. https://www.vagrantup.com, 2015.

[11] Ansible. http://www.ansible.com/home, 2015.

[12] chef. https://www.chef.io, 2015.

[13] Seth Hardy, Masashi Crete-Nishihata, Katharine Kleemola, Adam Senft, Byron Sonne, Greg Wiseman, Phillipa Gill, and Ronald J. Deibert. Targeted Threat Index: Characterizing and Quantifying Politically-Motivated Targeted Malware. In *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 527–541, San Diego, CA, August 2014. USENIX Association.

[14] Stevens Le Blond, Adina Uritesc, Cédric Gilbert, Zheng Leong Chua, Prateek Saxena, and Engin Kirda. A Look at Targeted Attacks Through the Lense of an NGO. In *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 543–558, San Diego, CA, August 2014. USENIX Association.

[15] Automated Malware Analysis - Cuckoo Sandbox. http://www.cuckoosandbox.org/, 2015.

[16] Kernel Based Virtual Machine. http://www.linux-kvm.org/page/Main_Page, 2015.

[17] Christopher Kruegel. Full System Emulation: Achieving Successful Automated Dynamic Analysis of Evasive Malware. In *Black Hat*, 2014.

[18] Shinsuke Miwa, Toshiyuki Miyachi, Masashi Eto, Masashi Yoshizumi, and Yoichi Shinoda. Design and implementation of an isolated sandbox with mimetic internet used to analyze malwares. In Terry V. Benzel and George Kesidis, editors, *DETER Community Workshop on Cyber Security Experimentation and Test 2007, Boston, Ma, USA, August 6-7, 2007*. USENIX Association, 2007.

[19] The Xen Project, the powerful open source industry standard for virtualization. http://www.xenproject.org/, 2015.

[20] OpenStack Open Source Cloud Computing Software. https://www.openstack.org/, 2015.

[21] VMWare vSphere. http://www.vmware.com/products/vi/, 2015.

[22] 津田侑, 神薗雅紀, 遠峰隆史, 安田真悟, 三浦良介, 宮地利幸, 衛藤将史, 井上大介, 中尾康二. 標的型攻撃のシナリオ再現環境の構築. 情報処理学会研究報告. CSEC, [コンピュータセキュリティ], Vol. 2014, No. 18, pp. 1–6, may 2014.

[23] 津田侑, 神薗雅紀, 遠峰隆史, 安田真悟, 三浦良介, 宮地利幸, 衛藤将史, 井上大介, 中尾康二. 標的型攻撃再現のための攻撃シナリオ定義インタフェースの実装. コンピュータセキュリティシンポジウム 2014 論文集, 第 2014 巻, pp. 450–457, oct 2014.

[24] ENCS. European network for cyber security. https://www.encs.eu, 2015.

[25] ICS-CERT. The industrial control systems cyber emergency response team. https://ics-cert.us-cert.gov, 2015.

[26] CYDER. 実践的サイバー防御演習. http://www.soumu.go.jp/menu_news/s-news/01ryutsu03_02000085.html, 2015.

[27] Hardening 10 MarketPlace. the 1st event of hardening project 2015. http://wasforum.jp/2015/03/hardening-10-marketplace/, 2015.