ECE/CS230
Computer Systems Security

Charalambos (Harrys) Konstantinou

https://sites.google.com/view/ececs230kaust/

Program security

Midterm feedback

- I highly appreciate your effort and collaboration – my door/”email” is always open

Updates

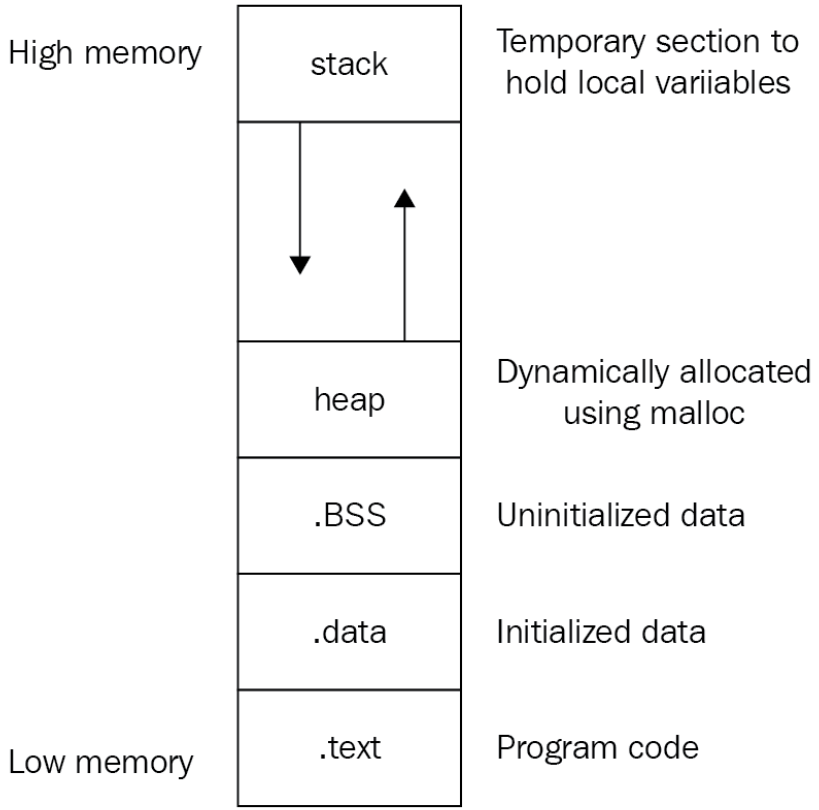
- Assingment 2 is due this Thursday
- Discussions as usual
- Today we will finish with program security – Assignments questions?
- Next week: Oct. 3 – midterm exam: 8:45am-10:30am
  - Midterm and final exams will be open-notes meaning that students can consult the following course materials: slides, handouts (including labs and assigned readings), lecture notes. Anything else is not allowed (except linguistic dictionaries). Kindle-type readers without internet access are allowed (check prior exam day with instructor) to avoid printing slides, etc.
  - All methods of evaluations are required. Students who do not show up for an exam or do not provide any assignment or participate in a discussion should expect a grade of zero on that item.

Buffer Overflows - Stack

- Now, we will focus on the stack, so stack overflows. The same concepts can be extended, but we don't talk about those

What is the stack? For x86

- Stack is on top of memory
- It grows down, but buffers go up (strange huh?)
- Stack not only contain local-variables, but also control data, "stack frames" for functions
- The last bullet is why stack overflows can be bad.



Intermission – Function Preamble / Postamble, Finale

- So we need to talk about the function preamble and finale (postamble? That is not a word I know) with respect to C and x86
- The preamble is a setup procedure whenever a function is called, the finale is the takedown

Preamble

When a function is called the following happens

- Function parameters are pushed onto the stack in reverse order
- The function is then called using the CALL instruction - The return address of the next instruction (the one after the function call) is pushed onto the stack. This is known as the return address
- The current stack frame base pointer (where the current frame starts) is pushed onto the stack

Postamble/Finale

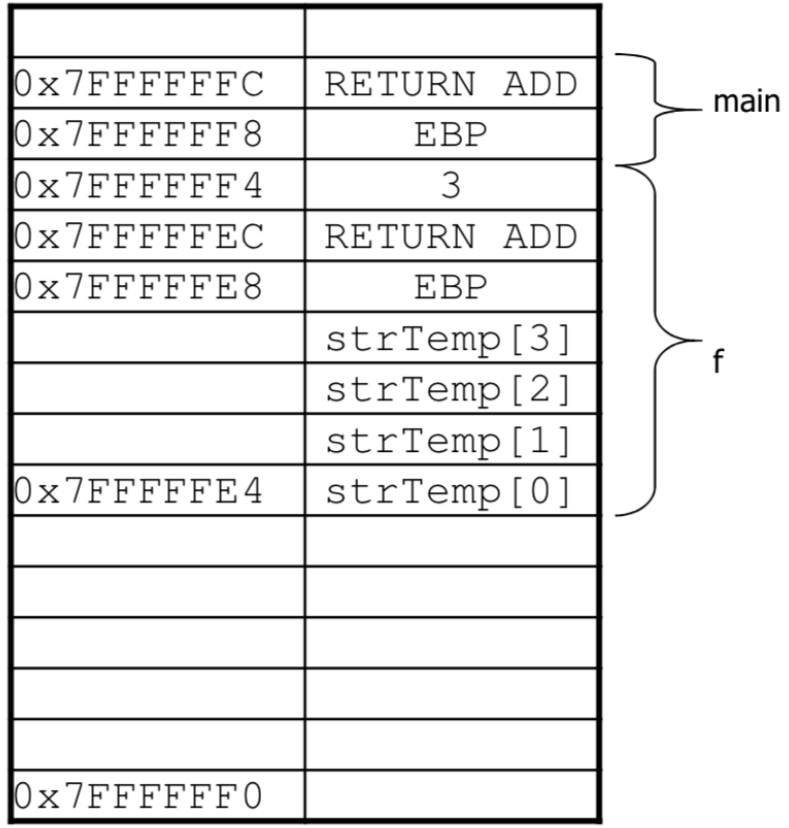
When a function ends the following takes place

- The saved stack frame base pointer is restored to the appropriate register (EBP)
- The Instruction Pointer is set to the saved return address. This way the next instruction will be the one after the function was called.

Stack Frames (Simplified)

```
void f(int a)
{
        char strTemp[4];
}
void main()
{
        f(3);
}
```

- Left column is the address
- Right column is contents
- I made up the addresses, but they are labeled correctly
- RETURN ADD is the saved return address, i.e. the address of the next instruction after the CALL instruction that started the current function
  - The idea is so that when the current function ends, RETURN ADD is the next instruction to run
- EBP holds the base of the previous stack frame (Extended Base Pointer)

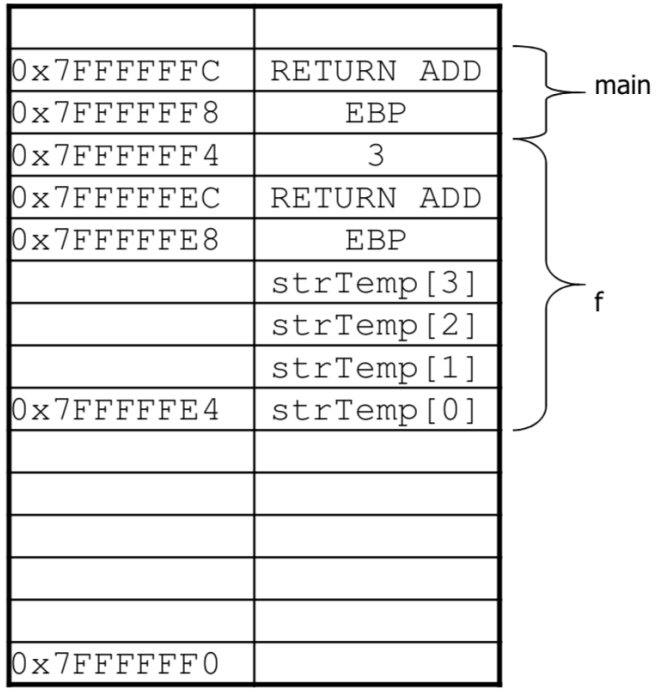


Notice something?

- On Lok's computer we were able to overwrite the next instruction
- What about in the stack frame from the previous slide?
  - No instructions ☹
  - BUT! There is that return address.. Maybe if we can just overwrite that with another address
  - What other address?
    - Any address
    - What about an address on the stack, like strTemp?

Stack Overflow - Example 6

- So lets now just pretend that strTemp is user inputted.
- What does the user need to input in order to make that first return address point back to the start of strTemp?
- “abcdabcd”[0x7FFFFFE4] is one possibility. There are plenty more.
  - That is interesting...
- What happens when this function ends?
  - The next instruction is now going to be 0x7FFFFFE4.
  - But that is this “abcd” thing, so it will probably just crash. Just like in Lok’s computer.



Example 6 Cont.

- So... if we were slick, then we will replace “abcd” with a valid instruction right?!
- Absolutely. Actually, if we were REALLY slick, we will replace it with a whole SERIES of instructions. Like instructions to open a “root shell”
  - A “root shell” is just a command prompt that has root privileges
  - Other possibilities are opening a port to listen to instructions
  - Anything else really

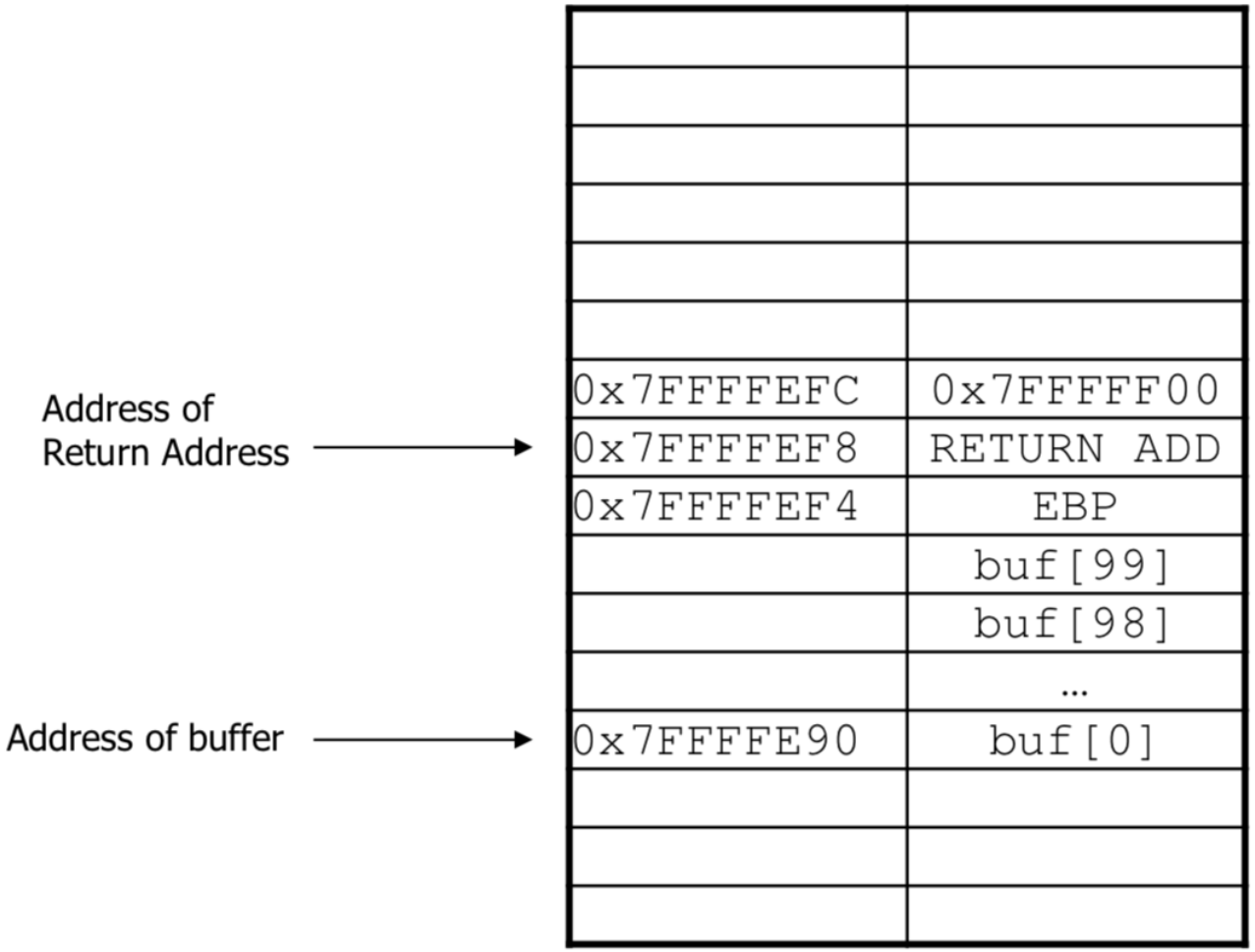
Shellcode

- The code to open a “shell” is known as – SHELLCODE
  (See http://shell-storm.org/shellcode/ for samples)
- Now that is interesting. But then there are a couple of difficult things.
  - First is how do we even know the address of strTemp in the first place?
  - Second is how do we know where the return address is, so I know how long my string is and also how to align things right?

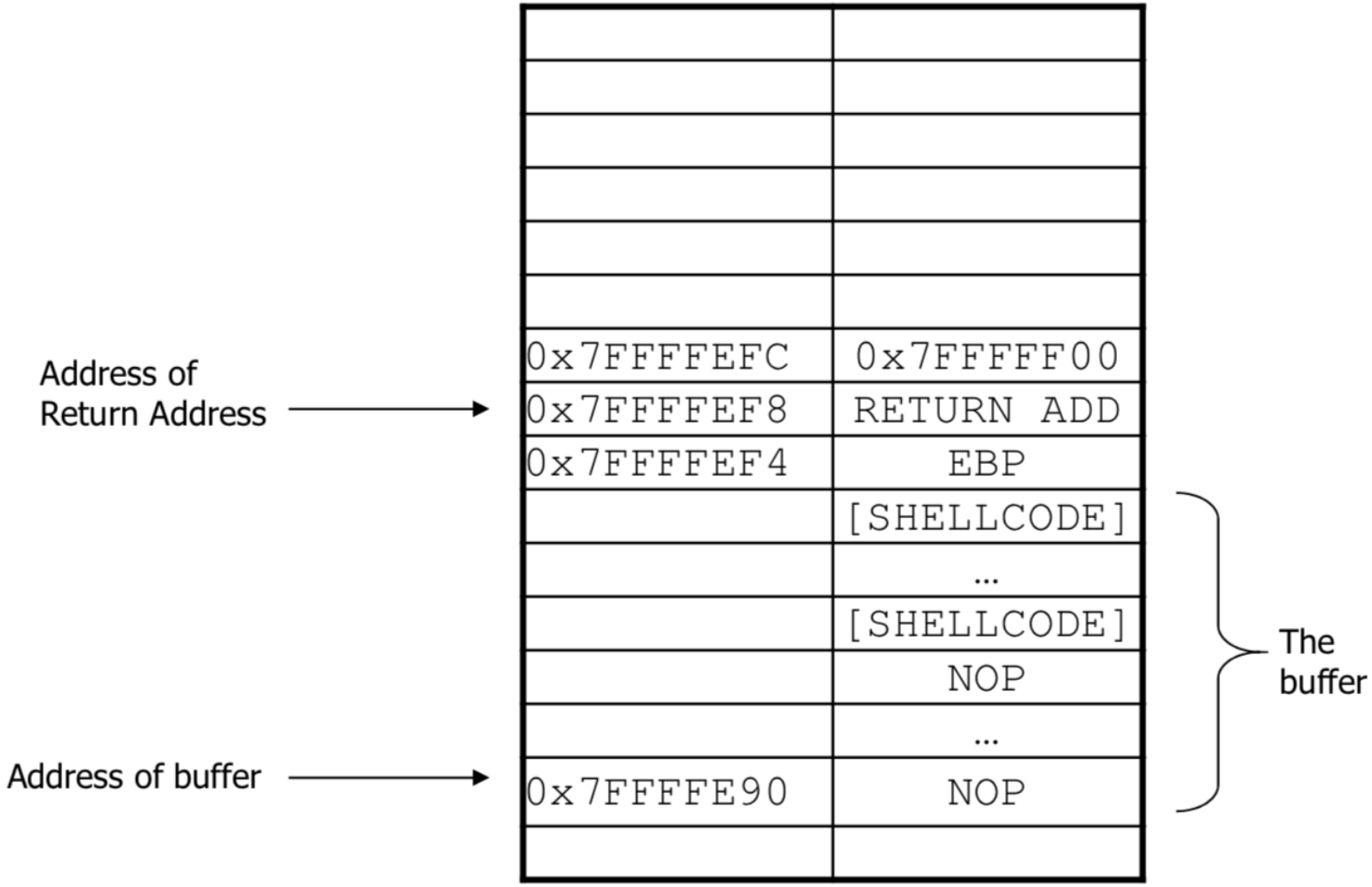
Address of strTemp

- The address of where your “shellcode” will be is difficult to obtain
  - You might be able to get it through debugging the program
  - You can try trial and error
  - But in general, there are well known ways (we’ll talk about mitigation strategies, one of which is called address space layout randomization)
- Okay, so the start of the “shellcode” is difficult to pin down, but what if there was a way to make the start of the shellcode span a LARGE area?
  - There is. In x86, instruction 0x90 is the NOP instruction, which is exactly 1 byte (which is good) and does absolutely nothing.

Some stack


Address of
Return Address
Address of buffer
0x7FFFFEFC
0x7FFFFF00
0x7FFFFEF8
RETURN ADD
0x7FFFFEF4
EBP
buf[99]
buf[98]
...
0x7FFFFE90
buf[0]

Stack with the buffer filled with 100 bytes of NOPs + SHELLCODE



NOPs

- Take for example the following memory contents [0x90][0x90]...[0x90][shellcode]
- What happens if lets say the next instruction was at the beginning of that chunk of memory?
  - Well the first instruction will be 0x90, which is NOP so nothing happens, and the next instruction is processed. Ohh wait that is a NOP as well, so then same thing and etc. etc. etc. until the shellcode instructions start to run
- Well what happens if the next instruction is, say the 5th 0x90 in that chunk?
  - Same exact thing as before, except less NOPs are processed.
- So, the moral of the story is that NOPs can be used to help alleviate this problem of having to know the exact address of the start of “shellcode”

What about the return address?

- So how do we know where the return address is so we can overwrite it?
  - Easiest way is to just keep trying... Repeat the address that we want the function to return to LOTS of times.
  - Example: [shellcode][0x7FFFFE92][0x7FFFFE92]...[0x7FFFFE92]
- Now some of you might have noticed that we might have an alignment problem
  - If we have an alignment problem, then the RETURN ADDRESS will not be overwritten with 0x7FFFFFE4 but that shifted.
  - Example with 2 byte alignment offset: 0xFFE47FFF

Egg

- So what did we learn?
- The easiest thing to do is to put a bunch of NOPs in the beginning then the SHELLCODE and then the address of one of those NOP instructions

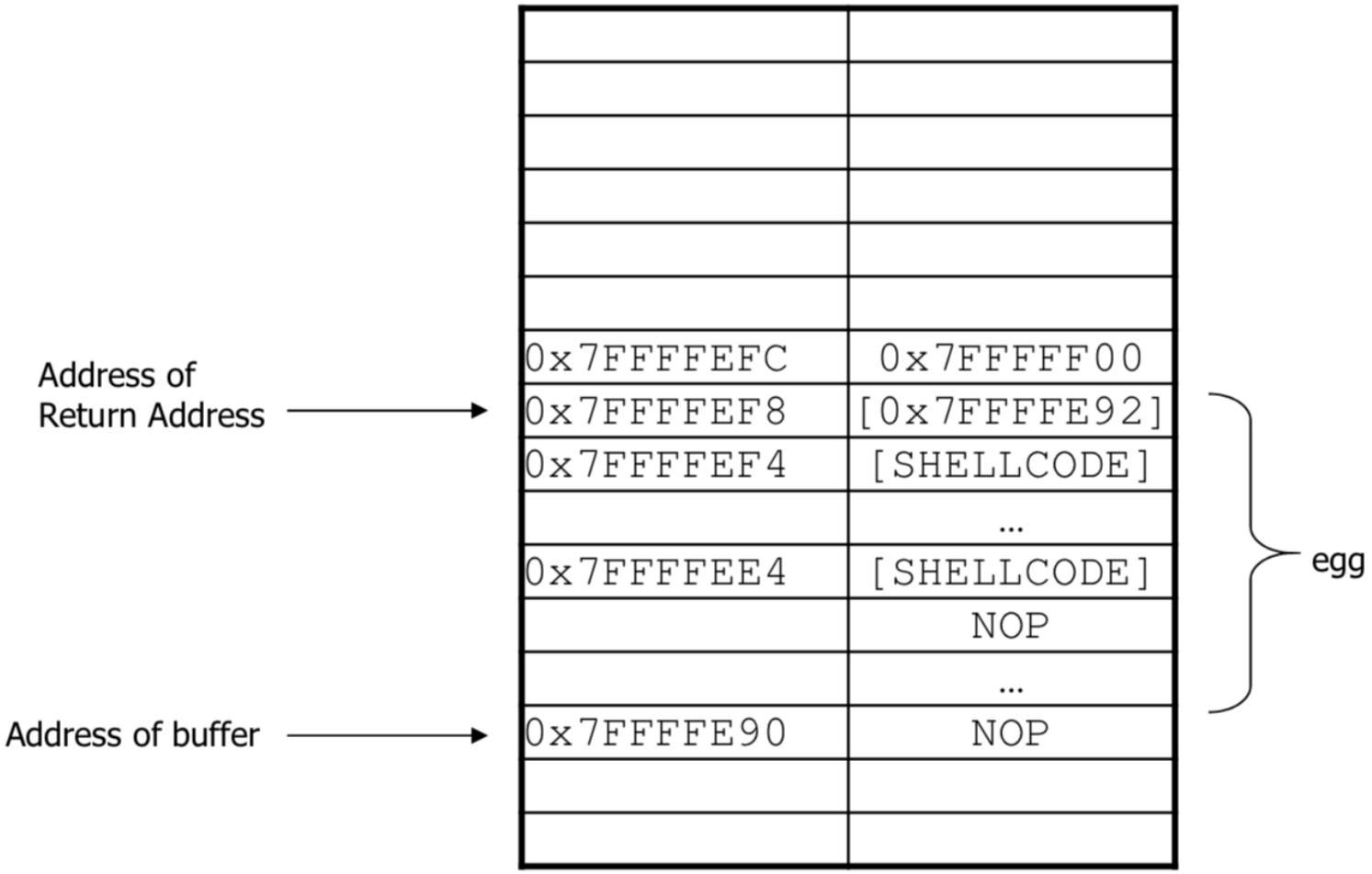
[NOP][NOP]...[NOP][SHELLCODE][ADD][ADD]...[ADD]

- This layout is known as an “egg”

Some questions

- Lets say that the shellcode is 20 bytes long
- How long does the egg have to be to overwrite the return address?
  - BUFFER+EBP+RETURNADDRESS=100+4+4=108
  - So if the egg is 108 bytes, we will overwrite ALL of the RETURN ADDRESS.
- How many NOPs can I possibly have?
  - EGG=NOPS+SHELLCODE+ADDRESS,
  - NOPS=EGG–SHELLCODE–ADDRESS=108–20–4=84
  - Thus, ADDRESS can point to ANY of the 84 NOPS and we are good.
- What happens if I used 85 NOPs?
  - Then there is an alignment problem. Only 3 bytes of the ADDRESS are overwriting the RETURN ADDRESS

Stack with an egg of size 108



A few things to notice

- strTemp was not large enough to hold that “egg” we were talking about.
  - So it is important for the buffer to be big enough (or at least from the start of the buffer to the saved return address)
- Also notice that the example was for the stack, there aren’t any saved return addresses elsewhere, like the heap right?
  - Nope there sure aren’t, but that doesn’t mean we can’t overwrite other interesting things
    - Like another variable. Notice that if there was another variable before strTemp then that would have been overwritten
    - Like a function pointer. (See the function-call table in C++ for example)
    - Or other things.
- Also I should point out that I completely ignored endianess (x86 likes to mess things up! It uses little-endian. Which means that address 0x7FFFFFE4 actually needs to be encoded as 0xE4FFFF7F in that string.)

A few things to notice

So how does that strTemp get populated with the egg anyways?

- In C most popular are unsafe/unbounded copy functions like strcpy(), memcpy() and gets()
- These should be replaced with “safer” versions like strncpy() that requires the user to insert a maximum number of elements to process

Static vs Dynamic Mitigation

- Two different types of techniques
- Dynamic ones require a system to be running
  - Mostly because you don’t “know” until runtime
  - Costly – Resource wise
  - Can be more “generic”
- Static ones can be performed “at rest”
  - Common ones are code analyzers
  - Very good for problems that are very well defined, more “specialized”
  - No runtime performance costs
  - Good for “known” problems

Buffer Overflows

- Static analysis is pretty useless
  - Very good at identifying “unsafe” function calls like strcpy
  - Only really effective with statically allocated buffers
  - Really, true effect will not be known until runtime though
  - Could have high false-positives
- Easiest “static” solution is to change programming languages
  - Like to Java where all bounds checking is built in

Other Static Techniques

- There exists other static techniques that are pretty effective
  - New libraries can be used that doesn’t implement or reimplements the “unsafe” functions
  - New compilers are introduced so that during compile time, all buffers can be bounded with the associated code and data structures

Runtime Techniques

- Static techniques aren't too interesting, so we will focus on dynamic ones
- Keep in mind that many of the dynamic techniques made the following observation
  - The reason why stack-overflows are so powerful is because control data (i.e. Return Address) is stored in-band with non-control data (like a buffer)
  - Control data can be overwritten very easily by non-control structures (by an instruction) – world read and writable characteristic of memory
  - This is an architectural (processor architecture) problem

Canary

- One of the first published solutions was called a “canary”, like the ones in coal mines
  - Put a chosen number like 0xDEADBEEF between the buffer and the important return address
  - When a stack overflow occurs trying to overwrite the return address, the number will be overwritten
  - Before returning from a function check to see if the number matches
    - If yes, then overflow did not occur
    - If no, then overflow occurred, and program stops.

Canary

- If the number is known, then it is easy to get around, just overwrite the canary with the same number
  - So randomize
  - The random number must be stored somewhere so what if we can find it? Good thing this is difficult!
- What if there are additional variables between the canary and the buffer? Then those variables can still be overwritten!!! The same applies to heap-based buffers.
- But, only protects the saved instruction pointer (return address) what about other pointers or addresses?
  - Use encryption. XOR all addresses canary.
  - But then what about “constant” addresses like library functions?

Canary

0x7FFFFFFC	RETURN ADD
0x7FFFFFF8	EBP
0x7FFFFFF4	3
0x7FFFFFEC	RETURN ADD
0x7FFFFFE8	EBP
	CANARY
	strTemp[3]
	strTemp[2]
0x7FFFFFE4	strTemp[1]
	strTemp[0]
0x7FFFFFF0	

No-eXecution Bit

- In addition to in-band data, this solution makes the observation that the “egg” contains the code to run. Since the buffer is on the stack, lets make the stack non-executable
  - Operating system (with help from CPU’s Memory Management Unit) allocates pages
  - Each page could be for any purpose, but OS knows if it’s a stack or heap or not, if it is, then mark the page as NX (note that the overhead is small, one extra bit for each page)
  - When the Instruction Pointer points to a page that has the bit set, then an error will occur and program should end

NX Cont.

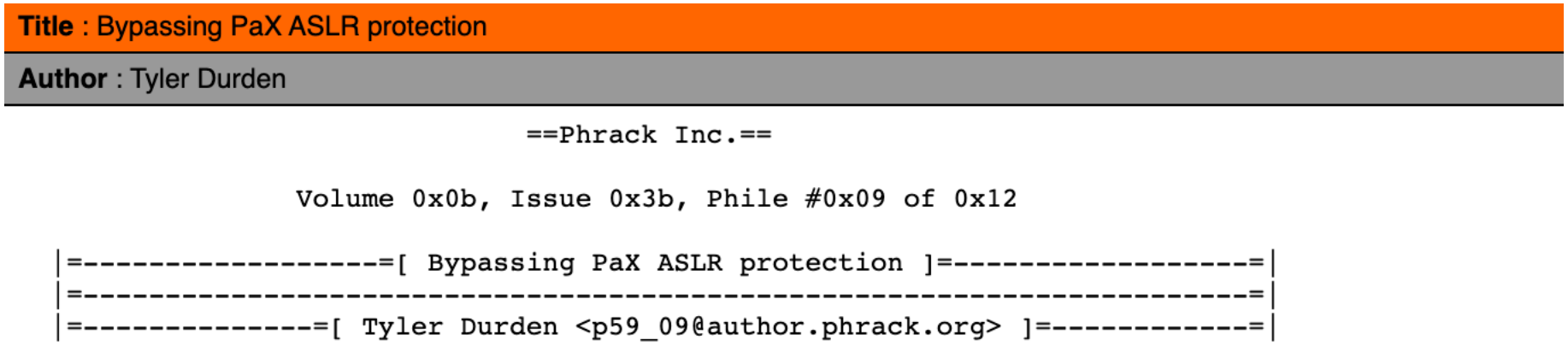
- Perhaps the simplest attack was to disable NX protection through an operating system call (but this is a configuration problem)
- Return-To-LibC
  - The response to NX was pretty simple, since the instruction pointer (return address) can't point to the stack (or any page that is marked with the bit) then lets overwrite the return address with the address of something that is known – a real function
  - The task is then to look for a real function with some parameters that we can set (by taking advantage of the stack layout) that will have some very interesting effects
  - Since LibC is the most common library with very useful functions like system() and exec() this kind of attack is normally known as a ret-libc (but any useful function will do though)
    - Ex. system("rm –rf /");

Address Space Layout Randomization

- As it turns out, people have found out ways to get around the no-execute bit
  - Though this is true, keep in mind that the attack vectors for the attacker are continually shrinking
- Researchers then observed that, wait a minute, all of these attacks require the attacker to know an address
  - Location of the return address or pointer
  - Location of a function (like libc functions) for ret-libc
- Solution: Why don’t we randomize the address layout?
  - Purpose is to achieve completely different addresses and memory layout each time a program is run

ASLR – good solution so far

- All you need to do is update the operating system
- Although there is possibility that ASLR can be bypassed due to implementation specific information http://phrack.org/issues/59/9.html#article



Title : Bypassing PaX ASLR protection

Author : Tyler Durden

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x09 of 0x12

|=----------=[Bypassing PaX ASLR protection]=----------=|
|=----------=[Tyler Durden <p59_09@author.phrack.org>]=----------=|

Defenses against buffer overflows – summary

- How might one protect against buffer overflows?
- Programmer: Use a language with bounds checking
  - And catch those exceptions!
- Compiler: Place padding between data and return address (“Canaries”)
  - Detect if the stack has been overwritten before the return from each function
- Memory: Non-executable stack
  - “W⊕X” (memory page is either writable or executable, but never both)
- OS: Stack (and sometimes code, heap, libraries) at random virtual addresses for each process
  - All mainstream OSes do this

Format string vulnerabilities

- Class of vulnerabilities discovered only in 2000
- Unfiltered user input is used as format string in `printf(), fprintf(), sprintf(), . . .`

Declaration of printf

- `int printf (const char *format, ...)`

Example

- `printf(":%s:\n", "Hello, world!");`

Format string vulnerabilities

More Examples

- `printf ("Characters: %c %c \n", 'a', 65);`
- `printf ("Decimals: %d %ld\n", 1977, 650000L);`
- `printf ("Preceding with blanks: %10d \n", 1977);`
- `printf ("Preceding with zeros: %010d \n", 1977);`
- `printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);`
- `printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);`
- `printf ("Width trick: %*d \n", 5, 10);`
- `printf ("%s \n", "A string");`

Format string vulnerabilities

- Class of vulnerabilities discovered only in 2000
- Unfiltered user input is used as format string in `printf(), fprintf(), sprintf(), . . .`
- `printf(buffer)` instead of `printf("%s", buffer)`
  - The first one will parse buffer for %'s and use whatever is currently on the stack to process found format parameters
- `printf("%s%s%s%s")` likely crashes your program
- `printf("%x%x%x%x")` dumps parts of the stack
- `%n` will write to an address found on the stack

FormatString.PDF

Incomplete mediation

- Inputs to programs are often specified by untrusted users
  - Web-based applications are a common example
  - “Untrusted” to do what?
- Users sometimes mistype data in web forms
  - Phone number: `01280807033`
  - Email: `harrys#kaust.edu.sa`
- The web application needs to ensure that what the user has entered constitutes a meaningful request

- This is called mediation

Incomplete mediation

- Incomplete mediation occurs when the application accepts incorrect data from the user

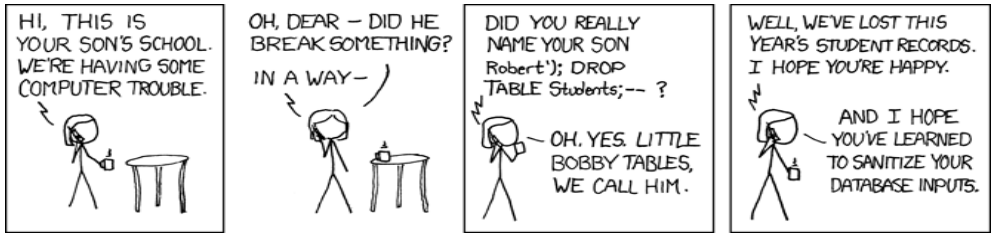
- Sometimes this is hard to avoid
  - Phone number: `012-808-0703`
  - This is a reasonable entry, that happens to be wrong
- We focus on catching entries that are clearly wrong
  - Not well formed
    - DOB: `1980-04-31`
  - Unreasonable values
    - DOB: `1876-10-12`
  - Inconsistent with other entries

Why do we care?

- What's the security issue here?

- What happens if someone fills in:
  - DOB: `98764874236492483649247836489236492`
    - Buffer overflow?
  - DOB: `'; DROP DATABASE users; --`
    - SQL injection?

https://xkcd.com/327/



- We need to make sure that any user-supplied input falls within well-specified values, known to be safe

Client-side mediation

- You've probably visited web sites with forms that do client-side mediation
  - When you click "submit", Javascript code will first run validation checks on the data you entered
  - If you enter invalid data, a popup will prevent you from submitting it

- Related issue: client-side state
  - Many web sites rely on the client to keep state for them
  - They will put hidden fields in the form which are passed back to the server when the user submits the form

Client-side mediation

- Problem: what if the user
  - Turns off Javascript?
  - Edits the form before submitting it? (Tampermonkey)
  - Writes a script that interacts with the web server instead of using a web browser at all?
  - Connects to the server “manually”?

    (telnet server.com 80)

- Note that the user can send arbitrary (unmediated) values to the server this way

- The user can also modify any client-side state

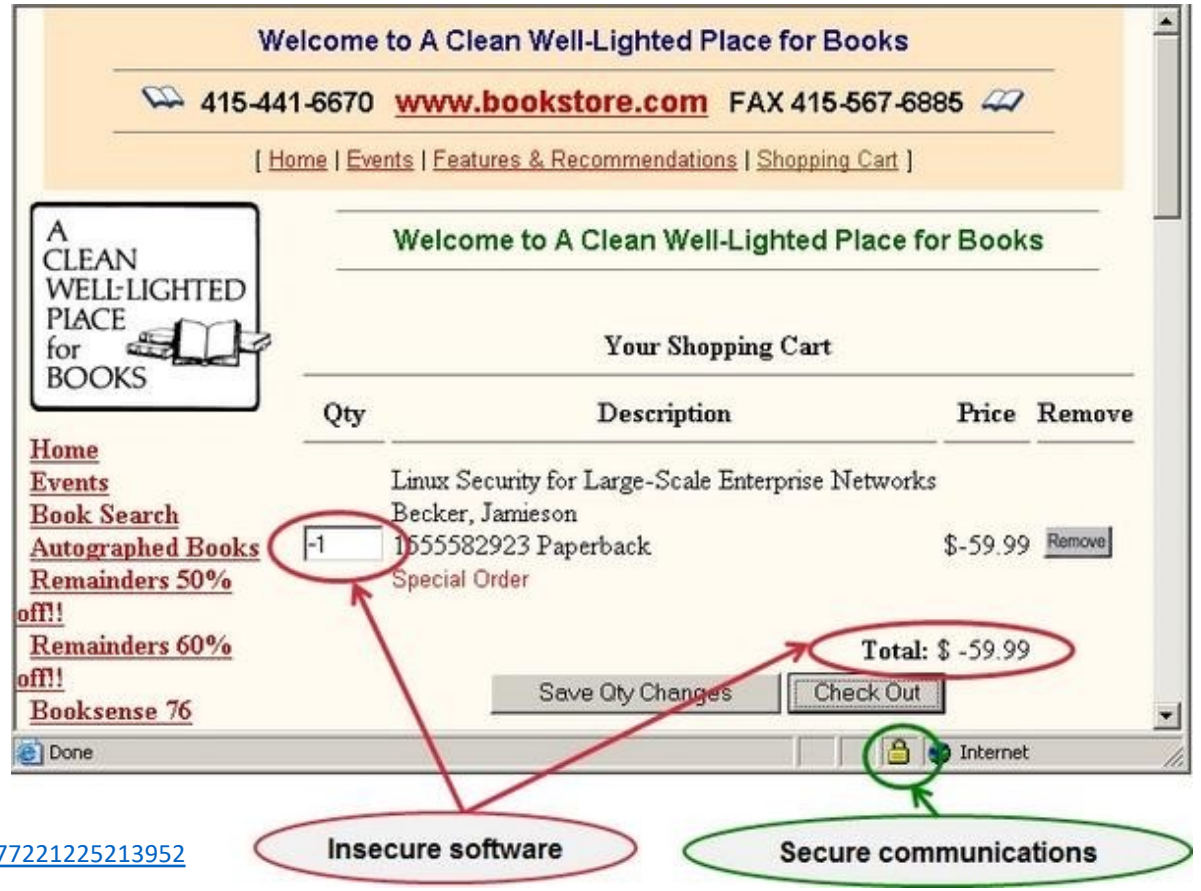
Example

- At a bookstore website, the user orders a copy of the course text. The server replies with a form asking the address to ship to. This form has hidden fields storing the user's order

```
<input type="hidden" name="isbn"  value="0-13-239077-9">
<input type="hidden" name="quantity"  value="1">
<input type="hidden" name="unitprice"  value="111.00">
```

- What happens if the user changes the "unitprice" value to "50.00" before submitting the form?

Another example



https://twitter.com/ericbaize/status/492777221225213952

Defenses against incomplete mediation

- Client-side mediation is an OK method to use in order to have a friendlier user interface but is useless for security purposes.
- You have to do server-side mediation, whether or not you also do client-side.
- For values entered by the user:
  - Always do very careful checks on the values of all fields
  - These values can potentially contain completely arbitrary 8-bit data (including accented chars, control chars, etc.) and be of any length
- For state stored by the client:
  - Make sure client has not modified the data in any way

TOCTTOU errors

- TOCTTOU (“TOCK-too”) errors
  - Time-Of-Check To Time-Of-Use
  - Also known as “race condition” errors
- These errors may occur when the following happens:
  1. User requests the system to perform an action
  2. The system verifies the user is allowed to perform the action
  3. The system performs the action
- What happens if the state of the system changes between steps 2 and 3?

Example

- A particular Unix terminal program is setuid (runs with superuser privileges) so that it can allocate terminals to users (a privileged operation)
- It supports a command to write the contents of the terminal to a log file
- It first checks if the user has permissions to write to the requested file; if so, it opens the file for writing
- The attacker makes a symbolic link:

```
logfile -> file_she_owns
```

- Between the “check” and the “open”, she changes it:

```
logfile -> /etc/passwd
```

The problem

- The state of the system changed between the check for permission and the execution of the operation

- The file whose permissions were checked for writeability by the user (`file she owns`) wasn't the same file that was later written to (`/etc/passwd`)
  - Even though they had the same name (`logfile`) at different points in time

- Q: Can the attacker really "win this race"?
- A: Yes.

Defenses against TOCTTOU errors

- When performing a privileged action on behalf of another party, make sure all information relevant to the access control decision is constant between the time of the check and the time of the action ("the race")
  - Keep a private copy of the request itself so that the request can't be altered during the race
  - Where possible, act on the object itself, and not on some level of indirection
    - e.g. Make access control decisions based on filehandles, not filenames
  - If that's not possible, use locks to ensure the object is not changed during the race

Outline

- Flaws, faults, and failures
- Unintentional security flaws
- Malicious code: Malware
- Other malicious code
- Nonmalicious flaws
- Controls against security flaws in programs

Malware

- Various forms of software written with malicious intent
- A common characteristic of all types of malware is that it needs to be executed in order to cause harm
- How might malware get executed?
  - User action
    - Downloading and running malicious software
    - Viewing a web page containing malicious code
    - Opening an executable email attachment
    - Inserting a CD/DVD or USB flash drive
  - Exploiting an existing flaw in a system
    - Buffer overflows in network daemons
    - Buffer overflows in email clients or web browsers

Types of malware

- Virus
  - Malicious code that adds itself to benign programs/files
  - Code for spreading + code for actual attack
  - Usually activated by users

- Worms
  - Malicious code spreading with no or little user involvement

Types of malware (2)

- Trojans
  - Malicious code hidden in seemingly innocent program that you download

- Logic Bombs
  - Malicious code hidden in programs already on your machine

Viruses

- A virus is a particular kind of malware that infects other files
  - Traditionally, a virus could infect only executable programs
  - Nowadays, many data document formats can contain executable code (such as macros)
    - Many different types of files can be infected with viruses now
- Typically, when the file is executed (or sometimes just opened), the virus activates, and tries to infect other files with copies of itself
- In this way, the virus can spread between files, or between computers

Infection

- What does it mean to “infect” a file?
- The virus wants to modify an existing
- (non-malicious) program or document (the host) in such a way that executing or opening it will transfer control to the virus
  - The virus can do its “dirty work” and then transfer control back to the host
- For executable programs:
  - Typically, the virus will modify other programs and copy itself to the beginning of the targets’ program code
- For documents with macros:
  - The virus will edit other documents to add itself as a macro which starts automatically when the file is opened

Infection

- In addition to infecting other files, a virus will often try to infect the computer itself
  - This way, every time the computer is booted, the virus is automatically activated
- It might put itself in the boot sector of the hard disk
- It might add itself to the list of programs the OS runs at boot time
- It might infect one or more of the programs the OS runs at boot time
- It might try many of these strategies
  - But it's still trying to evade detection!

Spreading

- How do viruses spread between computers?
- Usually, when the user sends infected files (hopefully not knowing they're infected!) to his friends
  - Or puts them on a p2p network
- A virus usually requires some kind of user action in order to spread to another machine
  - If it can spread on its own (via email, for example), it's more likely to be a worm than a virus

Payload

- In addition to trying to spread, what else might a virus try to do?
- Some viruses try to evade detection by disabling any active virus scanning software
- Most viruses have some sort of payload
- At some point, the payload of an infected machine will activate, and something (usually bad) will happen
  - Erase your hard drive
  - Subtly corrupt some of your spreadsheets
  - Install a keystroke logger to capture your online banking password
  - Start attacking a particular target website

Spotting viruses

- When should we look for viruses?
  - As files are added to our computer
    - Via portable media
    - Via a network
  - From time to time, scan the entire state of the computer
    - To catch anything we might have missed on its way in
    - But of course, any damage the virus might have done may not be reversable

- How do we look for viruses?
  - Signature-based protection
  - Behaviour-based protection

Signature-based protection

- Keep a list of all known viruses
- For each virus in the list, store some characteristic feature (the signature)
  - Most signature-based systems use features of the virus code itself
    - The infection code
    - The payload code
  - Can also try to identify other patterns characteristic of a particular virus
    - Where on the system it tries to hide itself
    - How it propagates from one place to another

Polymorphism

- To try to evade signature-based virus scanners, some viruses are polymorphic
  - This means that instead of making perfect copies of itself every time it infects a new file or host, it makes a modified copy instead
  - This is often done by having most of the virus code encrypted
    - The virus starts with a decryption routine which decrypts the rest of the virus, which is then executed
    - When the virus spreads, it encrypts the new copy with a newly chosen random key
- How would you scan for polymorphic viruses?

Behaviour-based protection

- Signature-based protection systems have a major limitation
  - You can only scan for viruses that are in the list!
  - But there are several brand-new viruses identified every day
    - One anti-virus program recognizes over *36 million* virus signatures
  - What can we do?

- Behaviour-based systems look for suspicious patterns of behaviour, rather than for specific code fragments
  - Some systems run suspicious code in a sandbox first

False negatives and positives

- Any kind of test or scanner can have two types of errors:
  - False negatives: fail to identify a threat that is present
  - False positives: claim a threat is present when it is not
- Which is worse?
- How do you think signature-based and behaviour-based systems compare?

Base rate fallacy

- Suppose a breathalyzer reports false drunkness in 5% of cases, but never fails to detect true drunkness.
- Suppose that 1 in every 1000 drivers is drunk (the base rate).
- If a breathalyzer test of a random driver indicates that he or she is drunk, what is the probability that he or she really is drunk?
- Applied to a virus scanner, these numbers imply that there will be many more false positives than true positives, potentially causing the true positives to be overlooked or the scanner disabled.

Worms

- A worm is a self-contained piece of code that can replicate with little or no user involvement
- Worms often use security flaws in widely deployed software as a path to infection
- Typically:
  - A worm exploits a security flaw in some software on your computer, infecting it
  - The worm immediately starts searching for other computers (on your local network, or on the Internet generally) to infect
  - There may or may not be a payload that activates at a certain time, or by another trigger

The Morris worm

- The first Internet worm, launched by a graduate student at Cornell in 1988
- Once infected, a machine would try to infect other machines in three ways:
  - Exploit a buffer overflow in the “finger” daemon
  - Use a back door left in the “sendmail” mail daemon
  - Try a “dictionary attack” against local users’ passwords. If successful, log in as them, and spread to other machines they can access without requiring a password
- All three of these attacks were well known!
- First example of buffer overflow exploit in the wild
- Thousands of systems were offline for several days

The Code Red worm

- Launched in 2001
- Exploited a buffer overflow in Microsoft's IIS web server (for which a patch had been available for a month)
- An infected machine would:
  - Deface its home page
  - Launch attacks on other web servers (IIS or not)
  - Launch a denial-of-service attack on a handful of web sites, including https://www.whitehouse.gov/
  - Installed a back door to deter disinfection
- Infected 250,000 systems in nine hours

The Slammer worm

- Launched in 2003, performed denial-of-service attack
- First example of a “Warhol worm”
  - A worm which can infect nearly all vulnerable machines in just 15 minutes
- Exploited a buffer overflow in Microsoft’s SQL Server (also having a patch available)
- A vulnerable machine could be infected with a single UDP packet!
  - This enabled the worm to spread extremely quickly
  - Exponential growth, doubling every 8.5 seconds
  - 90% of vulnerable hosts infected in 10 minutes

Conficker Worm

- First detected in November 2008
- Multiple variants
- Propagated a command-and-control style botnet
- Security experts had to generate and sinkhole C&C domains
- Number of infected hosts in 2009: 9-15 million, 2011: 1.7 million, 2015: 400,000

Stuxnet

- Discovered in 2010
- Allegedly created by the US and Israeli intelligence agencies
- Allegedly targeted Iranian uranium enrichment program
- Targets Siemens SCADA systems installed on Windows. One application is the operation of centrifuges
- It tries to be very specific and uses many criteria to select which systems to attack after infection

Stuxnet

- Very promiscuous: Used 4(!) different zero-day attacks to spread. Has to be installed manually (USB drive) for air-gapped systems.
- Very stealthy: Intercepts commands to SCADA system and hides its presence
- Very targeted: Detects if variable-frequency drives are installed, operating between 807-1210 Hz, and then subtly changes the frequencies so that distortion and vibrations occur resulting in broken centrifuges.

WannaCry

- Launched in May 2017, ransomware
- Infected 230,000 computers, including many of the British National Health Service
- Exploits a Windows SMB vulnerability originally discovered by the NSA
- NSA kept it secret (and exploited it?)
- The “Shadow Brokers” leaked it (and others) in April 2017
- Microsoft had released a patch for it a month ago (after being alerted by NSA? Somebody else?) but many systems remained unpatched
- Emergency patch for Windows XP and 8 in May 2017

Trojan horses

http://www.sampsonuk.net/B3TA/TrojanHorse.jpg

Trojan horses

- Trojan horses are programs which claim to do something innocuous (and usually do), but which also hide malicious behaviour

You're surfing the Web and you see a button on the Web site saying, "Click here to see the dancing pigs." And you click on the Web site and then this window comes up saying, "Warning: this is an untrusted Java applet. It might damage your system. Do you want to continue? Yes/No."

Well, the average computer user is going to pick dancing pigs over security any day.

And we can't expect them not to. — Bruce Schneier

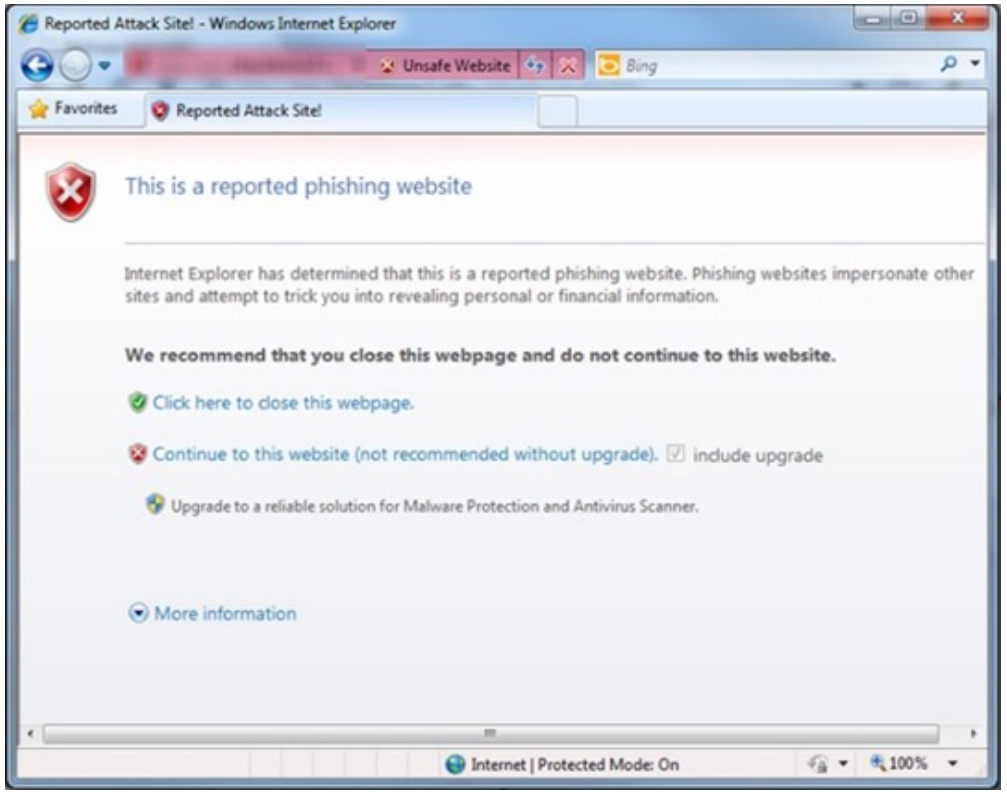
Trojan horses

- Gain control by getting the user to run code of the attacker's choice, usually by also providing some code the user wants to run
  - "PUP" (potentially unwanted programs) are an example
  - For scareware, the user might even pay the attacker to run the code
- The payload can be anything; sometimes the payload of a Trojan horse is itself a virus, for example

- Trojan horses usually do not themselves spread between computers; they rely on multiple users executing the "trojaned" software
  - Better: users share the trojaned software on p2p

Scareware

http://static.arstechnica.com/malware_warning_2010.png



Ransomware

https://en.wikipedia.org/wiki/WannaCry_ransomware_attack#/media/File:Wana_Decrypt0r_screensh

Ransomware

- Demands ransom to return some hostage resource to the victim

- CryptoLocker in 2013:
  - Spread with spoofed e-mail attachments from a botnet
  - Encrypted victim's hard drive
  - Demanded ransom for private key
  - Botnet taken down in 2014; estimated ransom collected between $3 million to $30 million

- Could also be scareware

Logic bombs

- A logic bomb is malicious code hiding in the software already on your computer, waiting for a certain trigger to "go off" (execute its payload)
- Logic bombs are usually written by "insiders", and are meant to be triggered sometime in the future
  - After the insider leaves the company
- The payload of a logic bomb is usually pretty dire
  - Erase your data
  - Corrupt your data
  - Encrypt your data, and ask you to send money to some offshore bank account in order to get the decryption key!

Logic bombs

- What is the trigger?

- Usually something the insider can affect once he is no longer an insider
  - Trigger when this particular account gets three deposits of equal value in one day
  - Trigger when a special sequence of numbers is entered on the keypad of an ATM
  - Just trigger at a certain time in the future (called a "time bomb")

Spotting Trojan horses and logic bombs

- Spotting Trojan horses and logic bombs is extremely tricky. Why?

- The user is intentionally running the code!
  - Trojan horses: the user clicked “yes, I want to see the dancing pigs”
  - Logic bombs: the code is just (a hidden) part of the software already installed on the computer
- Don’t run code from untrusted sources?
- Better: prevent the payload from doing bad things
  - More on this later

Outline

- Flaws, faults, and failures
- Unintentional security flaws
- Malicious code: Malware
- Other malicious code
- Nonmalicious flaws
- Controls against security flaws in programs

Other malicious code

- Web bugs (beacon)
- Back doors
- Salami attacks
- Privilege escalation
- Rootkits
- Keystroke logging
- Interface illusions

Web bugs

- A web bug is an object (usually a 1x1 pixel transparent image) embedded in a web page, which is fetched from a different server from the one that served the web page itself.

- Information about you can be sent to third parties (often advertisers) without your knowledge or consent
  - IP address
  - Contents of cookies (to link cookies across web sites)
  - Any personal info the site has about you

Web bug example

- On the quicken.com home page:
  - <IMG WIDTH="1" HEIGHT="1" src="http://app.insightgrit.com/1/nat?id=79152388778&ref=http://www.eff.org/ Privacy/Marketing/web bug.html&z=668951 &purl=http://quicken.intuit.com/">

- What information can you see being sent to insightgrit.com?

“Malicious code”?

- Why do we consider web bugs “malicious code”?
- This is an issue of privacy more than of security
- The web bug instructs your browser to behave in a way contrary to the principle of informational self-determination
  - Much in the same way that a buffer overflow attack would instruct your browser to behave in a way contrary to the security policy

Leakage of your identity

- With the help of cookies, an advertiser can learn what websites a person is interested in
- But the advertiser cannot learn person's identity
- ... unless the advertiser can place ads on a social networking site
- Content of HTTP request for Facebook ad:

```
GET [pathname of ad]  Host:ad.doubleclick.nt
Referer:https://www.facebook.com/profile.php?id=
123456789&ref=name
Cookie:  id=2015bdfb9ec...
```

Back doors

- A back door (also called a trapdoor) is a set of instructions designed to bypass the normal authentication mechanism and allow access to the system to anyone who knows the back door exists
  - Sometimes these are useful for debugging the system, but don't forget to take them out before you ship!

- Fanciful examples:
  - "Reflections on Trusting Trust" (mandatory reading)
  - "WarGames"

Examples of back doors

- Real examples:
  - Debugging back door left in sendmail
  - Back door planted by Code Red worm
  - Port knocking
    - The system listens for connection attempts to a certain pattern of (closed) ports. All those connection attempts will fail, but if the right pattern is there, the system will open, for example, a port with a root shell attached to it.
  - Attempted hack to Linux kernel source code

```
if ((options == (__WCLONE|__WALL)) &&
(current->uid = 0))
retval = -EINVAL;
```

Sources of back doors

- Forget to remove them
- Intentionally leave them in for testing purposes
- Intentionally leave them in for maintenance purposes
  - Field service technicians
- Intentionally leave them in for legal reasons
  - “Lawful Access”
- Intentionally leave them in for malicious purposes
  - Note that malicious users can use back doors left in for non-malicious purposes, too!

Salami attacks

- A salami attack is an attack that is made up of many smaller, often considered inconsequential, attacks
- Classic example: send the fractions of cents of round-off error from many accounts to a single account owned by the attacker
- More commonly:
  - Credit card thieves make very small charges to very many cards
  - Clerks slightly overcharge customers for merchandise
  - Gas pumps misreport the amount of gas dispensed

Privilege escalation

- Most systems have the concept of differing levels of privilege for different users
  - Web sites: everyone can read, only a few can edit
  - Unix: you can write to files in your home directory, but not in /usr/bin
  - Mailing list software: only the list owner can perform certain tasks

- A privilege escalation is an attack which raises the privilege level of the attacker (beyond that to which he would ordinarily be entitled)

Sources of privilege escalation

- A privilege escalation flaw often occurs when a part of the system that legitimately runs with higher privilege can be tricked into executing commands (with that higher privilege) on behalf of the attacker
  - Buffer overflows in setuid programs or network daemons
  - Component substitution (See attack on search path in textbook)
- Also: the attacker might trick the system into thinking he is in fact a legitimate (higher-privileged) user
  - Problems with authentication systems
    - "`-froot`" attack
  - Obtain session id/cookie from another user to access their bank account

Rootkits

- A rootkit is a tool often used by “script kiddies”

- It has two main parts:
  - A method for gaining unauthorized root / administrator privileges on a machine (either starting with a local unprivileged account, or possibly remotely)
    - This method usually exploits some known flaw in the system that the owner has failed to correct
    - It often leaves behind a back door so that the attacker can get back in later, even if the flaw is corrected
  - A way to hide its own existence
    - “Stealth” capabilities
    - Sometimes just this stealth part is called the rootkit

Stealth capabilities

- How do rootkits hide their existence?
  - Clean up any log messages that might have been created by the exploit
  - Modify commands like `ls` and `ps` so that they don't report files and processes belonging to the rootkit
  - Alternately, modify the kernel so that no user program will ever learn about those files and processes!

Example: Sony XCP

- Mark Russinovich was developing a rootkit scanner for Windows
- When he was testing it, he discovered his machine already had a rootkit on it!
- The source of the rootkit turned out to be Sony audio CDs equipped with XCP “copy protection”
- When you insert such an audio CD into your computer, it contains an `autorun.exe` file which automatically executes
- `autorun.exe` installs the rootkit

Example: Sony XCP

- The “primary” purpose of the rootkit was to modify the CD driver in Windows so that any process that tried to read the contents of an XCP-protected CD into memory would get garbled output
- The “secondary” purpose was to make itself hard to find and uninstall
  - Hid all files and processes whose names started with sys
- After people complained, Sony eventually released an uninstaller
  - But running the uninstaller left a back door on your system!

Keystroke logging

- Almost all of the information flow from you (the user) to your computer (or beyond, to the Internet) is via the keyboard
  - A little bit from the mouse, a bit from devices like USB keys

- An attacker might install a keyboard logger on your computer to keep a record of:
  - All email / IM you send
  - All passwords you type
- This data can then be accessed locally, or it might be sent to a remote machine over the Internet

Who installs keyboard loggers?

- Some keyboard loggers are installed by malware
  - Capture passwords, especially banking passwords
  - Send the information to the remote attacker

- Others are installed by one family member to spy on another
  - Spying on children
  - Spying on spouses
  - Spying on boy/girlfriends

Kinds of keyboard loggers

- Application-specific loggers:
  - Record only those keystrokes associated with a particular application, such as an IM client
- System keyboard loggers:
  - Record all keystrokes that are pressed (maybe only for one particular target user)
- Hardware keyboard loggers:
  - A small piece of hardware that sits between the keyboard and the computer
    - Works with any OS
    - Completely undetectable in software

Interface illusions

- You use user interfaces to control your computer all the time
- For example, you drag on a scroll bar to see offscreen portions of a document
- But what if that scrollbar isn’t really a scrollbar?
- What if dragging on that “scrollbar” really dragged a program (from a malicious website) into your “Startup” folder (in addition to scrolling the document)?
  - This really happened

Interface Illusion by Conficker worm

Interface illusions

- We expect our computer to behave in certain ways when we interact with “standard” user interface elements.
- But often, malicious code can make “nonstandard” user interface elements in order to trick us!
- We think we’re doing one thing, but we’re really doing another
- How might you defend against this?

Phishing

- Phishing is an example of an interface illusion
- It looks like you're visiting Paypal's website, but you're really not.
  - If you type in your password, you've just given it to an attacker
- Advanced phishers can make websites that look every bit like the real thing
  - Even if you carefully check the address bar, or even the SSL certificate!

Phishing Detection

- Unusual email/URL
  - Especially if similar to known URL/email
  - Email that elicits a strong emotional response and requests fast action on your part
- Attachments with uncommon names
- Typos, unusual wording
- No https (not a guarantee)

Man-in-the-middle attacks

- Keyboard logging, interface illusions, and phishing are examples of man-in-the-middle attacks
- The website/program/system you're communicating with isn't the one you think you're communicating with
- A man-in-the-middle intercepts the communication from the user, and then passes it on to the intended other party
  - That way, the user thinks nothing is wrong, because his password works, he sees his account balances, etc.

Man-in-the-middle attacks

- But not only is the man-in-the-middle able to see (and record) everything you're doing, and can capture passwords, but once you've authenticated to your bank (for example), the man-in-the-middle can hijack your session to insert malicious commands
  - Make a $700 payment to attacker@evil.com
- You won't even see it happen on your screen, and if the man-in-the-middle is clever enough, he can edit the results (bank balances, etc.) being displayed to you so that there's no visible record (to you) that the transaction occurred
  - Stealthy, like a rootkit

Outline

- Flaws, faults, and failures
- Unintentional security flaws
- Malicious code: Malware
- Other malicious code
- Nonmalicious flaws
- Controls against security flaws in programs

Covert channels

- An attacker creates a capability to transfer sensitive/unauthorized information through a channel that is not supposed to transmit that information.
- What information can/cannot be transmitted through a channel may be determined by a policy/guidelines/physical limitations, etc.

Covert channels

- Assume that Eve can arrange for malicious code to be running on Alice's machine
  - But Alice closely watches all Internet traffic from her computer
  - Better, she doesn't connect her computer to the Internet at all!

- Suppose Alice publishes a weekly report summarizing some (nonsensitive) statistics

- Eve can "hide" the sensitive data in that report!
  - Modifications to spacing, wording, or the statistics itself
  - This is called a covert channel

Side channels

- What if Eve can't get Trojaned software on Alice's computer in the first place?

- It turns out there are some very powerful attacks called side channel attacks
  - Eve watches how Alice's computer behaves when processing the sensitive data

  - Eve usually has to be somewhere in the physical vicinity of Alice's computer to pull this off
    - But not always!

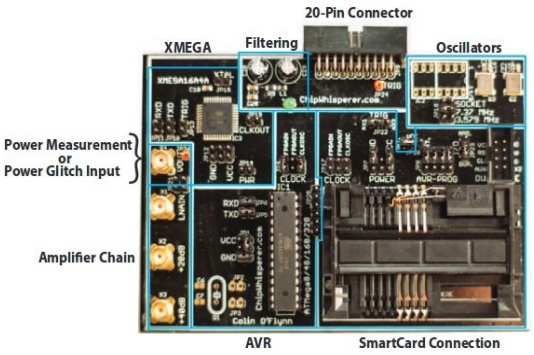
Potential Attack Vectors

- Bandwidth consumption

- ’Shoulder-surfing’
- Reflections

Potential Attack Vectors

- Timing computations
- Power consumption
- Electromagnetic emission
- Sound emissions
- Cache access
- Differential power analysis
- Differential fault analysis



Reflections: Scenario

- Alice types her password on a device in a public place
- Alice hides her screen
- But there is a reflecting surface close

Reflections

- Eve uses a camera and a telescope
- Off-the-shelf: less than $2k
- Photograph reflection of screen through telescope
- Reconstruct original image
- Distance: 10–30 m
- Depends on equipment and type of reflecting surface

Reflections: Defense

The picture so far

- We've looked at a large number of ways an attacker can compromise program security
  - Exploit unintentional flaws
  - Introduce malicious code, including malware
  - Exploit intentional, but nonmalicious, behaviour of the system

- The picture looks pretty bleak

- Our job is to control these threats
  - It's a tough job