SafeNet

Sentinel
Software Monetization Solutions

Java アプリケーションの著作権 およびリバースエンジニアリング対策

― 最新リリース Sentinel HASP 5.1 ―

WHITE PAPER

近年、開発者は公開されている .class ファイル形式とシンプルな命令セットによる使い勝手の良さからオープンソース開発環境として Java プラットフォームを選んでおり、その数は増え続けています。オープンソースアプローチの使用には多くの利点があるものの、コードは完全に公開されたままの状態です。長年にわたりソフトウェア開発に費やした時間と投資に基づく会社の知的財産（IP; Intellectual Property）への脅威が、今明らかになっています。

目次

Java オープンソースのリスクと弱点

- **自由に入手可能な JVM—**
  ハッカーは市販の逆アセンブラを利用してコードを調べ、JVM の動作を正確に把握できます。コードは完全にシースルーになっています
- **公開されている Java .class—**
  ハッカーは容易に入手できるツールを利用して .class ファイルを処理、修正、変換できます
- **命令数の少なさ—**
  JAVA の命令数は少ないため（ネイティブコードの最大 400 より少ない 200 未満）、ハッカーはコードをより迅速に解析してリバースエンジニアリングできます
- **制御のしやすさ—**
  ハッカーは特定のハードウェアレベルまで下りる必要がなく、より簡単にデータを見て解析できます

要旨

Java 環境はプラットフォームに依存しないため、開発者にますます人気となっています。しかし、Java のオープンソースコードと公開されている .class ファイル形式、またシンプルな命令セットは、悪意のあるコード解析にさらされたままの状態です。企業は市販のアプリケーションを Java に移行しているため、その重要な知的財産は、リバースエンジニアリング、コード操作、盗難の危険にさらされています。現在、.class ファイルの暗号化とソースコードやバイトコードレベルでの難読化の 2 つが、リバースエンジニアリングを防ぐために開発者が使用している主な対策です。それでもまだ脆弱性は残っています。

このホワイトペーパーでは、Java コードの脆弱性の本質や、既存のソリューションの限界、さらに Java コードの包括的なエンベロープ（難読化と暗号化の組み合わせ）がいかにして重要なアルゴリズムやその他の知的財産を保護する理想的なソリューションになるかについて検証します。

ますます人気の高まる Java アプリケーション

現在、世界の IT 企業のほとんどがある程度まで Java を受け入れています。数多くの製品が、Windows や UNIX プラットフォーム向けではなく、Java プラットフォーム向けに Java で記述されています。

他のプログラミング言語に勝る Java の最も重要な利点は、プラットフォームに依存しないという点です。つまり、ほぼすべてのアーキテクチャとオペレーティングシステムで動作します。アプリケーションを Intel や PowerPC などの特定のハードウェアアーキテクチャに関連付けるのではなく、Java はすべてのアーキテクチャに対して単一のコードベースを使用します。ソフトウェア開発者は、事前定義済みの Java API のソフトウェアパッケージを使用する Java 言語でプログラムを記述します。その後、開発者は Java コンパイラを使用して、これらのプログラムをコンパイルします。その結果、Java 仮想マシン（JVM）上で実行できるコンパイルされたバイトコードになります。JVM は、Java プログラムが実行されるソフトウェア環境です（たとえば、Java Runtime Environment）。ユーザは、JVM がある場合にのみ Java プログラムを実行できます。しかし、JVM は多くのプラットフォームで利用可能であり、Java を移植性の高い言語にしています。

リバースエンジニアリングに非常に脆弱な Java

「一度記述すればどこでも実行可能」であることは大きな利点ですが、この環境の設計法はハッカーにとってネイティブアプリケーションよりもはるかにリバースエンジニアリングしやすいものになっています。つまり開発者は、知的財産を失う極めて現実的な危険に直面しているのです。アプリケーションベースの仮想マシンがネイティブアプリケーションよりもリバースエンジニアリングしやすい理由は数多くあります。

オープンソースである JVM

Sun は JVM 用の無料で利用できるソースコードを作成しました。これはハッカーにとっては、コードを簡単に調べて仮想マシンの動作を簡単に正確に把握できることを意味します。

公開されている Java .class ファイル形式

先に述べたとおり、Java ソースコードはバイトコードにコンパイルされ、Java .class ファイルに格納されます。Java .class ファイル形式の仕様は公開されており、技術的な知識のある人なら誰でも .class ファイルを処理、修正、変換できるツールを容易に記述することができます。

JVM はハードウェアではなくソフトウェア

特定のプロセッサの専門的な理解を必要とする標準的なプログラミング言語と異なり、JVM はマイクロプロセッサのように動作し、オペレーティングシステムとコンピュータハードウェアで提供される組み込み機能を使用するアプリケーションです。ハッカーはハードウェアレベルまで下りる必要がないため、JVM を簡単に完全に制御できます。

たとえば標準のネイティブシステム開発言語でデバッグする場合、プロセッサを一時停止することは非常に難しく、プロセッサやデバッグ機能、利用可能なリングデバッガに関する専門知識が必要です。しかし、JVM 実行環境のソースは公開されているため、開発者は仮想プロセッサのあらゆる側面の完全な制御を提供する独自の仮想マシンを簡単に構築できます。このため、この実行環境で動作するあらゆるアプリケーションが容易に解析できます。

Sentinel のエンベロープの機能と利点

- **自動ファイルラッパー—**
  ファイル暗号化とネイティブコードの難読化によってソフトウェアのリバースエンジニアリングに対する強力な防御を提供します
- **ハードウェアへのアプリケーションの再結合—**
  アプリケーションがすぐにプロテクションキーによってハードウェアに結合されます
- **セキュアな通信チャネル—**
  Sentinel HASP は、プロテクションされたアプリケーションとプロテクションキーの間の通信にセキュアなチャネルを提供することで中間攻撃を排除します。Java Envelope はこの機能を利用して、ハッカーが通信を妨害してプロテクションキーから送り返されるデータにアクセスすることを防ぎます
- **実行時の復号化—**
  Sentinel HASP は、一度にすべての .class ファイルを仮想マシンにロードしません。実行時に要求されたファイルのみを復号化するため、ハッカーによってアプリケーション全体が再構築されることを防ぎます

ネイティブコードよりも命令数の少ない Java

JVM コードがリバースエンジニアリングしやすいもう 1 つの原因は、ネイティブアプリケーションよりも命令数が少ないとことにあります。命令数が少ない理由は、パフォーマンスにあります。JVM の使用はアプリケーションとネイティブプロセッサ間にソフトウェアの層を追加するため、パフォーマンスに悪影響を与えます。今のままプロセッサ実行速度が向上していくと最終的にこの問題は解決されますが、それはまだ先の話です。仮想マシンの開発者が実行速度を改善してきた方法の 1 つが、ネイティブプロセッサアセンブラと比べて小さいバイトコード命令セットを使用することでした。

ネイティブアプリケーションは最大 400 の命令で構成されますが、Java アプリケーションが通常使用するのは 200 未満です。命令がより少ないことは、クラッカーがリバースエンジニアリングのためにより素早くコードを解析できることを意味します。

これらの特性により、仮想マシンは他のタイプのアプリケーションよりもリバースエンジニアリング攻撃に対してはるかに脆弱になります。

脆弱性を高めるサードパーティの逆アセンブラ

JVM はリバースエンジニアリング攻撃に対して本質的に脆弱であるだけでなく、市販やフリーウェアの Java バイトコード逆アセンブラがますます入手しやすくなっており、コードのリバースエンジニアリングのプロセスをさらに簡潔にしています。

市場に出回っている多くの Java バイトコード逆アセンブラのうち代表的なものが、IDA と Eclipse 用バイトコードプラグインです。市販製品である IDA は、80x86 や MIPS を含む様々なプロセッサで動作するユニバーサルな逆アセンブラです。Eclipse 用バイトコードプラグインはフリーウェアです。これは、Java .class ファイルのバイトコードを逆コンパイルして、すべての opcode 命令を適切な順序で表示できます。

これらの製品はバイトコードからオリジナルのソースコードを完全に復元することはなさそうですが、オリジナルのソースコードと同等でバイトコードよりもはるかに読みやすいソースコードを復元します。ソースコードが復元されると、攻撃者はコード部分を簡単に取り出し、それを競合他社のアプリケーションで違法に使用したり、.class ファイル内でパッチをあてる場所を特定したりすることが容易にできます。

図 1 は、ハッカーが .class ファイルにパッチをあてる方法の例を示しています。スクリーンの上半分は、Java ソースコードの一部分を示しています。スクリーンの下半分は、バイトコードの逆アセンブルの出力を示しています。これはバイトコード命令のリストです。赤色で示した部分は、ソースコードの IF 構造の比較命令です。バイトコード命令「LCMP」には、16 進表記の 0x94 があります。またツールは、opcode が .class ファイル内のどこに位置しているかを特定します。ハッカーは、この情報とシンプルな 16 進エディタを使用して 1 分足らずで IF 分岐を改変できます。IF 条件がライセンスチェックであると仮定すると、ハッカーは検証されるライセンスが無効（たとえば期限切れ）である場合でも、条件を簡単に逆転して「True」を返すように命令することで違反を犯すことができます。この場合ハッカーは 1 バイトのパッチですべての作業を完了できます。

多くのアプリケーションはこの例よりも複雑ですが、バイトコードは複雑なアプリケーションでも理解しやすく単純です。

JVM は、一度アプリケーションを記述すればほぼどんなプラットフォームでもそれを実行できる能力を開発者に与えますが、ハッカーが簡単にソースコードをリバースエンジニアリング、操作、盗難できてしまうという大きなデメリットもあります。

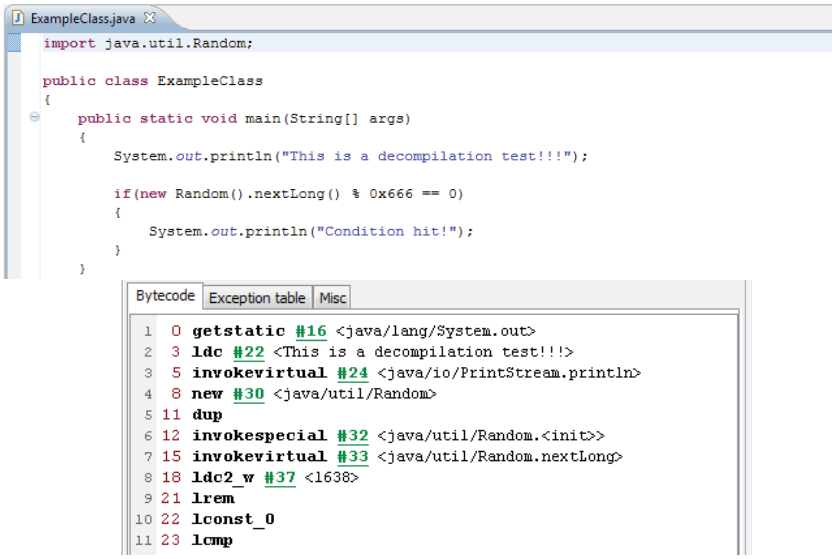

図 1 - バイトコードの逆アセンブルを行える Eclipse バイトコードプラグイン

リバースエンジニアリングを防ぐために搭載された対策が不十分である理由

ほとんどの仮想マシンにはリバースエンジニアリングを複雑化する機能が含まれています。Java では、ユーザはオリジナルファイルの改変を確実に防ぐために JAR アーカイブ内に準備されている各クラスにデジタル証明書を配置できます。この方法をとるしかありませんが、この機能は極めて簡単に取り除くことができ、攻撃シナリオの中でも少数の静的なパッチアプローチを防ぐだけです。このアプローチは実行時にメモリに適用されるパッチを防ぐことができません。

また Java は仮想マシンでバイトコードベリファイアを実行し、渡されたバイトコードを実行前に自動的に解析します。これにより、「不審な」コードの実行を防ぎ、バイトコードの挿入をより難しくしています。

しかし、これらの対策は攻撃者を苦しめたとしても、知的財産を完全に保護するまでには至りません。

リバースエンジニアリング攻撃の防御

開発者が静的な .class ファイルの解析とバイトコードの逆アセンブルを防ぐために一般的に使用する方法は、エンベロープです。これは、完全なファイル暗号化 / 復号化を適用して .class ファイルの解析を防ぎます。エンベロープにより、開発者は保護ファイルのオリジナルローダーを、暗号化 / 復号化を扱うカスタムローダーと交換します。暗号化は標準の Java .class ファイル形式から、「鍵」を処理する人以外が読めない形式へと変えるアルゴリズムを使用することで .class ファイルの解析を防ぎます。それでもなお .class ファイルのバイトコードはメモリ内の一カ所で暗号化されていない状態のままで残り、そこは通常システムクラスローダーがクラスをロードしようとする前に一瞬アクセス可能になります。ハッカーがそのメモリの場所を見つけることができたら、オリジナルの状態のクラスにアクセスできてしまいます。

ハッカーによるこのメモリの場所の攻撃を防ぐには、難読化と呼ばれている 2 つ目の手法が必要です。難読化は、オリジナルのコードと同じように振る舞う、より複雑で理解しにくいコードを生成します。以下は、Windows バイナリから抜粋した単純な 80x86 アセンブラコードの断片です。

Java は固有のセキュリティ対策をいくつか提供しますが、これらの機能は攻撃を完全に防ぐには不十分です。暗号化やコード難読化などの手法は攻撃者の行動を遅らせるために一般的に使用されていますが、それでも脆弱性は残ったままです。Java エンベロープは、暗号化とネイティブコードの難読化の組み合わせによって最も強力な保護を提供します。Sentinel HASP ソリューションを使用することにより、自家製ソリューションを開発することに貴重な時間と労力を費やすことなく、エンベロープの多くの利点を得ることができます。

```
text:0040C609 ; =============== S U B R O U T I N E =======================
text:0040C609
text:0040C609
text:0040C609 sub_40C609          proc near          ; DATA XREF: .rdata:004561A4o
text:0040C609
text:0040C609 arg_0         = dword ptr 4
text:0040C609 Src                     = dword ptr 0Ch
text:0040C609 Size                    = dword ptr 10h
   ext:0040C609

   push  ebx
push     esi
push     [esp+8+Size] ; Size
mov esi, [esp+0Ch+arg_0]
push     [esp+0Ch+Src] ; Src
push     0 ; int

push     dword ptr [esi+0Ch] ; int
call      sub_43AF3E

   mov ecx,          [esi+4]
mov edx,          [ecx]
xor       ebx, ebx
cmp eax, 1000h
setnle    bl
push     ebx
push     ecx
call      dword ptr [edx+1Ch]
add  esp, 18h
xor       eax, eax
pop       esi
inc       eax
pop       ebx
retn
```

これからコードをより理解しにくくするために、新しい命令を追加して他のものを交換することでコード断片の難読化を試みます。

以下は難読化したバージョンです。

```
text:0040C609 ; =============== S U B R O U T I N E =======================
text:0040C609
text:0040C609
text:0040C609 sub_40C609 proc near ; DATA XREF: .rdata:004561A4o
text:0040C609
text:0040C609 arg_0 = dword ptr 4
text:0040C609 Src = dword ptr 0Ch
text:0040C609 Size = dword ptr 10h
text:0040C609
push ebx
push esi
push [esp+8+Size] ; Size
mov esi, [esp+0Ch+arg_0]
push [esp+0Ch+Src] ; Src
mov ecx, 1024
```

Sentinel HASP は一度にすべての .class ファイルを仮想マシンにロードするのではなく、実行時に要求された .class ファイルを復号化するため、ハッカーによってアプリケーション全体を再構築されるのを防ぎます。

```
sub ecx, 1000
add ecx, 6
sub ecx, 30
push ecx

push dword ptr [esi+0Ch] ; int
call sub_43AF3E
push ecx
pop ecx

mov ecx, [esi+4]
mov edx, [ecx]
mov ebx, 2000
add ebx, 2000
add ebx, 2000
sub ebx, 6000
cmp eax, 1000h
mov eax, 1000h
setnle bl
push ebx
push ecx
call dword ptr [edx+1Ch]
add esp, 18h
xor eax, eax
pop esi
inc eax
pop ebx
retn
```

この例では算術命令を追加することによって opcode の断片を拡張しました。それによってコードはより読みにくくなりました。この断片からは容易に読むことができないため、ハッカーは何が起きているかを理解するために計算を行わなくてはなりません。

この例はまだ理解しやすいままの状態ですが、こうした難読化を必要なだけ繰り返し、この単純な断片を 1000 行以上ものコードに拡張してコードが何を行っているか理解しにくくすることが可能です。難読化を使用すると、攻撃者はオリジナルのコード断片を突き止めるための特殊なツールを構築せざるを得なくなります。これは決して簡単なことではありません。

数種類の難読化方式が命令セットに利用可能です。

プログラマはバイナリ内のすべての文字列を置き換えることでコードを難読化できます。これにより、リバースエンジニアリング攻撃を開始するのに格好の場所を見つけることがより困難になります。プログラマは、ジャンプをジャンクコードと戻り値に挿入することによって攻撃者を困惑させ、ソースコードやバイトコードを難読化することができます。

また開発者には、Const2Code 変換と呼ばれる手法を使用して、定数を難読化するオプションもあります。たとえば、秘密鍵はバイトセットとしてアプリケーション内に格納されることがあります。ハッカーは、そのバイトの場所を特定した場合のみ、それらにアクセスできます。秘密鍵などの定数を難読化するために、Const2Code アルゴリズムは定数を同じ定数になる複数の異なるコマンドに変換します。

たとえば、ソースコード断片の定数 cst = 0x12345678 を隠すために、その定数を複数の算術演算（add、sub など）に簡単に分割できます。A = 0x9ABCDF00; B=0x2 C=0x135799E00。ここで、これら 3 つの変数から定数 0x12345678 の戻り値を計算してみましょう。オリジナルの定数は、cst = C / B + A - 0x88888888 - A + 1000= 0x12345678 です。定数を計算するためにこの 1 つのルーチンしかアプリケーションに含まれていない場合、攻撃者は定数を単純に取り出せず、まず何が起きているのかを理解することから始めなければなりません。

ここで、ソースコードの難読化の例を見てみましょう。

図 2 には、秘密鍵を隠すために C2C-Algorithm によってすでに難読化されたオリジナルコードが含まれています。

ExampleClass.java ExampleClass.java

```
import java.util.Random;

public class ExampleClass
{
    public static int A_MAGIC_KEY = 0;

    public static void main(String[] args)
    {
        A_MAGIC_KEY = 0x101;

        for(int counter = 0; counter < 10; counter++)
        {
            A_MAGIC_KEY = A_MAGIC_KEY * (counter + 1) + 15 / 2;
        }

        System.out.println("This is a magic encryption key: " + A_MAGIC_KEY);

        System.out.println("This is a decompilation test!!!");

        if(new Random().nextLong() % 0x666 == 0)
        {
            System.out.println("Condition hit!");
        }
    }
```

ExampleClass.java ExampleClass.java *ExampleClass.java

```
public class ExampleClass
{
    public static int A_BANANA  = 0;
    public static int A_APPLE   = 0x0BADBABE;
    public static int A_CHICKEN = 0x123;
    public static int SALAD     = 0x123;
    public static int DRESSING  = 0x123;

    public static int A_MAGIC_KEY = 0;

    public static void main(String[] args)
    {
        A_CHICKEN = SALAD- DRESSING;

        System.out.println(A_CHICKEN);
        for(int private_static_int = 0; private_static_int < 1001; private_static_int++)
        {
            A_BANANA = A_APPLE;
            A_APPLE  = private_static_int;

            if(private_static_int > (SALAD - DRESSING) && private_static_int <= (SALAD - DRESSIN
            {
                A_CHICKEN = (A_CHICKEN * ((private_static_int - (SALAD - DRESSING) + 1))) + 15 /
                System.out.println(A_CHICKEN);
            }
        }
    }
```

図 2 – Java ソース例は難読化できます。秘密鍵を計算する基本の C2C-Algorithm によるシンプルなコード断片です。

図 3 – Java ソース例は、手動でソースコードが難読化されています。

図 3 に示したとおり、手動によるソースコードの難読化後の例はまったく異なって見えます。一見したところでは、手動で難読化されたコードが同じ秘密鍵を計算することに気付かないでしょう。

この場合、ソースコードが異なるだけでなく、生成されたバイトコードもまったく異なります。多数のバイトコードを生成できるため、ソースコードが難読化されたプログラムの解析は、さらに困難になります。

SafeNet Sentinel: より簡単なエンベロープ方法

Sentinel HASP Envelope は、ファイル暗号化とネイティブコード難読化によってソフトウェアのリバースエンジニアリングに対する強力な防御を提供する自動ファイルラッパーです。これにより、ソフトウェアに埋め込まれたアルゴリズム、企業秘密、専門的なノウハウをハッカーから確実に保護できます。Sentinel HASP は、Java アプリケーションをハードウェアプラットフォームに再接続することによって、非常にセキュアな知的財産保護を提供します。これにより攻撃者は Java アプリケーションだけでなく、保護された Java アプリケーションを解読するためにネイティブコードもリバースエンジニアリングすることを余儀なくされます。その結果、攻撃者はエンベロープされた Java コードを解読するために極めて高度な知識が必要になります。

LicensingLive! ™ (lahy' sun sing lahyv'), adj. n. [SAFENET, INTERACTIVE]

1. ソフトウェアのパッケージ化、価格設定、調達、提供、管理に関わるベストプラクティスと新たな課題をご紹介します。

2. ソフトウェアベンダ、業界アナリスト、ライセンスコンサルタント、テクノロジーベンダが集うフォーラムです。

リンクサイト：

Sentinel Online
www.Safenet-inc.com/sentinel

LicensingLive!
www.LicensingLive.com

Twitter
twitter.com/LicensingLive

LinkedIn
http://bit.ly/LinkedInLicensingLive

YouTube
http://www.youtube.com/user/LicensingLive

BrightTALK
http://www.brighttalk.com/channel/5572

本ホワイトペーパーの内容、製品・ソリューションについてのお問合せは下記までお願いいたします。

日本セーフネット株式会社

SRM ソリューション事業部
東京都港区新橋 6-17-17
御成門センタービル 8F
Tel: 03-5776-2751
Email: SalesRM-Japan@safenet-inc.com

Sentinel HASP は、保護されたアプリケーションと保護キーの間の通信にセキュアなチャネルを提供することで、中間者攻撃を排除します。Java Envelope はこの機能を利用して、ハッカーが通信を妨害して保護キーから送り返されるデータにアクセスするのを防ぎます。

Sentinel HASP は、一度にすべての .class ファイルを仮想マシンにロードするのではなく、実行時に要求された .class ファイルを復号化するため、ハッカーによってアプリケーション全体が再構築されるのを防ぎます。

最後に

JVM を使うと、開発者は一度アプリケーションを記述すれば、ほぼどんなプラットフォームでもそれを実行できますが、ハッカーが簡単にソースコードをリバースエンジニアリング、操作、盗難できてしまうという大きなデメリットもあります。市販されている多くの逆アセンブリがこのプロセスをさらに容易にしています。また Java はセキュリティを提供していますが、搭載されている機能では攻撃者を阻止するには不十分です。暗号化や難読化などの手法は攻撃者の行動を遅らせるために一般的に使用されていますが、それでも脆弱性は残ったままです。エンベロープは、暗号化とネイティブコードの難読化の組み合わせによって、知的財産の保護を可能にする最も強力なデータ保護を提供します。Sentinel HASP ソリューションを使用することで、一からソリューションを開発することに時間と労力を費やすことなく、エンベロープの利点を得ることができます。

SafeNet Sentinel ソフトウェア収益化ソリューション

SafeNet は、世界中のソフトウェアベンダとテクノロジーベンダに、革新的で信頼できるソフトウェアライセンシングおよびエンタイトルメント管理ソリューションを 25 年以上も提供してきました。

統合しやすく使いやすい、革新的な機能重視の Sentinel® ソフトウェア収益化ソリューションは、規模や技術要件、組織構造を問わず、どんな組織に対しても固有のライセンス有効化、施行、管理要件を満たすように設計されています。全体の収益性を向上させ、社内業務を改善し、競争力を維持し、顧客やエンドユーザとの関係を深めつつ、著作権侵害対策、IP 保護、ライセンス有効化、ライセンス管理の課題、あらゆるソフトウェア収益向上に関する課題に SafeNet はお客さまとともに取組んでいます。SafeNet には、進化し続けるマーケットに対応するため、新たな要件に適応し新たなテクノロジーを取り入れてきた実績があります。世界中の 25,000 以上のお客様が、Sentinel を選択することが、今日、明日、そしてその先のビジネスのやり方を発展させていく自由を手に入れることだと考えています。

インストールされたアプリケーション、組込みアプリケーション、クラウドアプリケーション用の SafeNet ソフトウェア収益化ソリューションの完全なポートフォリオに関する詳細、また無料評価版をダウンロードするには、下記をご覧ください。

http://www.safenet-inc.com/sentinel/

無料の Sentinel HASP 開発者キットをダウンロードするには、下記をご覧ください：

http://www3.safenet-inc.com/Special/hasp/safenet-hasp-srm-order/default.asp

SafeNet について

SafeNet は、1983 年に設立された、情報セキュリティ業界における世界的なリーダー企業です。SafeNet は、お客様の貴重な資産である ID や、トランザクション、通信、データ、ソフトウェアライセンスを IT セキュリティの視点から、情報ライフサイクル全般にわたり保護しています。SafeNet のお客様は、100 カ国以上、2 万 5 千を超える企業や政府機関に及び、その情報セキュリティの保護を SafeNet に委ねています。

日本セーフネットについて
日本セーフネット株式会社（http://jp.safenet-inc.com 代表取締役社長：酒匂 潔、本社：東京都港区）は、米国 SafeNet, Inc. の日本法人で、2001 年の設立以来、ネットワークやアプリケーションのセキュリティ製品の日本国内での販売、マーケティング、サポートを提供しています。