MS00-057 最終検証レポート

ＮＴＴ コミュニケーションズ株式会社
ＩＴ マネジメントサービス事業部
セキュリティオペレーションセンタ

2009 年 5 月 26 日

Ver. 1.1

1. 調査概要

Apache-Tomcat に存在する UTF-8 の冗長表現を使った Directory Traversal 問題(7 参考①)に関連して、MS00-057 も UTF-8 の冗長表現を使った手法であることから同様の検証を実施した。検証結果は以下となった。

- 3Byte の冗長な UTF-8 表現でも、問題が発現する
- 日本語 MS-Windows2000+IIS では、「.(ピリオド)」、「/(スラッシュ)」、「\(バックスラッシュ)」だけではなく、UNICODE で分離される「¥(円記号)」でも発現する

2. 前提情報や、対策方法などについて

本レポートでは、検証に直接関係すること、また検証結果からの考察以外は記述しない。
UTF-8 さらに、UTF-8 の冗長表現については、「7 参考①」で記した Web サイトを参照してほしい。
また、UNICODE によって分離された文字を使ったサニタイジング回避テクニックについても、「7 参考④」で記した Web サイトを参照してほしい。

さらに、MS00-057 自体が過去のセキュリティ問題であり、最新のサービスパックによって既に修正されている。よって、インターネット上に最新のサービスパックが適用されていない MS-Windows サーバは存在しないだろうと想定されるため、敢えて記述しない。しかしながら対策についての具体的記述については、「7 参考②」で記した Microsoft 社の Web サイトを参照してほしい。

3. MS00-057 についての検証結果

3.1. MS00-057 の概要

MS00-057 とは、サービスパックの適用されていない Microsoft Windows2000 上の IIS(Internet Information Service)において、「../」の「/(スラッシュ)」を冗長な UTF-8 表現とすることで、ディレクトリ・トラバーサルが可能となるセキュリティ問題である。
もし、CGI 実行権のある Web 仮想ディレクトリ上でこの攻撃が行われる場合、ディレクトリ・トラバーサルによって、Web 仮想ディレクトリと同一ドライブにある任意のコマンドが IUSER 権限で実行されることになる。

対策として、「7 参考②」で記した Microsoft 社が提供しているパッチを適用すること。また最新の MS-Windows(Windows2000SP4 以降)ではこのような問題は発現しない。
このような対策とは別に、MS-Windows システムがインストールされているフォルダ(一般的に C ドライブ)と、Web 仮想ディレクトリのあるフォルダを別のドライブ(例えば D ドライブ)に分離することも対策となる。

OS システムとデータ(Web コンテンツ)を別ドライブに分離し、可用性を高めていた MS-Windows システムはこの攻撃に耐性があったということになる。

3.2. 検証環境

以下の環境で、検証を行った。

- Microsoft-Windows2000 Server 日本語版 SP 未適用
- Microsoft-Windows2000 Server 英語版 SP 未適用

「Microsoft-Windows2000 Server 日本語版 SP 未適用」では全項目を行った。「Microsoft-Windows2000 Server 英語版 SP 未適用」では一部項目を実施した(3.6 参照)。

3.3. 検証内容(冗長な UTF-8 表現)

図 3.3-1～図 3.3-6 は、「/(スラッシュ)」「\(バックスラッシュ)」「.(ピリオド)」の 2Byte、および 3Byte の冗長な UTF-8 表現について検証した結果である。
図を見ると確認できるが、「/(スラッシュ)」「\(バックスラッシュ)」「.(ピリオド)」の 2Byte、および 3Byte の冗長な UTF-8 表現によって、MS00-057 問題が発現している。

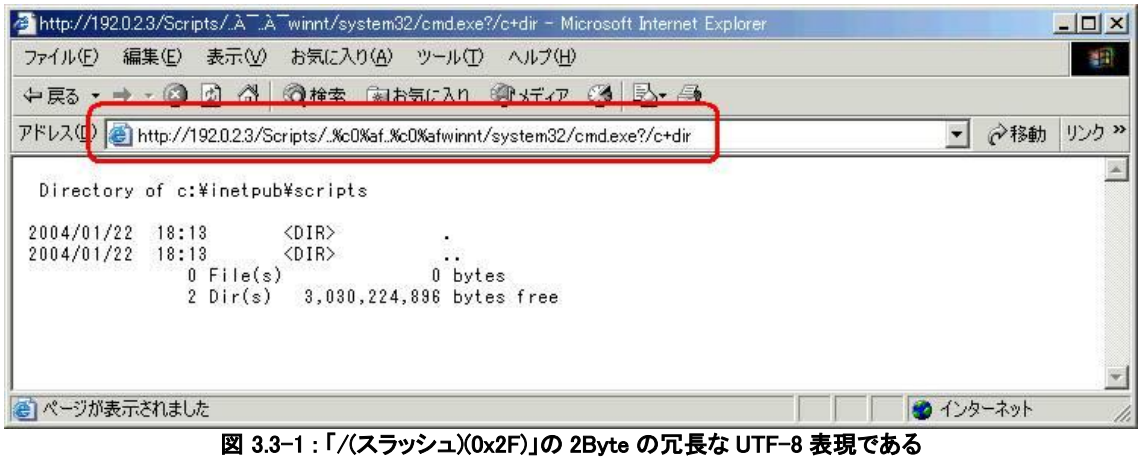

図 3.3-1 :「/(スラッシュ)(0x2F)」の 2Byte の冗長な UTF-8 表現である

「%C0%AF」の場合、問題が発現する

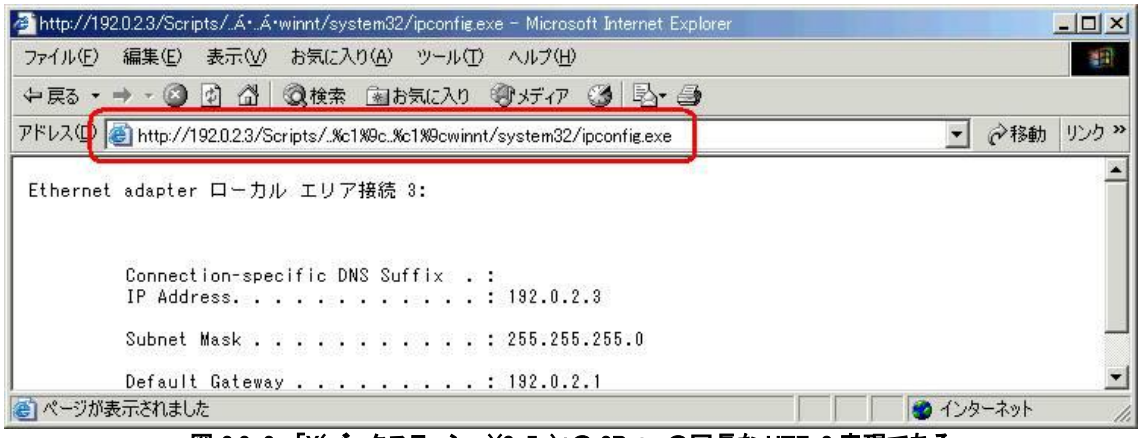

図 3.3-2 :「¥(バックスラッシュ)(0x5c)」の 2Byte の冗長な UTF-8 表現である

「%C1%9C」の場合、問題が発現する

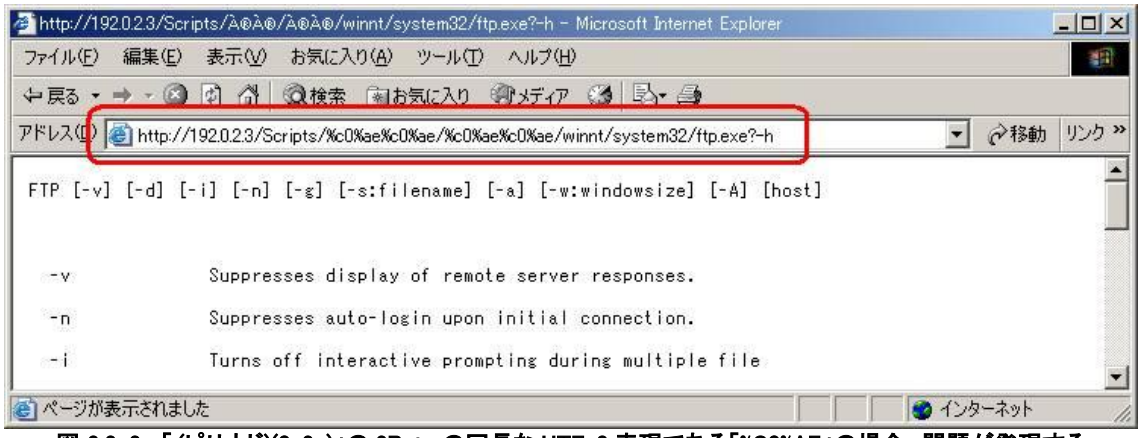

図 3.3-3 :「.(ピリオド)(0x2e)」の 2Byte の冗長な UTF-8 表現である「%C0%AE」の場合、問題が発現する

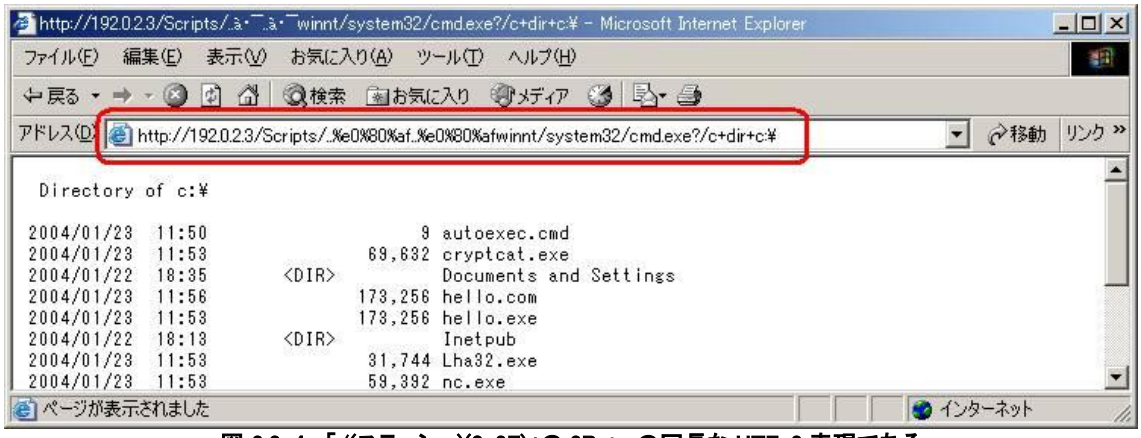

図 3.3-4 :「/(スラッシュ)(0x2F)」の 3Byte の冗長な UTF-8 表現である

「%E0%80%AF」の場合、問題が発現する

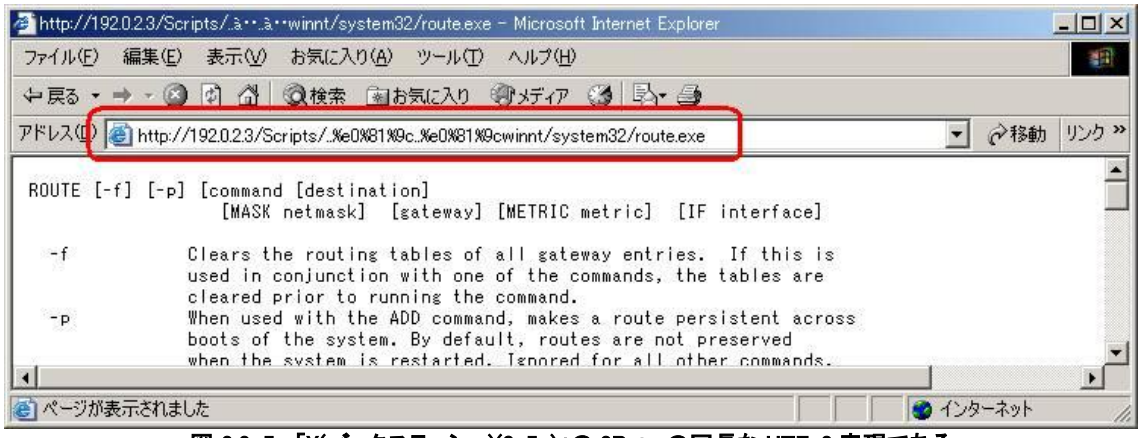

図 3.3-5 :「¥(バックスラッシュ)(0x5c)」の 3Byte の冗長な UTF-8 表現である

「%E0%81%9C」の場合、問題が発現する

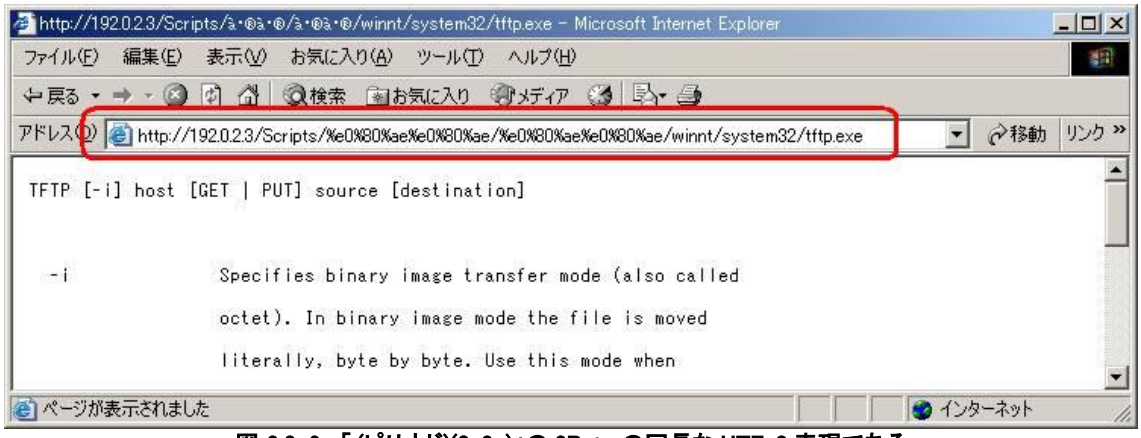

図 3.3-6 :「.(ピリオド)(0x2e)」の 3Byte の冗長な UTF-8 表現である

「%E0%80%AE」の場合、問題が発現する

3.4. 検証内容(UNICODE で分離された文字[U+00A5])

UNICODE 以前の文字コードでは、日本語の「¥(円記号)」と ASCII の「\(バックスラッシュ)」が同一のコードに割り当てられていたが、UNICODE では、別々のコードに割り当てられた。
つまり「\(バックスラッシュ)」は、U+005C に割り当てられ、「¥(円記号)」は U+00A5 に割り当てられた。
MS00-057 で UNICODE を使ったサニタイズ回避テクニックを用いてみたのが以下の検証結果である。

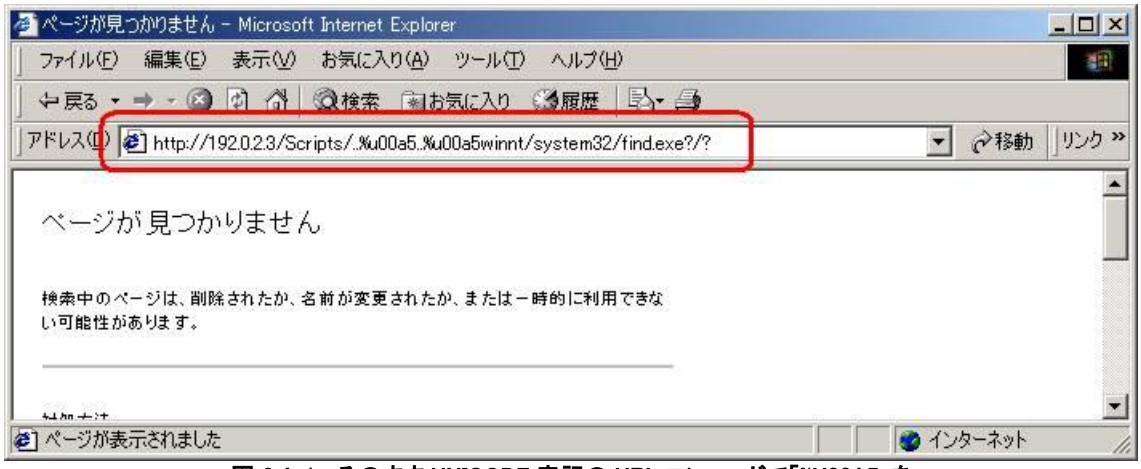

図 3.4-1 : そのまま UNICODE 表記の URL エンコードで「%U00A5」を

指定するだけでは問題は発現しない

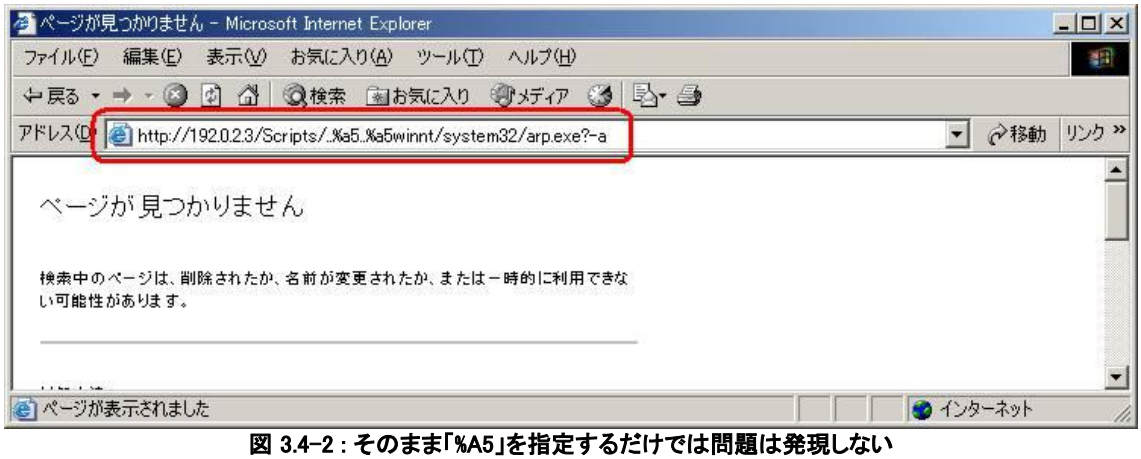

図 3.4-2 : そのまま「%A5」を指定するだけでは問題は発現しない

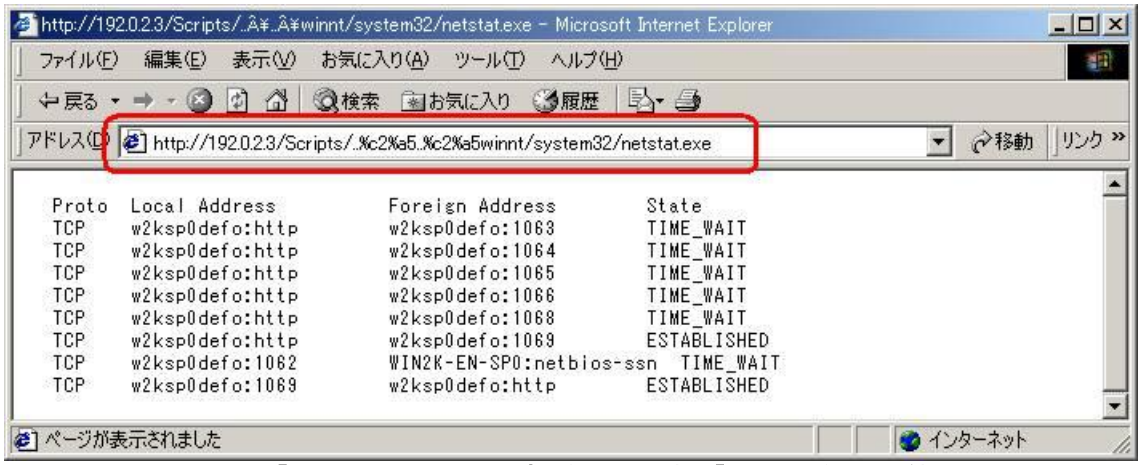

図 3.4-3 : 「¥(円記号[u+00A5])」の適切な UTF-8 表現「%C2%A5」を与えた結果。

MS00-057 では、UNICODE によるサニタイズ回避テクニックが通用する問題であったことが確認された

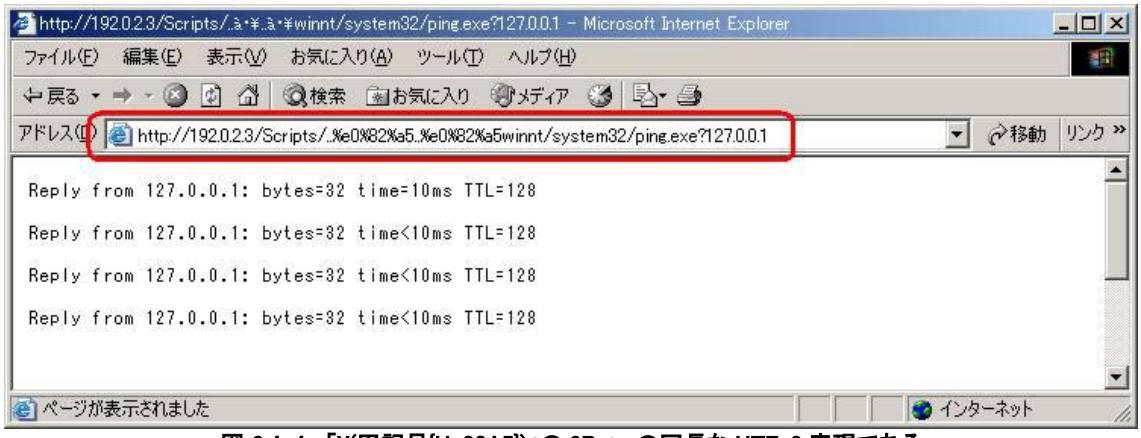

図 3.4-4 :「¥(円記号[U+00A5])」の 3Byte の冗長な UTF-8 表現である

「%E0%82%A5」の場合でも、問題が発現する

図 3.4-5 : 英語版 MS-Windows では「¥(円記号[U+00A5])」では、MS00-057 問題は発現しない

3.5. 検証内容(UNICODE で分離された文字[U+2025])

UNICODE では、U+2025 が、中点二つを示しているが、この文字を ASCII に変換すると、「.(ピリオド)」二個に展開される可能性がある。もし展開されるようであれば、UNICODE を使ったサニタイズ回避テクニックとして利用される危険性がある。
以下は、それについて検証した結果である。

図 3.5-1：そのまま UNICODE 表記の URL エンコードで「%U2025」を指定するだけでは問題は発現しない

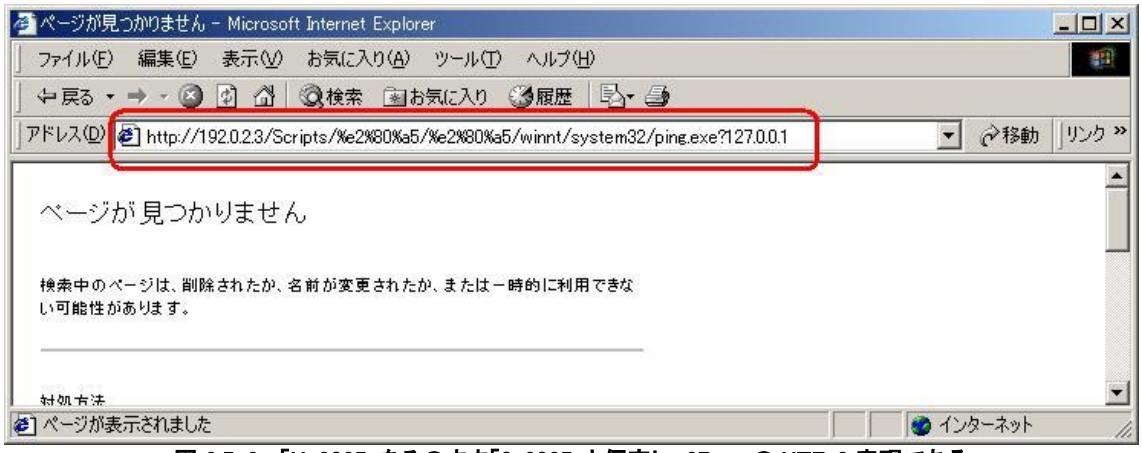

図 3.5-2：「U+2025」をそのまま「0x2025」と仮定し、3Byte の UTF-8 表現である「%E2%80%A5」でアクセスしたが、MS00-057 が再現しない

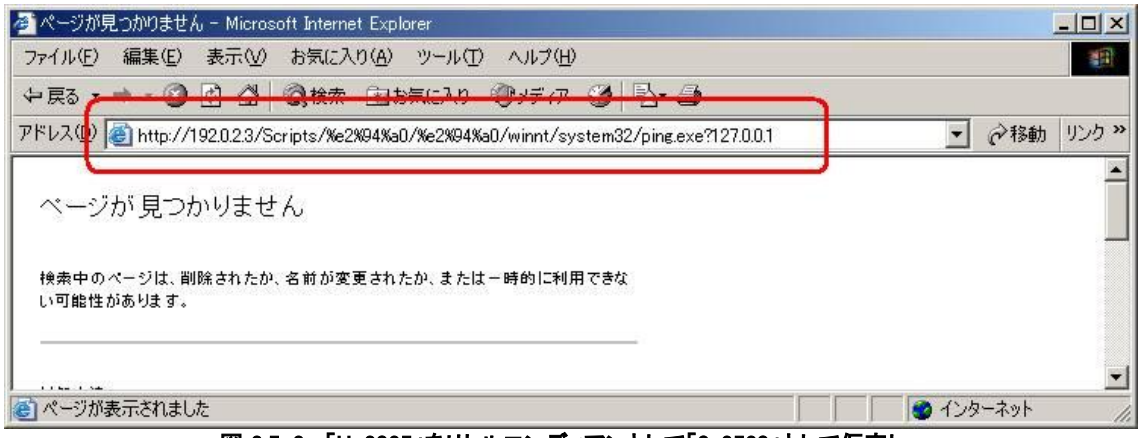

図 3.5-3：「U+2025」をリトルエンディアンとして「0x2520」として仮定し、3Byte の UTF-8 表現である「%E2%94%A0」でアクセスしたが、MS00-057 が再現しない

図 3.5-4：日本語 MS-Windows では「U+2025」は「0x8164」に置換される

U+2025 を、MS-Windows に添付されている「文字コード表」で確認すると、中点二つ(U+2025)は、Shift JIS の中二点(0x8164)と対応している(図 3.5-4)。

そこで、Shift-JIS とは無縁な英語版の MS-Windows+IIS で、U+2025 を試してみたのが以下の検証結果である。

図 3.5-5：英語版 MS-Windows+IIS でも UNICODE 表記の URL エンコードである

「%U2025」を与えても、MS00-057 は発現しない

図 3.5-6 : 「U+2025」をそのまま「0x2025」と仮定し、3Byte の UTF-8 表現である

「%E2%80%A5」でアクセスしたが、MS00-057 が再現しない

図 3.5-7 : 「U+2025」をリトルエンディアンとして「0x2520」として仮定し、

3Byte の UTF-8 表現である「%E2%94%A0」でアクセスしたが、MS00-057 が再現しない

3.6. まとめ

第三章の結果をまとめると以下のようになる

- 「/(スラッシュ)」、「\(バックスラッシュ)」、「.(ピリオド)」の 2Byte の冗長な UTF-8 表現で、MS00-057 は発現する
- 「/(スラッシュ)」、「\(バックスラッシュ)」、「.(ピリオド)」の 3Byte の冗長な UTF-8 表現で、MS00-057 は発現する
- UNICODE によって、分離された「¥(円記号)」を使った表現で、MS00-057 は発現する
  (日本語版の MS-Windows では発現したが、英語版の MS-Windows では発現しなかった)
- UNICODE によって、分離された「¥(円記号)」を 3Byte の UTF-8 表現で、MS00-057 は発現する
  (日本語版の MS-Windows では発現したが、英語版の MS-Windows では発現しなかった)
- UNICODE「U2025」では、発現しなかった

表にすると以下のようになった。

検証文字	検証対象の MS-Windows	
	日本語版	英語版
「/(スラッシュ)」の 2Byte の冗長な UTF-8 表現	発現	―
「/(スラッシュ)」の 3Byte の冗長な UTF-8 表現	発現	―
「\(バックスラッシュ)」の 2Byte の冗長な UTF-8 表現	発現	―
「\(バックスラッシュ)」の 3Byte の冗長な UTF-8 表現	発現	―
「.(ピリオド)」の 2Byte の冗長な UTF-8 表現	発現	―
「.(ピリオド)」の 3Byte の冗長な UTF-8 表現	発現	―
「¥(円記号)」の UTF-8 表現	発現	発現せず
「¥(円記号)」の 3Byte の UTF-8 表現	発現	発現せず
U+2025 を使ったサニタイジング回避テクニック	発現せず	発現せず

表 3.6-1：検証結果

4. 攻撃可能バイトと日本語

4.1. 攻撃可能バイトと日本語

冗長な UTF-8 をそのまま IDS/IPS/WAF でフィルタすることで、日本語環境の Web ページで障害が発生する可能性がある。

「.(ピリオド)」「/(スラッシュ)」「\(バックスラッシュ)」「¥(円記号)」らの、冗長な UTF-8 表現は、以下の表のように、別の日本語コードの文字と一致する場合がある。
よって、冗長な UTF-8 表現をそのままフィルタしてしまうと、例えば EUC-JP で展開している Web サイトでは「政」の字が入力できないなどの障害が発生する可能性がある。

第三章で登場した検証対象は、以下である。

MS00-057 を発現するのにキーとなった文字コード	HEX
「/(スラッシュ)」の ASCII コード	0x2F
「/(スラッシュ)」の 2Byte の冗長な UTF-8 表現	0xC0 0xAF
「/(スラッシュ)」の 3Byte の冗長な UTF-8 表現	0xE0 0x80 0xAF
「\(バックスラッシュ)」の ASCII コード	0x5C
「\(バックスラッシュ)」の 2Byte の冗長な UTF-8 表現	0xC1 0x9c
「\(バックスラッシュ)」の 3Byte の冗長な UTF-8 表現	0xE0 0x81 0x9C
「.(ピリオド)」の ASCII コード	0x2E
「.(ピリオド)」の 2Byte の冗長な UTF-8 表現	0xC0 0xAE
「.(ピリオド)」の 3Byte の冗長な UTF-8 表現	0xE0 0x80 0xAE
「¥(円記号)」の UTF-8 表現	0xC2 0xA5
「¥(円記号)」の 3Byte の UTF-8 表現	0xE0 0x82 0xA5

表 4.1-1：検証対象とその HEX コード

これらの組み合わせの中で、日本語として認識する組み合わせは以下の表となる。

	HEX	文字
1	0xC1	ﾁ
2	0xE0 0x81	焉
3	0x81 0x9C	●
4	0xAF	ﾂ
5	0xE0 0x80	烙
6	0xAE	ｮ
7	0xC2	ﾂ
8	0xA5	・
9	0xE0 0x82	烽
10	0x82 0xA5	ぇ
11	0x9C 0x5C	彌
12	0x9C 0xC1	愧
13	0x9C 0xE0	憫
14	0x9C 0xC0	慨
15	0x9C 0xA5	悋
16	0x9C 0xC2	慊

表 4.1-2 : Shift JIS の場合

	HEX	文字
1	0xC0 0xAF	政
2	0xC0 0xAE	成
3	0xC2 0xA5	促
4	0xA5 0xC1	チ
5	0xA5 0xE0	ム
6	0xA5 0xC0	ダ
7	0xA5 0xA5	ウ
8	0xA5 0xC2	ヂ

図 4.1-3 : EUC JP の場合

5. ビット値について(表)

文字コード	種類	ビット列	HEX
0x5C	ASCII	01011100	0x5C
	2Byte UTF-8	11000001 10011100	0xC1 0x9C
	3Byte UTF-8	11100000 10000001 10011100	0xE0 0x81 0x9C
0x2F	ASCII	00101111	0x2F
	2Byte UTF-8	11000000 10101111	0xC0 0xAF
	3Byte UTF-8	11100000 10000000 10101111	0xE0 0x80 0xAF
0x2E	ASCII	00101110	0x2E
	2Byte UTF-8	11000000 10101110	0xC0 0xAE
	3Byte UTF-8	11100000 10000000 10101110	0xE0 0x80 0xAE
0xA5	-	10100101	0xA5
	2Byte UTF-8	11000010 10100101	0xC2 0xA5
	3Byte UTF-8	11100000 10000010 10100101	0xE0 0x82 0xA5
0x2025	UTF-16(BE)	00100000 00100101	0x20 0x25
	3Byte UTF-8	11100010 10000000 10100101	0xE2 0x80 0xA5
0x2520	UTF-16(LE)	00100101 00100000	0x25 0x20
	3Byte UTF-8	11100010 10010100 10100000	0xE2 0x94 0xA0

表 5-1:ビット値

6. 検証作業者

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター
　佐名木 智貴

7. 参考

① Apache-Tomcat と冗長な UTF-8 表現(CVE-2008-2938 検証レポート) ver1.0
http://www.icto.jp/security_report/pdf/sr20080110.pdf
② 「正規化エラーによる、ファイルへの誤ったアクセス権の適用」の脆弱性に対する対策
(MS00-057)
http://www.microsoft.com/japan/technet/security/bulletin/ms00-057.mspx
③ Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability
http://www.securityfocus.com/bid/1806
④ Unicode とサニタイジング回避テクニック ver1.5
http://rocketeer.dip.jp/secProg/unicodebug006.pdf
⑤ RFC2279 - UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc2279.txt
⑥ ユニコードとセキュリティ
http://openmya.hacker.jp/hasegawa/public/20041030/unicode-and-security.pdf

8. 履歴

- 2008 年 10 月 27 日 : ver1.0 最初の公開
- 2009 年 05 月 26 日 : ver1.1 Web サイト移転に伴う最新版公開 URL の変更

9. 最新版の公開 URL

http://www.ntt.com/icto/security/data/soc.html#security_report

10. 本レポートに関する問合せ先

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター

e-mail: scan@ntt.com

以 上