redhat.

Security Enhancements in RHEL

(Beside SELinux)

Ulrich Drepper
Consulting Engineer

Security Risks

Today's program code is often not safe; Red Hat’s goal is to harden the OS

Problems:

- private information is locally or even over the Internet readable by too many people
- communication is not safe due to missing encryption and/or protocol errors
- authentication is not at all, not correctly, or not thoroughly performed
- programs contain bugs
- once exploited, the attacker often has complete access to the application and its code, or even to the entire system

How to Mitigate the Risk

- use file system access correctly (owner and group rights, ACL)
- use encrypted communication channels
- use safe authentication, maybe with central authority
- deploy more strict access control mechanisms (rule based access control, SELinux)
- log machine activity
- change system to make exploiting program bugs harder
- create programs in a way to make exploiting them harder

Here we concentrate on the last two points

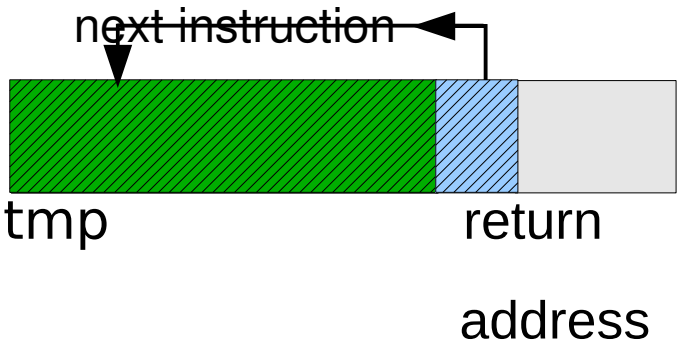
Buffer Overflow

Buggy code:

```
int match(const char *p)
{
  char tmp[MAXPASSWD];
  if (gets(tmp)==NULL)
    return 1;
  return strcmp(p, tmp);
}
```

Local variables much faster than `malloc()`

Stack layout:



Hardware Support

Processor manufacturers added to counter overflows:

- Previously could not protect against execution of readable memory
- Intel and AMD added NX support
  - In all 64-bit processors these days, in 32-bit Pentium4 since 2005Q1
  - Requires kernels with large memory support (PAE)
  - Also now available on PPC hardware
- Memory regions like stack can be marked read or read/write only
- Extends to kernel memory as well
- Creates some compatibility problems
  - Solved via ExecShield mechanisms

Exec-Shield

Developed by Red Hat, shipped in RHEL4

Goals

- mark as much memory as possible as not-executable
- keep binary compatibility
- do not limit address space

Implementation

- kernel keeps track of highest address with real code
- binaries are instrumented with information whether they need executable stacks or not
- kernel or C library make stack executable if necessary

Exec Shield (cont)

Avoid reproducible layout

- fixed addresses problem since they provide jump targets
- executable needs fixed address, but not DSOs
- stack and heap do not need fixed addresses

Exec Shield randomizes

- load addresses of DSO
- stack address, heap address

Often used DSOs are loaded in ASCII-safe area

Information exposed in /proc limited

Needs coordination with prelinking

Position Independent Executables

Exec Shield cannot randomize load address of executables

Solution: new kind of executables

- mixture between executable and DSO
- not without cost, but not as expensive as DSOs
- recommended for programs accessible through the network
- completely compatible with all Linux versions

Use -fpie/-fPIE for compilation and -pie at link-time

First PIEs shipped in RHEL3, extended to cover all network visible code in RHEL4

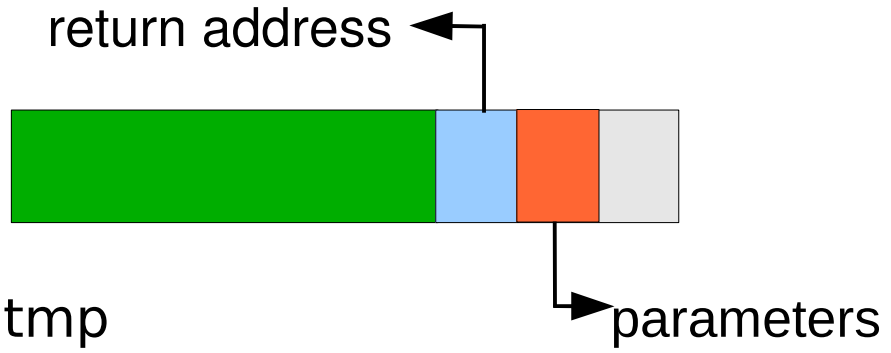
Stack Canaries

Automatically generated canary:

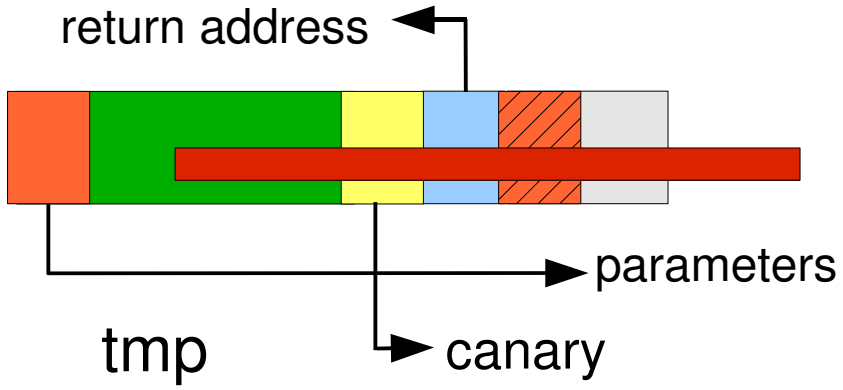
```
int match (const char *p)
{
  char tmp[MAXPASSWD];
  ...
}
```

- Canary checked before return value used
- Parameters copied to safety
- Non-arrays before arrays

Original stack layout:



Stack layout after:



Heap Overflows

The heap is the second main source of dynamic memory:

- Prone to the same overflow problems as the stack
  - Corrupt neighboring memory
  - Overwrite control data structures
- Explicit freeing introduces additional problem (double-free)
- Either problem used to allow intruder often to write arbitrary data at arbitrary address
- Much more robust implementation in glibc since RHEL4:
  - Detect most memory corruptions
  - Detect invalid pointers
  - Detect double free
  - Program is stopped before harm can be caused

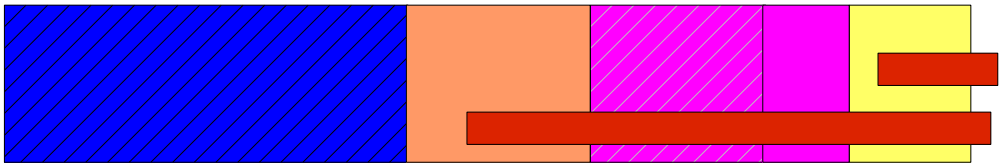
Automatic Fortification

Starting with Fedora Core 4 C sources can be automatically fortified:

- When compiled with _FORTIFY_SOURCE, checks are added transparently
  - Goal is to be unintrusive, not to catch all memory handling problems
  - Almost no performance loss
- Requires new gcc and glibc
- Works by tracking size of memory blocks whenever possible
  - Local memory allocated on stack is known to compiler
  - Calls to `malloc()` etc is recognized
- Special versions of functions writing to memory called with check boundaries before writing
- In RHEL5, all programs will be compiled this way

ELF Data Hardening

Traditional layout of an ELF file:



.bss overflow
.data overflow

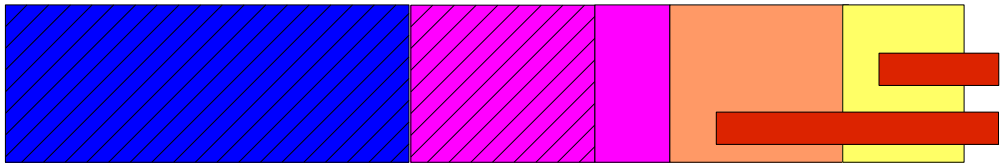
- ELF Data exposed
  - dynamic section
  - GOT and sometimes PLT

- Data unnecessarily writable

```
const char *const msgs[] = {
  “message1”, “message2”
};
```

ELF Data Hardening (cont)

Layout after changes in the linker:



- Enabled with -z relro linker option
- ELF Data before program data
- read-only section extended
- non-PLT GOT always read-only
- if -z now is additionally used entire GOT is read-only

For network accessible applications -z relro -z now advised

Questions?

Comments?

Contact:drepper@redhat.com

Papers: http://people.redhat.com/drepper/nonselsec.pdf

http://people.redhat.com/drepper/defprogramming.pdf