Wizard Bible 増刊号 No.2

2007 年 10 月 24 日

1 章：ぬるま湯 Java　2 章：XHR のリバースエンジニアリング　3 章：Wizard Bible の紹介

1 章：ぬるま湯 Java

金床 著

1.1. はじめに

ややあいまいな記憶だが、おそらく 2001 年ごろ、筆者はメイン言語を Java へと切り替えた。それまでは 3 年ほど C++(Borland C++ Builder　以下 BCB）を使っていた。気が付いてみれば 2006 年ももう後半であり、5 年近くの間殆ど Java だけで過ごしてきたことになる。この 5 年間仕事でも趣味でもコードを書き続けてきたが、そのほとんどすべての成果物が Java 製である。

最近になって（筆者の本業である）ウェブアプリ周辺では、Ruby が注目されている。耳にしている人も多いかと思うが、Ruby on Rails というフレームワークが起爆剤となり、世界中で爆発的に Ruby の使用人口が増えているようだ。また、Python や PHP などもこの 5 年でずいぶんメジャーになってきたようで、活気がどこからともなく伝わってくる。

プログラマには、どのプログラミング言語を使うのかという問題が常につきまとう。かつてサラリーマン時代に COBOL で仕事をしたことがあるが、その後 Perl を覚えたときに「これは生産性が 20 倍は違うぞ」と驚愕したものだ（といっても実際には COBOL と Perl では守備範囲が違うので、単純に比べるべきではないのだが）。どのプログラミング言語を選択するかでその後の効率は大きく変わってくる。

かつては「どの言語がいいのか？」という話題を見るたびに「より多くの言語を自分で使える状態にしておき、何か作るべきものを前にしたときに最適の言語を選ぶ姿こそが正しい」と思っていたのだが、いつのまにか「これも Java、あれも Java、それも Java」になってしまっていた。Java のぬるま湯に頭までつかっているのだ。最近になってだんだん他の言語も覚えてみたいなという欲求が出てきたのだが、その前になぜこんなにも Java だけで過ごすようになってしまったのか、少し考えてみることにした。

1.2. C++との出会い

1998 年頃はとにかく Windows 上でネイティブで動作するアプリケーションを作りたかったので、BCB を選択した。というか実は最初に Visual C++を買ったのだが、意味がわからずアプリケーションを作れなかったのである。いわゆる挫折である。プログラミングのプの字もわからなかった頃だったので、いきなり Visual C++は難しかった（…今さわっても理解できないような気もするが）。

BCB はやさしかった。難易度だけでなく、プログラマに対してやさしい開発環境だった。何もしなくても矢印のボタンをクリックするだけでウィンドウプログラムが起動するのだ。Visual C++で挫折した後だったのでこのやさしさは筆者の心をわしづかみにし、アンチマイクロソフトの旗印を鮮明にすることとなった。

その後まったく意味を理解しないまま（いわゆるコピペプログラミングで）何故かアプリケーションを作り続けることができ、おそらく数万行ほどは「意味はわからないが目的の動作をしている」ソースコードを書き続けた。今考えると、この段階では BCB が用意してくれる VCL というクラスライブラリをただ使っている状態だった。

転機が訪れたのは書店で「C++ FAQ」という本を購入したときだ。この時点ではもうかなりのボリュームのソースコードを書いた経験があり、また VCL についてはかなり使いこなせていた（と思っていた）ので、「オレは C++は結構できるぜ」とすさまじい勘違いをしていたのである。しかしこの本を読んでも一体何が書いてあるのかさっぱり理解できなかった。答えではなく、質問の意味もわからなかったのである。「デストラクタって何だ？」というようなレベルであった。つまりその時点では C++という言語は殆ど何も理解できておらず、ただ使いやすく設計されていた VCL を使ってちょこちょこ作業ができる程度のレベルだったのだ。

「C++ FAQ」を読み進めるうちに自分が作ったプログラムが（主にメモリリーク系の）欠陥だらけ、問題だらけであることがわかってしまい、「今まで作っていたのは何だったんだ」という強い脱力感にさいなまされた。しかし何とか頑張ってその時点ではじめて「言語としての」C++を勉強しはじめ、他の書籍も片っ端から読み漁り、徐々にまともな C++プログラマへと進歩していった。

1.3. Java との出会い

C++は間違いなく「最強の」そして「最狂の」プログラミング言語である。オブジェクト指向的に使うことで規模の大きいアプリケーションも作れるのに、インラインアセンブラで CPU レベルでのプログラミングも可能であり、またテンプレートを使いこなせば想像を絶する高度なプログラミングが可能となる（ちなみに筆者はテンプレートについては STL を使う程度のレベルまでしか理解できなかった）。そして実行速度も速く、多くのプラットフォームでサポートされている。

しかし欠点も多い。まず何と言っても「落とし穴」が多すぎる。普通にのんきにプログラミングしていると、かならず足をずっぽり取られてしまうのだ。C++で罠を避けるためには常に頭の中で「この場合はアレに気をつけないといけない…」「こういうケースではコレはタブーで…」と最大限の注意を払わねばならないのだ。これは非常に疲れる。

また、クロスプラットフォームのコードを書くのことが実質不可能である。これは同時にインターネット上で参考になりそうな、あるいは利用できそうなソースコードを見つけても、手元の環境ではコンパイルできず動かない、というケースが多いことも意味する。BCB 時代にはさんざんこれで泣かされた。高いスキルがあれば移植できることもあるのだろうが、当時の筆者には難しい話

であった。

さらにセキュリティの問題がある。バッファオーバーフローやフォーマットストリングバグなど、コンピュータを乗っ取ることのできる致命的なセキュリティホールを作る可能性が高い。

そんな C++との激しい戦いで消耗しつつあった筆者の前に、Java が姿を現した。

1.4. 言語と開発環境

このころ（そして今もだが)、筆者にとってはプログラミングはそれ自体が目的ではなく、ソフトウェアを完成させることが目的となっていた。そのため言語仕様だけでなく、どのようなクラスライブラリが使用できるのかという点が非常に重要になっていた。

Java に手を出して最初の印象は「クラスライブラリが信じられないほど充実している」というものだ。VCL も悪くないライブラリだったのだが、筆者が特に作りたかったネットワーク関係（ソケット関係）のライブラリはイマイチだった（そのためかなり苦労して自作した)。

しかし Java は違った。最初から完璧な形でソケット関連のライブラリが用意されていたのである。これには感動した。そしてこのクラスライブラリが「標準」であり、多くのプラットフォームで同じコードが稼働することに驚いた。

言語としての Java も筆者を魅了した。C++にあったようなたくさんの落とし穴は殆ど見あたらず、気軽なプログラミングが可能となっていた。C++で感じていた「う〜ん、これはちょっと問題なんじゃないのか」という点がことごとく解決されていたのである。「C++の反省をふまえて開発された」という点が筆者のニーズとぴたりと一致した。

しかし開発環境はいまいちだった。BCB でプログラミングを覚えた筆者にとって、統合開発環境（IDE）はあって当たり前のものだ。Java を勉強する際は javac などを使うコマンドラインでのプログラミングからはじめたものの、javac の起動はあまりにも重く、「Java は重い」というイメージを増幅することとなった。

同じく Borland から出ていた Borland JBuilder を使ってみたものの、Swing で作られた IDE はあまりにも重く、またネイティブ GUI とのルック&フィールの違いは違和感が大きすぎた。

そこで出会ったのが Eclipse である。

1.5. Eclipse

Eclipse で最初に驚いたのはコンパイルがあっという間に終わることだ。BCB ではコンパイルというのは数分、時には 10 分かかるものだったのだが、Eclipse ではエディタ上で変更点を保存するとほぼその瞬間に終わるのである。「Java は重い」という常識が覆された。

また、ネイティブ GUI で動作するので使っていて気持ちがいい。エディタに vi を使えないという点は残念なのだが、それを補ってあまりある魅力に溢れている。Eclipse を使い始めたことで、「言語と開発環境は一体だ」と再認識した。

筆者が考える Java の魅力は、Eclipse あってのものだ。エディタで編集中にリアルタイムでコンパイルが走り、エラーとなる箇所に赤線が入る。保存を行えばほぼその瞬間にコンパイルは完了しているので、すぐに実行し動作を確認することができる。つまりコンパイル型言語とインタプリタ型言語のおいしいところを両取りしているのだ。

いわゆる「コンパイル型言語」では変更するたびにコンパイルする時間が必要となるので、その間にプログラマの思考が中断されてしまうと言われる。そのため「インタプリタ型言語」は保存して即実行できるから思考が中断されず、多くのプログラマに支持されているのだ。しかし「インタプリタ型言語」は実行するまで文法エラーがあるかどうかわからないという欠点を持つ。特に筆者のようにセンスがないプログラマは何度実行してもエラー、エラー、エラーとなってしまい、そのたびに「何行目でエラーだ？」とエラーメッセージを読み解いてエディタに戻って…という作業を繰り返すはめになる。

しかし Eclipse ならば編集中に文法エラーを発見でき、思考を中断されずに即実行できる。これはあまり大きな声で語られていないが、Java+Eclipse という組み合わせを決定的に魅力的なものにしている最大の要因ではないだろうか。

他にも関数の名前を変更したいときなどに使えるリファクタリングの機能やクラス間の継承関係を把握できる機能など、今や筆者にとっては「必須」となってしまった機能が Eclipse には満載されている。筆者にとって「Java」とは同時に「Eclipse での開発」を意味するものとなっている。

1.6. GUI

しばらく CUI のプログラミングとウェブアプリケーションの開発ばかりやっていたのだが、やはり GUI のソフトウェアも作りたい。しかし現状、Java での GUI 開発はまだまだ問題がある状態である。BCB で可能だった「マウスでボタンなどの部品を選び、ぺたぺた貼り付けて GUI を作る」というプログラミングはまだ実現できていないのだ。

Java が標準でサポートしている Swing のような GUI は、筆者にとっては受け入れられないものだ。あの違和感と動作のもったり感は、とうてい使いたくなるものではない。苦労して作ったアプリケーションがあのような使い心地では、満足できないのである。

そこで最近は SWT というライブラリを使っている。これは Eclipse も使っている、Java でネイティブな GUI を使うアプリケーションを作成できるものだ。これはなかなか悪くない。ただ現状では開発環境が追いついていないため、どうしてもソースコード内でボタンのサイズなどの数値を直接入力するスタイルを取らざるをえず、開発効率はとても悪い。Eclipse にプラグインがあるのだが、バグが多く使い物にならない状態だ。

1.7. その他いろいろ

そして筆者はウェブアプリケーション開発で食っている人間なので、やはりサーブレットエンジンのような効率的に開発されたサーバーサイドの仕組みは魅力的だ。C++などでももちろんウェブアプリケーションを作ることはできるのだが、かなり効率が悪いと言わざるを得ないだろう。また、RDBMS と連携する箇所も多いので、JDBC のように多くの RDBMS をサポートするライブラリが使えるのもありがたい。これらの要素は言語が広く使われてはじめて実現される側面もあるため、「シェアが広い」という点が言語自体の魅力になるという考え方もできるだろう。

1.8. まとめ

以上により、筆者がプログラミング言語と開発環境に求めるのは次のような点である。

- クラスライブラリが充実しており、また高いクオリティを持つ
- ネットワークプログラミングが可能である（できれば SSL もネイティブでサポートする）
- エディタの保存と同時にコンパイルが（高速で）終わり、エラー箇所がすぐわかる
- 関数名の変更などのリファクタリングの機能がある
- オブジェクト指向をサポートしている
- ネイティブ GUI のアプリケーションを開発可能である
- クロスプラットフォームである（Windows と Linux がサポートされていればとりあえず満足）
- バッファオーバーフローなどを起こすことができない
- 関数の中身を実行時に書き換えることができる
- ウェブ用の効率的なエンジンが存在する
- RDBMS との相性がいい

このように情報を整理してみると、やはり Java が頭ひとつ抜き出て魅力的であることがわかる。しかしあまりにもぬるま湯で刺激がないことも確かだ。そこで、Ruby や Lisp などの勉強もしてみようかと思っている（Lisp の勉強は少しだが開始した）。Ruby については Eclipse 並に充実した IDE が登場することを期待している。

5 年後には一体何を使っているだろうか。

2.1. はじめに

XHR とは XMLHttpRequest のことである。これを使うと、ウェブページ中に埋め込まれた次のような JavaScript のコードにより、比較的自由度の高い HTTP 通信を行うことができる（このコードは IE でのみ動作する）。

```
var request1 = new ActiveXObject("Microsoft.XMLHTTP");
request1.open( 'GET', 'http://www.jumperz.net/index.php',false );
request1.send( null );
alert( request1.responseText );
```

しかし XHR が存在することで、ウェブアプリケーションに XSS 脆弱性が存在する場合、XHR に TRACE メソッドを送信させることで XST（Cross Site Tracing）攻撃が可能となる。この場合、Basic 認証の情報や HttpOnly で発行された Cookie が攻撃者に奪われる可能性がある。そのため Firefox や Opera の最近のバージョンでは XHR を使用する際にメソッドとして TRACE は指定できないようになっている。IE でも同様に比較的新しいバージョンでは TRACE が禁止されている。

また、メソッドと同様に、XHR の送信先となるポート番号もセキュリティ制限の対象とすべきだ。例えば 25 番ポートに接続できてしまうと、XHR を悪用してスパムメールを送信されてしまう（ユーザのブラウザにスパムメールを送信させる）可能性が考えられる。つまり次のようなコードは失敗するべきと考えられる。

```
request1.open( 'GET', 'http://www.jumperz.net:25/index.php',false );
```

さてこのように

- TRACE メソッドは禁止
- ポート番号について、例えば 25 番などは禁止

というセキュリティ制限がかけられている XHR だが、どうも IE についてはこの制限のかけ方がお粗末であるように思える。そこで今回はリバースエンジニアリング祭りをおこない、IE の XHR 実装がどこでどのように行われているのかを調べてみた。

2.2. メソッドの制限

TRACE メソッドを禁止するべき場所は open 関数の第 1 引数だ。この引数はずばり HTTP リクエストのメソッドを指定するもので、普通は GET や POST が指定される。HTTP のメソッドというのはアルファベットの大文字と相場が決まっているので（よい子は相場じゃなくて RFC を見てくだちい）、とりあえずまともなプログラマなら[A-Z]{1,20}のような感じで入力をチェックし、その上で「TRACE」ではないことを確認するだろう。

しかし、さすがマイクロソフト様。なんと次のようなおどろきのコードで TRACE が飛んでしまうのだ。

```
request1.open( '\nTRACE', 'http://www.jumperz.net/index.php',false );
```

この場合、送信される TCP のデータストリームの内容は最初に余分な 1 バイト（0x0D）が付いた TRACE メソッドの HTTP リクエストとなる。しかし Apache や IIS はこれを受け付けてしまうため、XST は成立してしまう。つまりこの IE の XHR における TRACE の禁止は、簡単に回避されてしまうのだ。

さらにひどいことに、次のようなコードさえ動いてしまう。

```
var method = "GET\x09/index.php\x09HTTP/1.0\r\nHost:hogehoge\r\nX-Foo:";
request1.open( method, 'http://www.jumperz.net/index.php',true );
request1.send( null );
```

このテクニックを使うとクライアントサイドでの HRS をはじめ、様々な問題が引き起こされるだろう。

2.3. ポートの制限

Firefox や Opera では XHR で接続可能なポートについても厳しく制限しており、スクリプトを含むページと同じポートにのみ接続可能のようだ。しかし IE ではこの実装も奇妙であり、21 番ポートや 25 番ポートには接続できないのだが、22 番ポートや 23 番ポートには接続できてしまう。接続できたからといってそれほど深刻な問題が発生するわけではないのだが、メソッドの禁止の実装と同じく、どのように実装されているのかが気になるところである。

2.4. 祭り開始

まずは IE における XHR の実装がどの DLL で行われているのかを調べてみることにした。system32 ディレクトリにおいて「grep -i xmlhttprequest *.dll」としてみると、msxml3.dll などのいくつかの DLL ファイルがヒットする。

TRACE の禁止が明示的に行われているようであれば「grep -i trace msxml3.dll」でヒットしそうに思えたが、これはヒットしない。また TRACE というのはよく使われる単語なので、「grep TRACE *.dll」などとしてしまうと大量の DLL ファイルがヒットしてしまい、結局どの DLL において TRACE の禁止が実装されているのかはわからない。

そこで Dependency Walker というツールを使って msxml3.dll が依存している DLL をリストアップしてみることにした。リストアップされた中で、まずは winhttp.dll というファイルが怪しいように思えた。しかしこのファイルの中身を TRACE で検索してみても「ERROR_SYSTEM_TRACE」という文字列がヒットするだけで、TRACE メソッドの制限が行われているわけではないようだ。そこで、次に怪しいと思われる wininet.dll を調べてみた。すると HttpOpenRequest という関数の中で、それらしい箇所を見つけることができた。

```
7667421E   8B35 B4136676   MOV ESI,DWORD PTR DS:[<&SHLWAPI.StrCmpNIA>]   ;
SHLWAPI.StrCmpNIA
76674224   6A 05           PUSH 5
76674226   68 58426776     PUSH WININET.76674258   ; ASCII "TRACE"
7667422B   53              PUSH EBX
7667422C   FFD6            CALL ESI
7667422E   85C0            TEST EAX,EAX
76674230   0F84 DBC30100    JE WININET.76690611
76674236   6A 05           PUSH 5
76674238   68 50426776     PUSH WININET.76674250   ; ASCII "TRACK"
7667423D   53              PUSH EBX
7667423E   FFD6            CALL ESI
76674240   85C0            TEST EAX,EAX
76674242  ^0F85 09F5FFFF    JNZ WININET.76673751
76674248   E9 C4C30100     JMP WININET.76690611
```

このように、StrCmpNIA という API を使って何か（おそらくユーザが指定したメソッド）と「TRACE」という文字列を比べているのである。「ビンゴ！」というやつだ。

さて「これは何だ？」と思ったのは「TRACK」の部分である。TRACE と同様に、TRACK というメソッドも禁止しているのだ。そこでウェブを検索してみると、どうやら IIS のオリジナル実装で、TRACE と同様の働きをする HTTP のメソッドということである。XST では TRACE と同様に TRACK も問題になる、と指摘している文章を見つけることができた。TRACK というメソッドの存在は知らなかったので、これは勉強になった。

ここで StrCmpNIA の第 3 引数として「5」を指定している（PUSH 5 の部分）ので、文字列の比較は 5 文字目までしか行われない。そのため面白いことに、ユーザが「TRACEAAAA」や「TRACKBBBB」のような文字列を指定した場合にも、これは禁止される。

2.5. 一般の開発者を無視した実装

さて鋭い読者なら同様の疑問を持っているかと思うが、この実装には大きな問題が存在している。TRACE と TRACK の禁止は明らかに XST 対策である。XST 対策は IE の XHR の実装に対して行われるべきだが、実際には wininet.dll の中で行われてしまっている。

wininet.dll というのは IE 以外のソフトウェアからも利用されるライブラリであり、一般のアプリケーション開発者が HTTP クライアントを作る際にも利用する可能性が高い。これは例えば次のようなコード（本稿を書く際に使ったもの）で行われる。

```
#include <windows.h>
#include <wininet.h>
#include <stdio.h>

int main( int argc, char* argv[] )
{
HINTERNET hInternet;
HINTERNET hSession;
HINTERNET hReq;
```

```
hInternet = InternetOpen(
   "TEST",
   INTERNET_OPEN_TYPE_DIRECT,
   NULL,
   NULL,
   0 );

hSession = InternetConnect(
   hInternet,
   "www.jumperz.net",
   80,
   NULL,
   NULL,
   INTERNET_SERVICE_HTTP,
   0,
   0 );

hReq = HttpOpenRequest(
   hSession,
   "TRACE",
   "/index.php",
   NULL,
   NULL,
   NULL,
   0,
   NULL );

if( hReq == NULL )
   {
   printf( "null" );
   }

InternetCloseHandle( hInternet );
InternetCloseHandle( hSession );
InternetCloseHandle( hReq );

return 0;
}
```

wininet.dll の中で TRACE と TRACK が禁止されてしまったため、一般の開発者が作成する IE とは関係のない HTTP クライアントアプリケーションも影響を受けてしまう。つまりこれらのアプリケーションからも TRACE や TRACK の送出が禁止されてしまうのだ。これは開発者を無視した暴挙であるように感じる。TRACE を送出するような専用のクライアントアプリケーションがそれほど多く存在しているとは思わないが、本来ならば msxml3.dll 内などで実装すべき処理である。

2.6. ポートの制限

ポートの制限についても同様に wininet.dll 内で行われていることがわかった。これは以下の箇所である（ちなみにこちらは少し古いマシンで調査したので、2.4 で示したリストとは DLL のバージョンが異なる）。

```
7020DDAA  0FB74424 04    MOVZX EAX,WORD PTR SS:[ESP+4]
7020DDAF  83E8 00        SUB EAX,0
7020DDB2  74 1E          JE SHORT WININET.7020DDD2
7020DDB4  83E8 15        SUB EAX,15
7020DDB7  74 19          JE SHORT WININET.7020DDD2
7020DDB9  83E8 04        SUB EAX,4
7020DDBC  74 14          JE SHORT WININET.7020DDD2
7020DDBE  83E8 55        SUB EAX,55
7020DDC1  74 0F          JE SHORT WININET.7020DDD2
7020DDC3  83E8 09        SUB EAX,9
7020DDC6  74 0A          JE SHORT WININET.7020DDD2
7020DDC8  83E8 18        SUB EAX,18
7020DDCB  74 05          JE SHORT WININET.7020DDD2
7020DDCD  33C0           XOR EAX,EAX
7020DDCF  C2 0400        RETN 4
```

7020DDB4 の 15 は 10 進数で 21 である。EAX にはユーザが指定したポートが入っており、まず 21 を減算してそれが 0 であるかどうかを調べている。つまりユーザが FTP のポートである 21 を指定した場合にはここで引っかかる。つぎにさらに 4 を減算してそれが 0 であるかどうかを調べている。つまり SMTP のポートである 25 を指定した場合にはここで引っかかる。以下同様に比較が行われており、結果として禁止されているポートは 21/25/110/119/143（FTP/SMTP/POP3/NNTP/IMAP）であることがわかった。

ちなみにこの箇所については、0x05 で示したソースコードの 80 の部分（ポート番号）を接続可能なポート（22 番など）と接続が禁止されているポート（21 番など）に変更した exe をそれぞれ OllyDbg でトレースしてログを取り、ログの内容を比較することで特定することができた。メソッドの制限はただ単に「TRACE」という文字列を検索するだけで見つけることができたが、こちらは少々手間がかかった。

このように、ポートについてもメソッドの場合と同じように wininet.dll 内に直接制限が設けられている。しかし HTTP クライアントが 21 番ポートなどに接続したい場合というのはあまり考えられないので、メソッドとは異なり、こちらの副作用はほとんどないと考えられる。

2.7. 祭りの後のまとめ

今回は IE の XHR の挙動について、実装を行っている箇所をひとりリバースエンジニアリング祭りによって特定し、実際に読んでみることで、どのようにセキュリティ制限が行われているのか、またその影響などを把握することができた。開発者を無視して強引に（しかも不完全に）TRACE を禁止するというマイクロソフトの手法は衝撃的ですらある。筆者は元々WININET のような MS 依

存のライブラリの利用は避けているが、これはやはり正解であったと思うことになった。

ところでKG先生主催のリバースエンジニアリング祭りが秋にも開催されるとのことなので、国内最高レベルのリバースエンジニアリング関連情報を提供するWizard Bibleの読者・著者の皆様にはふるって参加していただきたい。筆者は今回ついにOllyDbgでのトレースログの取り方を覚え、また本稿のネタを調べるのに丸一晩かかったほどのハイスキルなので、参加しても学ぶべきことがあるか疑問だが、いちおう参加しようと思っている。

3.1. Wizard Bibleとは

1章、2章で取り上げた記事は、それぞれWizard Bible vol.29（2006年11月リリース）、Wizard Bible vol.34（2007年6月リリース)から抜粋したものです。

Wizard Bible（http://wizardbible.org/）とはインターネットで毎月公開されているオンライン雑誌です。内容は幅広い意味でのセキュリティを扱っております。また、誰でも無料で読むことができます。

3.2. Wizard Bible vol.36の紹介

著者・読者の方々のおかげで、2007年9月にVol.36を公開することができました。増刊号を手にとっている皆様にも雰囲気がわかるように、Vol.36の目次を紹介します。ただし、この場では著者名は伏せておきました。

- マニアックJavaプログラミング第7回: ～Javaクラッキング～
- Windowsシステムプログラミング 番外編 ～Java～
- Windowsシステムプログラミング Part2 ～デバッグ～
- リバースエンジニアリング実践 khallengeへのチャレンジ その1
- リバースエンジニアリング実践 khallengeへのチャレンジ その2
- はじめてのハッキング ～バッファオーバフローの利用～
- 基礎暗号学講座・第11回 ～原始元判定アルゴリズム～

これはあくまで一例であり、過去にも多くの記事を公開しております。興味がある方は是非Webサイトにアクセスしてみてください。

3.3. Wizard Bible vol.37の原稿募集

Wizard Bible vol.37の原稿を現在募集してします。締め切りは10月29日（月曜日）です。技術的な記事だけに限らず、セキュリティのイベントのレポートなどでも構いません。もちろん、Black Hat Japan 2007のレポート記事も募集しています。どうか宜しくお願い致します。

なお､募集やりリース時の告知などはhttp://akademeia.info/で行っています。