Secure Programming

- Introduction
- Morris Worm
- HB Gary Hack Background
- Details of Break-In
- Details of Break-In Continued
- Sequel
- The Security Problem
- Terminology
- Types of Malicious Programs
- Principle of Least Privilege
- Passwords
- Passwords Continued
- Program Bugs Which Affect Security
- Trivial Buffer Overflow Example
- Trivial Buffer Overflow Example Log
- Problematic Library Functions and Alternatives
- Buffer Overflow Attacks
- Buffer Overflow Mitigation
- Untrusted Data Used for Executing Other Programs
- Validating Data
- Validating Data Continued

- Environmental Variables Validation
- Race Conditions
- Avoiding Race Conditions
- Randomness
- Security Organizations / Publications
- References

Introduction

- Notorious incidents as motivation.
- The problem.
- Terminology.
- Security bugs
- Security organizations / publications.

Morris Worm

- 1988, first hack which received attention in the popular press.
- Replicating worm.
- Overflow’d `gets()` buffer in `fingerd` daemon.
- Misused `DEBUG` command to `sendmail` daemon.
- Ran dictionary attack against publicly readable `/etc/passwd` file.
- Given a password, would use `.forward` and `.rhosts` file to break into other hosts where user had accounts.
- Was supposedly meant for innocent research, but a bug caused indiscriminate propagation.

HB Gary Hack Background

- WikiLeaks gained world-wide prominence in 2010 with releasing, among other dumps, US State Department emails in Nov 2010.
- At end of 2010, bowing to political pressure, major payment processors stopped processing donations to WikiLeaks.
- The *hacktivist* group **Anonymous** mounted *distributed-denial-of-service* attacks on web sites of payment processors.
- At beginning of 2011, CEO Aaron Barr of security company **HBGary Federal** publicized the fact that he could reveal the identity of Anonymous.
- Anonymous took down the HBGary Federal website, extraced 40K emails from email server, deleted 1TB of backup data.

Details of Break-In

- HBGary Federal website was running a proprietary *content-management system* which has an *SQL injection flaw*. By providing specially crafted inputs, attackers were able to run arbitrary queries against database, and accessed login and password table.

- Passwords were hashed using a fast hash algorithm (MD5) without any *salt*, making them amenable to a *rainbow-table attack* (comparing the hashed password with a table of precomputed hashes for passwords).

- CEO and COO had passwords which were cracked. Same passwords were used on other machines and email, twitter, etc.

- Used COO password to access Linux support machine which was accessed using password-based ssh (rather than public/private key access).

- Linux system had not been patched and contained a well-known security flaw which allowed access as root! Deleted backups and research data.

Details of Break-In Continued

- Barr’s password allowed access to Google Apps as **administrator**. Allowed attackers to examine email archives.

- Email contained root password for another machine `rootkit`!. But remote root access not possible.

- With access to email account, anonymous sent an email to system administrator requesting ssh access be allowed (gave possible root passwords in email!). Administrator set things up, changed password to `changeme123` and verified user-name!!

- Attackers dumped user database for all users who ever registered on `rootkit.com`. Contained crackable passwords.

Sequel

- Extremely poor security practices for a commercial company and even worse for a security company.
- Anonymous members arrested 2012 (ringleader was arrested in 2011 and was cooperating with the FBI).
- HBGary received additional business!!
- HBGary purchased by defense contractor ManTech.
- CEO Barr stepped down from HBGary. Took position as *cybersecurity director* at another federal contractor. Fired 2012 for concentrating on social media and Anonymous.
- In 2013, list of `rootkit.com` user-names/passwords allowed possible identification of Chinese hackers suspected in other hacking incidents.

The Security Problem

- A user (person or program) with limited authorization interacts with a program which has or may (temporarily) get different authorization.
- By choosing certain inputs or interacting with the program in a certain way, the user forces the program to take an unintended action using the program’s authorization.
- Ultimate exploit is to spawn a shell with program’s authorization.

Terminology

Some of these definitions are adapted from the Jargon Dictionary at http://catb.org/jargon/:

Hacker

Person who enjoys exploring the intricacies of programmable systems, skilled in programming (quickly), expert.

Cracker

Person who breaks security on a computer system. Journalists often misuse the term *hacker* to refer to a *cracker*.

Root Kit

A collection of scripts which allows a cracker to break into a machine and (maybe) get `root` access.

Script Kiddie

A cracker who merely uses *root kits* and other pre-written scripts to crack into a system. Does not really have much of an understanding of the technology.

Types of Malicious Programs

Trojan Horse

A program which provides normal useful functionality while also performing hidden malicious activity. Eg. a `login` program which allows users to login normally, while storing plaintext passwords in a file.

Worm

A program that propagates itself over a network, reproducing itself as it goes.

Virus

A program that searches out other programs and *infects* them by embedding a copy of itself in them, so that they become Trojan horses. When these programs are executed, the embedded virus is executed too, thus propagating the *infection*. This normally happens invisibly to the user. Unlike a worm, a virus cannot infect other computers without assistance. It is propagated by vectors such as humans trading programs with their friends.

Principle of Least Privilege

- Always use lowest possible privilege.
- When running set[ug]id programs, use privileged user only for portions of execution where it is strictly necessary, reverting to non-privileged user wherever possible.
- When done with privileged operations, revert to non-privileged user permanently.

Passwords

- Passwords should never be stored in plaintext.
- Typically, passwords are hashed (not encrypted) using a one-way hashing algorithm like MD5 or SHA2. Only hash is stored. When user logs in, entered password is hashed and compared with stored hash; if they match, then login is allowed, else denied.
- A responsible administrator should never have access to a plaintext password.
- Initially, Unix passwords stored in world-readable `/etc/passwd`. Now stored in `/etc/shadow` with restricted readability.
- Simple hash amenable to *dictionary attack* where hashes are precomputed for many common passwords; if precomputed hash matches stored hash (got by accessing `/etc/passwd` or stealing `/etc/shadow` using other hacks), then password is dictionary word.

Passwords Continued

- Rainbow-tables facilitate dictionary attack by allowing compact storage of dictionary-word/hash mapping.

- Dictionary attacks made harder by adding a random salt to each password before hashing. The salt is stored along with the hash to allow matching an entered password.

- In 2000’s, brute-forcing passwords becoming possible because of extremely fast hardware (sometimes using GPUs).

- Current best practice is to use purposely slow hash algorithms like `bcrypt`.

Program Bugs Which Affect Security

- Buffer overflows. No checking on whether a buffer overflows ... hence specially crafted inputs can overflow the buffer overwriting other memory, which can compromise security. Notorious in C/C++ programs. Can be avoided by using safe programming languages like Java or Perl.
- Untrusted data used in executing other programs.
- Untrusted data used as `printf()` format strings (exploit can read arbitrary locations and even write (using %n) arbitrary locations).
- Environmental variables.
- Race conditions. Program checks something and then acts based on the result of the check. The check and act are not atomic, allowing a malicious user to change the result of the check before the act.

- Randomness. Cryptographic schemes often depend on a source of randomness. If the source of randomness is compromised, then the security of the cryptographic scheme is also compromised.

Trivial Buffer Overflow Example

The following program is a trivial example which illustrates how overflowing a buffer can over-write sensitive data.

```
enum {
  NAME_SIZE = 40,
  SENSITIVE_SIZE = 16,
};


struct {
  /* buffer to be overflowed */
  char name[NAME_SIZE];

  /* sensitive data */
  char sensitive[SENSITIVE_SIZE];
} data;

int
main() {
  strcpy(data.sensitive, "sensitive");
  printf("Enter your name: "); fflush(stdout);
  scanf("%s", data.name);
  printf("sensitive = %s\n", data.sensitive);
  return 0;
}
```

Trivial Buffer Overflow Example Log

```
$ ./overflow
Enter your name: John Smith
sensitive = sensitive
$ ./overflow
Enter your name: 0123456789012345678901234567890123456789cracked
sensitive = cracked
$
```

Problematic Library Functions and Alternatives

- Replace `gets()` with `fgets()`.
- Replace `strcpy()` with `strncpy()`.
- Replace `strcat()` with `strncat()`.
- Replace `scanf()`, `sscanf()`, `fscanf()`, `vscanf()`, `vsscanf()`, `vfscanf()` with precision specifiers like `%.10s` or custom parse code.
- Replace `sprintf()`/`vsprintf()` with `snprintf()`/`vsnprintf()` or precision specifiers like `%.10s`.

Always validate untrusted input and make sure buffers are big enough to hold input.

Buffer Overflow Attacks

- Typical attack attempts to overflow a stack-allocated buffer so as to change return address stored in stack frame.

- Changed return address transfers control to hostile code which may try `execl("/bin/sh", "/bin/sh", 0x00)`.

- If hostile code is installed in the stack, then its execution can be avoided by making the stack non-executable.

- If hostile code is installed in the heap, then execution possible.

Buffer Overflow Mitigation

- Non-executable stack.
- Address space layout randomization (ASLR).
- Network scanning to detect known buffer overflow attacks or NOP sleds.

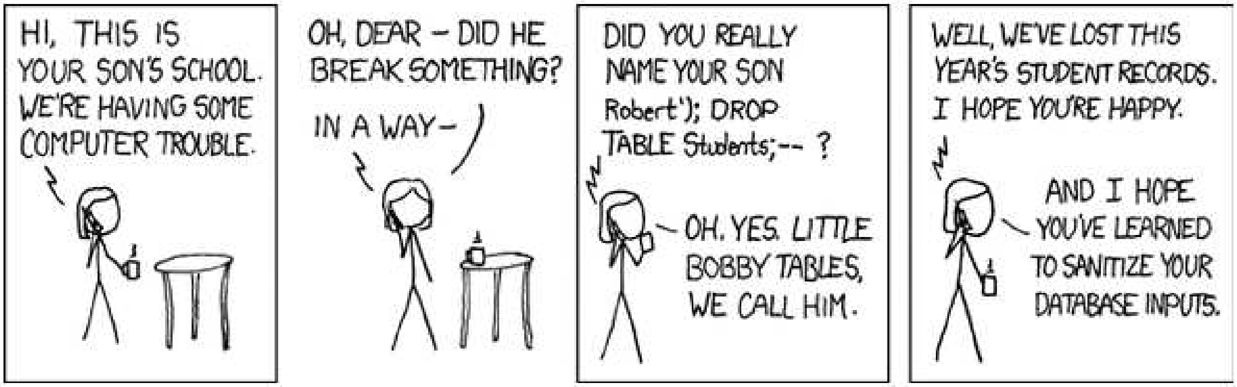
Untrusted Data Used for Executing Other Programs

Abstraction of typical unsecure code:

```
#define COMMAND       "cat "
void unsafeCat(char buf[], int bufLen) {
  strcpy(buf, COMMAND);
  printf("Enter file name: "); fflush(stdout);
  fgets(buf, bufLen - strlen(COMMAND), stdin);
  system(buf);
}
```

Assume program runs as root and user provides file name as `some_file; rm -rf /`!!

Validating Data


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

(Source: http://xkcd.com/327/)

Validating Data Continued

- Besides checking length of data (to prevent buffer overflows), must additionally validate all untrusted data.

- Safer to explicitly permit legal inputs rather than try to exclude illegal inputs.

- In C, passing untrusted unchecked data to `system()` or `popen()`, `exec()` can cause this problem.

- In other languages like Perl, shell-languages, constructs like back-quotes (capture output of a command) have this problem.

- Often a problem for CGI programs.

- A solution: Perl *taints* untrusted data and does not allow it to be passed to one of the above.

Environmental Variables Validation

- If a program runs other programs, then it needs to redefine the `PATH` variable.
- Also critical for `LD_LIBRARY_PATH` which can be used to force a program to load trojan’ed libraries (see http://linuxmafia.com/faq/Admin/ld-lib-path.html).
- Best to clear out the environment and explicitly define needed environmental variables to known safe values.

Race Conditions

- Consider program which wants to create a temporary file to append data to.
- It first checks to see if the temporary file exists and then creates it.
- In the time between the check and the creation, another program creates a symbolic link to a system file with the temporary filename.
- This can lead to corrupted binaries and a denial-of-service attack.

Avoiding Race Conditions

- Check and consequent action must be atomic.
- Preferable to use `fstat()`, `fchown()`, and `fchmod()` rather than a series of operations containing `stat()`, `chown()` and `chmod()`.

Randomness

- If a *random* cryptographic key is predictable, then cyptography can be cracked.
- Most computers are deterministic and do not have a good source of randomness (exceptions are hardware white-noise generators).
- Usually `/dev/random` abstracts out some decent source of randomness, like time between user keystrokes and hardware random number generators.

Security Organizations / Publications

- Bugtraq mailing list. Administered by securityfocus.com. Reports of security related bugs. Low signal-to-noise ratio, but has useful information.
- CERT. Federally funded. Affiliated with SEI at CMU. More selective.
- RISKS Digest. Weekly. Contains discussions on all technological risks.
- Phrack Magazine. Cracker magazine. Poor presentation, but strong on technology.

References

Text, Ch. 38.

David A. Wheeler, *Secure Programming for Linux and Unix HOWTO*, at http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO.html.

Wikipedia article on the Morris Worm.

John Viega, Gary McGraw, *Building Secure Software*, Addison-Wesley, 2002.

Peter Bright, *Anonymous speaks: the inside story of the HBGary hack*, Ars-Technica, at http://arstechnica.com/tech-policy/news/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack.ars/: strongly recommended.

Peter Bright, *With arrests, HBGary hack saga finally ends*, Ars-Technica, at http://arstechnica.com/tech-policy/news/2012/03/the-hbgary-saga-nears-its-end.ars.

Nate Anderson, *How Anonymous accidentally helped expose two Chinese hackers*, Ars-Technica at http://arstechnica.com/security/2013/02/how-anonymous-accidentally-helped-expose-two-chinese-hackers/.

There was also non-technical coverage in the mainstream media. A initial NY Times summary is a Eric Lipton and Charles Savage, *Hackers Reveal Offers to Spy on Corporate Rivals*, New York Times, Feb 11, 2011, at http://www.nytimes.com/2011/02/12/us/politics/12hackers.html?scp=1&sq=hbgary&st=cse: main-stream press coverage in the New York Times.

Humorous take on Colbert Report at http://www.theatlanticwire.com/opinions/view/opinion/Stephen-Colbert-Tackles-the-HBGary-Story-7132.