

CS5412: DANGERS OF CONSOLIDATION

Lecture XXIII

Ken Birman

Are Clouds Inherently Dangerous?

2

- Gene Spafford, famous for warning that the emperor has no clothes fears that moving critical information to the cloud could be a catastrophe
- His concern?
 - ▣ Concentration of key resources creates a “treasure chest” that adversaries can focus upon and attack
 - ▣ Risk of a virus spreading like wildfire
- Core issue: Clouds create *monocultures*



What Constitutes a “Monoculture”?

3

monoculture: An environment in which the predominance of systems run apparently identical software components for some or all services.

- ▣ Such systems share vulnerabilities, hence they are at risk to rapid spread of a virus or other malware vector.



Cloned babies

Cloned plants



Forms of monocultures

4

- Large numbers of instances of identical programs or services (includes applications, not just the O/S)
- Wide use of the same programming language or scripting tool
- Any standard defines a kind of monoculture

Current example: OpenSSL

5

- SSL (renamed Transport Layer Security: TLS) is a standard used to negotiate security keys for secure TCP communication
 - ▣ Involves use of keys from certificate authorities to encrypt communication, including passwords
 - ▣ Used for connections to https websites
- Issue: OpenSSL was an open source effort
 - ▣ And open development: anonymous contributors
 - ▣ One of those contributors introduced a bug in ~2012

What was the bug?

6

- OpenSSL has a heart beat protocol
 - “If you are still there, send me XX bytes to prove it”
 - Normally XX was small, like 16, but the client could actually specify the value **YELLOW SUBMARINE**
 - With big values a buffer read overrun caused OpenSSL to corrupt memory...
- And, in that case, the data was decrypted

```
(uint8_t) [
00000000 02 00 79 68 65 61 72 74 62 6c 65 65 64 2e 66 69 | ..yheartbleed.FL|
00000010 6c 69 70 70 6f 2e 69 6f 59 45 4c 4c 4f 57 20 53 | ll,pp,loYELLOW S|
00000020 55 42 4d 41 52 49 4e 45 39 dc dc 5c 69 e8 80 17 | UBMARINE9...\i...|
00000030 ab 05 6f 8e 51 aa a7 33 20 19 9f 48 29 95 2e 00 | ...o,Q...3...H)...|
00000040 05 00 05 01 00 00 00 00 00 0a 00 08 00 05 00 17 | .....|
00000050 00 18 00 19 00 0b 00 02 01 00 08 0d 00 0a 00 08 | .....|
00000060 04 01 04 03 02 01 02 03 ff 01 00 01 00 25 32 32 | .....NZZI|
00000070 25 32 43 25 32 32 64 61 74 61 25 32 06 0d fe d0 | %2CX22datdK2...|
00000080 73 49 14 1a e5 bb cd 58 89 0f b7 53 | aI....X...SI|
}
```

Central lesson learned?

7

- In the cloud community, majority solutions often dominate and become de-facto standards
- Everyone then uses them: They are “presumed to be the best (because widely used), hence widely used...
- And if one of those shared elements is buggy, every system using them is at risk of compromise

Taking the larger view

Three categories of attack

□ **Configuration attacks.**

- Exploit aspects of the configuration. Vulnerability introduced by system administrator or user who installs software on the target.
- Includes compiling SNDMAIL with the back door enabled

□ **Technology attacks.**

- Exploit programming or design errors in software running on the target. Vulnerability introduced by software builder.
- Here hacker breaks in via buggy code

□ **Trust attacks.**

- Exploit assumptions made about the trustworthiness of a client or server. Vulnerability introduced by system or network architect.
- Hacker abuses legitimate access, like a hospital worker who peeks at Lindsey Lohan's medical records

Monoculture: A defense for configuration attacks.

A carefully constructed, fixed, system configuration would be an effective defense against configuration attacks.

- ❑ System configuration (today) is hard to get right and thus is best done by experts. Having one or a small number of “approved” configurations would allow that.
- ❑ Configuration attacks are considered “low hanging fruit” and thus likely are the dominant form of attack today.
- ❑ Configurations change not only because a system administrator installs software but also from a user visiting web sites or interacting with web services that cause software downloads.
 - To rule-out such downloads could be a serious limitation on system functionality. Such downloads often bring vulnerabilities, though.

So monocultures help... for one case

10

- Question becomes: what percent of attacks leverage configuration mistakes?
 - ▣ ... *nobody knows!*
 - ▣ But gray-hat hackers assure us that things like standard passwords are a very common problem

Viruses love monocultures

11

- Earliest Internet Worm was launched at Cornell!
 - ▣ A brief episode of notoriety for us
 - ▣ Worm exploited variety of simple mechanisms to break into computer systems, then used them as a springboard to find other vulnerable systems and infect them
 - ▣ It had a simple trick to prevent itself from reinfecting an already infected system: checked for a “lock” file
 - But even if present, reinfected with a small probability
 - Idea was to jump back onto systems that might have been fixed by system admin team but who left the lock in place

Monocultures are a known risk

12

- Vast majority of computer viruses and worms operate by exploiting software bugs
 - ▣ For example, failure to check boundaries on arrays
 - ▣ Very common in code written in C++ or C because those languages check automated boundary checks
 - ▣ Nothing stops an input from overrunning the end of the array
- What lives beyond the end of an array?



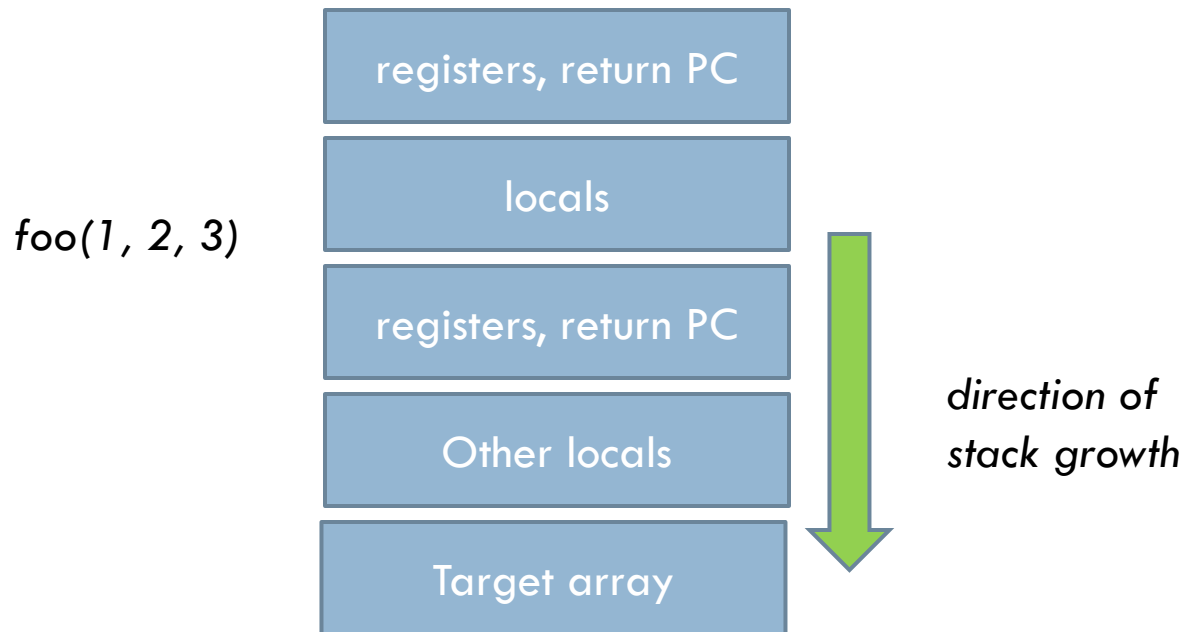
Beyond the end...

13

- Two cases to consider
 - ▣ Array is on the stack (local to some active method)
 - ▣ Array is in the program's data or BSS area, or was allocated from the heap

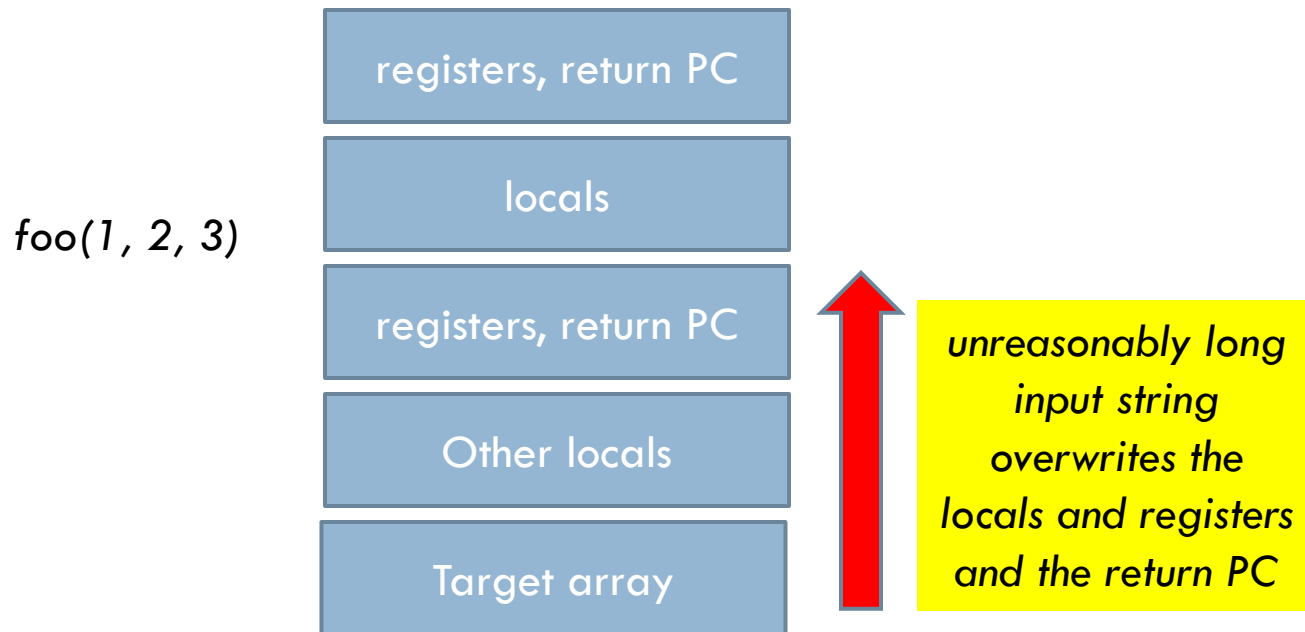
Stacks grow “downwards...”

14



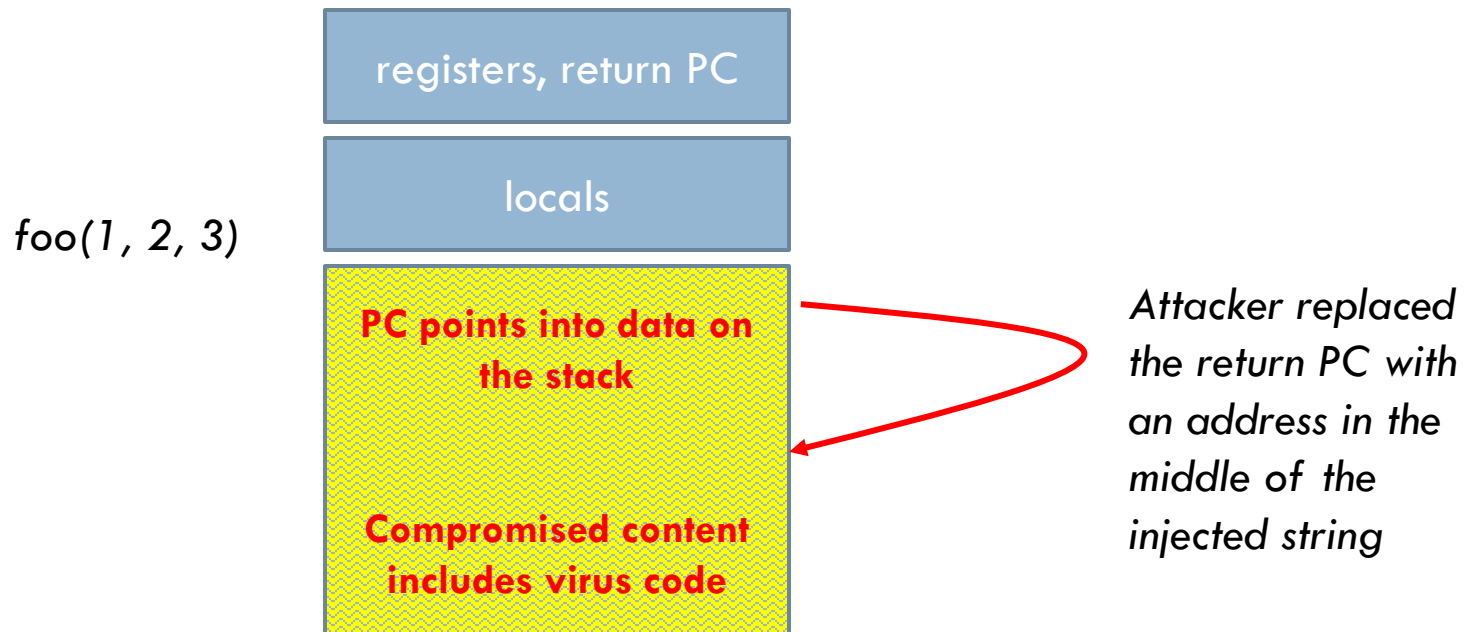
Stacks grow “downwards...”

15



Stacks grow “downwards...”

16



Why does this attack work?

17

- Attacker needs to be able to predict
 - Where the target string lives in memory
 - How the stack is arranged
 - What the code that reads the string will do

- Trick is to get the code to jump into the data read from the attacker

Bootstrapping concept

18

- The hacker doesn't have much "room" for instructions
- So typically this logic is very limited: often just code to read a longer string from the network and then execute that longer code
 - ▣ In effect, the initial attack is a bootstrap program
 - ▣ It loads and launches a more serious program

Example

19

- String loads code that simply allocates a much bigger object, reads from the same input source into it, and jumps to the start
- Allows the attacker to send a multi-GB program that would be way too large to “fit” within the stack
 - ▣ Trick is to take over but not trigger exceptions
 - ▣ If the attack causes the program to throw an exception, someone might notice

What about data/heap?



20

- Here attacker might be in a position to overwrite other adjacent variables on which the program is dependent
 - ▣ This does assume some “predictability” in memory layout!
 - ▣ We could perhaps replace a filename it reads or one it writes with filenames the attacker would prefer that it use instead, or with network URLs
 - ▣ Of course the program will now be a very sick puppy but it might last just long enough to do the I/O for the attacker
 - ▣ That I/O becomes a “point of leverage” that the attacker exploits like the first domino in a long line...

Example “attack opportunity”

21

- Any program that works with strings in C or C++ is at risk even if we length-check inputs

```
void unsafe(char *a, char *b)
{
    char tmp[32];
    strcpy(tmp, a);
    strcat(tmp, b);
    return(strcmp(tmp, "foobar"));
}
```

- Problem here isn't with the input length per-se but with the assumption in “unsafe” that the combined string fits in tmp

Why not just fix the compiler?

22

- People have modified C to check array bounds
 - ▣ This only helps in limited ways
- C and C++ and Fortran are *unsafe by design* because of pointer aliasing
 - ▣ They let us treat an object of one type as if it was of some other type
 - ▣ And they impose no real boundary checking at all
- Fixing the language would break many programs that are in wide use: we would need to fix them too

Other examples of attacks

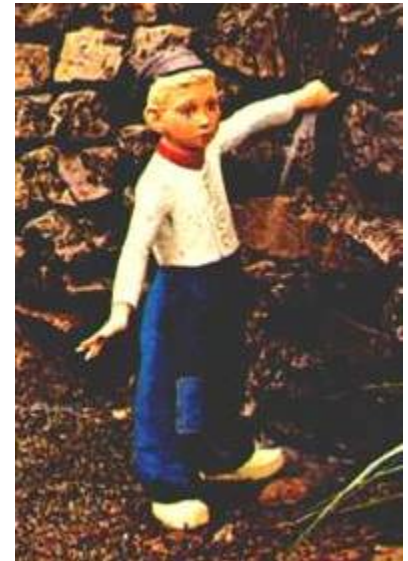
23

- ❑ Back doors, such as debug or maintenance features
- ❑ Passwords left with their default values
- ❑ Automated file or patch download features that can be tricked into overwriting system files
- ❑ Code that has built-in features that can be misused to trick the program into executing unusual logic
- ❑ ... in fact the list is really endless!

Broader problem

24

- We simply don't have a good way to create things that are correct, by construction, ground up
 - ▣ Lacking those, trying to find problems in existing code is like trying to plug a leak in a dam
- At best we can prove properties of one thing or another but the assemblage invariably has holes!
 - ▣ Or they sneak in over time



Cloud “permissiveness”

25

- Anyhow, it makes no sense to imagine that we would tell people how to build cloud applications
- With EC2 we just hand Amazon an executable
 - ▣ How will it know if the binaries were compiled using the right compiler?
 - ▣ What if the version of the compiler matters?
 - ▣ Generally not viewed as a realistic option
- In fact when C and C++ run on .NET many of these overflow issues are caught, but “managed” C or C++ will reject all sorts of classic programs as buggy

How to attack a cloud

26

- A good firewall can block many kinds of attacks
- But something will get through eventually, we can't avoid every possible risk and close every possible virus exploit
- And once the virus breaks in, it compromises every single accessible instance of the same code

What can we do about these issues?

27

- Today: Focus on these kinds of viral attacks
- Thursday: Look at the bigger picture

First, let's stop the stack attack...

28

- How can we do that?
 - ▣ The attacker is taking advantage of knowledge of the program behavior and flaws
 - ▣ An “unpredictable” program would have crashed but not been so easy to compromise
 - ▣ Can we take a program written in C or C++ and make it behave less predictably without causing it to crash?

Stack randomization

29

- Idea is simple:
 - ▣ Modify the runtime to randomly allocate chunks of memory (unpredictable size) between objects on stack
 - ▣ We can also add a chunk of unpredictable size to the bottom of the stack itself

- Attacker countermeasures?
 - ▣ May be possible to use a “block” of jump instructions, no-ops to create code that can run in a “position independent manner”
 - ▣ Or might guess the offset and try, try again... If the datacenter doesn't notice the repeated crashes a few hundred tries might suffice to break in

.NET has *automated* diversity

30

- If enabled, a wide variety of randomization mechanisms will be employed
- Just a bit in the runtime environment you can set
- But important to retest programs with stack randomization enabled
 - ▣ Some programs *depend* on bugs, other issues!

More recent work on diversity

31

- Diverse OS can scramble the number of system calls
- Placement of segments in memory can be varied
- Code can be dynamically relinked to reorder the placement of compiled code and data elements

But this can't stop all attacks

32

- For example, database “code injection” attacks have a similar approach and yet don't rely on array overflow:
 - Intended code
 - `SELECT * FROM users WHERE name = '' + userName + '';`
 - Limits query to data for this user
 - Attacker sends a “faulty” name argument:
 - `' or '1'='1`
 - `SELECT * FROM users WHERE name = ` ' or '1'='1;`
- There are many examples of this kind because many programs exchange messages that involve application-specific programming languages

Blocking SQL query injection?

33

- This is easy:
 - ▣ Read the input
 - ▣ Then “clean it up”
 - ▣ Then pass it in to the application

- As long as the developer uses the right tools these issues don't arise
 - ▣ But not every developer cooperates

Other ideas: Castro and Costa

34

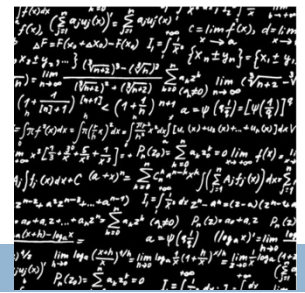
- One project at Microsoft monitors program crashes
 - ▣ Each time a crash happens they look to see what input caused the program to fail
 - ▣ In one project they create virus “signatures”
 - ▣ In another they automatically combine these to create a pattern, more and more selective, for blocking the input strings that cause the problem
 - ▣ Use gossip, rapidly and robustly disseminate the fix together with a “proof” of the bug that triggers it

Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham, Vigilante: End-to-End Containment of Internet Worms, in ACM Symposium on Operating Systems Principles (SOSP), Brighton, UK, Oct 2005

OSDI, Spring 2005

What kind of “proof”?

35

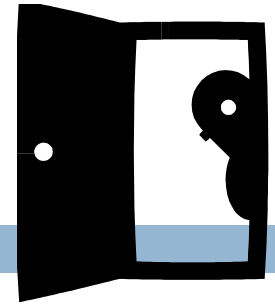


- Before installing a patch, verify that problem is real
 - ▣ Proof: Example of an input that will cause a crash or some other form of compromise
 - ▣ Verification: Try it inside a virtual machine

- One issue: if the filter is too broad, it might block legitimate inputs that wouldn't cause a crash

- We want to block the attack but not legitimate users

Back door attacks



36

- Some attacks don't actually compromise a program
 - ▣ For example, the early Internet worm operated by exploiting a feature in the original SNDMAIL program
 - ▣ Code was written by Eric Allman and was unstable for the first few years
 - So he needed ways to see what the problem was
 - Included a debug feature allowing him to use SNDMAIL as a kind of remote FTP program to access files on remote system... and SNDMAIL runs with elevated priority...
 - Internet worm used this “feature” as one of its attack vectors

Stack diversity doesn't stop these...

37

- Backdoor attacks use legitimate features of a program, or perhaps debug features, to ask program to do things it was programmed to do!
 - ▣ The program isn't really malfunctioning or compromised
 - ▣ But it still does things for us that allow breakin
 - ▣ For example, can use SNDMAIL to copy a modified program on top of /etc/init in Linux
 - ▣ This modified program might work normally, but always allow logins from Evil.Hacker with password "Gotcha"
 - ▣ Better compiler won't help...

Neither would better checking tools

38

- A back door is a problem with the specification
 - ▣ The program shouldn't have functionality that replaces arbitrary files with code downloaded from the network, or copied from other places, or even with code "created" within the program itself
 - ▣ Yet it is very hard to pin down the rules we need to check to achieve confidence!

The ultimate back door

Who is this man?

Do you trust his
Software?



The ultimate back door

Ken Thompson

Co-Creator of
UNIX and C

Turing Award: 1983

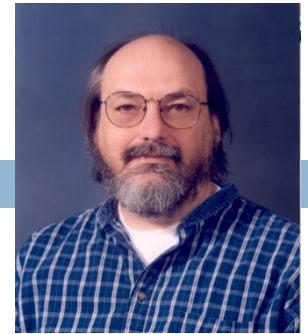


The ultimate back door

41

- Ken Thompson discussed hidden back doors in a famous Turing Award lecture
 - ▣ He considered the Unix login program
 - ▣ Showed how a *macro substitution* could insert a back door
 - ▣ Then pointed out that the macro preprocessor could have a back door that does the macro substitution
 - ▣ Then he applied this to the macro preprocessor itself
 - ▣ Ended up with a vanilla-looking Unix system that would always allow him to log in but where those lines of code could only be discovered by examining the byte code

The ultimate back door



42

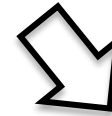
```
#define DESCOFF (6)
#define VALOFF (8)
#define STABSIZE (12)

/* Print ABFD's stabs section STABSECT_NAME (in `stabs'),
   using string table section STRSECT_NAME (in `strtab'). */

static void
print_section_stabs (abfd, stabsect_name, strsect_name)
    bfd *abfd;
    const char *stabsect_name;
    const char *strsect_name;
{
    int i;
    unsigned file_string_table_offset = 0, next_file_string_table_offset = 0;
    bfd_byte *stabp, *stabs_end;

    stabp = stabs;
    stabs_end = stabp + stab_size;

    printf ("Contents of %s section:\n\n", stabsect_name);
    printf ("Symnum n_type n_othr n_desc n_value n_strx String\n");
    -uu-:---F1  objdump.c      68% (1825,0)  (C/l Abbrev Fill)-----
```

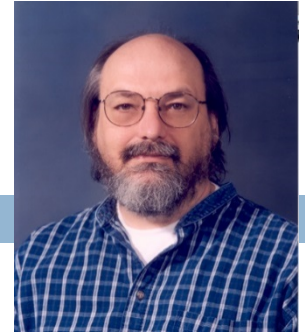


Compiler



011001001111010

The ultimate back door



43

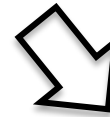
```
#define DESCOFF (6)
#define VALOFF (8)
#define STABSIZE (12)

/* Print ABFD's stabs section STABSECT_NAME (in `stabs'),
   using string table section STRSECT_NAME (in `strtab'). */

static void
print_section_stabs (abfd, stabsect_name, strsect_name)
    bfd *abfd;
    const char *stabsect_name;
    const char *strsect_name;
{
    int i;
    unsigned file_string_table_offset = 0, next_file_string_table_offset = 0;
    bfd_byte *stabp, *stabs_end;

    stabp = stabs;
    stabs_end = stabp + stab_size;

    printf ("Contents of %s section:\n\n", stabsect_name);
    printf ("Symnum n_type n_otrh n_desc n_value n_strx String\n");
    -uu-:---F1  objdump.c      68% (1825,0)  (C/l Abbrev Fill)-----
```



Compiler



011001001111010

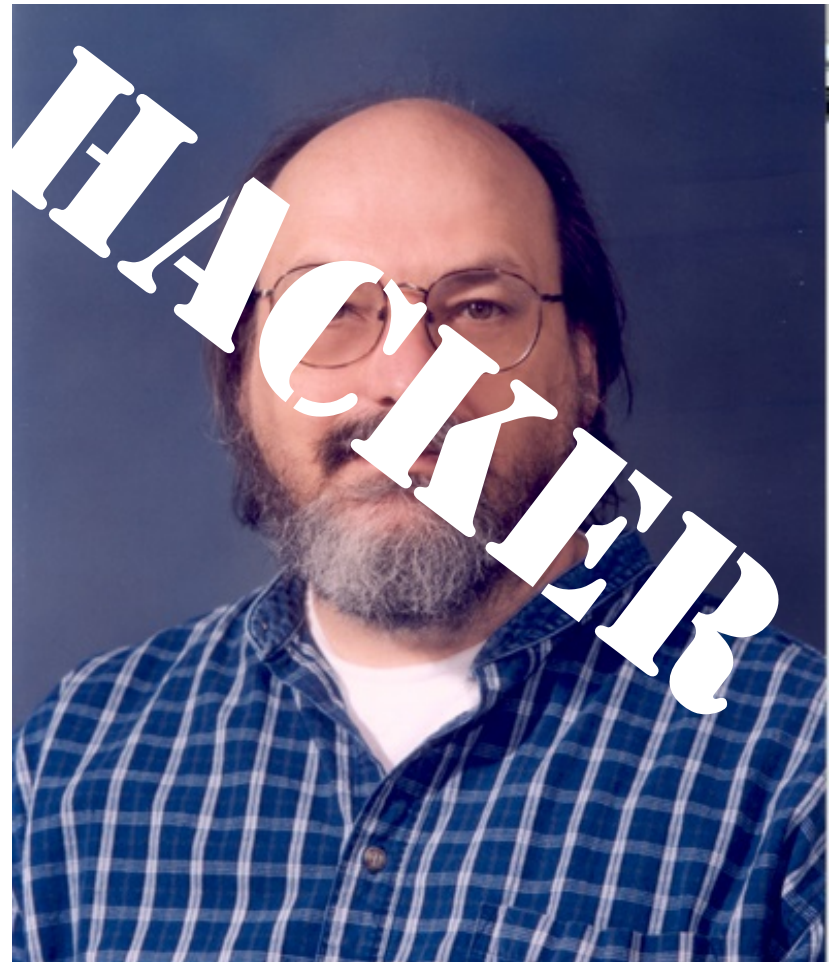
```
...
if(program == "login")
    add-login-backdoor();
if(program == "compiler")
    add-compiler-backdoor();
```

The ultimate back door

Ken Thompson

Co-Creator of
UNIX and C

Turing Award: 1983



The ultimate back door

45

- In general, covert “virtualized” platforms lurk in many settings
 - ▣ Virus could virtualize your machine
 - ▣ Attacker with serious resources could sneak a monitoring component into your printer or the disk drive itself
 - ▣ Even the network could potentially “host” a covert computing device and its own stealth network!
- Very hard to really secure modern computing systems. Cloud actually helps because many operators have resources to build their own specialized hardware

Recent concern

46

- Even common devices can have backdoors
 - ▣ Line printers often have a wide variety of network APIs
 - ▣ Network routers and firewalls
 - ▣ The virtualization platforms that operate the cloud
 - ▣ Web browsers...

- We seem to be surrounded by insecure components

What about virtualization as a tool?

47

- By running the user's code in a virtual machine the cloud gives us a way to firewall the user from other users
 - ▣ We share a machine but I can't see your work and you can't see mine
 - ▣ Virtualization code needs to block things like putting the network into promiscuous mode ("monitoring" mode)
 - ▣ Forces us to trust the VM hypervisor and the hardware that supports virtualization, but gives "containment"
- Now a virus can only harm the user that "let it in"

Other forms of diversity

48

- Run different products that offer equivalent functionality, like two versions of an email server
 - ▣ Strange finding: researchers have shown that for many applications, even versions created separately share bugs!
- Consider morphing the system calls: code would need to be compiled on a per-instance basis but would protect against attacks that require attacker to know local system call numbering
- Vary thread scheduling order dynamically

Combining multiple methods

- This is sometimes called “defense in depth”
- The first line of defense is the dynamically managed firewall: ideally, attack won't get in
 - ▣ But if it does, randomization has some chance of defeating the attack one step later
 - ▣ Each new obstacle is a hurdle for the attacker
- Will this stop attacks? Only simple ones... but most attacks use simple methods!

Defense in depth

50



... but talented attackers still win

51



How can anyone trust the cloud?

52

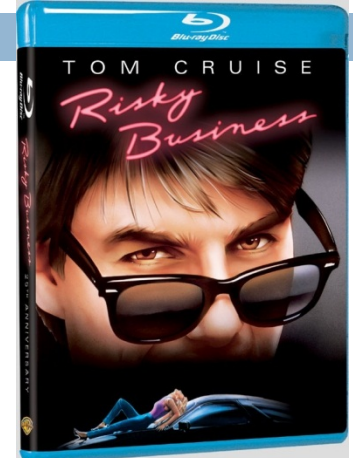
- The cloud seems so risky that it makes no sense at all to trust it in any way!
- Yet we seem to trust it in *many* ways
- This puts the fate of your company in the hands of third parties!



For all its virtues, the cloud is risky!

53

- Categories of concerns
 - ▣ Client platform inadequacies, code download, browser insecurities
 - ▣ Internet outages, routing problems, vulnerability to DDoS
 - ▣ Cloud platform might be operated by an untrustworthy third party, could shift resources without warning, could abruptly change pricing or go out of business
 - ▣ Provider might develop its own scalability problems
 - ▣ Consolidation creates monoculture threats
 - ▣ Cloud security model is very narrow and might not cover important usage cases



But the cloud is also good in some ways

54

- With a private server, DDoS attacks often succeed
 - ▣ In contrast, it can be hard to DDoS a cloud
 - ▣ The DDoS operator spends real money and won't want to waste the cash
 - ▣ Thus because cloud is hard to DDoS, cloud emerges as a very good response to DDoS worries

More good news

55

- Diversity can compensate for monoculture worries
- Elasticity is a unique capability not seen in other settings
- Ability to host and compute on massive data sets is very valuable
 - ▣ Obviously, only of value if task is suited this style of massive parallism, but many do fit the model
- ... the list goes on

So the cloud is tempting

56

- And cheaper, too!

- What's not to love?
 - ▣ Imagine that you work for a large company that is healthy and has managed its own story in its own way
 - ▣ Now the cloud suddenly offers absolutely unique opportunities that we can't access in any other way
 - ▣ Should you recommend that your boss drink the potion?

To cloud, or not cloud...

57

- ... maybe that's the question
- ... or maybe there is no other choice anymore