There are many many fuzzers

- American Fuzzy Lop
  - Mutation-based white-box fuzzer

- SPIKE
  - A library for creating network-based fuzzers

- Burp Intruder
  - Automates customized attacks against web apps

- And many more… (BFF, Sulley, …)
You fuzz, you crash. Then what?

Try to find the **root cause**

Is there a smaller input that crashes in the same spot? (Make it easier to understand)

Are there multiple crashes that point back to the same bug?

Determine if this crash represents an **exploitable vulnerability**

In particular, is there a buffer overrun?

---

**Finding memory errors**

1. **Compile** the program with **Address Sanitizer (ASAN)**
   - Instruments accesses to arrays to check for overflows, and use-after-free errors
   - [https://code.google.com/p/address-sanitizer/](https://code.google.com/p/address-sanitizer/)

2. **Fuzz it**

3. Did the program **crash with an ASAN-signaled error**? Then worry about exploitability

   - Similarly, you can **compile with other sorts of error checkers** for the purposes of testing
     - E.g., `valgrind memcheck` [http://valgrind.org/](http://valgrind.org/)
Writing and testing for secure code

- Know the systems and libraries you use
  - `printf("%n");` !!

- Assume the worst; code and design defensively
  - Think defensive driving

- Use **pen testing** tools to help automate
  - Assume that attackers will; seek to have at least as much information as they!

Now…

Continuing with Software Security

Required reading:
“StackGuard: Simple Stack Smash Protection for GCC”

Optional reading:
“Basic Integer Overflows”
“Exploiting Format String Vulnerabilities”

Malware

Malicious code that is stored on and runs on a victim’s system

• How does it get to run?
  • Attacks a user- or network-facing vulnerable service
    - E.g., using techniques you learned the past couple weeks
  • Backdoor: Added by a malicious developer
  • Social engineering: Trick the user into running/clicking/installing
  • Trojan horse: Offer a good service, add in the bad
  • Attacker with physical access downloads & runs it

Potentially from any mode of interaction (automated or not), provided sufficient vulnerability

Malware

What can it do?

• Virtually anything, subject only to its permissions
  • Brag: “APRIL 1st HA HA HA HA YOU HAVE A VIRUS!”
  • Destroy:
    - Delete/mangle files
    - Damage hardware (more later this lecture)
  • Crash the machine, e.g., by over-consuming resources
    - Fork bombing or “rabbits”: while(1) { fork(); }
  • Steal information (“exfiltrate”)
  • Launch external attacks
    - Spam, click fraud, denial of service attacks
  • Ransomware: e.g., by encrypting files
  • Rootkits: Hide from user or software-based detection
    - Often by modifying the kernel
    - Man-in-the-middle attacks to sit between UI and reality
Malware

When does it run?

• Some delay based on a trigger
  • **Time bomb**: triggered at/after a certain time
    - On the 1st through the 19th of any month…
  • **Logic bomb**: triggered when a set of conditions hold
    - If I haven’t appeared in two consecutive payrolls…
  • Can also include a **backdoor** to serve as ransom
    - “I won’t let it delete your files if you pay me by Thursday…”

• Some attach themselves to other pieces of code
  • **Viruses**: run when the user initiates something
    - Run a program, open an attachment, boot the machine
  • **Worms**: run while another program is running
    - No user intervention required

Self-propagating malware

• **Virus**: propagates by arranging to have itself *eventually* executed
  • At which point it creates a new, additional instance of itself
  • Typically infects by altering *stored* code
  • User intervention required

• **Worm**: *self*-propagates by arranging to have itself *immediately* executed
  • At which point it creates a new, additional instance of itself
  • Typically infects by altering *running* code
  • No user intervention required

The line between these is thin and blurry
Some malware uses both styles
Technical challenges

• **Viruses: Detection**
  • Antivirus software wants to detect
  • Virus writers want to avoid detection for as long as possible
    • *Evade* human response

• **Worms: Spreading**
  • The goal is to hit as many machines and as quickly as possible
    • *Outpace* human response

Viruses
Viruses

• They are **opportunistic**: they will eventually be run due to user action

• Two *orthogonal* aspects define a virus:
  1. How does it **propagate**?
  2. What else does it do (what is the “**payload**”)?

• General infection strategy:
  • Alter some existing code to include the virus
  • Share it, and expect users to (unwittingly) re-share

• Viruses have been around since at least the 70s

---

**How viruses infect other programs**

Entry point

Original program

Entry point

Original program

Virus

Take over the entry point
Viruses are classified by what they infect

- **Document viruses**
  - Implemented within a formatted document
  - Word documents (very rich macros)
  - PDF (Acrobat permits javascript)
  - (Why you shouldn’t open random attachments)

- **Boot sector viruses**
  - Boot sector: small disk partition at a fixed location
  - If the disk is used to **boot**, then the firmware loads the boot sector code into memory and runs it
  - What’s **supposed** to happen: this code loads the OS
  - Similar: AutoRun on music/video disks
  - (Why you shouldn’t plug random USB drives into your computer)

- **Memory-resident viruses**:
  - “Resident code” stays in memory because it is used so often

---

**Viruses have resulted in a technological arms race**

The key is **evasion**

Mechanisms for **evasive propagation**

Want to be able to claim wide coverage for a long time

Mechanisms for **detection and prevention**

Want to be able to claim the ability to detect many viruses
How viruses propagate

• First, the virus looks for an **opportunity to run**. *Increase chances* by attaching malicious code to something a user is likely to run
  • autorun.exe on storage devices
  • Email attachments

• When a virus runs, it looks for an **opportunity to infect** other systems.
  • User plugs in a USB thumb drive: try to overwrite autorun.exe
  • User is sending an email: alter the attachment
  • Viruses can also proactively create emails ("I Love You")

Detecting viruses

• **Method 1: Signature-based detection**
  • Look for bytes corresponding to injected virus code
  • Protect other systems by installing a recognizer for a known virus
  • In practice, requires fast scanning algorithms

• This basic approach has driven the multi-billion dollar antivirus market

• #Recognized signatures is a means of **marketing** and competition
  • But what does that say about how important they are?
Um.. thanks?

FEATURE

Antivirus vendors go beyond signature-based antivirus

Robert Westervelt, News Director

This article can also be found in the Premium Editorial Download "Information Security magazine: Successful cloud migrations require careful planning." Download it now to read this article plus other related content.

Security experts and executives at security vendors are in agreement that signature-based antivirus isn't able to keep up with the explosion of malware. For example, in 2009, Symantec says it wrote about 15,000 antivirus signatures a day; that number has increased to 25,000 antivirus signatures every day.

"Signatures have been dying for quite a while," says Mikko H. Hypponen, chief research officer of Finnish-based antivirus vendor, F-Secure. "The sheer number of malware samples we see every day completely overwhelms our ability to keep up with them."

Security vendors have responded by updating their products with additional capabilities, such as file reputation and heuristics-based engines. They're also making upgrades to keep up with the latest technology trends, such as virtualization and cloud computing.
You are a virus writer

• Your goal is for your virus to spread far and wide

• How do you avoid detection by antivirus software?
  1. **Give them a harder signature to find**

---

**How viruses infect other programs**

- **Entry point**
  - **Original program**

- **Entry point**
  - **Virus**
    - **Original program**
    - **j**mp

- **Entry point**
  - **Original program**
  - **j**mp

- **Entry point**
  - **Original program**
    - **etc.**

- **“Appending”**
- **“Surrounding”**
- **Confuse scanners**
- **Overwrite uncommonly used parts of the program**
You are a virus writer

• Your goal is for your virus to spread far and wide

• How do you avoid detection by antivirus software?
  1. Give them a harder signature to find
  2. Change your code so they can’t pin down a signature

Mechanize code changes:

Goal: every time you inject your code, it looks different

Polymorphic viruses
Polymorphic viruses

When used properly, encryption will yield a different, random output upon each invocation.

Crypting services

Iteratively obfuscate the code (encrypt + jmp + …)

Until the obfuscated code is “fully undetectable”

2013: Web-based crippling services
One charged $20 to “remain undetected for more than 7 days”
Polymorphic viruses: Arms race

Now you are the antivirus writer: how do you detect?

• Idea #1: Narrow signature to catch the decrypter
  • Often very small: can result in many false positives
  • Attacker can spread this small code around and jmp

• Idea #2: Execute or statically analyze the suspect code to see if it decrypts.
  • How do you distinguish from common “packers” which do something similar (decompression)?
  • How long do you execute the code??

Now you are the virus writer again: how do you evade?

Polymorphic countermeasures

• Change the decrypter
  • Oligomorphic viruses: change to one of a fixed set of decrypters
  • True polymorphic viruses: can generate an endless number of decrypters
    - e.g., brute force key break
    - Downside: inefficient
Metamorphic code

• Every time the virus propagates, generate a *different version* of the code
  • Code *semantically* the same
  • But it works slightly differently
    - Different machine code instructions
    - Different algorithms to achieve the same thing
    - Different use of registers
    - Different constants….

• How would you do this?
  • Include a code rewriter with your virus
  • Add a bunch of complex code to throw others off (then just never run it)
ZPerrx can directly reorder the instructions in its own code.

Figure 7: Zperrx inserts JMP instruction into its code

---

a. An early generation:

```
C7060F000055  mov  dword ptr [esi],55000000h
C746048BEC5151  mov  dword ptr [esi+0004],5151EC8Bh
```

b. And one of its later generations:

```
BF0F000055  mov  edi,55000000h
893E  mov  [esi],edi
5F  pop  edi
52  push  edx
B640  mov  dh,40
BA8BEC5151  mov  edx,5151EC8Bh
53  push  ebx
8BDA  mov  ebx,edx
895E04  mov  [esi+0004],ebx
```

c. And yet another generation with recalculated ("encrypted") "constant" data.

```
BB0F000055  mov  ebx,55000000h
891E  mov  [esi],ebx
5B  pop  ebx
51  push  ecx
B9CB00C05F  mov  ecx,5FC000C0h
81C1C0BB91F1  add  ecx,F191EC0h ; ecx=5151EC8Bh
899E04  mov  [esi+0004],ecx
```

Figure 6: Example of code metamorphosis of Win32/Exe
Polymorphic

When can AV software successfully scan?
Figure 8: A partial or complete snapshot of polymorphic virus during execution cycle

Metamorphic

When can AV software successfully scan?
Figure 10: T-1000 of Terminator 2