INTERNET SECURITY SYSTEMS

# Trust No-one, Not Even Yourself OR The Weak Link Might Be Your Build Tools David Maynor Research Engineer

**ISS X-Force R&D** 

## Thank god my source tree is safe! **CINTERNET SECURITY SYSTEMS** "Developers normally expect attacks against their code, just not while it is being built"

### • Simple security holes are becoming a thing of the past.

- Strcpy() and gets() problems are all but extinct.
- Heap overflows can make reliable compromise across platforms and patch levels hard.
- Increase in built-in stack protection.

# Thank god my source tree is safe! (cont)

- Developers becoming better educated, they can find their own "low hanging fruit."
  - Increased security awareness has forced developers to consider security in the design process.
  - More educated bug hunters lead to a higher discovery rate.

# Thank god my source tree is safe! (cont)

- New security technologies making remote attacks less likely to succeed.
  - Widespread use of IDS/IPS/firewall/gateway antivirus technologies
  - Stateful inspection and deep threat analysis technologies becoming commonplace
  - Remote attacks becoming less likely to succeed even with 0day
    - HTTP Proxies make things like connect back shells over port 80 less effective
    - NAT makes connecting directly to target machines harder

# Thank god my source tree is safe! (cont)

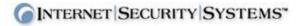
### • Where are the weak links in security now?

- Development is outsourced more
- Cost cutting is making strange bedfellows
- Open source projects are gaining more popularity in mission critical roles.

## My compiler? You MUST be joking! THERNET SECURITY SYSTEMS\* "The weak link might not be in you code content, but how you build it."

## Is it possible?

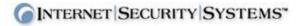
- Can attackers really backdoor code as it is being built?
- Yes, otherwise this would be a boring speech
- Will it be noticed?
  - Depends on the payload
  - Different affects on different file formats
  - Subtle OS changes like patching can break it



# My compiler? You MUST be joking! (cont)

#### Is it easy?

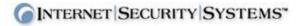
- No. This is a very complex attack.
- Requirements before one could even hope to succeed
  - Access to build machine
  - Expert knowledge of compiler and output file format
  - Expert creation of payload
    - Payload is the code that is being added, this can range from shell access to remote tracking



# My compiler? You MUST be joking! (cont)

#### • What can the results of an attack like this yield?

- Email encryption program
  - A copy of the plaintext is saved during creation of the ciphertext.
  - A different key is used that the intended
- SSL
  - Weaken server keys
  - Allow for sniffing of ssl communications
- Banking application
  - Create secret store of personal information
  - Transmission of information to 3<sup>rd</sup> parties
- Kernel
  - Allow for unauthorized elevated privileges
  - Allow process to be hidden from sysadmins and users



# My compiler? You MUST be joking! (cont)

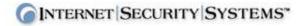
#### How portable is this?

- Across operating systems?
  - Win32 vs. linux vs. \*nix
    - Depends on the actual payload
    - More than likely not
- Across file formats?
  - PE vs. ELF vs. COFF
    - This depends on where the payload is hidden
    - More than likely not
- Across architectures?
  - RISC vs. CISC
    - This depends on how the payload is encoded.
    - More than likely not

I use gcc, can I be affected by this? INTERNET SECURITY SYSTEMS\* "Open source tools may appear to be easy but still present a challenge."

### A brief overview of gcc.

- Where does it come from? Who writes it?
  - http://gcc.gnu.org
  - 1.0 released May 23, 1987
  - Current version (as of writing) 3.4.0
  - Written by the Free Software Foundation
- What is it?
  - More of a suite than a single tool.
    - Supports C, C++, Objective-C, java, ada, fortran frontends
    - List of backend support at http://gcc.gnu.org/backends.html



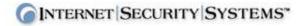
#### What does gcc actually do to code?

- Phases of compiling
- Points where gcc modifies original code
- Optimizations

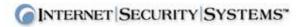


#### How can an attacker use this to their advantage?

- Best Places to attack?
  - \_start
    - glibc-2.3/sysdeps/i386/elf/start.S
    - It set up initial environment variables
    - Sets up command line arguments
    - Calls main()
- Analysis of frontend/backend for attack points
  - Things to consider
    - Breaking the program
    - compatibility



- The payload
  - C code
  - X86 asm
  - "shellcode"

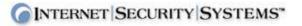


### EXAMPLE: Linking fun

 Add a stub to \_start to call code in object file that is automatically added by a trojaned linker.

### EXAMPLE: \_start fun

 Code added to \_start that creates a single udp packet every time the program is run.



## My compiler is not open source, I must be safe...right?

"How to trojan a compiler you do not have the source for..."

### Visual Studio 6.0

- Written by Microsoft
- Integrated development environment, compiler, assembler, linker.
- Used for windows development only, no cross compiling abilities.

### Weak links?

- crt0.c
  - From the comments at the beginning of the file: "This is the actual startup routine for apps. It calls the user's main routine [w]main() or [w] WinMain after performing C Run-Time Library initialization."
  - Its in C, does not require asm to craft a payload.



### My compiler is not open source, I must be safe...right? (cont)

- Payload code:
  - EXAMPLE: code in C++
  - EXAMPLE: code is asm
  - EXAMPLE: Adding code before main() or winmain()



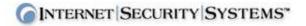
### I use an obscure compiler, I MUST be safe! "Auditing less popular compilers for attack points."

### • LCC

- http://www.cs.princeton.edu/software/lcc/
- Covered in book "A Retargetable C Compiler "
  - Awesome book
  - Overheard at party "It's the new dragon book"
- Popular for learning compiler internals

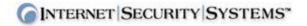
### How it differs from Visual Studio and gcc

- Less popular, not often used for mission critical apps
- Less optimazations

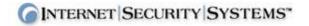


#### Binary analysis

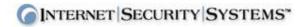
- Best way to learn about something is use it:
  - Build simple "hello world" program with Icc



- Use nm to examine symbols created by lcc

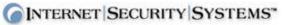


- Use objdump to examine code generated by compiler



### How to interpert your findings.

- Determining what the compiler does to the code
- Finding stuff you didn't write
- Finding where the compiler stores its code



# Thankfully there are only basic attacks!!

"Aside from simple code injection, what else could be done?"

#### Advance attack methods

- Adding code to getopt()
- Replacing safe functions with unsafe versions

### Dependent attack

- Do nothing if DEBUG is defined
- Only attack if there if it is a socket app
- Only attack if it is a setuid app

# Thankfully there are only basic attacks!! (cont)

EXAMPLE: bye-bye bounds checking

# Thankfully there are only basic attacks!! (cont)

 Tools compilers work with and how they can turn against you!

- Linker
- Assembler
- Libtool
- ar

# Other than 0wn1ng things, is this useful?

"There are often better ways to do these things, but in case of last resort, they work."

### Tracking code

- Every binary built with the compiler has a machine specific hash added for better forensics.
- Every binary built has code added that creates a UDP packet that is sent to an arbitrary address.
  - Useful for honeypots
  - Internal apps that should not leave a company



## How do I detect this?

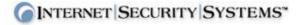
"Creating the problem is easy, creating the solution is...not."

#### Stack backtrace

- Standard library code should look the same
- Backtrace comparison of ELF bin should yield same known good results.

### Signatures for compiler operations

- Optimizations
- standard functions
- Step by step verification of code at runtime



## Thanks!!

- This speech was inspired by Ken Thompson's excellent piece for the ACM: *Reflections on Trusting Trust*.
  - http://www.acm.org/classics/sep95/