Practical Win32 and UNICODE exploitation

Lessons learned when the Cisco™ guys went to Windows™ land
Agenda

- Vulnerabilities in wide char environments
  - Stack based buffer overflows
  - Format strings
- Return address selection
  - Unicode addressable
  - SEH return
  - A generic return address solution
- Shellcode in UNICODE
  - Simple stack run
  - Venetian shell code
- Annual Phenoelit 0day
A normal overflow *yawn*

- User data overflows stack saved registers
- Frame Pointer (FP) is overwritten and ends up in EBP
- Return address pop’d from stack and ends up in EIP
- The rest is history

```
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
```

```
Oh, Dr. Watson, what a pleasure to meet you
```

```
POP EBP
EBP = 0x41414141
EIP = 0x41414141
```

```
Arg 0
Arg 1
```
A wide char overflow

- User data gets transformed into wide char
- Wide char data overwrites saved register data
- FP gets corrupted
- RET gets corrupted
- What now?

```
POP EBP
EBP = 0x00410041
RET
EIP = 0x00410041
```
Overflows compared

- Normal overflow overwrites FP and return address with user data
  - Limited modification of user data takes place
  - All 4 bytes of FP and return address can be influenced
  - Exception when executing 0x41414141

- Wide char or UNICODE overflows overwrite the same data (FP, RET)
  - User data is modified to at least 50%
  - Only 2 of 4 bytes of FP and RET can be influenced
  - Exception when executing 0x00410041
Most people claim that it’s byte (0x41) to byte with leading zero (0x0041)... wrong.

On Win32, the transformation is done by

```c
int MultiByteToWideChar(
    UINT CodePage, // PAGE!!!
    DWORD dwFlags,
    LPCSTR lpMultiByteStr, // source
    int cbMultiByte,
    LPWSTR lpWideCharStr, // destination
    int cchWideChar
);
```
<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>..</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>01</td>
<td>0100</td>
<td>..</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>02</td>
<td>0200</td>
<td>..</td>
<td>0200</td>
<td>0200</td>
</tr>
<tr>
<td>03</td>
<td>0300</td>
<td>..</td>
<td>0300</td>
<td>0300</td>
</tr>
<tr>
<td>04</td>
<td>0400</td>
<td>..</td>
<td>0400</td>
<td>0400</td>
</tr>
<tr>
<td>05</td>
<td>0500</td>
<td>..</td>
<td>0500</td>
<td>0500</td>
</tr>
<tr>
<td>06</td>
<td>0600</td>
<td>..</td>
<td>0600</td>
<td>0600</td>
</tr>
<tr>
<td>07</td>
<td>0700</td>
<td>..</td>
<td>0700</td>
<td>0700</td>
</tr>
<tr>
<td>08</td>
<td>0800</td>
<td>..</td>
<td>0800</td>
<td>0800</td>
</tr>
<tr>
<td>09</td>
<td>0900</td>
<td>..</td>
<td>0900</td>
<td>0900</td>
</tr>
<tr>
<td>0A</td>
<td>0A00</td>
<td>..</td>
<td>0A00</td>
<td>0A00</td>
</tr>
<tr>
<td>0B</td>
<td>0B00</td>
<td>..</td>
<td>0B00</td>
<td>0B00</td>
</tr>
<tr>
<td>0C</td>
<td>0C00</td>
<td>..</td>
<td>0C00</td>
<td>0C00</td>
</tr>
<tr>
<td>0D</td>
<td>0D00</td>
<td>..</td>
<td>0D00</td>
<td>0D00</td>
</tr>
<tr>
<td>0E</td>
<td>0E00</td>
<td>..</td>
<td>0E00</td>
<td>0E00</td>
</tr>
<tr>
<td>0F</td>
<td>0F00</td>
<td>..</td>
<td>0F00</td>
<td>0F00</td>
</tr>
</tbody>
</table>
## Windows transformation tables

<table>
<thead>
<tr>
<th></th>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:</td>
<td>1000 ..</td>
<td>1000 ..</td>
<td>1000 ..</td>
<td>1000 ..</td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>1100 ..</td>
<td>1100 ..</td>
<td>1100 ..</td>
<td>1100 ..</td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>1200 ..</td>
<td>1200 ..</td>
<td>1200 ..</td>
<td>1200 ..</td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>1300 ..</td>
<td>1300 ..</td>
<td>1300 ..</td>
<td>1300 ..</td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>1400 ..</td>
<td>1400 ..</td>
<td>1400 ..</td>
<td>1400 ..</td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>1500 ..</td>
<td>1500 ..</td>
<td>1500 ..</td>
<td>1500 ..</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>1600 ..</td>
<td>1600 ..</td>
<td>1600 ..</td>
<td>1600 ..</td>
<td></td>
</tr>
<tr>
<td>17:</td>
<td>1700 ..</td>
<td>1700 ..</td>
<td>1700 ..</td>
<td>1700 ..</td>
<td></td>
</tr>
<tr>
<td>18:</td>
<td>1800 ..</td>
<td>1800 ..</td>
<td>1800 ..</td>
<td>1800 ..</td>
<td></td>
</tr>
<tr>
<td>19:</td>
<td>1900 ..</td>
<td>1900 ..</td>
<td>1900 ..</td>
<td>1900 ..</td>
<td></td>
</tr>
<tr>
<td>1A:</td>
<td>1A00 ..</td>
<td>1A00 ..</td>
<td>1A00 ..</td>
<td>1A00 ..</td>
<td></td>
</tr>
<tr>
<td>1B:</td>
<td>1B00 ..</td>
<td>1B00 ..</td>
<td>1B00 ..</td>
<td>1B00 ..</td>
<td></td>
</tr>
<tr>
<td>1C:</td>
<td>1C00 ..</td>
<td>1C00 ..</td>
<td>1C00 ..</td>
<td>1C00 ..</td>
<td></td>
</tr>
<tr>
<td>1D:</td>
<td>1D00 ..</td>
<td>1D00 ..</td>
<td>1D00 ..</td>
<td>1D00 ..</td>
<td></td>
</tr>
<tr>
<td>1E:</td>
<td>1E00 ..</td>
<td>1E00 ..</td>
<td>1E00 ..</td>
<td>1E00 ..</td>
<td></td>
</tr>
<tr>
<td>1F:</td>
<td>1F00 ..</td>
<td>1F00 ..</td>
<td>1F00 ..</td>
<td>1F00 ..</td>
<td></td>
</tr>
</tbody>
</table>
# Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>21:</td>
<td>2100</td>
<td>!</td>
<td>2100</td>
<td>!</td>
</tr>
<tr>
<td>22:</td>
<td>2200</td>
<td>&quot;</td>
<td>2200</td>
<td>&quot;</td>
</tr>
<tr>
<td>23:</td>
<td>2300</td>
<td>#</td>
<td>2300</td>
<td>#</td>
</tr>
<tr>
<td>24:</td>
<td>2400</td>
<td>$</td>
<td>2400</td>
<td>$</td>
</tr>
<tr>
<td>25:</td>
<td>2500</td>
<td>%</td>
<td>2500</td>
<td>%</td>
</tr>
<tr>
<td>26:</td>
<td>2600</td>
<td>&amp;</td>
<td>2600</td>
<td>&amp;</td>
</tr>
<tr>
<td>27:</td>
<td>2700</td>
<td>'</td>
<td>2700</td>
<td>'</td>
</tr>
<tr>
<td>28:</td>
<td>2800</td>
<td>(</td>
<td>2800</td>
<td>(</td>
</tr>
<tr>
<td>29:</td>
<td>2900</td>
<td>)</td>
<td>2900</td>
<td>)</td>
</tr>
<tr>
<td>2A:</td>
<td>2A00</td>
<td>*</td>
<td>2A00</td>
<td>*</td>
</tr>
<tr>
<td>2B:</td>
<td>2B00</td>
<td>+</td>
<td>2B00</td>
<td>+</td>
</tr>
<tr>
<td>2C:</td>
<td>2C00</td>
<td>,</td>
<td>2C00</td>
<td>,</td>
</tr>
<tr>
<td>2D:</td>
<td>2D00</td>
<td>-</td>
<td>2D00</td>
<td>-</td>
</tr>
<tr>
<td>2E:</td>
<td>2E00</td>
<td>.</td>
<td>2E00</td>
<td>.</td>
</tr>
<tr>
<td>2F:</td>
<td>2F00</td>
<td>/</td>
<td>2F00</td>
<td>/</td>
</tr>
</tbody>
</table>
### Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>30:</td>
<td>3000</td>
<td>0.</td>
<td>3000</td>
<td>0.</td>
</tr>
<tr>
<td>31:</td>
<td>3100</td>
<td>1.</td>
<td>3100</td>
<td>1.</td>
</tr>
<tr>
<td>32:</td>
<td>3200</td>
<td>2.</td>
<td>3200</td>
<td>2.</td>
</tr>
<tr>
<td>33:</td>
<td>3300</td>
<td>3.</td>
<td>3300</td>
<td>3.</td>
</tr>
<tr>
<td>34:</td>
<td>3400</td>
<td>4.</td>
<td>3400</td>
<td>4.</td>
</tr>
<tr>
<td>35:</td>
<td>3500</td>
<td>5.</td>
<td>3500</td>
<td>5.</td>
</tr>
<tr>
<td>37:</td>
<td>3700</td>
<td>7.</td>
<td>3700</td>
<td>7.</td>
</tr>
<tr>
<td>38:</td>
<td>3800</td>
<td>8.</td>
<td>3800</td>
<td>8.</td>
</tr>
<tr>
<td>3A:</td>
<td>3A00</td>
<td>:.</td>
<td>3A00</td>
<td>:.</td>
</tr>
<tr>
<td>3B:</td>
<td>3B00</td>
<td>;.</td>
<td>3B00</td>
<td>;.</td>
</tr>
<tr>
<td>3C:</td>
<td>3C00</td>
<td>&lt;.</td>
<td>3C00</td>
<td>&lt;.</td>
</tr>
<tr>
<td>3D:</td>
<td>3D00</td>
<td>=.</td>
<td>3D00</td>
<td>=.</td>
</tr>
<tr>
<td>3E:</td>
<td>3E00</td>
<td>&gt;.</td>
<td>3E00</td>
<td>&gt;.</td>
</tr>
<tr>
<td>3F:</td>
<td>3F00</td>
<td>?.</td>
<td>3F00</td>
<td>?.</td>
</tr>
</tbody>
</table>
### Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>40:</td>
<td>4000 @.</td>
<td>4000 @.</td>
<td>4000 @.</td>
<td>4000 @.</td>
</tr>
<tr>
<td>41:</td>
<td>4100 A.</td>
<td>4100 A.</td>
<td>4100 A.</td>
<td>4100 A.</td>
</tr>
<tr>
<td>42:</td>
<td>4200 B.</td>
<td>4200 B.</td>
<td>4200 B.</td>
<td>4200 B.</td>
</tr>
<tr>
<td>43:</td>
<td>4300 C.</td>
<td>4300 C.</td>
<td>4300 C.</td>
<td>4300 C.</td>
</tr>
<tr>
<td>44:</td>
<td>4400 D.</td>
<td>4400 D.</td>
<td>4400 D.</td>
<td>4400 D.</td>
</tr>
<tr>
<td>45:</td>
<td>4500 E.</td>
<td>4500 E.</td>
<td>4500 E.</td>
<td>4500 E.</td>
</tr>
<tr>
<td>46:</td>
<td>4600 F.</td>
<td>4600 F.</td>
<td>4600 F.</td>
<td>4600 F.</td>
</tr>
<tr>
<td>47:</td>
<td>4700 G.</td>
<td>4700 G.</td>
<td>4700 G.</td>
<td>4700 G.</td>
</tr>
<tr>
<td>48:</td>
<td>4800 H.</td>
<td>4800 H.</td>
<td>4800 H.</td>
<td>4800 H.</td>
</tr>
<tr>
<td>4A:</td>
<td>4A00 J.</td>
<td>4A00 J.</td>
<td>4A00 J.</td>
<td>4A00 J.</td>
</tr>
<tr>
<td>4B:</td>
<td>4B00 K.</td>
<td>4B00 K.</td>
<td>4B00 K.</td>
<td>4B00 K.</td>
</tr>
<tr>
<td>4C:</td>
<td>4C00 L.</td>
<td>4C00 L.</td>
<td>4C00 L.</td>
<td>4C00 L.</td>
</tr>
<tr>
<td>4D:</td>
<td>4D00 M.</td>
<td>4D00 M.</td>
<td>4D00 M.</td>
<td>4D00 M.</td>
</tr>
<tr>
<td>4E:</td>
<td>4E00 N.</td>
<td>4E00 N.</td>
<td>4E00 N.</td>
<td>4E00 N.</td>
</tr>
<tr>
<td>4F:</td>
<td>4F00 O.</td>
<td>4F00 O.</td>
<td>4F00 O.</td>
<td>4F00 O.</td>
</tr>
</tbody>
</table>
### Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>50:</td>
<td>5000 P.</td>
<td>5000 P.</td>
<td>5000 P.</td>
<td>5000 P.</td>
</tr>
<tr>
<td>51:</td>
<td>5100 Q.</td>
<td>5100 Q.</td>
<td>5100 Q.</td>
<td>5100 Q.</td>
</tr>
<tr>
<td>52:</td>
<td>5200 R.</td>
<td>5200 R.</td>
<td>5200 R.</td>
<td>5200 R.</td>
</tr>
<tr>
<td>53:</td>
<td>5300 S.</td>
<td>5300 S.</td>
<td>5300 S.</td>
<td>5300 S.</td>
</tr>
<tr>
<td>54:</td>
<td>5400 T.</td>
<td>5400 T.</td>
<td>5400 T.</td>
<td>5400 T.</td>
</tr>
<tr>
<td>56:</td>
<td>5600 V.</td>
<td>5600 V.</td>
<td>5600 V.</td>
<td>5600 V.</td>
</tr>
<tr>
<td>57:</td>
<td>5700 W.</td>
<td>5700 W.</td>
<td>5700 W.</td>
<td>5700 W.</td>
</tr>
<tr>
<td>58:</td>
<td>5800 X.</td>
<td>5800 X.</td>
<td>5800 X.</td>
<td>5800 X.</td>
</tr>
<tr>
<td>59:</td>
<td>5900 Y.</td>
<td>5900 Y.</td>
<td>5900 Y.</td>
<td>5900 Y.</td>
</tr>
<tr>
<td>5A:</td>
<td>5A00 Z.</td>
<td>5A00 Z.</td>
<td>5A00 Z.</td>
<td>5A00 Z.</td>
</tr>
<tr>
<td>5B:</td>
<td>5B00 [ .</td>
<td>5B00 [ .</td>
<td>5B00 [ .</td>
<td>5B00 [ .</td>
</tr>
<tr>
<td>5C:</td>
<td>5C00 \ .</td>
<td>5C00 \ .</td>
<td>5C00 \ .</td>
<td>5C00 \ .</td>
</tr>
<tr>
<td>5D:</td>
<td>5D00 ] .</td>
<td>5D00 ] .</td>
<td>5D00 ] .</td>
<td>5D00 ] .</td>
</tr>
<tr>
<td>5E:</td>
<td>5E00 ^ .</td>
<td>5E00 ^ .</td>
<td>5E00 ^ .</td>
<td>5E00 ^ .</td>
</tr>
<tr>
<td>5F:</td>
<td>5F00 _ .</td>
<td>5F00 _ .</td>
<td>5F00 _ .</td>
<td>5F00 _ .</td>
</tr>
<tr>
<td>ASCII</td>
<td>ANSI</td>
<td>OEM</td>
<td>UTF7</td>
<td>UTF8</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>60</td>
<td>6000</td>
<td>6000</td>
<td>6000</td>
<td>6000</td>
</tr>
<tr>
<td>61</td>
<td>6100</td>
<td>6100</td>
<td>6100</td>
<td>6100</td>
</tr>
<tr>
<td>62</td>
<td>6200</td>
<td>6200</td>
<td>6200</td>
<td>6200</td>
</tr>
<tr>
<td>63</td>
<td>6300</td>
<td>6300</td>
<td>6300</td>
<td>6300</td>
</tr>
<tr>
<td>64</td>
<td>6400</td>
<td>6400</td>
<td>6400</td>
<td>6400</td>
</tr>
<tr>
<td>65</td>
<td>6500</td>
<td>6500</td>
<td>6500</td>
<td>6500</td>
</tr>
<tr>
<td>66</td>
<td>6600</td>
<td>6600</td>
<td>6600</td>
<td>6600</td>
</tr>
<tr>
<td>67</td>
<td>6700</td>
<td>6700</td>
<td>6700</td>
<td>6700</td>
</tr>
<tr>
<td>68</td>
<td>6800</td>
<td>6800</td>
<td>6800</td>
<td>6800</td>
</tr>
<tr>
<td>69</td>
<td>6900</td>
<td>6900</td>
<td>6900</td>
<td>6900</td>
</tr>
<tr>
<td>6A</td>
<td>6A00</td>
<td>6A00</td>
<td>6A00</td>
<td>6A00</td>
</tr>
<tr>
<td>6B</td>
<td>6B00</td>
<td>6B00</td>
<td>6B00</td>
<td>6B00</td>
</tr>
<tr>
<td>6C</td>
<td>6C00</td>
<td>6C00</td>
<td>6C00</td>
<td>6C00</td>
</tr>
<tr>
<td>6D</td>
<td>6D00</td>
<td>6D00</td>
<td>6D00</td>
<td>6D00</td>
</tr>
<tr>
<td>6E</td>
<td>6E00</td>
<td>6E00</td>
<td>6E00</td>
<td>6E00</td>
</tr>
<tr>
<td>6F</td>
<td>6F00</td>
<td>6F00</td>
<td>6F00</td>
<td>6F00</td>
</tr>
</tbody>
</table>
## Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>71:</td>
<td>7100 q.</td>
<td>7100 q.</td>
<td>7100 q.</td>
<td>7100 q.</td>
</tr>
<tr>
<td>72:</td>
<td>7200 r.</td>
<td>7200 r.</td>
<td>7200 r.</td>
<td>7200 r.</td>
</tr>
<tr>
<td>73:</td>
<td>7300 s.</td>
<td>7300 s.</td>
<td>7300 s.</td>
<td>7300 s.</td>
</tr>
<tr>
<td>74:</td>
<td>7400 t.</td>
<td>7400 t.</td>
<td>7400 t.</td>
<td>7400 t.</td>
</tr>
<tr>
<td>75:</td>
<td>7500 u.</td>
<td>7500 u.</td>
<td>7500 u.</td>
<td>7500 u.</td>
</tr>
<tr>
<td>76:</td>
<td>7600 v.</td>
<td>7600 v.</td>
<td>7600 v.</td>
<td>7600 v.</td>
</tr>
<tr>
<td>77:</td>
<td>7700 w.</td>
<td>7700 w.</td>
<td>7700 w.</td>
<td>7700 w.</td>
</tr>
<tr>
<td>78:</td>
<td>7800 x.</td>
<td>7800 x.</td>
<td>7800 x.</td>
<td>7800 x.</td>
</tr>
<tr>
<td>79:</td>
<td>7900 y.</td>
<td>7900 y.</td>
<td>7900 y.</td>
<td>7900 y.</td>
</tr>
<tr>
<td>7A:</td>
<td>7A00 z.</td>
<td>7A00 z.</td>
<td>7A00 z.</td>
<td>7A00 z.</td>
</tr>
<tr>
<td>7B:</td>
<td>7B00 {</td>
<td>7B00 {</td>
<td>7B00 {</td>
<td>7B00 {</td>
</tr>
<tr>
<td>7C:</td>
<td>7C00</td>
<td>7C00</td>
<td>7C00</td>
<td>7C00</td>
</tr>
<tr>
<td>7D:</td>
<td>7D00 }</td>
<td>7D00 }</td>
<td>7D00 }</td>
<td>7D00 }</td>
</tr>
<tr>
<td>7E:</td>
<td>7E00 ~.</td>
<td>7E00 ~.</td>
<td>7E00 ~.</td>
<td>7E00 ~.</td>
</tr>
<tr>
<td>7F:</td>
<td>7F00 ..</td>
<td>7F00 ..</td>
<td>7F00 ..</td>
<td>7F00 ..</td>
</tr>
</tbody>
</table>
### Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>AC20</td>
<td>C700</td>
<td>80FF</td>
<td>0000</td>
</tr>
<tr>
<td>81</td>
<td>8100</td>
<td>FC00</td>
<td>81FF</td>
<td>0000</td>
</tr>
<tr>
<td>82</td>
<td>1A20</td>
<td>E900</td>
<td>82FF</td>
<td>0000</td>
</tr>
<tr>
<td>83</td>
<td>9201</td>
<td>E200</td>
<td>83FF</td>
<td>0000</td>
</tr>
<tr>
<td>84</td>
<td>1E20</td>
<td>E400</td>
<td>84FF</td>
<td>0000</td>
</tr>
<tr>
<td>85</td>
<td>2620 &amp;</td>
<td>E000</td>
<td>85FF</td>
<td>0000</td>
</tr>
<tr>
<td>86</td>
<td>2020</td>
<td>E500</td>
<td>86FF</td>
<td>0000</td>
</tr>
<tr>
<td>87</td>
<td>2120 !</td>
<td>E700</td>
<td>87FF</td>
<td>0000</td>
</tr>
<tr>
<td>88</td>
<td>C602</td>
<td>EA00</td>
<td>88FF</td>
<td>0000</td>
</tr>
<tr>
<td>89</td>
<td>3020 0</td>
<td>EB00</td>
<td>89FF</td>
<td>0000</td>
</tr>
<tr>
<td>8A</td>
<td>6001 `</td>
<td>E800</td>
<td>8AFF</td>
<td>0000</td>
</tr>
<tr>
<td>8B</td>
<td>3920 9</td>
<td>EF00</td>
<td>8BFF</td>
<td>0000</td>
</tr>
<tr>
<td>8C</td>
<td>5201 R</td>
<td>EE00</td>
<td>8CFF</td>
<td>0000</td>
</tr>
<tr>
<td>8D</td>
<td>8D00</td>
<td>EC00</td>
<td>8DFF</td>
<td>0000</td>
</tr>
<tr>
<td>8E</td>
<td>7D01 }</td>
<td>C400</td>
<td>8EFF</td>
<td>0000</td>
</tr>
<tr>
<td>8F</td>
<td>8F00</td>
<td>C500</td>
<td>8FFF</td>
<td>0000</td>
</tr>
</tbody>
</table>
### Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>90:</td>
<td>9000</td>
<td>C900</td>
<td>90FF</td>
<td>0000</td>
</tr>
<tr>
<td>91:</td>
<td>1820</td>
<td>E600</td>
<td>91FF</td>
<td>0000</td>
</tr>
<tr>
<td>92:</td>
<td>1920</td>
<td>C600</td>
<td>92FF</td>
<td>0000</td>
</tr>
<tr>
<td>93:</td>
<td>1C20</td>
<td>F400</td>
<td>93FF</td>
<td>0000</td>
</tr>
<tr>
<td>94:</td>
<td>1D20</td>
<td>F600</td>
<td>94FF</td>
<td>0000</td>
</tr>
<tr>
<td>95:</td>
<td>2220</td>
<td>F200</td>
<td>95FF</td>
<td>0000</td>
</tr>
<tr>
<td>96:</td>
<td>1320</td>
<td>FB00</td>
<td>96FF</td>
<td>0000</td>
</tr>
<tr>
<td>97:</td>
<td>1420</td>
<td>F900</td>
<td>97FF</td>
<td>0000</td>
</tr>
<tr>
<td>98:</td>
<td>DC02</td>
<td>FF00</td>
<td>98FF</td>
<td>0000</td>
</tr>
<tr>
<td>99:</td>
<td>2221</td>
<td>D600</td>
<td>99FF</td>
<td>0000</td>
</tr>
<tr>
<td>9A:</td>
<td>6101</td>
<td>DC00</td>
<td>9AFF</td>
<td>0000</td>
</tr>
<tr>
<td>9B:</td>
<td>3A20</td>
<td>F800</td>
<td>9BFF</td>
<td>0000</td>
</tr>
<tr>
<td>9C:</td>
<td>5301</td>
<td>A300</td>
<td>9CFF</td>
<td>0000</td>
</tr>
<tr>
<td>9D:</td>
<td>9D00</td>
<td>D800</td>
<td>9DFF</td>
<td>0000</td>
</tr>
<tr>
<td>9E:</td>
<td>7E01</td>
<td>D700</td>
<td>9EFF</td>
<td>0000</td>
</tr>
<tr>
<td>9F:</td>
<td>7801</td>
<td>9201</td>
<td>9FFF</td>
<td>0000</td>
</tr>
</tbody>
</table>
## Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0:</td>
<td>A000</td>
<td>E100</td>
<td>A0FF</td>
<td>0000</td>
</tr>
<tr>
<td>A1:</td>
<td>A100</td>
<td>ED00</td>
<td>A1FF</td>
<td>0000</td>
</tr>
<tr>
<td>A2:</td>
<td>A200</td>
<td>F300</td>
<td>A2FF</td>
<td>0000</td>
</tr>
<tr>
<td>A3:</td>
<td>A300</td>
<td>FA00</td>
<td>A3FF</td>
<td>0000</td>
</tr>
<tr>
<td>A4:</td>
<td>A400</td>
<td>F100</td>
<td>A4FF</td>
<td>0000</td>
</tr>
<tr>
<td>A5:</td>
<td>A500</td>
<td>D100</td>
<td>A5FF</td>
<td>0000</td>
</tr>
<tr>
<td>A6:</td>
<td>A600</td>
<td>AA00</td>
<td>A6FF</td>
<td>0000</td>
</tr>
<tr>
<td>A7:</td>
<td>A700</td>
<td>BA00</td>
<td>A7FF</td>
<td>0000</td>
</tr>
<tr>
<td>A8:</td>
<td>A800</td>
<td>BF00</td>
<td>A8FF</td>
<td>0000</td>
</tr>
<tr>
<td>A9:</td>
<td>A900</td>
<td>AE00</td>
<td>A9FF</td>
<td>0000</td>
</tr>
<tr>
<td>AA:</td>
<td>AA00</td>
<td>AC00</td>
<td>AAff</td>
<td>0000</td>
</tr>
<tr>
<td>AB:</td>
<td>AB00</td>
<td>BD00</td>
<td>ABFF</td>
<td>0000</td>
</tr>
<tr>
<td>AC:</td>
<td>AC00</td>
<td>BC00</td>
<td>ACFF</td>
<td>0000</td>
</tr>
<tr>
<td>AD:</td>
<td>AD00</td>
<td>A100</td>
<td>ADFF</td>
<td>0000</td>
</tr>
<tr>
<td>AE:</td>
<td>AE00</td>
<td>AB00</td>
<td>AEFF</td>
<td>0000</td>
</tr>
<tr>
<td>AF:</td>
<td>AF00</td>
<td>BB00</td>
<td>AFFF</td>
<td>0000</td>
</tr>
</tbody>
</table>
## Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0: B000</td>
<td>9125 .%</td>
<td>B0FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B1: B100</td>
<td>9225 .%</td>
<td>B1FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B2: B200</td>
<td>9325 .%</td>
<td>B2FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B3: B300</td>
<td>0225 .%</td>
<td>B3FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B4: B400</td>
<td>2425 $%</td>
<td>B4FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B5: B500</td>
<td>C100 ..</td>
<td>B5FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B6: B600</td>
<td>C200 ..</td>
<td>B6FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B7: B700</td>
<td>C000 ..</td>
<td>B7FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B8: B800</td>
<td>A900 ..</td>
<td>B8FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>B9: B900</td>
<td>6325 c%</td>
<td>B9FF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>BA: BA00</td>
<td>5125 Q%</td>
<td>BAFF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>BB: BB00</td>
<td>5725 W%</td>
<td>BBFF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>BC: BC00</td>
<td>5D25 ]%</td>
<td>BCFF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>BD: BD00</td>
<td>A200 ..</td>
<td>BDFF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>BE: BE00</td>
<td>A500 ..</td>
<td>BEFF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>BF: BF00</td>
<td>1025 .%</td>
<td>BFFF</td>
<td>0000 ..</td>
<td></td>
</tr>
<tr>
<td>ASCII</td>
<td>ANSI</td>
<td>OEM</td>
<td>UTF7</td>
<td>UTF8</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>C0</td>
<td>C000</td>
<td>1425</td>
<td>.%</td>
<td>C0FF</td>
</tr>
<tr>
<td>C1</td>
<td>C100</td>
<td>3425</td>
<td>4%</td>
<td>C1FF</td>
</tr>
<tr>
<td>C2</td>
<td>C200</td>
<td>2C25</td>
<td>,%</td>
<td>C2FF</td>
</tr>
<tr>
<td>C3</td>
<td>C300</td>
<td>1C25</td>
<td>.%</td>
<td>C3FF</td>
</tr>
<tr>
<td>C4</td>
<td>C400</td>
<td>0025</td>
<td>.%</td>
<td>C4FF</td>
</tr>
<tr>
<td>C5</td>
<td>C500</td>
<td>3C25</td>
<td>&lt;%</td>
<td>C5FF</td>
</tr>
<tr>
<td>C6</td>
<td>C600</td>
<td>E300</td>
<td></td>
<td>C6FF</td>
</tr>
<tr>
<td>C7</td>
<td>C700</td>
<td>C300</td>
<td></td>
<td>C7FF</td>
</tr>
<tr>
<td>C8</td>
<td>C800</td>
<td>5A25</td>
<td>Z%</td>
<td>C8FF</td>
</tr>
<tr>
<td>C9</td>
<td>C900</td>
<td>5425</td>
<td>T%</td>
<td>C9FF</td>
</tr>
<tr>
<td>CA</td>
<td>CA00</td>
<td>6925</td>
<td>i%</td>
<td>CAFF</td>
</tr>
<tr>
<td>CB</td>
<td>CB00</td>
<td>6625</td>
<td>f%</td>
<td>CBFF</td>
</tr>
<tr>
<td>CC</td>
<td>CC00</td>
<td>6025</td>
<td>`%</td>
<td>CCFF</td>
</tr>
<tr>
<td>CD</td>
<td>CD00</td>
<td>5025</td>
<td>P%</td>
<td>CDFF</td>
</tr>
<tr>
<td>CE</td>
<td>CE00</td>
<td>6C25</td>
<td>1%</td>
<td>CEFF</td>
</tr>
<tr>
<td>CF</td>
<td>CF00</td>
<td>A400</td>
<td></td>
<td>CFFF</td>
</tr>
</tbody>
</table>
### Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0: D000</td>
<td>F000</td>
<td>D0FF</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>D1: D100</td>
<td>D000</td>
<td>D1FF</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>D2: D200</td>
<td>CA00</td>
<td>D2FF</td>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>D3: D300</td>
<td>CB00</td>
<td>D3FF</td>
<td>1300</td>
<td></td>
</tr>
<tr>
<td>D4: D400</td>
<td>C800</td>
<td>D4FF</td>
<td>1400</td>
<td></td>
</tr>
<tr>
<td>D5: D500</td>
<td>3101</td>
<td>D5FF</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>D6: D600</td>
<td>CD00</td>
<td>D6FF</td>
<td>1600</td>
<td></td>
</tr>
<tr>
<td>D7: D700</td>
<td>CE00</td>
<td>D7FF</td>
<td>1700</td>
<td></td>
</tr>
<tr>
<td>D8: D800</td>
<td>CF00</td>
<td>D8FF</td>
<td>1800</td>
<td></td>
</tr>
<tr>
<td>D9: D900</td>
<td>1825</td>
<td>D9FF</td>
<td>1900</td>
<td></td>
</tr>
<tr>
<td>DA: DA00</td>
<td>0C25</td>
<td>DAFF</td>
<td>1A00</td>
<td></td>
</tr>
<tr>
<td>DB: DB00</td>
<td>8825</td>
<td>DBFF</td>
<td>1B00</td>
<td></td>
</tr>
<tr>
<td>DC: DC00</td>
<td>8425</td>
<td>DCFF</td>
<td>1C00</td>
<td></td>
</tr>
<tr>
<td>DD: DD00</td>
<td>A600</td>
<td>DDFF</td>
<td>1D00</td>
<td></td>
</tr>
<tr>
<td>DE: DE00</td>
<td>CC00</td>
<td>DEFF</td>
<td>1E00</td>
<td></td>
</tr>
<tr>
<td>DF: DF00</td>
<td>8025</td>
<td>DFFF</td>
<td>1F00</td>
<td></td>
</tr>
</tbody>
</table>
Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0:</td>
<td>E000</td>
<td>D300</td>
<td>E0FF</td>
<td>0000</td>
</tr>
<tr>
<td>E1:</td>
<td>E100</td>
<td>DF00</td>
<td>E1FF</td>
<td>0100</td>
</tr>
<tr>
<td>E2:</td>
<td>E200</td>
<td>D400</td>
<td>E2FF</td>
<td>0200</td>
</tr>
<tr>
<td>E3:</td>
<td>E300</td>
<td>D200</td>
<td>E3FF</td>
<td>0300</td>
</tr>
<tr>
<td>E4:</td>
<td>E400</td>
<td>F500</td>
<td>E4FF</td>
<td>0400</td>
</tr>
<tr>
<td>E5:</td>
<td>E500</td>
<td>D500</td>
<td>E5FF</td>
<td>0500</td>
</tr>
<tr>
<td>E6:</td>
<td>E600</td>
<td>B500</td>
<td>E6FF</td>
<td>0600</td>
</tr>
<tr>
<td>E7:</td>
<td>E700</td>
<td>FE00</td>
<td>E7FF</td>
<td>0700</td>
</tr>
<tr>
<td>E8:</td>
<td>E800</td>
<td>DE00</td>
<td>E8FF</td>
<td>0800</td>
</tr>
<tr>
<td>E9:</td>
<td>E900</td>
<td>DA00</td>
<td>E9FF</td>
<td>0900</td>
</tr>
<tr>
<td>EA:</td>
<td>EA00</td>
<td>DB00</td>
<td>EAFF</td>
<td>0A00</td>
</tr>
<tr>
<td>EB:</td>
<td>EB00</td>
<td>D900</td>
<td>EBFF</td>
<td>0B00</td>
</tr>
<tr>
<td>EC:</td>
<td>EC00</td>
<td>FD00</td>
<td>ECFF</td>
<td>0C00</td>
</tr>
<tr>
<td>ED:</td>
<td>ED00</td>
<td>DD00</td>
<td>EDFF</td>
<td>0D00</td>
</tr>
<tr>
<td>EE:</td>
<td>EE00</td>
<td>AF00</td>
<td>EEFF</td>
<td>0E00</td>
</tr>
<tr>
<td>EF:</td>
<td>EF00</td>
<td>B400</td>
<td>EFFF</td>
<td>0F00</td>
</tr>
</tbody>
</table>
## Windows transformation tables

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ANSI</th>
<th>OEM</th>
<th>UTF7</th>
<th>UTF8</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0:</td>
<td>:AD00</td>
<td>:F0FF</td>
<td>:0000</td>
<td></td>
</tr>
<tr>
<td>F1:</td>
<td>:B100</td>
<td>:F1FF</td>
<td>:0100</td>
<td></td>
</tr>
<tr>
<td>F2:</td>
<td>1720</td>
<td>:F2FF</td>
<td>:0200</td>
<td></td>
</tr>
<tr>
<td>F3:</td>
<td>:BE00</td>
<td>:F3FF</td>
<td>:0300</td>
<td></td>
</tr>
<tr>
<td>F4:</td>
<td>:B600</td>
<td>:F4FF</td>
<td>:0400</td>
<td></td>
</tr>
<tr>
<td>F5:</td>
<td>:A700</td>
<td>:F5FF</td>
<td>:0500</td>
<td></td>
</tr>
<tr>
<td>F6:</td>
<td>:F700</td>
<td>:F6FF</td>
<td>:0600</td>
<td></td>
</tr>
<tr>
<td>F7:</td>
<td>:B800</td>
<td>:F7FF</td>
<td>:0700</td>
<td></td>
</tr>
<tr>
<td>F8:</td>
<td>:B000</td>
<td>:F8FF</td>
<td>:0000</td>
<td></td>
</tr>
<tr>
<td>F9:</td>
<td>:A800</td>
<td>:F9FF</td>
<td>:0100</td>
<td></td>
</tr>
<tr>
<td>FA:</td>
<td>:B700</td>
<td>:FAFF</td>
<td>:0200</td>
<td></td>
</tr>
<tr>
<td>FB:</td>
<td>:B900</td>
<td>:FBFF</td>
<td>:0300</td>
<td></td>
</tr>
<tr>
<td>FC:</td>
<td>:B300</td>
<td>:FCFF</td>
<td>:0000</td>
<td></td>
</tr>
<tr>
<td>FD:</td>
<td>:B200</td>
<td>:FDFF</td>
<td>:0100</td>
<td></td>
</tr>
<tr>
<td>FE:</td>
<td>:A025</td>
<td>:FEFF</td>
<td>:0000</td>
<td></td>
</tr>
<tr>
<td>FF:</td>
<td>:A000</td>
<td>:FFFF</td>
<td>:0000</td>
<td></td>
</tr>
</tbody>
</table>
What can be addressed?

- With 2 out of 4 bytes to influence, one gets roughly 65535 different addresses instead of 4294967295
- ~86% of this address space is not mapped
  - Useless, unless Dr. Watson is invited for dinner
- Threads make fixed addresses unreliable
- Currently, the most common method is blowing up the heap
  - Heap of the target process is artificially inflated
  - More memory gets mapped
  - Higher chances to find an addressable section
  - Method most widely used by KF/Snosoft
What about format strings?

- Format string bugs in pure wchar functions (swprintf, fwprintf) are perfectly fine.
  - ...of course, with the usual addressing problems
- For ASCII versions, remember that at least every second byte gets modified
- Format arguments work on a minimum of two bytes (%s, %x, etc.)
- Three byte format args are possible (%3d),
  - but the middle byte becomes 0x00 most of the time...
  - This in turn means „end of string“ 😞
- With OEM, you got a lot of „%“ anyway
  - But again, a lot of 0x00 bytes as well
Format strings in color

- User input:
  ![User input diagram]

- Wide char by ANSI:
  ![Wide char by ANSI diagram]

- *printf() call: „“
  - 0x00 byte encountered, end of string

- *wprintf() call: „AAAA_23FEFE23“
  - Format argument is wide char
Return address selection

- Blowing up the heap
  - Doesn't always work
  - Is unreliable since it's unpredictable
  - Might blow up other things...

- Finding a JMP/CALL <reg>
  - If a register points to code we can use, all we need is a jump or call to that register
  - Much more reliable
  - Must be addressable in our wide char scenario
  - Very limited number of directly addressable code sections
Return address selection - SEH

- If no register points to data we can influence, a JMP/CALL <reg> doesn't help
- Overflowing further, code added structured exception handlers (SEH) might be present
- Upon an exception, the overwritten SEH address is called (code execution)
- Upon execution of (attacker provided) SEH, EBX points to SEH record

Overflow

Pointer to next SEH record

SE handler

Attacker provided
SEH, again in color

1. Overflow up to SEH address
2. Trigger exception (not hard 😊)
3. Get code from ntdll.dll
4. Enjoy

AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AAA
AA
Return to register, but how?

- Useful JMP/CALL <reg> sequences in wide char addressable locations are very hard to find.
- Solution: pure simple brute force
  - Search the entire mapped address space for wide char addressable locations
  - Search from those locations ...
    - Bail if memory access occurs
    - Print result if JMP/CALL <reg> is found
    - Recourse if CALL/JMP <imm> is found

→ Find all addressable JMP/CALL <reg>
... while at it ...

... put an end to those return address issues
- Also support search for JMP/CALL <reg> in ASCII overflows
- Support automatic handling of forbidden characters such as 0x00
- Support stack-return as well
  - If a pointer to your buffer is further up in stack, adjust stack by n bytes and return
- Support saving the return addresses
- Support diffing of return addresses

→ Phenoelit OllyUni Plugin for OllyDbg

Phenoelit
OllyUni finding example

- UNICODE return addresses that are not directly reachable:

```plaintext
0x00420153 is addressable by the sequence 0x429C in the ANSI table
0x0042015D is not Unicode addressable, but contains CALL EBX
```

```plaintext
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00420153</td>
<td>57</td>
</tr>
<tr>
<td>00420154</td>
<td>8D45 E8</td>
</tr>
<tr>
<td>00420157</td>
<td>68 30957100</td>
</tr>
<tr>
<td>0042015C</td>
<td>50</td>
</tr>
<tr>
<td>0042015D</td>
<td>FFD3</td>
</tr>
</tbody>
</table>

PUSH EDI
LEA EAX, DWORD PTR SS:[EBP-18]
PUSH LIBRFC32.00719530
PUSH EAX
CALL EBX
```
The shell code dilemma

- Returning in our buffer works now, but look what happened to the shell code ...

<table>
<thead>
<tr>
<th>E8 00000000</th>
<th>CALL 004015C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5D</td>
<td>POP EBP</td>
</tr>
<tr>
<td>64:8B0D 000000</td>
<td>MOV ECX,DWORD PTR FS:[0]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E8 00000000</th>
<th>CALL 004015C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>ADD BYTE PTR DS:[EAX],AL</td>
</tr>
<tr>
<td>0000</td>
<td>ADD BYTE PTR DS:[EAX],AL</td>
</tr>
<tr>
<td>005D 00</td>
<td>ADD BYTE PTR SS:[EBP],BL</td>
</tr>
<tr>
<td>64:008B 000D00</td>
<td>ADD BYTE PTR FS:[EBX+D00],CL</td>
</tr>
<tr>
<td>0000</td>
<td>ADD BYTE PTR DS:[EAX],AL</td>
</tr>
<tr>
<td>0000</td>
<td>ADD BYTE PTR DS:[EAX],AL</td>
</tr>
<tr>
<td>0000</td>
<td>ADD BYTE PTR DS:[EAX],AL</td>
</tr>
</tbody>
</table>
Solutions

- Injected 0x00 (or other) bytes have to be planned for
  - Create code with the 0x00 bytes being part of it
    → considered hard & not practical
- Create code that uses „padding“ to get rid of the annoying 0x00 bytes
  - Intel architecture helps us with the variable length commands
  - Effectively use every second command for the real work
Stack walk code

- The 0x00 byte padding method can be used for very simple things, if a register points to a ASCII version of our data:
  50 :push eax
  00 6D 00 :add byte ptr [ebp],ch
  C3 :ret

- Intermix some INC/DEC instructions to adjust the register where required
Venetian Shellcode

- First published as „Creating Arbitrary Shellcode In Unicode Expanded Strings“ by Chris Anley (chris@nextgenss.com)
- Chris dubbed the method „venetian shell code“ due to the fact that the 0x00 gaps are closed like a venetian blind
- Only (to me) known public implementation from Dave Aitel in makeunicode2.py
  - Unfortunately it needs a lot of fixing (sorry ;)
  - A commercial version is included in CANVAS and might be working better.
Venetian code in color

1. Set one register to the start of your real shell code
2. Pad 3 bytes
3. Modify the 0x00 byte
4. Pad 3 bytes
5. Increase your pointer register
6. Goto 2
Venetian Shellcode [2]

- Using opcodes with 0x00 in them, one can write code to fill the gaps:
  - add byte ptr [EAX],<value> with opcode 80 00 ?? to set values where 0x00 bytes are
  - add byte ptr [EBP],CH with opcode 00 6D 00 to „realign“ after each „real“ instruction
  - xchg EAX,ESP with opcode 94 to get a pointer to the code into EAX
Venetian Shellcode [3]

- Size is a problem. An example code from the original paper:

```assembly
40 :inc eax
00 6D 00 :add byte ptr [ebp], ch
40 :inc eax
00 6D 00 :add byte ptr [ebp], ch
80 00 75 :add byte ptr [eax], 75h
00 6D 00 :add byte ptr [ebp], ch
```

- This means 14 bytes to set one byte. Although the next one is for free (unmodified original), that’s a lot of code.

- The shell code string in the above example would be: `\x40\x6d\x40\x6d\x80\x75\x6D`
The size of the venetian shellcode increases ~10-14 bytes per byte real shell code

- Real shell code should be as small as possible
- Dave Aitel uses a second stage code, which scans the stack for the final bind shell code
- Becomes XXL for higher numbers of not usable chars (filter problem)

The result is a 3 stage code:
1. UNICODE venetian code that creates second stage
2. Second stage code that searches for the bind shell
3. Your well known bind shell code

remote root
Venetian shellcode generator

- Instant UNICODE enabled instructions should be preferred, since no realignment is needed
  - NOP instruction ADD AL,0 = 0400
  - Byte modification instruction
    INC byte ptr DS:[EAX] = FE00
  - MOV instruction (needs realignment)
    MOV byte ptr DS:[EAX],<val> = C600xx

- Placing the second stage code multiples of 256 after the venetian code decreases size further
  - Adding to EAX is only possible for byte 2 and 4 due to „ADD EAX,11002200“ being 05 00 22 00 11
  - Numbers <256 can only be added to EAX using INC, which means 4 bytes (INC+padding) per one step
Only searching on stack is unreliable
- Most often, a totally unrelated injection vector transports the third stage shell code
- Using the SEH, the full 4GB memory space can be scanned for the third stage code
  1. start at 0x00000000
  2. Install SEH (5)
  3. Loop-scan for your code
     - If found goto 4
     - If exception, SEH is triggered
  4. Execute third stage
  5. [SEH] increase counter
  6. [SEH] goto 3
Second stage code [2]

call near .getdelta
.getdelta:
    pop ebp
    mov ebx,ebp
    add ebx,.seh
    sub ebx,0x5
    mov edi,SEARCHFROM
.outsearch:
    pusha
    push ebx
    push dword [fs:0]
    mov [fs:0],esp
    mov esi,edi
    search:
        cmp dword [esi],SEARCHPATTERN
        jz .RunThird
        inc esi
        jmp short .search
    .RunThird:
        inc esi
        inc esi
        inc esi
        inc esi
        jmp esi
    .seh
        mov dword esp,[esp+8]
        pop dword [fs:0]
        add esp,0x4
        popa
        add edi,0x00001000
        add edi,0xFFFFF000
        jmp .outsearch
Phenoelit „vense“ generator

- Release of a working UNICODE shell code generator in 379 lines of Perl
- ~16%-20% smaller venetian code than Dave‘s makeunicode2.py
- Some handling of forbidden or unreliable characters (ANSI code page)
- Second stage shell code for full memory search of third stage code
Putting it all to use ...
SAP Internet Transaction Server

- Three tier architecture to Internet-enable SAP R/3
  - WGate plugin for web server (IIS ISAPI, NES Plugin or CGI)
  - AGate service, communicating between WGate and R/3 – acting as middleware
- (relatively) easy to install, runs on Windows NT/2000 or Linux
- Directly connected to the existing SAP R/3
This installation type is appropriate for production systems where additional security is desirable. In this case, you can install a second firewall between WGate and AGate, as shown below. For even greater security, you could also place a firewall between AGate and R/3.

**Increased ITS Security**

- **HTTP**
- **HTTPS Generic Firewall**
- **Web Host**
- **SNC Encryption SAP router Firewall**
- **ITS Host**
- **R/3 Application Server**
- **TCP/IP**
Vulnerabilities in ITS

- **WGate**
  - Format string vulnerability in logging for higher „trace level“ (SAPisch for log level)

- **AGate (directly exploitable through WGate)**
  - Buffer overflow in ~command parameter
  - Buffer overflow in ~runtimemode parameter
  - Buffer overflow in ~session parameter
  - Buffer overflow in HTTP Content-Type field (ASCII)

- **Info leak in AGate**
  - ~command=AgateInstallCheck gives all installed DLLs, their path and exact version 😊
Break for administrative stuff

- Fix information for the listed vulnerabilities
  - SAP Notes component BC-FES-ITS
  - Notes 678526, 678523, 569011
  - We have *no idea* what that means, so don’t ask us what’s 0day and what’s FFday

- SAP Advisories and Patches require
  - Registration
  - Customer-, Partner- or Installation number

- Apparently, all is well in ITS 6.20 PL7, 6.10 PL30 and 4.6 PL463
Techniques used

- AGate ~runtimemode overflow
  - SEH based return to UNICODE addressable CALL EBX
  - Phenoelit Venetian code
  - Second stage SEH memory scan code
  - Halvar‘s TCP backconnect

- AGate ~command overflow
  - Direct return to UNICODE addressable CALL
  - Dave‘s Venetian code
  - Halvar‘s TCP backconnect

- AGate Content-Type overflow is boring ASCII, but makes good use of the Ret-Addr diffing
Techniques used [2]

- WGate format string (not UNICODE)
  - Format string generator for shell codes:
    ```
    for ($i=0; $i<length($sc); $i++) {
        $char = substr($sc,$i,1);
        $delta = (0x100 | ord($char)) –
                   ($initial & 0xFF);
        $format .= sprintf("%%%uu%%n",$delta);
        $initial = ord($char);
    }
    
    All you need to know is the number of bytes written before your format string is hit
    
    Result:
    %361u%n%24u%n%256u%n%256u%n%256u%n%256u%n%351u%n%240u%n%256u%n%256u%n...
    ```
Another note regarding SAP

- We had the hope that ITS is an exception...
- mySAP.com architecture vulnerabilities:
  - Buffer overflow while handling HTTP Host tag in Message Server
  - Buffer overflow while handling HTTP Host tag in Web Dispatcher
  - Buffer overflow while handling HTTP Host tag in Application Server
- No details, since it’s not fixed yet (or we don‘t know – remember, the login thing...)

Phenoelit
UNICODE overflows are exploitable – pretty much like everything else

You can use your CPU power for reliable return address search instead of looking for little green people on mars.

And still: Just because a platform is obscure, it does not mean it’s not going to be exploited one day or another.

Get all the stuff at http://www.phenoelit.de/whatSAP/
DA GREEZ: Phenoslit

HALLAP GERA DAVE

NGS JEFF & Ping

415 605-5

The Royal Hacking Society