ECE590
Computer and Information Security

Fall 2018

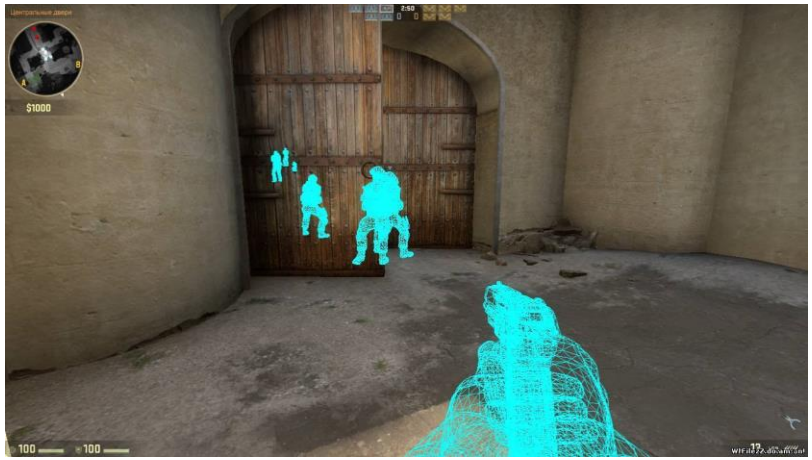
Reverse Engineering

Tyler Bletsch
Duke University

With additional content by Jiaming Li, NC State University, 2015

What is software reverse engineering?

- Determine and possibly change program logic
  - “Logic” ≠ Just observed behavior
- Ethics
  - Useful for good:
    - Analyze malware
    - Understand undocumented legacy code
    - Watch/read/play the damn crap you paid for
  - Useful for evil
    - “Crack” software (remove restrictions)
    - Find exploits
    - Cheat at games



Types of tools

- Disassembler
  - Turn compiled program into assembly
  - Not perfect
  - Static tool
- Debugger
  - Step through running program
  - Dynamic tool
- Hex editor
  - Make changes to binaries
- Monitoring tools
  - Watch system calls, library calls, etc.

Also, **decompiler**:

- Attempts to turn assembly back into source code.
- Usually awful at machine code, but managed code (e.g. Java, Python) can produce decent results.

Examples of tools

- Linux:
  - Disassember: objdump (free), **IDA Pro** (free and paid versions)
  - Debugger: **gdb** and its front-ends
  - Hex editor: okteta, bless, lots more…
  - Monitoring: **strace**, ltrace
- Windows:
  - Disassembler: **IDA Pro** (free and paid versions)
  - Debugger: WinDBG (basic), **OllyDbg** (shareware), SoftICE ($1000+)
  - Hex editor: XVI32, Notepad++ with plugin, etc.
  - Monitoring tools: Process Monitor, Explorer, and more.
- X86 in general: A hypervisor (VMware, KVM, etc.)

IDA Pro eats basically anything

Debug or disassemble? Both.

- Disassembler gives **static** results
  - Good overview of program logic
  - But need to “mentally execute” program
  - Difficult to jump to specific place in the code
- Debugger is **dynamic**
  - Can set break points
  - Can treat complex code as “black box”
  - Not all code disassembles correctly
- Disassembler **and** debugger both required for any serious SRE task

From "Computer Science 654 Lecture 5: Software Reverse Engineering" by Wayne Patterson, Howard Univ. 2009.

Example 1: HW2 auto-grader

- Python decompiles very easily


```
1  # Embedded file name: cryptotest.py
2  import subprocess
3  import sys, os
4  import zlib
5  import getpass
6  import socket
7  import datetime
8  import hashlib
9  from StringIO import StringIO
10 suppress_output = True
11 HEADER = 'cryptotest v1.1.0 by Dr. Tyler Bletsch (tkbletsc@ncsu.edu)'
12 color_cmd = '33'
13 color_status = '32'
14 color_tests = '44;96'
15
16 def hash_self():
17     return hash_file(sys.argv[0])
18
19
20 def hash_file(filename):
21     with open(filename, 'rb') as fp:
22         m = hashlib.md5()
23         m.update(fp.read() + 'yg' + 'slt')
24         return m.hexdigest()
```

Example 2: Minecraft

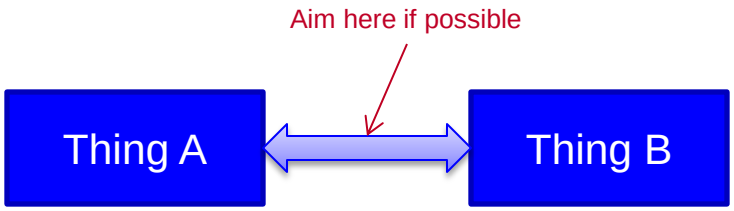
- Minecraft is a Java program, no mod support
- All mods use something like the Mod Coder Pack (MCP):

> "Use MCP to decompile the Minecraft client and server jar files.
>
> Use the decompiled source code to create mods for Minecraft.
>
> Recompile modified versions of Minecraft.
>
> Reobfuscate the classes of your mod for Minecraft."

- Entire mod community is built on reverse engineering!

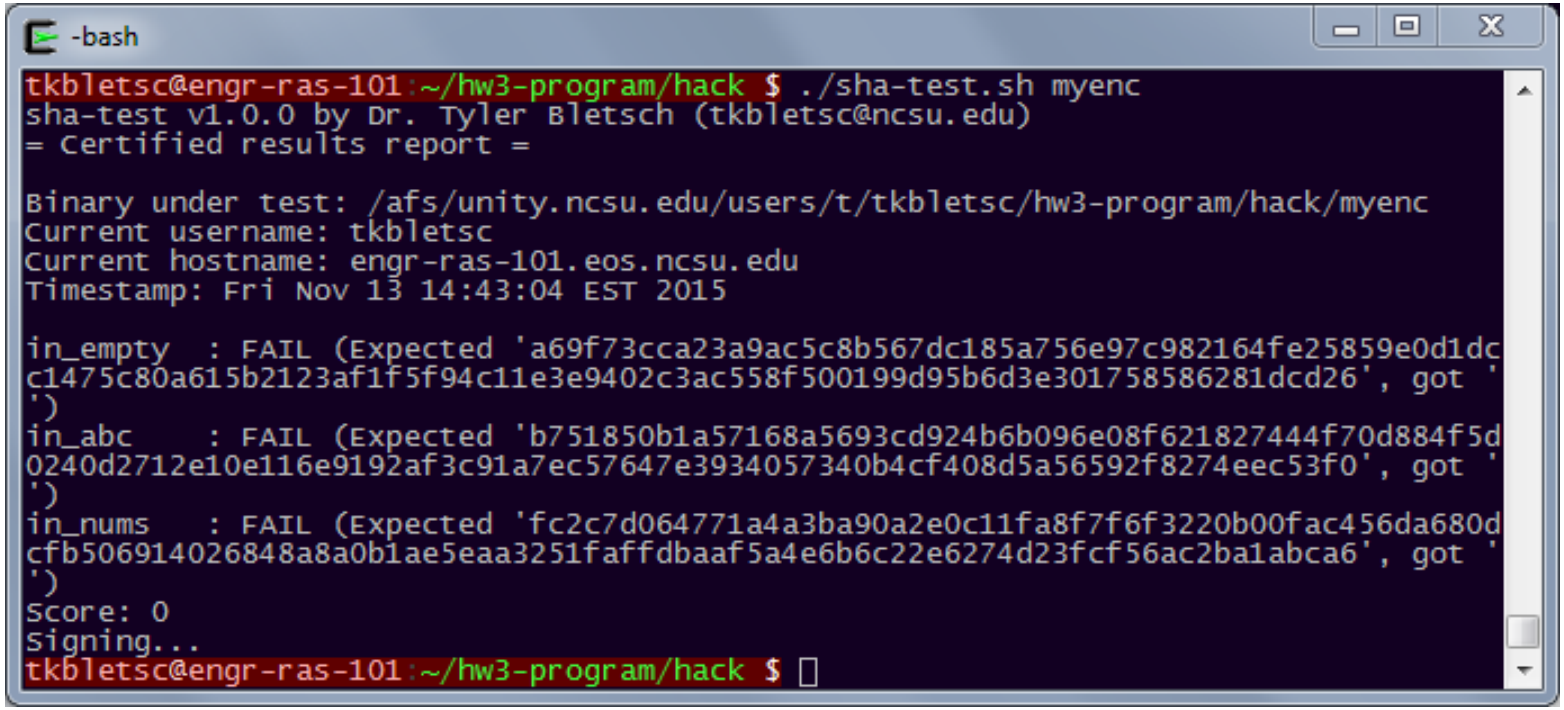
Examining multi-component systems

- Weaknesses often at the seams – where parts of system come together
  - Most visible, often exploitable
  - Example: SQL injection
- If not the seams, at least focus on the least protected part



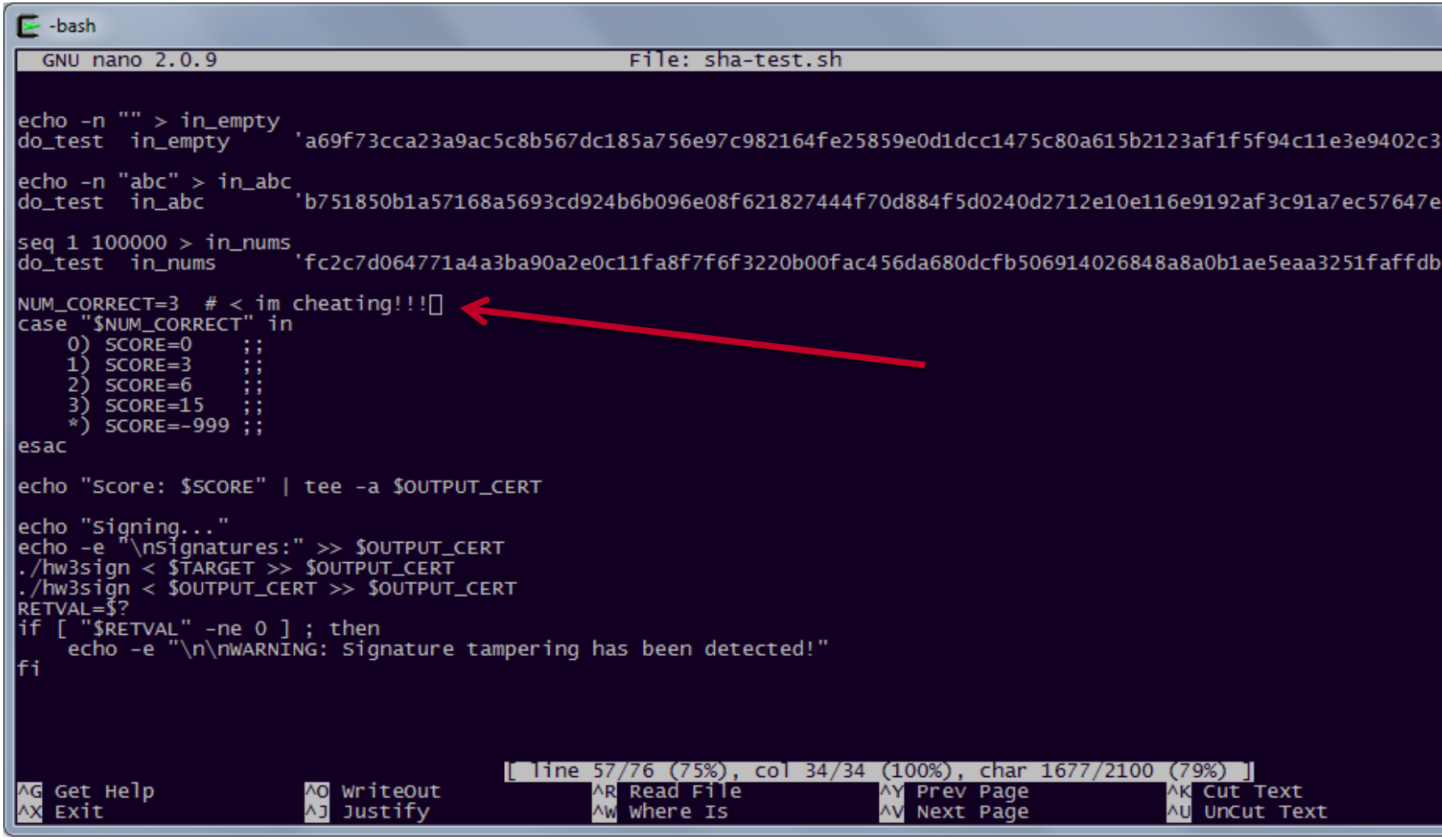
Example 3: HW3 auto-grader

- Files:
  - Binary: **`hw3sign`**
  - Shell script: **`sha-test.sh`**
- Normal usage:



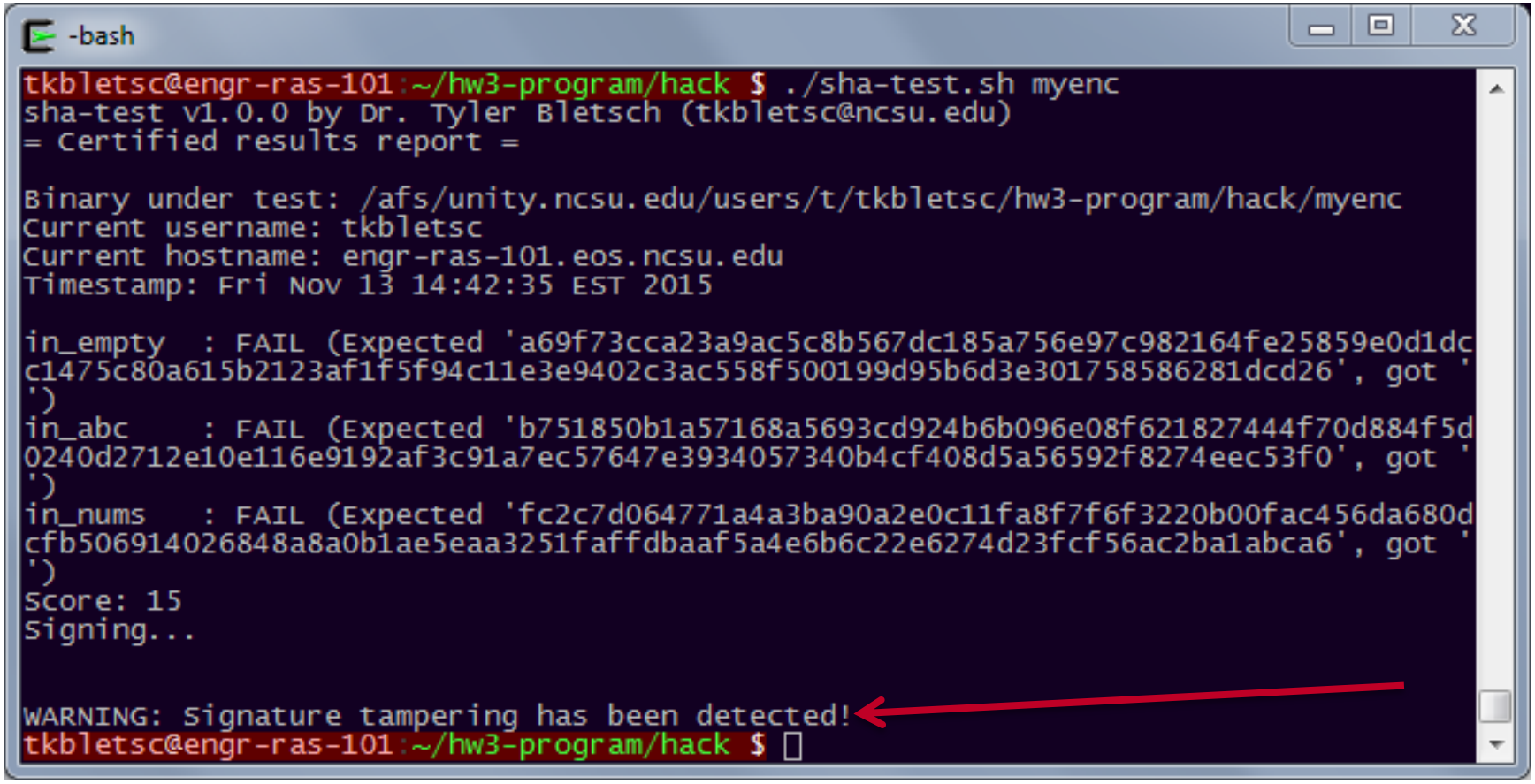
Example 3: HW3 auto-grader

- Naïve attack: Just change the script



Example 3: HW3 auto-grader

- Naïve attack: Just change the script
  - Failed: hw3sign must be checking it somehow.



Example 3: HW3 auto-grader Topology

`hw3sign`

`sha-test.sh`

Example 3: HW3 auto-grader

- Could look at behavior with strace:

```
$ strace -f -o trace.txt ./sha-test.sh myenc
        ...
$ cat trace.txt
4127  execve("./sha-test.sh", ["./sha-test.sh", "myenc"], [/* 46 vars */]) = 0
4127  brk(0)                                = 0x1700000
4127  mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
    = 0x7f55d5a17000
4127  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
    directory)
4127  open("/etc/ld.so.cache", O_RDONLY) = 3
4127  fstat(3, {st_mode=S_IFREG|0644, st_size=210058, ...}) = 0
4127  mmap(NULL, 210058, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f55d59e3000
4127 close(3)                               = 0
        ...
```

- But hw3sign never appears to open sha-test.sh:

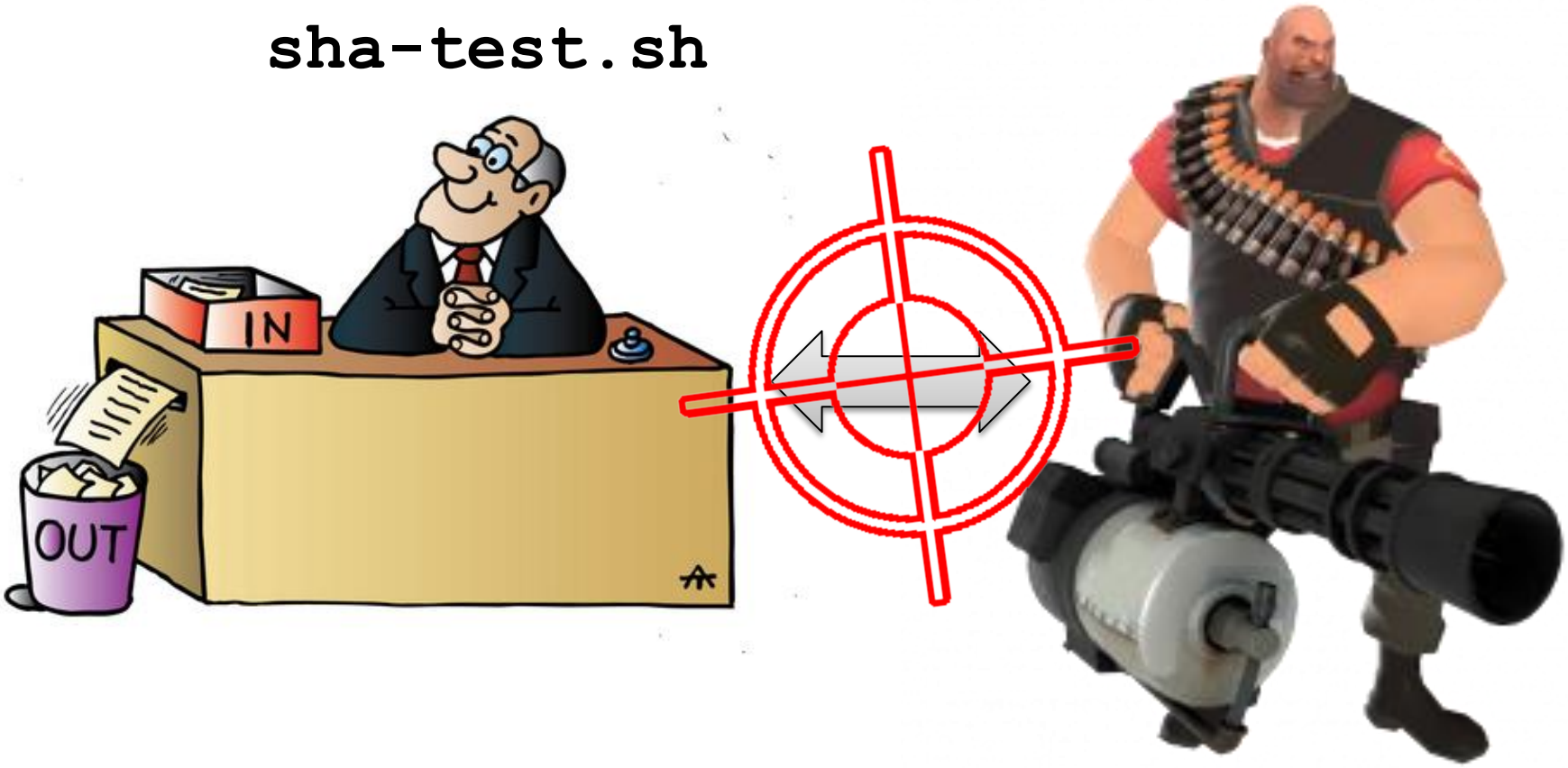
```
$ grep open trace.txt | grep sha-test.sh
4127  open("./sha-test.sh", O_RDONLY)    = 3
```

  - This one line is from when sha-test.sh itself is started
  - There's more mystery here that I'll leave to you…

Example 3: HW3 auto-grader
Best place to attack?

`hw3sign`

`sha-test.sh`



Example 3: HW3 auto-grader

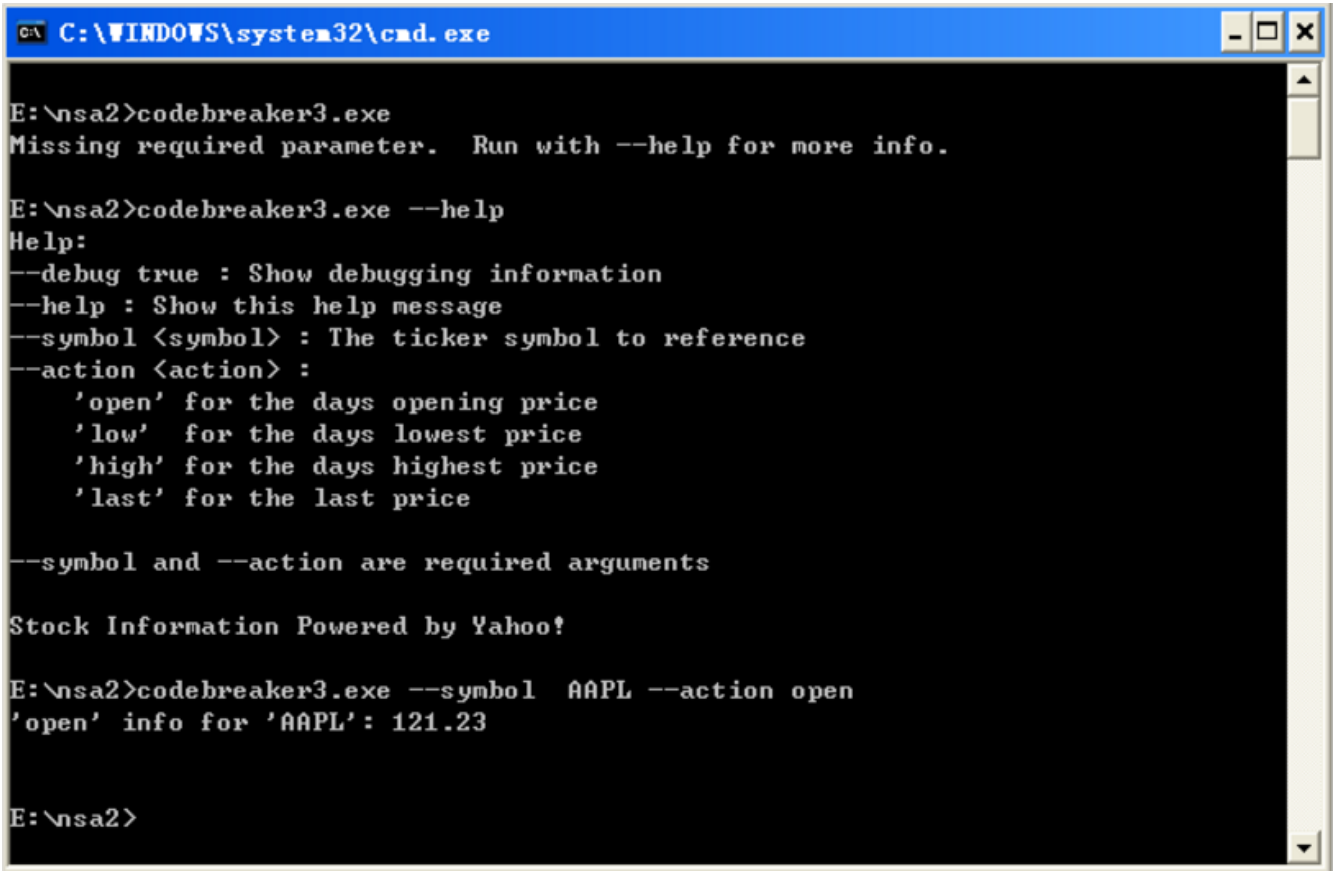
- Two past successful student attacks
- **Black box attack:**
  - hw3sign signs the binary, then the certificate itself
  - What if we ask it to “test” a doctored certificate as a binary – it will sign it for us! No understanding needed!
- **Chameleon attack:**
  - Add cheating to sha-test.sh; also add code to copy a legit sha-test.sh over itself before doing signings
  - Malicious behavior occurs then hides before check occurs

Example 4: NSA Codebreaker challenge

- Scenario:
  - Terrorists using a cryptography program to decrypt/authenticate messages from leadership
  - What we have:
    - The program: **`codebreaker3.exe`**
    - A member's key: **`tier1_key.pem`**
    - A text file with a hidden message: **`tier1_msg.txt`**
  - At first glance, the program appears to simply check stock information, but that's a ruse.
  - Need to reverse engineer it:
    Challenge has 4 tasks, we'll do 2.

Codebreaker Task 1: Decrypt

- Need to decode message we have.
- The program:



Adapted from content by Jiaming Li, NC State University, 2015

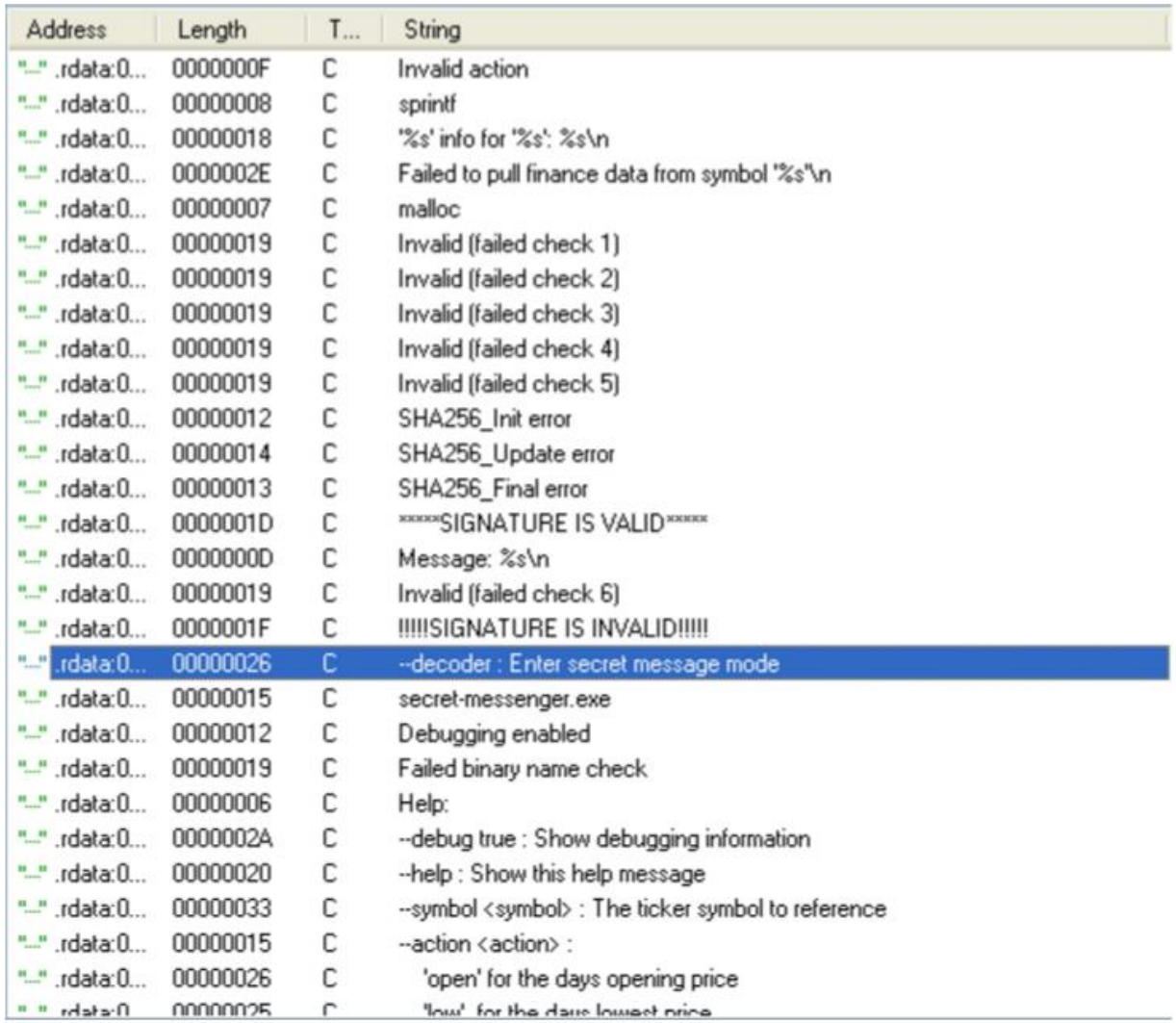
Codebreaker Task 1: Decrypt

- Do static analysis with IDA Pro
  - Load binary
  - Confirm binary format options
  - Process:
    - Code is disassembled
    - Call graph of assembly code built
    - All memory references are cross-referenced, especially string literals

Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 1: Decrypt

- Do static analysis with IDA Pro, check the all string information

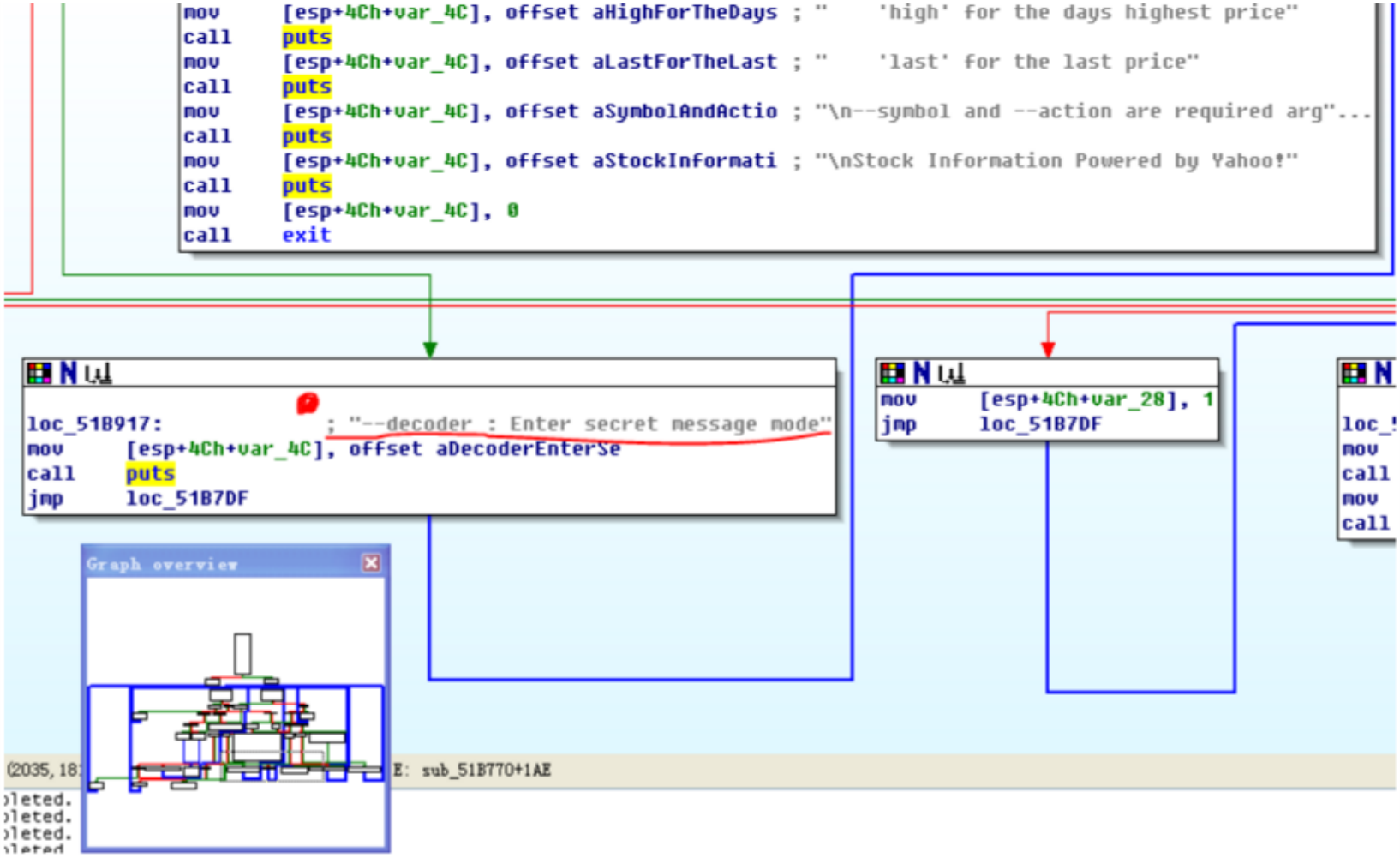


Address	Length	T...	String
"..." .rdata:0...	0000000F	C	Invalid action
"..." .rdata:0...	00000008	C	sprintf
"..." .rdata:0...	00000018	C	'%s' info for '%s': %s\n
"..." .rdata:0...	0000002E	C	Failed to pull finance data from symbol '%s'\n
"..." .rdata:0...	00000007	C	malloc
"..." .rdata:0...	00000019	C	Invalid (failed check 1)
"..." .rdata:0...	00000019	C	Invalid (failed check 2)
"..." .rdata:0...	00000019	C	Invalid (failed check 3)
"..." .rdata:0...	00000019	C	Invalid (failed check 4)
"..." .rdata:0...	00000019	C	Invalid (failed check 5)
"..." .rdata:0...	00000012	C	SHA256_Init error
"..." .rdata:0...	00000014	C	SHA256_Update error
"..." .rdata:0...	00000013	C	SHA256_Final error
"..." .rdata:0...	0000001D	C	*****SIGNATURE IS VALID*****
"..." .rdata:0...	0000000D	C	Message: %s\n
"..." .rdata:0...	00000019	C	Invalid (failed check 6)
"..." .rdata:0...	0000001F	C	!!!!!SIGNATURE IS INVALID!!!!!
"..." .rdata:0...	00000026	C	--decoder : Enter secret message mode
"..." .rdata:0...	00000015	C	secret-messenger.exe
"..." .rdata:0...	00000012	C	Debugging enabled
"..." .rdata:0...	00000019	C	Failed binary name check
"..." .rdata:0...	00000006	C	Help:
"..." .rdata:0...	0000002A	C	--debug true : Show debugging information
"..." .rdata:0...	00000020	C	--help : Show this help message
"..." .rdata:0...	00000033	C	--symbol <symbol> : The ticker symbol to reference
"..." .rdata:0...	00000015	C	--action <action> :
"..." .rdata:0...	00000026	C	'open' for the days opening price
"..." .rdata:0...	00000025	C	'low' for the days lowest price

Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 1: Decrypt

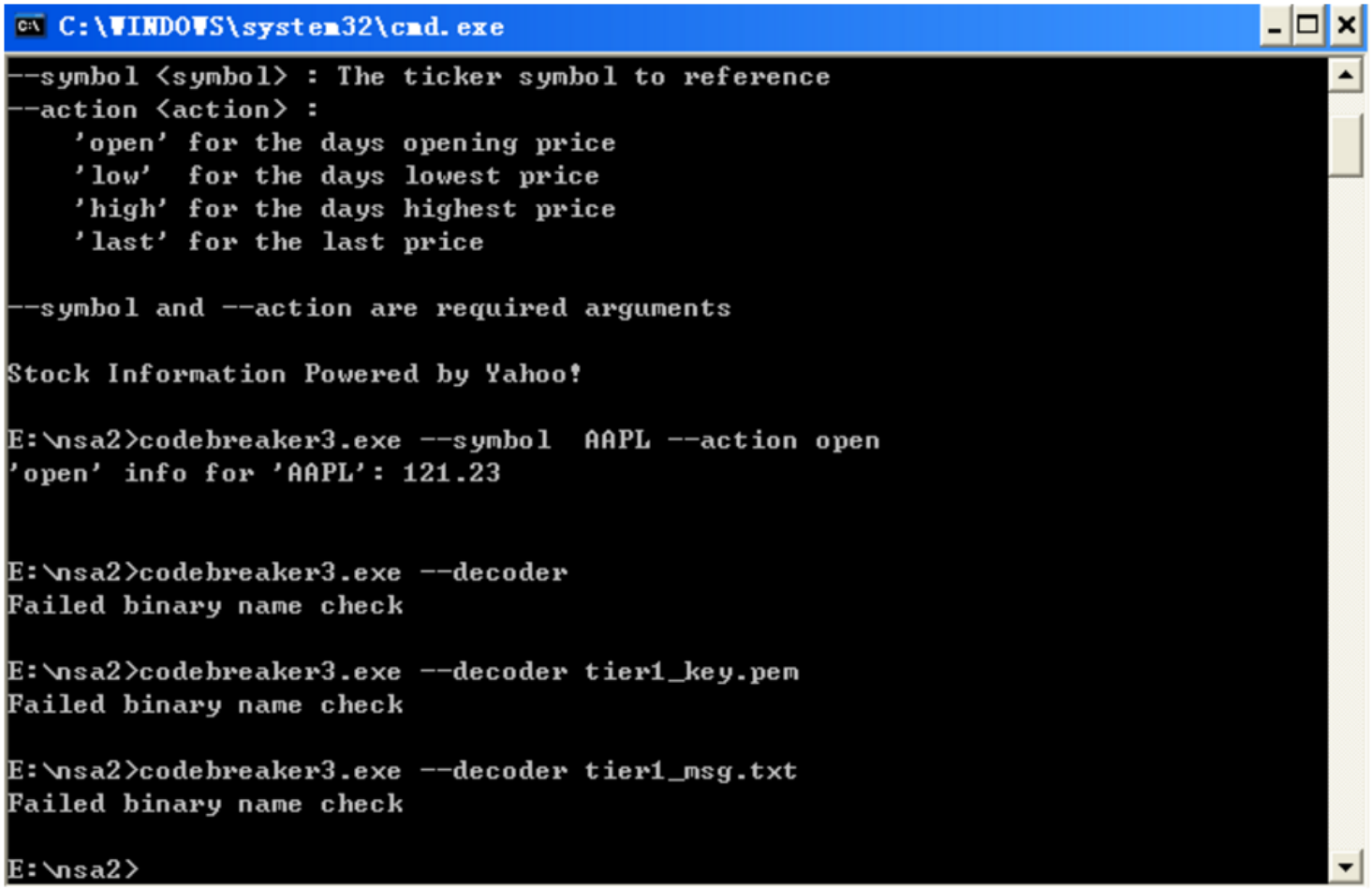
- Press x, this leads us to the location where this string appears:



Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 1: Decrypt

- OK, let's try "decoder" parameter:



Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 1: Decrypt

- We need to find where “Failed binary name check” appears:

```
loc_51B9A6:                ; "Failed binary name check"
mov     [esp+4Ch+var_4C], offset aFailedBinaryNa
call    puts
mov     [esp+4Ch+var_4C], 1
call    exit
```

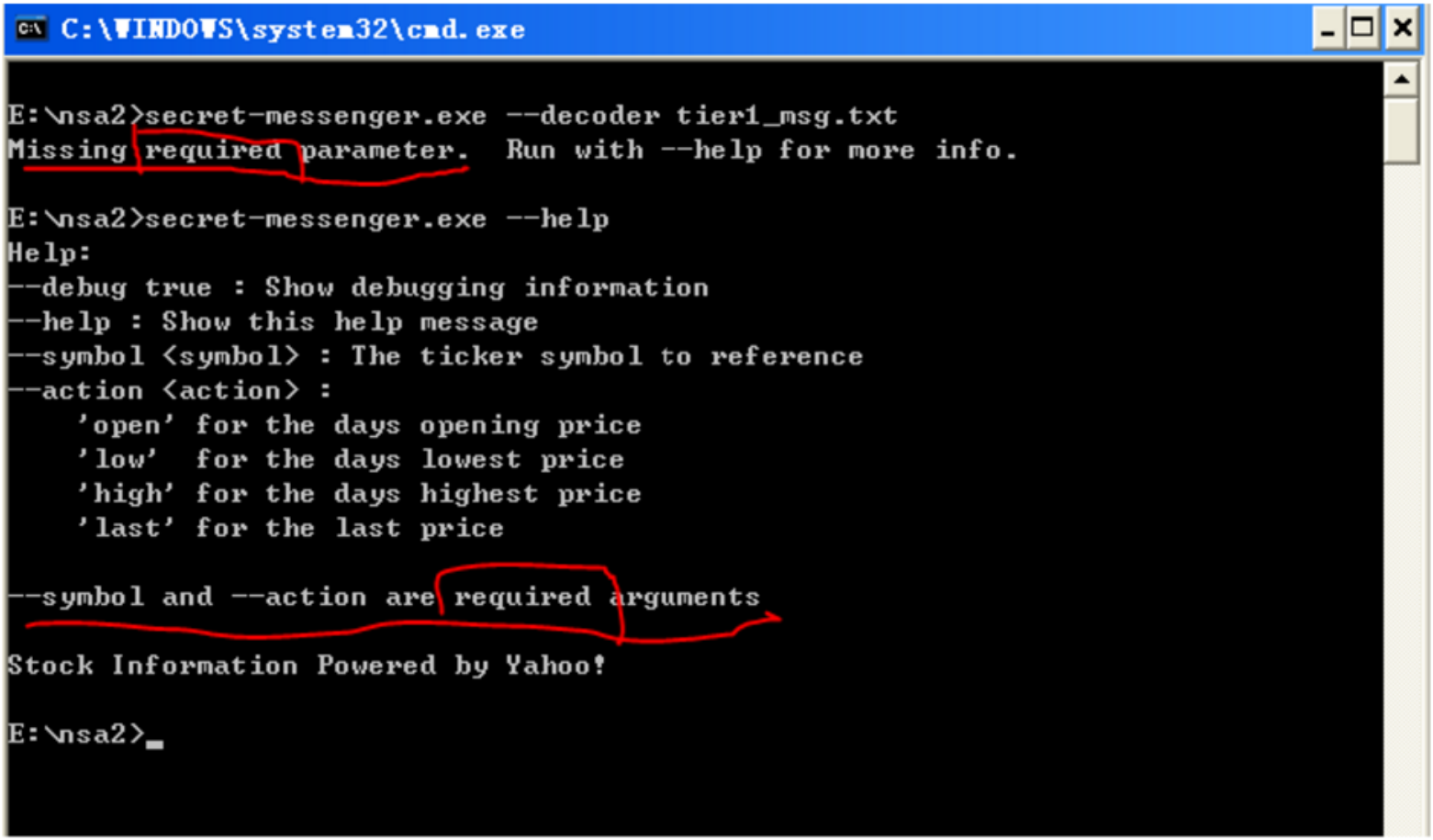
- and this comes from:

```
loc_51B928:
mov     [esp+4Ch+var_44], 15h
mov     [esp+4Ch+var_48], edi
mov     [esp+4Ch+var_4C], offset aSecretMessenge ; "secret-messenger.exe"
call    memcmp
test    eax, eax
jnz     short loc_51B9A6
```


Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 1: Decrypt

- Then we change our program name to "secret-messenger.exe" and try again:



Codebreaker Task 1: Decrypt

- Ideas?

- Let's jam the stuff into symbol and action fields

```
E:\nsa2>secret-messenger.exe --symbol tier1_key.pem --action tier1_msg.txt --dec
oder
*****SIGNATURE IS VALID*****
Message: Meet at 22:00 tomorrow at our secure location.  Come alone, and do not
tell anyone - this meeting is sensitive, as leadership will be present.  To auth
enticate yourself, mention the pass code hpbl8rlmmhfkcadi6nj8 at the door.
*****SIGNATURE IS VALID*****

E:\nsa2>_
```

Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 2: Bypass access limitation

- We've collected a **new message file** - this one to a different field operative **whose key we also have**.
- Each operative has their own decrypt tool, each tool will only decrypt content "addressed" to a its owner.
- Need to defeat this access limitation to decrypt the message.

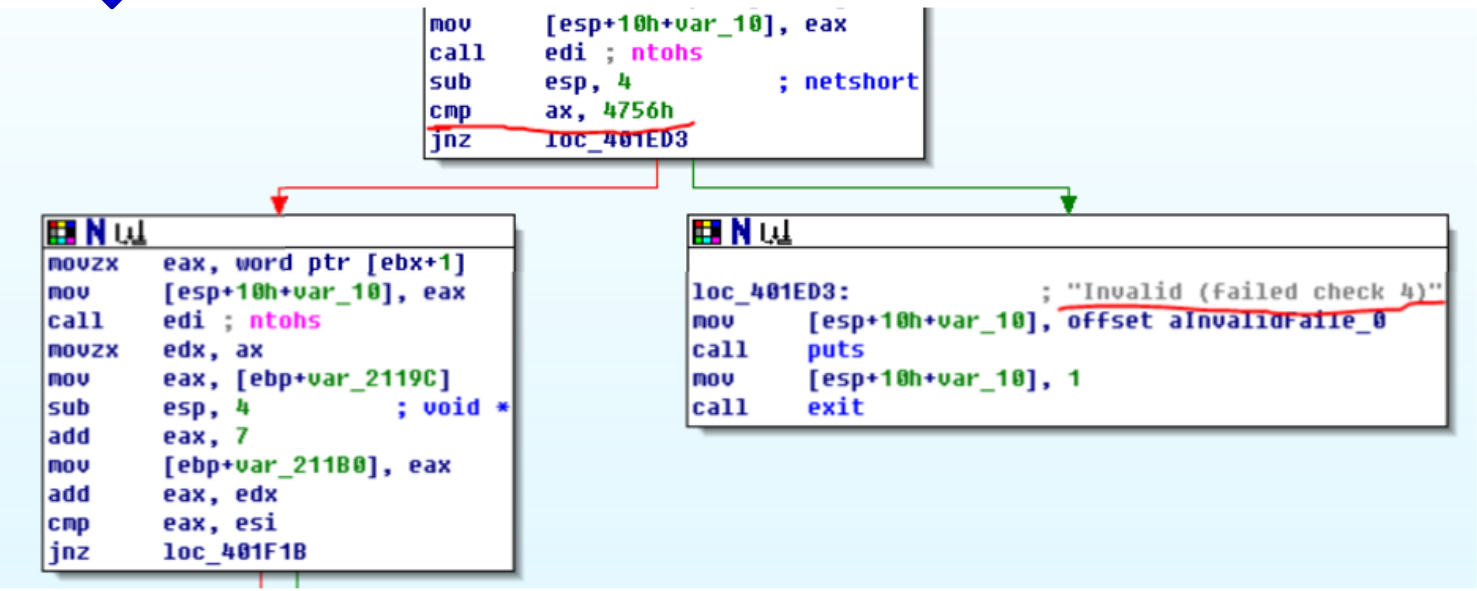
```
E:\nsa2>secret-messenger.exe --symbol tier2_key.pem --action tier2_msg.txt --dec
oder
Invalid (failed check 4)

E:\nsa2>
```

Codebreaker Task 2: Bypass access limitation

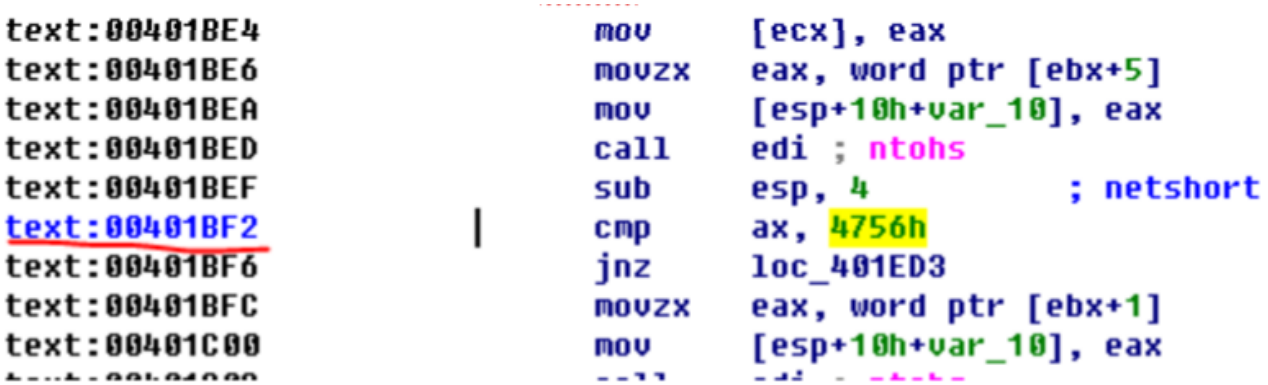
- Let's go back to IDA to find where this error appears:

Address	Length	T...	String
"..." .rdata:0...	0000000F	C	Invalid action
"..." .rdata:0...	00000008	C	sprintf
"..." .rdata:0...	00000018	C	'%s' info for '%s': %s\n
"..." .rdata:0...	0000002E	C	Failed to pull finance data from symbol '%s'\n
"..." .rdata:0...	00000007	C	malloc
"..." .rdata:0...	00000019	C	Invalid (failed check 1)
"..." .rdata:0...	00000019	C	Invalid (failed check 2)
"..." .rdata:0...	00000019	C	Invalid (failed check 3)
"..." .rdata:0...	00000019	C	Invalid (failed check 4)



Codebreaker Task 2: Bypass access limitation

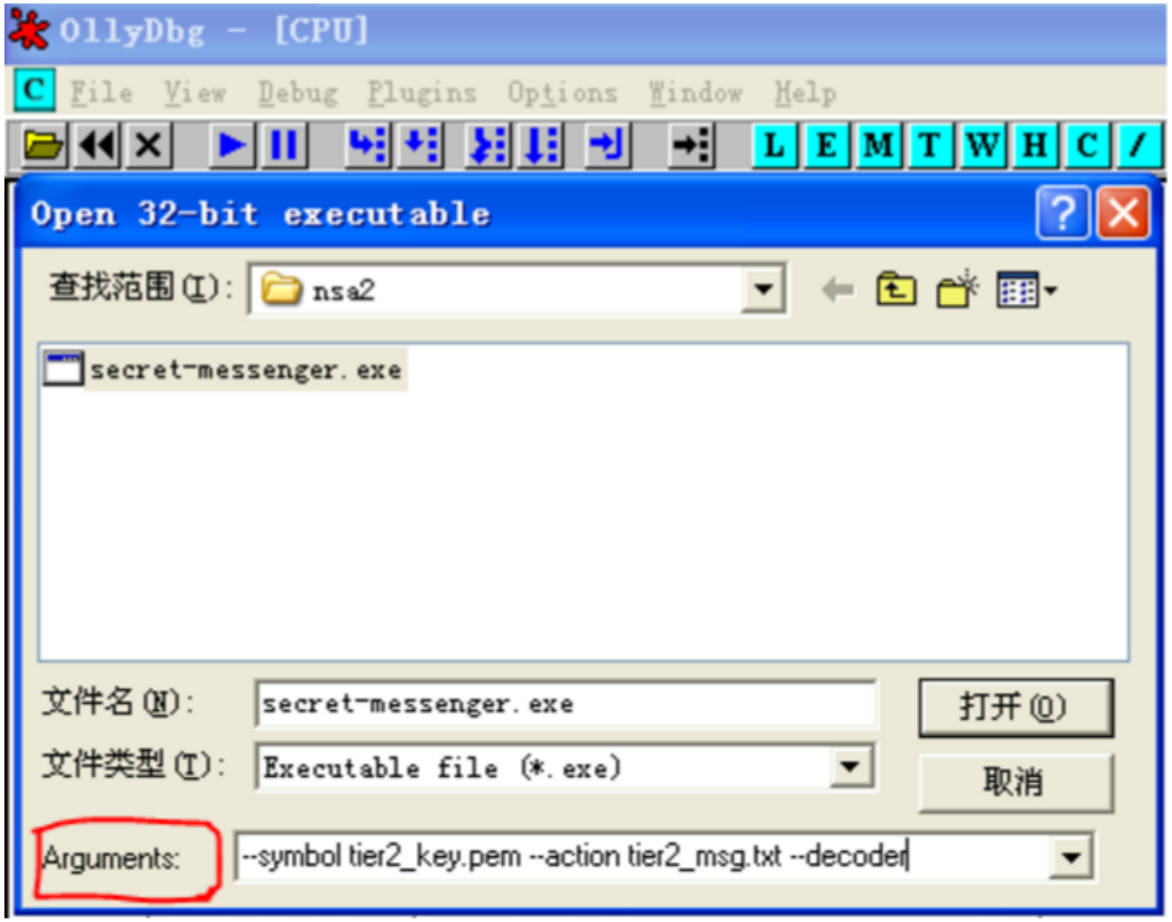
- Note down the address of “cmp ax,4756h”, press SPACE:



- How to test if this is the check?
- How to bypass the check?

Codebreaker Task 2: Bypass access limitation

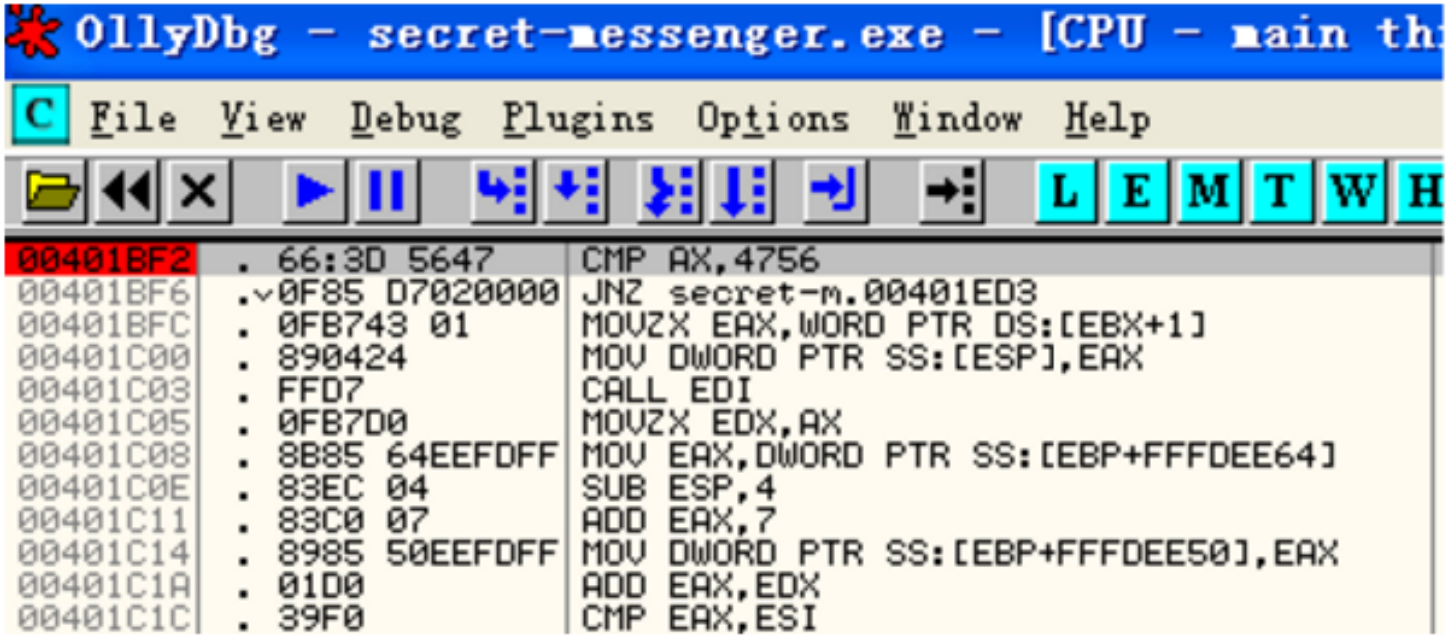
- In order to bypass this check as easily as possible, we can just modify the assembly code or change the specific flag during execution. Load the program with ollydbg:



Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 2: Bypass access limitation

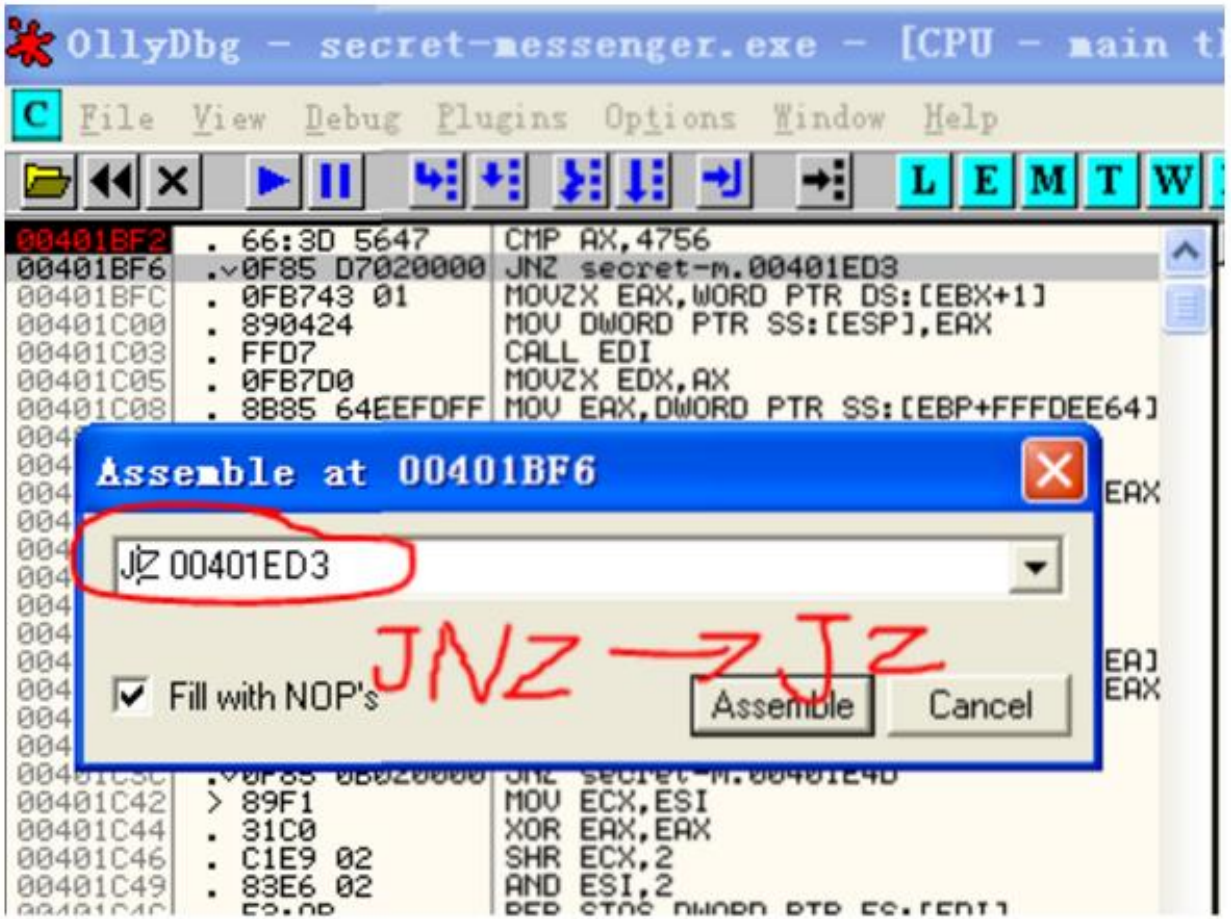
- Press CTRL+g go to the address 00401bf2, press F2 set breakpoint:



Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 2: Bypass access limitation

- Let's run the program and it will stop at this breakpoint, press F8 to run one more step and we modify the conditional JUMP instruction manually:



Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Task 2: Bypass access limitation

- Then, right click → copy to executable→ all modification, so we just saved our new program, let's try to run it:

```
E:\nsa2\modified>secret-messenger.exe --symbol tier2_key.pem --action tier2_msg.
txt --decoder
*****SIGNATURE IS VALID*****
Message: Our plans have been set into motion - Member number 392 is ready to car
ry out his tasking, and in 2 weeks time the window of opportunity will be open.
 If it is necessary to abort the action, the authentication code to use is 43moh
by6j8p7y32353mc.
*****SIGNATURE IS VALID*****

E:\nsa2\modified>
```

Adapted from content by Jiaming Li, NC State University, 2015

Codebreaker Tasks 3 and 4

- Task 3: Analyze decryption logic and develop a compatible encryption tool

- Task 4: Spoof messages so they appear to come from group leadership. Tell all recipients:
  "*Leadership has arranged a meeting with the local authorities…Meet at the city police station at 18:00. Be discreet, and come unarmed as to not draw attention.*" (LOL)

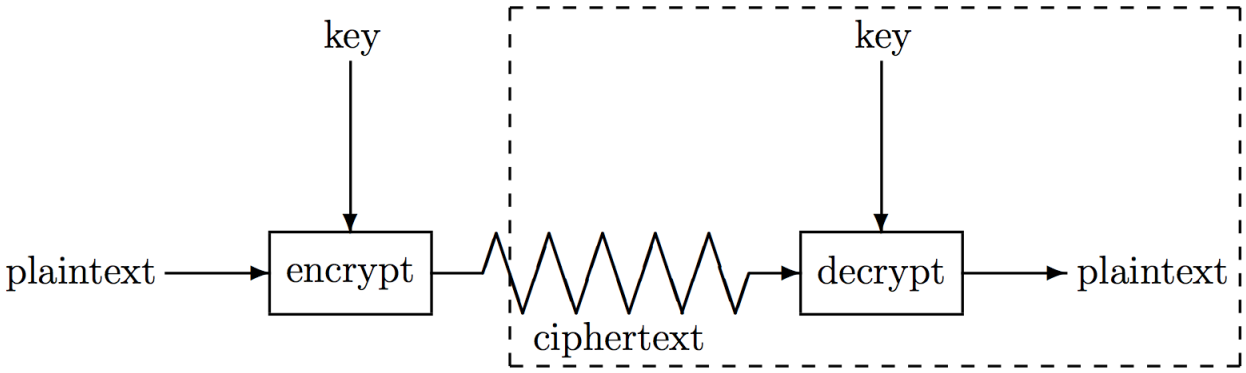
Adapted from content by Jiaming Li, NC State University, 2015

Anti-reverse engineering

- **Basics:**
  - Turn off debug symbols (-g)
  - Strip other symbols (e.g. “strip” tool on *NIX)
  - Consider static linking (no external calls to standard libraries to trace)
- **Anti-disassembly:**
  - Encrypted or self-modifying code
  - Code riddled with junk that is jumped over
    - Can especially confuse x86 assemblers due to variable-length instructions
- **Anti-debugging:**
  - Identify if debugger is in use (effects on real time, use of debug registers, etc.) and act differently
  - Use threads in complex ways to get less deterministic behavior
- **Tamper resistance:**
  - Hash parts of ones own code/data and verify periodically
  - The verification code is also code, though…
- **Obfuscation:**
  - Include lots of unreachable code to increase work the reverse engineer must do

DRM: Digital Rights Management

- Attempt to restrict what users can do with *data they have* on a *computer they own*
- Almost every implementation looks like this:



- Cu
- Pro

DRM deployment process