セキュリティーレポートサンプル

ご注意

この文書は株式会社スプラインのセキュリティ評価サービスで

納品されるレポートのサンプルです。

団体、個人名、サイト名、URL 等はすべて架空の物です。

目次

1. 概要

1-1.調査対象

http://●▲■.com/mail/ 以下の URL よりアクセスされる PHP スクリプト。およびそれらスクリプトによって利用されるデータベース。

1-2.調査概要

当該システムに対し、主にクロスサイトスクリプティング、および SQL インジェクションによる脆弱性に注目し調査を行った。また社内にシステムのクローンを作成し、実際にクロスサイトスクリプティング、および SQL インジェクションを行い、危険が現実に存在する事を確認した。

1-3.調査責任者

株式会社●■●■　Ａ田Ｂ太朗

1-4.結論

当該システムの利用を続ける事により、悪意ある第三者がデータベースに対し壊滅的なダメージを与える可能性がある。
よって早急にシステムの入れ替えを行う事が望ましい。

2. クラッキング手順の詳細

2-1. データベース破壊の危険性

内部情報に詳しくない外部の人間が、DB を破壊できる事を確認した。

2-1-1. 実現方法

以下の URL にアクセスする事により、「●▲■メルマガ」に登録する全会員の ID、パスワード、氏名等、すべてのデータを削除する事ができる。

http://●▲■.com /mail/?no=5;delete%20from%20user

※ 大変危険性の高いURLなので、どんな事があっても絶対にアクセスしてはいけません。

2-1-2. 発覚時の影響

すべてのデータが破壊されるため、危険性はこの上なく高い。実行される事により運営が不可能になる。

2-1-3. 発覚する可能性の度合い

このセキュリティホールが発覚する可能性は、低いとは言いきれない。ある程度のプログラミングスキルとセキュリティに関心がある人間（以下、仮に「ジョージ」とする）なら、比較的簡単にセキュリティホールの存在を知る事が出来る。また、ジョージは不正アクセスする事なくサーバーを調査する事が可能であり、実際の破壊活動は見知らぬ第三者に委託する事ができる。

2-1-4. 原因

GET 変数"no"をサニタイジング（無効化）していない事が原因である。

2-1-5. 対策

mail//php/phpstartup.php の 53 行目以降を、以下のように修正する事で問題を回避する事ができる

旧＞

```
function get_artist($no,$con){
        $sql = "select n_artist from artist where no = $no";
        $result = pg_exec($sql);
        return pg_result($result,0,0);
}
function get_table($no,$con){
        $sql = "select n_table from artist where no = $no";
        $result = pg_exec($sql);
        return pg_result($result,0,0);
}
```

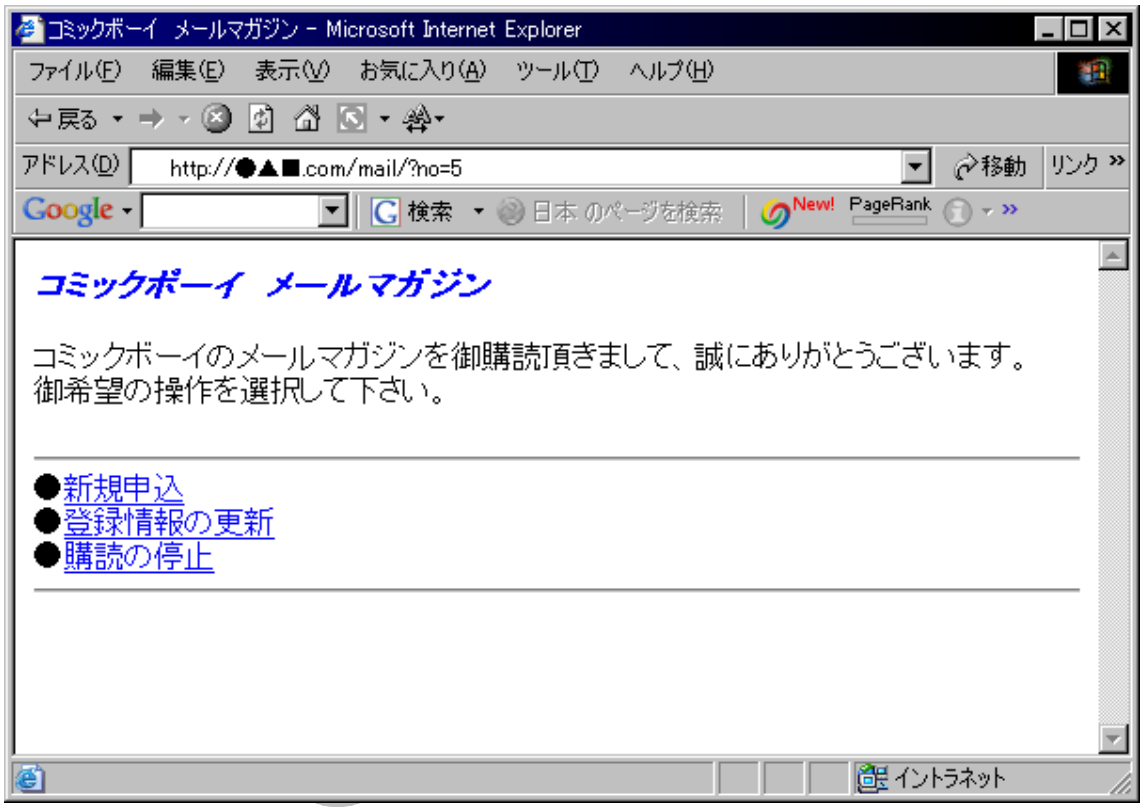
新＞

```
function get_artist($no,$con){
        $no = (int)$no; // $no を数値に変換する事でサニタイジング
        $sql = "select n_artist from artist where no = $no";
        $result = pg_exec($sql);
        return pg_result($result,0,0);
}
function get_table($no,$con){
        $no = (int)$no; // $no を数値に変換する事でサニタイジング
        $sql = "select n_table from artist where no = $no";
        $result = pg_exec($sql);
        return pg_result($result,0,0);
}
```

2-1-6. セキュリティホール発見に至るまでの経緯のシミュレーション

以下に、内部情報に精通していないジョージが、セキュリティホールを発見し、破壊活動を行うまでの過程をシミュレートする。

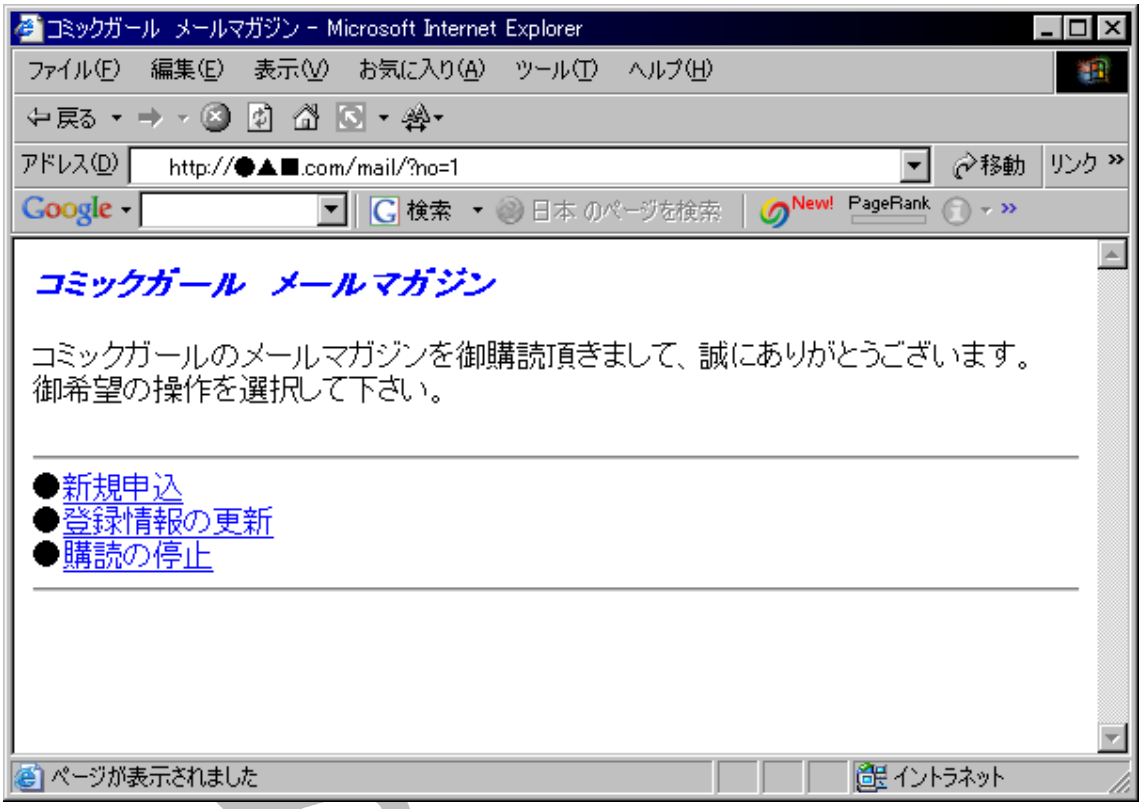
漫画家「コミックボーイ」に興味のあるジョージは、ある日コミックポータルサイト http://●▲■.com/ を訪れ、コミックボーイのメールマガジンの登録ができる事を知った。登録ページの URL は http://●▲■.com /mail/?no=5 であり、画面は以下のような物だった。



ジョージは URL に付与しているパラメーター**no=5** に興味を持った。この数字を変える事によって何か新しい情報が得られると思ったジョージは、ブラウザのアドレスバーに以下の文字列を入力した。

http://●▲■.com /mail/?no=1

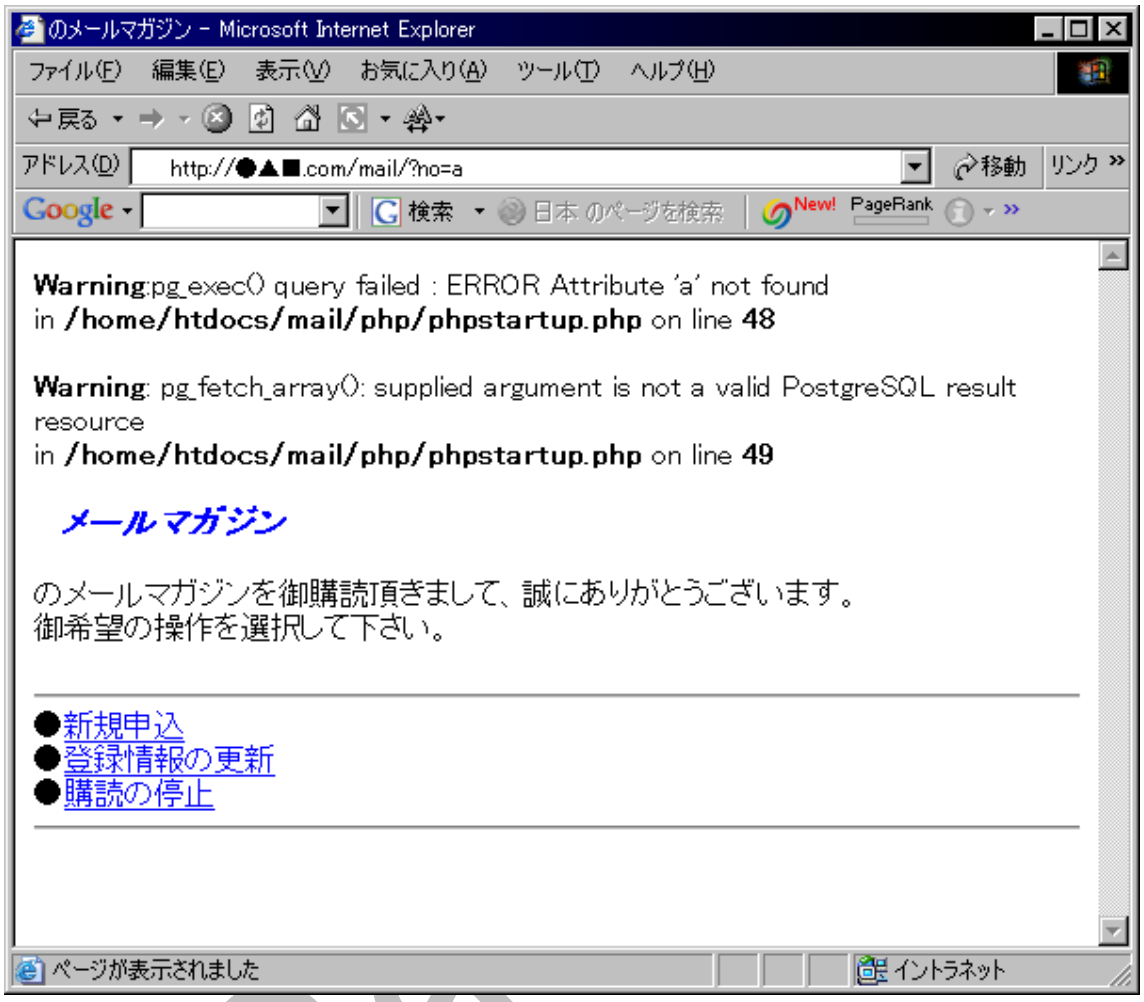
すると以下のような画面が表示された。



コミックガールが別の漫画家である事を知っていたジョージは URL パラメーターの**"no="**の後の番号がそれぞれの漫画家に与えられた番号であると認識した。そこでジョージは他にはどんな漫画家が登録されているのかに興味を持ち、次々と番号を変え登録されている漫画家を調べた。また、**"no="**の後に数値以外の物を入力するとどうなるのか興味を持ち、次のアドレスにアクセスした。

http:// ●▲■.com /mail/?no=a

そしてジョージは以下の画面出力を得る事が出来た。



オープンソースに精通しているジョージは、ここで以下の情報を得る。

- このシステムは php スクリプトによって動作している。
- このシステムはデータベースに PostgreSQL を利用している。
- このシステムは URL に与えられたパラメーターを利用してデータベースに問い合わせを行うが、その際スクリプト内部でエラーチェックを行っていないようだ。
- "Attribute 'a' not found"というメッセージから、URL に入力された文字列が直接データベースに引き渡されている。数値を指定していれば問題ないが、非数値である'a'を入力した事によってエラーメッセージが画面に表示された。

これらの情報から、ジョージはこのスクリプトが以下のような SQL を利用していると推測した。

```
select * from テーブル名 where id=[URL で指定したパラメーター]
```

パラメーターには数値だけでは無く、あらゆる情報を入力できる。だとするならば、データベースに特別な指令を与える事で、このシステムを破壊する事ができるのではないだろうか？
もはやコミックボーイの事よりシステムクラッキングへ興味が移行しているジョージは、SQL インジェクションの可能性をさぐるために以下の URL を入力した。

テスト１
http:// ●▲■.com/mail/?no=5;abc

テスト２
http:// ●▲■.com /mail//?no=5;select%201

２つの URL を入力した所、テスト１の場合には**"parse error at or near "abc"**と表示されるが、テスト２の場合にはエラーが表示されなかった。テスト２においてジョージは、データベースに対し無害な SQL コマンドを発行しようとしている。こうする事で SQL が解釈されているかどうかを探るのが目的だった。テスト２においてジョージの推測が正しければ、実行される SQL は以下の通りとなる。

select * from テーブル名 where id=5;select 1

‘a’や’abc’を入力した時にはエラーメッセージが表示されるのに、テスト２のケースではエラーが表示されないという事は、セミコロン以下の情報がデータベースへの入力として受け付けられたと考えられる。これは SQL インジェクションを行える可能性が高い事を意味している。
日頃面白くない事が続いているジョージは、このシステムを破壊する事にした。しかし自分が逮捕される事は避けたかったため、ジョージは他人に破壊させる事を思い付いた。これを実現するために、多段階プロキシを利用し大手コミュニティサイトに以下の書き込みを行った。

超ふざけたサイト発見！
http:// ●▲■.com /mail/?no=5;delete%20from%20member
http:// ●▲■.com /mail/?no=5;delete%20from%20master
http:// ●▲■.com /mail/?no=5;delete%20from%20user
http:// ●▲■.com /mail/?no=5;delete%20from%20user_master

ジョージはテーブルを破壊するために DELETE コマンドを利用したいと思ったが、実際に存在するテーブルの名前までは分からなかったので、ありそうなテーブル名を列挙して書き込みを行った。そしてジョージの思惑通り、大切な user テーブルが危機にさらされる事になってしまった。
興味を持った第三者が上記 URL のうち３番目をクリックすると、それだけでユーザー情報がすべて消えてしまうだろう。

……

ある日より http:// ●▲■.com からメールマガジンが送られてこなくなってしまった。サイトのトップページには障害発生のためメールマガジンはしばらく休みになると記載されている。
世界のどこかで、ジョージはほくそ笑んでいる事だろう。

2-1-7. 考察

上記で述べた例のように、ジョージは内部精通者では無いにも関わらずシステムのセキュリティホールを見つける事が出来る。ジョージと同等の考察を行える人物は PHP と PostgreSQL に関する一般的な知識を持っている必要があるが、これはそれほど高いスキルではない。また、エラーメッセージ等からシステムの推測も容易である。

ジョージは SQL インジェクションの調査過程においてサーバーのアクセスログに痕跡を残す事になるが、各々の行程の違法性は低いと考えらる。自分のアドレスバーに任意の文字列を入力する行為だけで誰かを罰する事は、おそらく不可能だろう。

また実際に攻撃を行う際、ジョージは掲示板を利用している。このため破壊活動は善意の第三者に行わせる事が可能になる。掲示板の書き込みで破壊活動を促す事はなんらかの幇助罪が適用できると思われるが、当方は法の専門家では無いため定かではない。また多段プロキシを利用して掲示板に書き込みを行ったとすれば、実際にジョージがどこの誰なのかを特定する事は極めて難しい。

影響の大きさを考えれば、早急な対応が望まれる。

3. 根底原因に関する考察

斯様なセキュリティーホールが発見された場合、兎角責任の標的になるのが実際にコーディングしたプログラマである。しかし、プログラムコーディング技術とセキュリティ技術は異なる分野の技術であり、知識である。例えば完全にオフラインのスタンドアロンなプログラムであれば、セキュリティ技術をまったく必要としない事も多い。

プログラムコーディングは開発行程全体のうち、ほんの一部である。小規模システムにおいては、コーディングが開発行程のほぼ 100%をしめているように思える事もあるが、それが大きな間違いである事は、今回のレポートからも明らかである。どのようなポリシーでどのようなシステムを開発するかと言う事を明確に指示できる人間が開発スタッフの上流行程に存在していれば、今回のような問題を避ける事ができた。例えば一言プログラマに対し「クロスサイトスクリプティングと SQL インジェクションについて調べて欲しい。こういう事が起こらないようなシステムを開発して欲しい」と指示を出す事が可能だったとしたら、それだけでも問題を軽減できた可能性がある。あるいはテスト専門会社にプログラムのテストを依頼する事でも問題を避けられた可能性が高い。

セキュリティーホールを無くしたいと思ったら、一人の開発者に責任のすべてを負わせる形では絶対にうまく行かない。バグと同じく、セキュリティ技術にかなり精通したプログラマであっても、ケアレスミスからセキュリティーホールを作ってしまう事がある。このような避け難いヒューマンエラーを回避するためには、開発行程内にテスト用の期間と予算を多めに取る事、セキュリティホールに対応した社内フレームワークを構築する事、セキュリティ技術に精通した人間を上流行程に配置する事等が必要である。