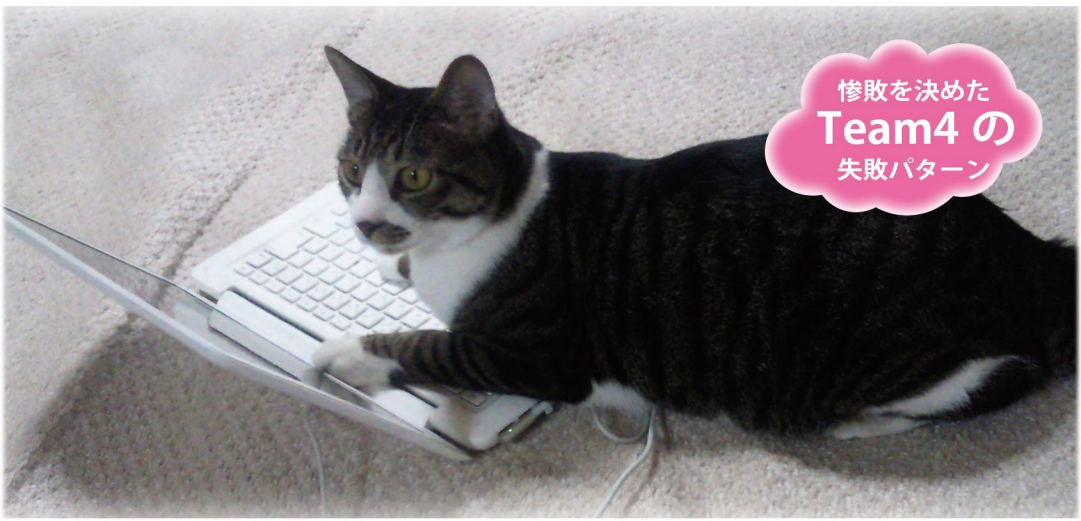
めもおきば

Tech Report

[めもおきばてっくれぽーと]
クラウドとセキュリティ、
TL 流速を支える
エンジニアの総合誌

2015 December 12

たたかっています。Hardening project



手軽さと完全従量課金が人気！

クラウド四方山話 サーバレスアーキテクチャ

新定番！ 自動プロビジョニング実行環境最新解説

セキュリティ好きが伝える

２０１５年振り返り ２０１６年予想

めもおきば TechReport 2015.12

— 目次 —

Aki @ nekoruri

Hardening 10 ValueChain 体験記

Hardening Project という競技をご存知ですか。

技術用語として Hardening（ハードニング）とは、「脆弱性を減らすことでシステムのセキュリティ堅牢にすること」（Wikipedia）を指します。サービスを提供するエンジニアであれば誰でもやっている当たり前のことですね。この Hardening――すなわち「守る技術」を競うため、WASForum[1]が 2012 年から実施しているセキュリティ・イベントが Hardening Project です。

以前から Hardening Project の存在は知っていたのですが、某ゆるふわ勉強会[2]の人たちに誘われて、今年 2015 年 11 月に行われた Hardening 10 ValueChain（以下「h10v」）で初参加してきました。結果としては最下位という惨敗でしたが、とても価値のある経験を得られたので少しでもそれが伝わればと思います。

どんなイベント？

ざっくりと言えば、各チームは「株式会社はーどにんぐ」の一員となり、脆弱性のある EC サイト環境を競技時間のあいだにできる限り堅牢にして外部からの攻撃に耐え、最終的に競技終了時点での売り上げを競うという競技形式のイベントです。ただ、こんな感じのふわっとした理解のまま参加すると、あっさり負けます（ソースは自分）。

今回の h10v の公式サイトには以下のように書かれています。

> 防御技術力のみならず、サイトの運営を維持する総合力、つまり攻撃に対する堅牢化、それと同時に売り上げの確保、ダウンタイムの最小化、そのためのコミュニケーションスキルのすべてを加味して勝者が決まる[3]

このような評価軸を実現するため、ここに書かれた様々な「総合力」が無ければ「売上」というスコアが上がらないように、競技全体が設計されています。

---

[1] Web Application Security Forum http://wasforum.jp/

[2] #ssmjp http://ssm.pkan.org/

[3] http://wasforum.jp/hardening-project/

競技そのものが「Hardening Day」として丸一日掛けて行われますが、さらにその振り返りとして「Softening Day」が翌日にあります。各チームの取り組みを発表し、攻撃側からの答え合わせを経て表彰式が行われます。この Softening Day の内容は YouTube にて公開されています[4]。

それでは、実際の h10v における競技の詳細を紹介します。

全体像

h10v では、10 人が 1 チームとなり、全 6 チームで競います。チームごとに用意されたテーブルに集まって 8 時間を戦い抜きます。人数や競技時間は、各回のテーマに応じて変わることがあるようです。前回（Hardening 10 MarketPlace）では 1 チーム 5-6 人だったようです。

各チームには、複数の仮想サーバで構成される競技環境が用意されます。各テーブルに用意された有線 LAN を経由して競技時間中のみ競技環境に接続できます。詳しくは後ほど詳しく紹介しますが、EC サイトをはじめとするいくつかのシステムが用意されています。この EC サイトに対して、運営側の用意したクローラーが「お客様」として定期的に訪問して商品を買っていきます。この「お客様」による売上金額こそが、スコアとして順位付けの対象となります。あくまで競技としては、システムの堅牢化はあくまで手段に過ぎず、購入数を上げることが目的となります。

この購入数を決めるのが「繁盛レベル」です。繁盛レベルは、運営側（後に紹介する kuromame6）が規定のルールに従って操作しているパラメータで、リアルタイムで会場前面のスクリーンに映し出されます。この繁盛レベルは「評判」を模しているため、たとえば情報漏洩事件があると一気に叩き落とされ、適切な対応をすれば回復してきます。また、事件そのものを未然に防ぐことに成功すれば上がるようです。上手く社長や顧客などへの技術アピールに成功すればさらなる向上が見込めます。これが「システムの堅牢化と安定運用」を「売上」として評価するための枠組みの肝です。

堅牢化したはずのシステムに対して「外部からの攻撃」を行うのが、運営側の「kuromame6」と呼ばれる技術チームです。彼らは 8 時間という競技時間の間、あの手

[4] https://youtu.be/I1Um1l_9Vw0　https://youtu.be/nWPwbCbktIY

この手でシステムの脆弱性を突いてきます。実際に行われた攻撃について後ほど紹介します。kuromame6 は攻撃だけでなく、競技環境の構築・運用や評価なども担当します。

「段取り八分」という言葉があるように事前準備も重要です。今回は「参加者配付資料」が前日の夕方に配布されました。それだけでなく、そもそも h10v の開催告知からも、前回（Hardening 10 Marketplace）の発展であることが読み取れるため、競技環境の大枠そのものは大きく変わっていないだろうということが予想できます。したがって、これらの事前情報を基にした準備をすることで、8 時間という短い時間を、何十何百時間にも拡張することができます。

今回は 10 人という大きなチーム（前回は 5-6 人）ですので、チームの体制作りやタスクマネジメントも重要です。募集要項[5]においても明言されています。

> すべてのチームは「やることが多すぎて手がでなかった」ということは激減し、むしろ「チームワークで乗り越えられた！」という経験ができる可能性が飛躍的に向上することでしょう。

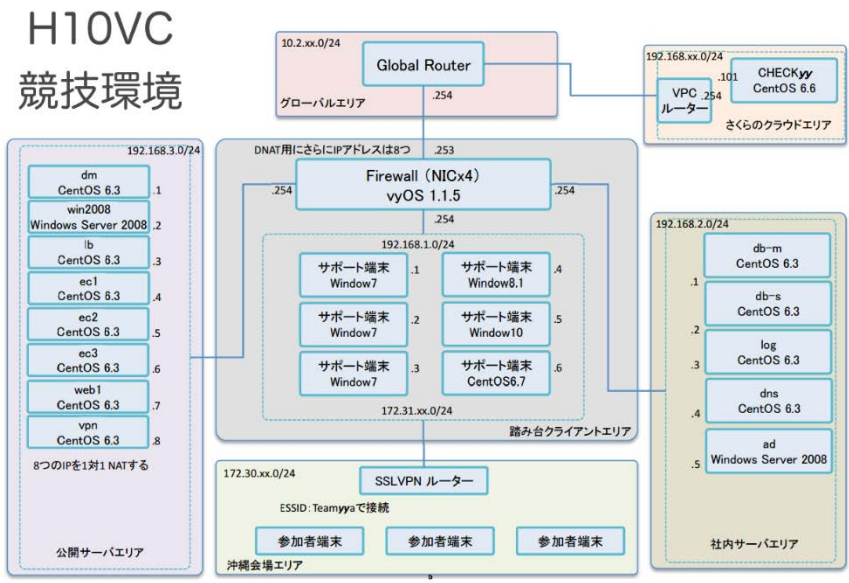
人数が多いだけでなく、自分のチームに足りない部分を補うための「マーケットプレイス」もまた用意されています。マーケットプレイスにお金を落とすことで、カスペルスキーや NEC の WAF などのソフトウェアを購入したり、プロのセキュリティエンジニアを時間で契約したりといったことができます。

これが h10v という競技の全体像です。上手く機能するチームを組み立てて事前準備を進め、攻撃側の kuromame6 に先んじて渡されたシステムを素早く堅牢化し、もし脆弱性を突かれたなら適切に対応するというだけでなく、繁盛レベルの増減条件を見極めて積極的に EC サイトの価値を上げていく――ここまでやって始めて勝利できる、そんな感じの総合競技です。

[5] http://wasforum.jp/hardening-project/h10v-application/

サーバ環境

以下が h10v で各チームに提供される競技環境です[6]。



12

各チームは大きく分けて 4 つのネットワークのサーバ・クライアントを管理します。これらはグローバルエリアのルータを介して、仮想的な「インターネット」に接続されています。kuromame6 からの攻撃や、EC サイトへのクローラーはグローバルエリアを介して「インターネット」側からやってきます。「社長」や「お客様」などと、「インターネット」を介してメールをやりとりすることも可能です。ただし、現実のインターネットへの接続は制限されているため、外部サイトから直接ファイルを持ってくることができないことには注意が必要です。

[6] http://www.iij.ad.jp/company/development/tech/techweek/pdf/151112_2.pdf

■　公開サーバエリア

公開サーバエリアには、EC サイト本体や「株式会社はーどにんぐ」のコーポレートサイトなどを提供するサーバ群が設置されています。また、外部との連絡を行うための DNS 兼メールサーバもあります。報告がされないと繁盛レベルは上がらないため、EC サイトなどのウェブサーバだけを守っていれば良いわけでは無く、実際にこの DNS サーバに対しても攻撃が行われました。

■　社内サーバエリア

データベースサーバや、Active Directory サーバなどが置かれています。DB サーバが落とされれば当然ながら公開サーバエリアで提供している様々なサービも停止しますし、Active Directory サーバが被害を受けると、この後に紹介する踏み台クライアントにも影響します。

■　踏み台クライアントエリア

公開サーバと並んで重要なのが、ここに設置されたサポート端末です。h10v では参加者の端末から直接接続して良いのはサポート端末のみというルールがあります。したがって、サポート端末が被害を受けてしまうと公開サーバなどに対して操作ができなくなってしまいます。

唯一 SSH が可能で事実上の作業機点となる CentOS 端末は 1 台しかありませんし、5 台の Windows 端末にはそれぞれメールが設定され、ここでしかメールの送受信ができません。しつこいようですが h10v では報告こそが評価の根幹ですので、CentOS 端末だけでなく Windows 端末もまた、EC サイトと同じかそれ以上に死ぬ気で守る必要があります。

■　さくらのクラウドエリア

ファイアウォールの外から「インターネット」を介して外部から自チームの環境を動作するために、「確認用サーバ」が設置されています。ごく普通の CentOS 環境のサーバが用意されているようなのですが、今回うまく活かすことができませんでしたので詳細は不明です。

実際の攻撃

h10v で実際に行われた代表的な攻撃が以下の通りです。

11 時	標的型攻撃
12 時	業務アプリからの情報漏洩
13 時	DD4BC（DDoS for Bitcoin、30 分間の NW 遮断）
14 時	Malvertising（広告からのマルウェア感染）
15 時	潜んでいたマルウェアの再活動
16 時	応募フォームの Web アプリからの情報漏洩

これからそれぞれの攻撃について紹介していきます。具体的な攻撃方法は明かされていないため、自チームで把握していた内容や、Softening Day などで公表されている内容からの推測をベースにしたもので、正確である保証はありません。

標的型攻撃

前回はメールによる標的型攻撃があったものの誰も「踏んで」くれなかったため、今回は「誰かが踏んだ体でそこから起動する」というマルウェア感染後という攻撃シナリオになりました。時間になると「マルウェアに感染して外部に攻撃しているからなんとかしろ」という連絡が外部の組織からやってきます。それと同時に一気に繁盛レベルも引き下げられ、適切な対応が進めば回復するという仕組みになっています。

開始から 1 時間ということもあり、これを事前に対策して予防できたチームは無かったように思います。私たちも含めて、カスペルスキーの導入で予防するつもりが間に合わなかったというチームが多いようです。

標的型攻撃で送り込まれたマルウェアは、後ほど再び大活躍します。

業務アプリからの情報漏洩

社長の売上把握や、在庫追加などのバックヤード業務を行うためのウェブアプリが用意されていますが、これがインターネットに全公開されており、そこに表示されていた「最

近の購入者情報」に含まれる個人情報が漏洩します。この情報漏洩も「外部からの連絡」という形で発覚します。

漏洩した情報は以下の組み合わせです。

1. 受注日
2. 購入者の氏名
3. 商品名および単価、数量、金額

参加者配付資料の「社内システム運用ルール」に「情報漏えい発生時の対応」というそのままの項目があるので、基本的にはそれにしたがって対応することになります。

現実でも同じですが、重要なことは、「いつから情報が漏れているか？」をいち早く特定することです。そのためには、この EC サイトのアプリケーションとしての構造を素早く把握しなくてはいけません。また、ユーザへの通知を行うためには、漏えい対象のユーザのメールアドレスを取得してお詫びメールを送るといった運用作業も必要となってきます。だんだんと、総合的なサービスの運用能力が問われるという Hardening Project の本質とが見えてきます。

DD4BC（DDoS for Bitcoin、30 分間の NW 遮断）

13 時に競技環境のネットワークに突然アクセスができなくなります。DD4BC と呼ばれる「DDoS 止めて欲しければ Bitcoin ちょーだい♡」という攻撃（を模した意図的なネットワーク遮断）です。ネットワークが遮断されてすぐ、kuromame6 から DDoS を受けた旨の共有と、30 分という復旧見込み時間がアナウンスされます。

この 30 分という「何もできない時間」を有効活用し、これまで 3 時間の振り返りと残り 5 時間に向けた体制作りが各チームに求められています。休み時間と思ってカレーをのんびり食べている場合ではありません。

Malvertising（広告からのマルウェア感染）

2015 年後半で最も話題になったマルウェア感染手法である Malvertising ですが、h10v でも当然のように仕掛けられました。コーポレートサイトに貼られた広告で 3 回に 1 回だけマルウェアが落ちてくるようになっていました（ブラウザ側の脆弱性による drive-by-

download 攻撃）。事前対応ができていなければ、ユーザからの問い合わせという形で発覚します。

教科書的な対応は、広告の管理者に連絡をして、不正な広告が表示されない新しいリンク先 URL を受け取るというものでした。参加者配付資料に「広告サーバ管理者」としてメールアドレスが掲載されていることから、深読みできたチームもあったかも知れません。

ユーザへの対応だけで無く、あわてて状況を確認しようとソフトウェア更新が不十分なサポート端末などでコーポレートサイトを開いてマルウェアに感染してしまえば、その端末が標的型攻撃の経路になりかねません。したがって、発覚後のサーバ側の対応だけでなく、サポート端末側のソフトウェアアップデートや、ウイルス対策ソフト等での防御もまた重要となります。

潜んでいたマルウェアの再活動

標的型攻撃で仕込まれたマルウェアが再び活動を始め、管理者パスワードが Pepper ちゃんにより突然読み上げられたり、Windows 端末を使っていたらマルウェア経由で話しかけられたりなど、潜んでいたマルウェアによる心温まる [7]攻撃がありました。

実際には、Windows 7／Windows 8 サポート端末では脆弱なパスワード・キャッシュから平文パスワードを奪取する攻撃が行われたようです。CentOS 端末ではシェルにキーロガーが仕込まれており、キーロガーの除去もしくは通信の遮断を行わない限り、root パスワード変更などを含むあらゆる操作が kuromame6 に筒抜けになっていたようです。

実際のビジネスが組織で行われることを考えれば、これはとても恐ろしいことです。誰か一人が標的型攻撃の実行ファイルを踏んでさえしまえば、そこを足がかりに最終的には管理権限が完全に奪取されてしまいます。その末路が日本年金機構であったりソニー・ピクチャーズだったりするわけです。強制アクセス制御や、操作端末の制限などの「ぱっと見面倒そう」な最小権限の原則も、こういった実例を目の当たりにすると必要性が実感できました。

正直、もうこのあたりから切羽詰まりすぎて記憶が飛んでいます。つらい。

[7] 黙って悪用されて死体すら残らず殺されるよりは余程マシ

応募フォームの Web アプリからの情報漏洩

「ありがち」なウェブアプリの SQL インジェクションが最後にやって来ました。Java で組まれたウェブアプリで入力値をバリデーションもエスケープも無く SQL にぶっ込んでしまう素敵フォームです。

このフォーム脆弱性への対応は、かなり各チームで対応が別れたのが面白いところです。ソースコードもサーバ内に同梱されていたので、そもそもの脆弱性自体を直したチームもあれば、フォーム自体を 1 から書き直してデータベースを利用しないメール送信フォームにしてしまったチーム、WAF を入れて対応を保留したチームと様々です。どれが正解ということはなく、それぞれのチームが持つスキルに合わせた「最適な対応」の判断が求められます。

結果と反省

我らが Team4「セキュ子の部屋」は残念ながら圧倒的最下位という結果に終わりました。全体の結果は以下の通りです。

順位	見込み販売力	技術点	顧客点	対応点	経済点
1 位： 現地集合検知改ざん	31,142,860	2450	3140 [1 位]	3540 [1 位]	7300 [1 位]
2 位： ちゃんぷるー	28,799,800	2150	2570	2880	1300 [3 位]
3 位： SUNSUNSUN	28,367,320	3200 [1 位]	2570	2770	1100
4 位： ちゅーばしんか	25,991,372	2750 [3 位]	2950 [3 位]	3210	1600 [2 位]
5 位： 秘密結社にんじんしりしり	25,145,656	2900 [2 位]	2570	3320 [2 位]	1100
6 位： セキュ子の部屋	14,841,380	2300	3140 [1 位]	3320 [2 位]	500

「見込み販売力」はいわゆるクローラーによる売上金額です。あくまで見込み販売力によって順位は決まりますが、それ以外の 4 つの評価点は以下のように設けられています。

技術点	脆弱性への対応、マルウェアの対応など
対応点	インシデント復旧時間、サイト全体の稼働率など
顧客点	サポート能力、ユーザ保護の取り組み
経済点	サービスや製品をたくさん使って市場を活性化したかなど

私たちのチームは売上では最下位だったものの、外部対応を担当した方の圧倒的努力のおかげで、顧客点・対応点では高い評価を得ることができています。その一方で、結果発表でもかなり突っ込まれていましたが、売上 2 位のチームが実は技術点では最下位だったりするのも、競技であるが故というところでしょうか。

優勝したチームは、マネージャはホワイトボードと付箋紙でタスク管理に注力して一切 PC を開かない、初動でマーケットプレイスからセキュリティエンジニア 2 名を競技時間まるまる買い占めて 1 チーム 12 名に増員[8]するなどという徹底したマネジメントとチームビルディングで勝利を掴みました。また、前日資料を元に、各サーバで起こりうる問題と対策の洗い出しなども行っていたようです。

脆弱なままのサービスは提供しないという選択

私たちのチームは「危険だとわかっているシステムは、まず安全性を確認してからオープンするのがお客様のためだ」という立場から、競技開始からまず初動作戦としてサービスを止めて緊急メンテナンス画面を出して脆弱性の対策に振り切るという選択をしました。脆弱性を塞ぎ未然に防ぐことで繁盛レベルを稼ぎ、最終的な売上で他を引き離すという作戦です。負け惜しみでは無いですが、この選択そのものは間違っていなかったものの、そのメリットを活かせなかったのが敗因です。

まず事前準備の不足から脆弱性の対策に時間が掛かり、ずるずると「メンテナンス時間」が伸びてしまうという最悪なパターンに陥りました。まれに良く聞く話ですが、大変よろしくないです。これは「いつまでに・何ができなかったら・どうする」という復旧判断ができないから起こります。あらかじめ所持スキルにより担当システムの分担はしていましたが、こういった判断を最終決定する「責任者」を一人きちんと決めていなかったのが原因です。これもまれに良く聞く話ですね。恐ろしいことです。

[8] その後「ガチャ」景品で 10 分間だけもう一人追加され、一瞬だけ 13 人体制でした。

マルウェアによる攻撃も絶対あるだろうという予想から、まずは Windows 端末にカスペルスキーを導入して守るという方針を立てたにもかかわらず、そもそもソフトウェアパッケージが大きくダウンロードに時間が掛かったり、Windows 10 に入れようとして対応 OS の都合で入らなかったりなど問題が多く発生しました。Windows 端末が使えなくなるとメール送受信ができなくなると言う焦りの空回りから、結局カスペルスキー一つに 16 時ぐらいまで引っ張っています。

さらに、最初に「正しい動作」を確認せずにいきなり Wordpress のバージョンアップに特攻した結果、EC サイトの本体である Wordpress のショッピングカートプラグインの不具合を踏み抜き、正しい在庫数が表示されないという問題が起きました。これは Softening Day で答え合わせされるまで原因が全くわからず、競技中は kuromame6 による情報改ざん攻撃だと思い込み調査をしていました。在庫数がわからないため、適切なマーケットプレイスからの「仕入れ」もできず、結果として最後に一気に売上が落ちるというダメージを食らいました。無駄な調査に人員も取られました。

結果として、端末のマルウェアや業務アプリといったサービス停止で防げなかったインシデントの対応に追われたあげく、競技開始後から 3 時間もサービスを止めたにもかかわらず脆弱性への対応が不完全なままのサービス再開を余儀なくされました。そして、一時的に他のチームより繁盛レベルを高く維持できた時間帯がありつつも、売っているのに仕入れていないという問題でまったく売上を得ることができませんでした。

次回勝つために

負けたら悔しいのが人間です。ならば次にどうするかを具体的に考えます。

kuromame6 からの講評では、技術点が最も高かったチームですら評価ポイントの 1/3 もクリアできていないようです。評価で重きを置かれるポイントは毎回変わるとはいえ、純粋に堅牢化の技術的そのものにもまだまだ根本的にひっくり返す余地がありそうです。優勝チームを見習ったチームビルディング・マネジメントと、万全の事前準備による瞬発力のある堅牢化技術が合わさることで最強になれそうです。

何も考えずに脆弱性診断

全ての意志決定には十分な情報が必要となります。脆弱性診断に長けたメンバーが居るチームが今回の技術点 1 位を取っているとおり、やはり適切な脆弱性診断による現状把握が大事です。競技環境の構成がある程度わかっている以上、ツールでできるような脆弱性診断はなかば機械的に競技開始時点で開始すべきでしょう。

Softening Day での発表によれば、今回は EC サイトやコーポレートサイトが Wordpress ベースだったため専用の脆弱性スキャンツールである WPScan が有用だったようです。またサポート端末については一般的な脆弱性スキャナである Nessus だけでも色々発見できていたそうで、自分のチームに脆弱性診断の専門家が居なかった場合でもツールの使い方を予習しておくだけでかなりの成果が見込めそうです。

今回もそうだったように、Hardening Project ではその趣旨から「最近注目されている脆弱性」を取り上げる方向がありますので、それを発見できるような脆弱性診断方法を逆算して自動化して持って行ければ、さらに一歩先へ行けそうです。

あらかじめ仕込まれたマルウェアへの対応

今回のように、競技開始時点で脆弱性だけで無くマルウェア本体までが仕掛けられていた場合、本質的には「何も信じられない」という状態から始めることになります。たとえば既設のバックドアでリアルタイムに介入されることまで想定すると何もできなくなってしまいます。ですが、物理的に新しいサーバを追加できない競技環境では、競技を成立させるために「現実的なゆるさ」があり上手くそこに乗っかっていく必要があります。

何も信頼できない環境に安全な橋頭堡を築いていくのはある種のパズル的な要素がありますが、例えばシェルや SSH を信頼できる静的リンクバイナリとして持ち込んだり、脆弱性診断でなんとかしたりといった手が考えられます。Windows であれば、ひとまず機械的にオープンソースのアンチウイルスソフトを導入すると言った手もありそうです。

「なにかされても被害を出さない」という封じ込めの考え方も有効そうです。全ての通信が通るルータは VyOS で手が出せるので、マルウェア自身への対策と合わせて実施するのが良さそうです。

作業環境の改善

競技環境はとにかくサーバの種類が多いため、ただ単純に SSH やリモートデスクトップで一台ずつログインして手作業でコマンドを売って回るといくらあっても時間が足りません。1 台 2 分の作業でも 13 台の CentOS サーバに一台ずつ廻るだけで 30 分近くも掛かってしまいます。

せっかく SSH が使えるのであれば、漏れたら最後また変えるしかないパスワードでは無く、参加者 PC から出さなければ絶対に漏れない公開鍵認証を利用することで、パスワードのコピペという手間からも解放され大幅に作業効率が向上します。例えば競技用の SSH プライベート鍵を共有しておけばよいでしょう。

こういった作業環境の改善は、かなり属人性が強く個人のノウハウが詰まっていることが多いのですが、そういったテクニックをあらかじめチーム内で共有することで作業の初速が上がります。初速を上げることは、最初の攻撃シナリオの実施までの時間が稼げることにもなり、何倍にも効いてきます。

Wordpress の管理画面などはブラウザからのアクセスが必要になりますが、CentOS で VNC サーバを利用するなど Windows 端末に頼りすぎない作業環境を作り込むことも効果が大きそうです。

自らのサービスの正しい把握

サービスを正しく運用するためには、それを実現するシステムの正しい把握が必要不可欠です。今回の EC サイトであれば、Wordpress のどういったプラグインでどのように実現されているか、売り上げ管理システムとの関係はどのようなものか、売上と仕入れの関係はどうなっているのか、そういったことを専門で面倒を見る「サービス担当者」が必要です。

事前の作り込み

こういったことを考えながら、事前に公開されている情報をもとに、競技環境を予測してそれに対する事前準備を作り込むことが勝利への第一歩です。

1. 更新ファイルの用意
   Wordpress であれば最新版の ZIP パッケージや、競技環境のディストリビューションのアップデート RPM パッケージを持ち込むことで、素早いソフトウェア更新ができます。
2. ファイアウォールのルール
   通すべき通信を予測して iptables や VyOS の ACL 案を作り込んでおき、素早く不必要な通信を記録・遮断することで、マルウェアの被害を防ぐことができます。
3. 認証情報の共有
   パスワードの再設定、ユーザの整理は行うとして、変更後のパスワード一覧をあらかじめ乱数で作っておき、競技に使う SSH プライベート鍵をチームメンバーで共有しておくことで、これらの再設定を全て前もって自動化することができます。
4. 脆弱性診断の準備
   競技環境に即した脆弱性診断ツールを用意しておきます。細かいことですが、存在しているシステムの全体がわかっているのであれば、例えば診断先 URL などもあらかじめ設定しておくと貴重な時間を有効活用できます。
5. 開幕ぶっぱなしスクリプト
   これら用意した道具を全てのサーバに流し込む作業をスクリプト化しておきます。SSH が使える環境であれば Fabric が有用ではないかと感じています。

感想

長々と今回の h10v という Hardening Project の紹介をしましたが、いかがだったでしょうか。楽しさと悔しさが伝われば幸いです。

次回は 2016 年 6 月に沖縄で開催されそうな雰囲気です。

是非一緒に参加して優勝を勝ち取りましょう！

クラウド四方山話「サーバレスアーキテクチャ」

IT 業界で今年もっとも流行ったバズワードが「IoT」だとすれば、パブリッククラウド業界で今年後半もっともバズったのが「サーバレスアーキテクチャ」ではないかと思います。特に 10 月に行われた re:Invent（AWS のカンファレンス）あたりから一気に叫ばれるようになったように感じます。

他のバズワードでもありがちな明確な定義が無いマーケティング用語ですが、2012 年あたりから「Serverless」という単語が明確なフレーズとして使われ始め[9]、AWS Lambda がそれをアピールに利用したことで一気に広がったようです。

「サーバ」でイメージされる範囲が広いのでサーバレスアーキテクチャに対しても様々な理解があるようですが、大きく二つのパターンに整理できるようです。

1. アプリケーション実行環境として、サーバという抽象化単位を隠蔽したマネージドサービスを利用するパターン
2. アプリケーションのビジネスロジックをフロント（ブラウザやスマホ・デスクトップアプリ）に移したうえで、データストアなどのコンポーネントを直接利用するパターン

前者はソフトウェアのアーキテクチャ、後者はシステム全体のアーキテクチャという点で視点そのものが大きく異なります。きっかけとなった AWS Lambda は前者の認識をもとに「サーバレス」を謳っていますが、クラウドコンピューティング全体の方向性としては後者における「サーバレス」化も進んでいくのではないかと考えています。この記事では、それぞれの定義における「サーバレス」が、現状どういったものであるかを整理し、課題と方向性を議論します。

[9] おそらくこのあたりでしょうか。
Why The Future Of Software And Apps Is Serverless

サーバレスアーキテクチャと PaaS

「サーバレス」化を盛んに言い始めたのは AWS Lambda ですが、その流れの発端は 2008 年にリリースされた Google App Engine です。どちらも、下にあるサーバのプロビジョニング（仮想サーバインスタンの作成と実行環境の構成）を隠蔽してステートレスなアプリケーションを動かすことができるアプリケーション実行環境です。どちらも一見似ていますが、その発展の仕方は大きく異なります。

Google App Engine はもとよりウェブアプリが大前提となっています。時刻をトリガとするスケジュールタスクはありますが、Cloud Pub/Sub からのプッシュや Cloud Storage の変更通知などの Google Cloud Platform 内でのコンポーネント間連携もすべて HTTP 経由で呼び出されます。

対して AWS Lambda は 2014 年に最初にリリースされたときには、Amazon S3 や Amazon DynamoDB の更新や Amazon Kinesis 経由でのメッセージをトリガとして、非同期に実行したアプリケーションを実装するための、いわばコンポーネント間連携の枠組みでした。それが、まず 4 月に同期呼び出しが実装されるとともに AWS Mobile SDK を経由してモバイルアプリから直接呼出せるようになり、API のバックエンドとしての道が開けました。さらに、7 月に Amazon API Gateway の登場で普通のウェブ API が AWS Lambda で実装できるようになり一気にその可能性が注目されるようになりました。

最終的にはどちらも、ステートレスなアプリケーションを受け取り、上手い具合にプロビジョニングされた実行環境を提供してくれ、外部からの HTTP リクエストへの応答や他のコンポーネントと連携可能という、似たような立ち位置に置かれています。

サーバレスアーキテクチャと PaaS の違いが良く議論されます。Heroku に代表されるいわゆるアプリケーション PaaS という分野は、技術的に十分成熟しプロビジョニングやオートスケールがマネージドサービスとして自動で行われることは当たり前となっています。ステートレスなアプリケーションが PaaS 上でスケールさせやすいということも以前から知られており、代表的な PaaS 事業者である Heroku の共同創業者 Adam Wiggins 氏が「The Twelve-Factor App」[10]として具体的な設計方針をまとめています。

[10] http://12factor.net/ja/

つまり、実行環境がステートレスであることを強制しているかどうかにかかわらず、そもそもスケールさせやすいアプリケーションを書くにはステートレスであるべきといえます。このように、アプリケーション自体においてそもそもサーバという単位を考えることはなくなりつつあり、いわゆるサーバレスアーキテクチャと従来からの PaaS に大きな違いは無いように見えます。

アプリケーション実行環境としてのサーバレス化

それでは何が「これがサーバレスアーキテクチャだ」と呼べるような実行環境の特徴かといえば、それは実行環境のプロビジョニングとオートスケールが真に隠蔽された結果、実際に利用したリソースに限りなく近い課金が行われることではないかと考えます。

Google App Engine ではあくまでインスタンス単位で動作しますが、分単位で自動オートスケールと課金が行われるため、余剰リソースがほとんど無く実際に利用したリソースに近い課金が行われます。あくまでリソースの追加・削除がインスタンス単位でしか行えないだけだという見方もできます。AWS Lambda に至っては、実際にアプリケーションが消費した CPU 時間に対して 100 ミリ秒単位で課金されます（Google App Engine もプレビュー時代は CPU 時間に課金されていました）。

利用者視点で見ると、この消費した CPU 時間への課金というところが、サーバレスアーキテクチャの一番の特徴と言えます。どんなに賢いオートスケールのアルゴリズムであっても、まともな応答時間でサービスを提供するためには余剰リソースが必要になるのに対して、消費した CPU 時間への課金であればいくら裏で余分に余計にリソースを確保していたとしても使わなければ良いので、もはや裏側でどんなオートスケールが走っているかを（正常に動いている限り）考える必要は一切無くなります。

その一方で、読込と初期化に何秒もかかるような大規模なフレームワークを使うにはまだまだ課題が大きそうです。Google App Engine ではダミーのリクエストを投げて「サーバを暖める」ための Warm up Requests が提供されています。一方でオートスケールを完全に隠蔽した AWS Lambda では、使用頻度の低いアプリケーションの初期化に時間が掛かってしまうという報告もあります [11]。これは、高レベルな AWS コンポーネントとの連

[11] AWS Lambda for Java
https://medium.com/@quodlibet_be/aws-lambda-for-java-5d5e954d3bdf#.drsn6gtv9

携を前提とした、初期化処理がほとんど必要無いような Microservice での利用を前提としているが故の割り切りとも言えるでしょう。

ほんとうのサーバレスアーキテクチャにむけて

「確保した量」から「使用した量」へのシフトによりサーバという単位を完全に忘れようとするのがアプリケーション実行環境におけるサーバレスアーキテクチャですが、提供されたサービスの組み合わせだけでサービスを組み立てることで、「アプリケーションサーバ」という存在自体を無くしてしまう方向での技術革新もだいぶ進みました。

スマートフォンアプリや SPA（Single Page Application）の普及により、サーバは認証と REST API によるデータの送受信だけを提供すれば良いという時代になりつつあります。REST API はその性質からデータストアと相性が良く、さらに各クラウド事業者が提供する認証基盤の SDK と組み合わせることができます。例えば Facebook でログインして Amazon Cognito の ID 連携を経由して DynamoDB や S3 上の自分の領域に直接データを保存するといったことが実際に可能です [12]。

パブリッククラウドが提供するデータストアコンポーネントは全体として圧倒的なスケーラビリティを持っているため、アプリケーションレベルごときの突発的負荷などにも軽く対応できます。そのため、実際にはバックエンドでは AWS Lambda などでアプリケーションが動いていたとしても、ユーザとの接点はデータストアと直接やりとりするようなタイプのシステムが増えていくと予想しています。

API としてユーザに提供したいインターフェースと、実際のデータモデルの間を埋めるものとして、Amazon API Gateway や Microsoft Azure API Management のように API 管理層が間に挟まり、データ形式の変換、キャッシュなどを行うコンポーネントが提供されています。Amazon API Gateway はもっぱら AWS Lambda の呼び出しという味方をされることが多いですが、本来はもっと大きな可能性を持っています。

---

[12] S3 + Lambda + Cognito を使って、簡単お問い合わせシステム構築 - Qiita
http://qiita.com/takuma_yoshida/items/3bd3af8b09307d25981a
Cognito 認証による DynamoDB FGAC #アドカレ 2015 | Developers.IO
http://dev.classmethod.jp/cloud/aws/dynamodb-fgac/

サーバレス時代の ID 連携

もう一つ重要なものが、ユーザの識別と認証です。

従来のウェブアプリなどでは、暗号化したブラウザクッキーやセッション DB などのサーバ側アプリケーションのみがアクセスできるセッション変数などを使うことで、リクエストをまたいでユーザを識別していました。そして、アプリケーション自体はデータストアに対する全ての操作権限を持ち、アプリケーションがユーザの ID をみて可能な操作のみを実装していました。ところがアプリケーション処理がユーザ側の端末で行われるようになると、アプリケーションの実装自体を信頼できなくなります。

古くから OpenID で ID 連携を提供し現在も OpenID Connect など自ら認証基盤と ID 連携を提供する Google をはじめとして、外部のソーシャルログインとも連携できる Amazon Cognito、Microsoft Azure Active Directory B2C など、各事業者は他のコンポーネントと ID 連携できるような枠組みを揃えています。

これは、アプリケーション自体に権限を持たせるのでは無く、「アプリケーションを利用しているユーザ」ごとに信頼して権限を持たせるという流れに沿ったものです。アプリケーションの上で ID とパスワードを直接入力するのでは無く、ブラウザ等を介して認証を提供するプロバイダーと直接 ID・パスワードのやりとりをして、あらかじめ必要最低限の権限だけを与えられたアクセスキーのみをアプリケーションに渡します。これでアプリケーションはどれだけ頑張っても、あらかじめ認可された範囲の操作しかできなくなります。

セキュリティの基本は「最小権限の原則」ですが、スマートフォンを筆頭にクライアント側アプリケーションの流れが来ている中、これからの数年で認証基盤と ID 連携を中心に「最小の権限だけを与えるとはそもそもどういう事か」という見直しが進んでいくものと思います。

おまけ：「薄く広い」アーキテクチャ

プロビジョニングの話が出ましたが、最近私が注目しているのが「薄く広い」アーキテクチャです。判る人には Google BigQuery と言えばピンと来るかも知れません。Google BigQuery を簡単に紹介すると、RDBMS っぽく SQL でクエリが書けるデータストアですが、膨大な件数のテーブルに複雑なクエリを投げても一瞬で結果が帰ってくると言う夢のような代物です。ただしその分制約も多いです。

Google BigQuery がやっているのは究極の力業です。データを数万台以上の膨大な数のサーバに分散保存し、クエリもその全てのサーバで一気に走らせてフルスキャンした結果を集約するというものです。これは Google 自身が持つ巨大なデータストアを共有で相乗りしているから可能となるスケールアウトの手法です。あまりぴったりくる訳語を見かけなかったので、ひとまず私はこのようなアーキテクチャを「薄く広い」アーキテクチャと呼んでいたりします。

サーバレスアーキテクチャも消費 CPU という結果ベースの課金であることが特徴と書いてきましたが、BigQuery の課金モデルも同様に「実際にスキャンしたディスクの容量」と大変分かりやすくなっています。

一番重要な点は、BigQuery のように「薄く広い」アーキテクチャでしか実現できないことがあり、これらはパブリッククラウド事業者のような大規模クラスタを共有できる状況でしか実装ができないというところです。今のところはまだありませんが、アプリケーション動作環境でも「薄く広い」アーキテクチャでないと実現が難しい「何か」が出てきたときが、本当の変革の時なのでは無いかと思います。その点で、IoT 業界がやろうとしていることの規模感には期待をしていたりします。

2015 年振り返りと 2016 年予想

今年個人的に心に残ったテーマと、来年盛り上がりテーマをつらつら書きます。

Superfish/eDellRoot

PC ベンダーが、独自ツールのために脆弱な独自ルート証明書を OS に導入していた事件です。サイトでも、「Superfish/eDellRoot が危険な理由」として取り上げました。

http://d.hatena.ne.jp/nekoruri/20150220/superfish

擁護のしようも無く、ひどい事件でした。しかも 2 月の Lenovo（Superfish）で勘弁してくれよと思っていたら、11 月には DELL からも同様の問題が見つかるというまさかの二段落ちです。

どちらも独自ルート証明書に共通の秘密鍵を付けてしまったために脆弱になってしまったのですが、細かいところで違いがありました。Lenovo の Superfish はそもそも HTTPS サイトにも広告を差し込むために MITM（Man-in-the-middle）攻撃を仕掛けるのが目的でした。したがって、各 PC に秘密鍵が置かれていることが必要になります。こういったアドウェアを仕掛けることの是非はさておき、これを安全に実装するのであれば、PC ごとに秘密鍵を変えてルート証明書を生成する必要がありました。

一方で DELL の eDellRoot（および同時に発覚した DSDTestProvider）はサポート用のツールに含まれていたもので、細かい利用目的は明かされていないようですが、おそらくはオンラインサポート時にサポート用システムに接続する時に何らかの理由で独自ルート証明書を使っていたのでは無いかと考えられます。そのためルート証明書を導入したが、その時に秘密鍵ごとインポートしてしまったのでは無いかという想像です。こちらは、単純に証明書だけをインポートすれば安全だったと言えます。

どちらも、あまりにもカジュアルにルート証明書を弄ったために、痛いしっぺ返しを食らうことになりました。

うるう秒

残念ながら当面の存続が決定してしまった「うるう秒」ですが、今年の 7 月 1 日に挿入されました。サイトでも「うるう秒まとめ」「Linux のうるう秒おさらい」として取り上げました。

http://d.hatena.ne.jp/nekoruri/20150521/leapsecond

http://d.hatena.ne.jp/nekoruri/20150629/leapsecond2

また、一緒に検証をした某氏が某技術ブログで取り組み内容を書いています。

http://ameblo.jp/principia-ca/entry-12046176524.html

ほとんどの環境では 59 秒が二周繰り返されるだけでそれほど気にしなくても良かったはずなのですが、前回 2012 年 7 月のうるう秒で Linux カーネルの不具合でサービス停止にも繋がる大きな問題があったため、今回は真面目な検証と対応を余儀なくされました。

うるう秒は黙っていると「Leap Indicator」として入り込んできます。うるう秒による不具合を完全に防ぐためには、かなりの努力が必要でした。その苦労を「非うるう秒三原則」としてまとめています。下位の ntpd と同期させたまま、ゆっくり時間をかけて 1 秒を受け入れるスクリプトも技術ブログの方で紹介されているので、万が一次回のうるう秒があった時には参考にしてください。

あと、CentOS 4 はそろそろなくしましょう（真顔）

通信の最適化

だいぶ前から少しずつ声が大きくなり、ついに今年の夏に爆発した「通信の最適化」の議論がありました。携帯電話事業者が、帯域幅を絞るために通信内容に含まれる画像や動画の再圧縮を行っていたというものです。

この問題は、直感的にはアウトっぽく見えるけど、法律的にはそこまで単純では無いという興味深いテーマです。大きく二つの論点があります。ひとつめは、大前提として、そもそも「通信の秘密」は侵害しているがその違法性を別の理由によって阻却しているという建前があり、この建前が本当に正当なものかという話です。もう一つは、「画像や動画

の再圧縮」によってアプリケーションで問題が出ている、という現実的かつ実装上の問題です。

他にも、数多くの論点が存在して自分が気にしている論点を主張しあうのでカオスだったという経緯で、サイトでも「「通信の最適化」の論点」として取り上げました。

http://d.hatena.ne.jp/nekoruri/20150715/transcoding

TLS デフォルト時代

もはや TLS 無しでは機能しないインターネットに大きく二つの変化がありました。

一つめは、正確には昨年末の話ですが SSL 3.0 プロトコルそのものに脆弱性が見つかり、「SSL」と呼ばれるトランスポート暗号化が全て「オワコン」になったことです。脆弱性対応のためにはサーバ側で SSL 3.0 自体の対応を切る必要があったため、この一年でだいぶ SSL 3.0 の非対応化が進みました。2015 年 12 月 4 日時点の SSL Pulse によれば、昨年 10 月には 100%近かったのが 30%前後まで落ちたようです。

SSL 3.0 もオワコンしたので、そろそろ「いわゆる SSL」とよばれるこの TCP 暗号化トランスポートプロトコルのことは TLS と呼ぶべきでしょう。TLS で暗号化されたウェブサイトのことは HTTPS サイトで良いと思います。

もう一つは、常時 HTTPS 化です。2014 年 8 月に Google が「HTTPS をランキング シグナルに使用します」というアナウンスを出してから流れが始まり、Apple がさらに攻めた「App Transport Security（ATS）」を打ち出して業界全体で話題になりました。

ATS では HTTPS 化だけでなく、Perfect Forward Secrecy（PFS）という性質を持つ暗号化アルゴリズムの利用を強制します。これは SSL 証明書の秘密鍵が漏えいした場合でも、それだけではあらかじめ保存しておいた過去の通信を復号することはできないという性質です（秘密鍵を取得してから MITM することは可能）。昔は、まさか「とりあえずトラフィック全保存しておこう」なんて攻撃者は現実的には居ないだろうと思われていたのですが、NSA 情報漏洩事件で実際に行っている可能性が指摘されてからは PFS が一気に注目されるようになりました。

もはやあまり DNS も通信路も信頼できる牧歌的な時代では無いので、HTTPS デフォルト化は時代の流れですね。HTTP/2 でもユーザ側の通信は原則 TLS ですし。

日本年金機構

情報セキュリティで 2015 年最大の事件が、5 月に発生した日本年金機構からの情報漏洩であることに異存がある人は少ないでしょう。

既に調査結果報告も出ています[13]が、とにかく「現場担当者のリテラシー」などという信頼できないものに頼らず、システムとしての情報セキュリティが重要であるという認識が広まればと思います。

セキュリティ人材の育成

手前味噌ながら、セキュリティ・キャンプ全国大会にて「クラウドセキュリティ基礎」「クラウドセキュリティ演習」として、サービス運用者から見たクラウド環境でのサービス運用とセキュリティ施策の話をする機会を頂きました。

いろいろな大人の事情（察せ）で、事前課題がさくらのクラウド、当日の演習が AWS になってしまったのは大変後悔しているところです。きちんと同じ環境で課題を出せていれば、当日はもっと深いところまで遊べると思うので、もし次回また話す機会を頂けたらもっともっと練り込んでクラウドとセキュリティの運用の楽しい話をしたいです。

また、まさか 2015 年にもなってインターネット回線が落ちるとは思わなかったので、AWS に接続できない場合フォールバック策を作り込んでいなかったのも痛かったです。実のところ、配布 PC 上には Vagrant や基本的な Docker のイメージ群など直前で内容を差し替えてオフラインでもなにかできるくらいには仕込みはしていたのですが、完全に準備不足でシナリオ作りまで手が回りませんでした。ごめんなさい。

Docker と HashiCorp

もう今年の動きは zembutsu さんのスライド[14]を上から見てくださいという感じのインフラ DevOps 界隈です。

---

[13] http://www.nenkin.go.jp/info/index.html

[14] http://www.slideshare.net/zembutsu/presentations

丁度１年前に、「あと半年もすれば仮想ネットワークベースのコンテナ間接続手法が出揃う気がするから、特に Docker に愛のある人で無ければ、それまでは単に１ホスト１コンテナなただのデプロイ手法として既存の管理方法をベースにノウハウ貯めるか、PaaS 経由で使うぐらいでちょうど良さそう。」という予想を立てていたのですが、だいたいその通りになった感じがします。

Docker はその構成上「ホスト」に捕らわれる場面が多かったのですが、オーバレイネットワーク等でのネットワーク仮想化も現実感を増してきました。個人的にはあまり魅力を感じませんが、Docker コンテナをベースにしたサーバレスアーキテクチャみたいな選択肢が来そうな気がしています。

フォグコンピューティング・エッジコンピューティング

ここからは来年の注目キーワードです。

今年はものづくり系に楽しいおもちゃが振ってきた一年でした。安価な契約と M2M 向けの付加価値を提供する SORACOM やさくらの IoT プラットフォーム、5 ドルコンピュータ Rasberry Pi Zero、1 万円を切った Windows 10 搭載スティック PC、とにかく「小っちゃいって事は便利だねっ」を体現するデバイスが立て続けに登場しました。

この数年はクラウドに以下に集約して最適化を図るかという流れが大きかったですが、小さくてもそれなりの計算機資源と接続性を持つコンピュータが揃ってきたので、来年あたりからはクラウド上の計算機資源をさらに活かすために、こちら側にもコンピュータをばらまくというモデルが広がるのでは無いかと予想しています。

こういったモデルを、Cisco は雲より近い所にあるフォグコンピューティング、NTT はネットワークの端っこにあるエッジコンピューティングと呼んでいるそうです。

マイナンバーと情報セキュリティマネジメント

マイナンバー、ついに１月から利用が始まります。

個人的には、マイナンバーにはまず政府内での名寄せの実現に注力してもらい、マイナンバーカードの利活用の話はもうすこし冷静にまともな議論を経てから実施して欲しいところです。

制度設計やシステム設計にいろいろ議論が尽きないマイナンバーですが、ひとまず動き出してしまえばかなり厳しく法整備がされているので、マイナンバー管理に合わせて情報セキュリティマネジメントへの理解が進むのではないか、というより進んで欲しいです。

また、物議を醸したIPAの「情報セキュリティマネジメント試験」も来年春から実施されます。余裕があったら挑戦してみようかなと思っているところです。

Back to the Futureにおける「未来」

今年は、皆さん大好き（ですよね？）なBack to the Futureにおける「未来」の年でした。スピルバーグの息子はジョーズ19を作っていませんが、ジョージ・ルーカスはまだ現役でスターウォーズの新作作って3Dで公開した、そんな2015年でした。

1985年に想像した2015年は、車が空を飛ばない以外は案外あたっていたのでは無いかなと感じます。今から30年後の2045年は一体どういう社会になっているでしょうか。

あとがき

こんにちはこんにちは！

初めてのコミケサークル参加です！

お酒が入るとなんか深そうにみえることを喋っているようなキャラなのですが、なんとなく普段から考えていることをアウトプットする機会と気合いがなかなか作れないので、同人誌という形で締め切りを作って書いてみることにしてみた結果がこの本です。少しでも楽しみながら何かを残せていれば幸いです。

時間とページ数の都合で落ちてしまったのですが、次回は手を動かす系の記事マシマシで書きたいと思っています。具体的には、アプリケーションコンテナや IoT 連携サービスの各社比較とか、そういうのを運用視点でがっつり見ていきたいです。また、来年も情報セキュリティは趣味と言い張って好き勝手やっていこうと思っているので、そちらでも継続してアウトプットし続けられたらと思います。

めもおきば TechReport 2015.12

発行日　2015 年 12 月 31 日
コミックマーケット 89

著者　Aki @nekoruri
aki@nekoruri.jp

発行　めもおきば

http://d.hatena.ne.jp/nekoruri/

堅牢化が捗る、大規模エミュレーション基盤

StarBED by NICT

2784550958018