Operating Systems

23. Security

Paul Krzyzanowski

Rutgers University

Spring 2015

Threats

Computer security… then

Issue from the dawn of computing:

- Colossus at Bletchley Park: breaking codes
- ENIAC at Moore School: ballistic firing tables
- single-user, single-process systems
- data security needed
- physical security

Computer security… now

- Sensitive data of different users lives on the same file servers
- Multiple processes on same machine
- Authentication and transactions over network
  - open for snooping
- We might want to run other people's code in our process space
  - Device drivers, media managers
  - Java applets, Flash code
  - Downloaded software
  - … not just from trusted organizations
    (also, do you trust a trusted organization?)

Systems are easier to attack

Automation

- Data gathering
- Mass mailings

Distance

- Attack from your own home

Sharing techniques

- Virus kits, virus obfuscation kits (*crypting* services)
- Hacking tools

Penetration

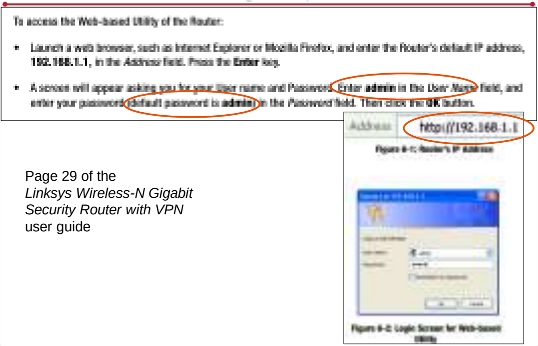
Guess a password

- system defaults, brute force,
  dictionary attack

Crack a password

- Online vs. offline
- Precomputed hashes (see rainbow tables)
  - Defense: Salt

Penetration: Guess/get a password



To access the Web-based Utility of the Router:

- Launch a web browser, such as Internet Explorer or Mozilla Firefox, and enter the Router's default IP address, **192.168.1.1**, in the *Address* field. Press the **Enter** key.
- A screen will appear asking you for your User name and Password. Enter **admin** in the *User Name* field, and enter your password (default password is **admin**) in the *Password* field. Then click the **OK** button.

Figure 6-1: Router's IP Address

Figure 6-2: Login Screen for Web-based Utility

Page 29 of the
Linksys Wireless-N Gigabit Security Router with VPN
user guide

Penetration

Social engineering

- people have a tendency to trust others
- identify corporate/school organizational structure
- facebook, twitter, blogs, personal home pages
- look through dumpsters for information
- impersonate a user
- Phishing: impersonate a company/service

Penetration

Trojan horse

- program masquerades as another
- Get the user to click on something, run something, enter data

```
All CS iLab and Graduate machine hostnames have changed
from XXX.rutgers.edu to XXX.cs.rutgers.edu.
________________________________________________________________

Use your Rutgers University username and password to log into any
CS iLab and Graduate domain machine.
________________________________________________________________

If you can not log in please verify that you have acknowledged the Department
of Computer Science Academic Integrity Policy at the following address.

You must acknowledge this policy once every academic calendar year.

http://www.cs.rutgers.edu/policies/academicintegrity/index.php?page=4

To confirm that you have submitted a record, log onto the above web
page.  A message stating "Acknowledgment received on ..." should
appear at the bottom.  If you see a message stating "To acknowledge
this policy, you must be logged in." you need to first log in.  If you
  see a message stating "You have no entry as of ...", we have no record
of your acknowledgement and you must submit a record to regain access
to your account.
________________________________________________________________
pxk@ls.cs.rutgers.edu's password:
Permission denied, please try again.
pxk@ls.cs.rutgers.edu's password:
```

Phishing

Masqueraded e-mail

Subject: Attn: Web/E-mail Account Holder,
Date: April 20, 2014

Attn: Web/E-mail Account Holder,

This message is from the University Webmail Messaging Center to all email account owners.

We are currently carrying out scheduled maintenance,upgrade of our web mail service and we are changing our mail host server,as a result your original password will be reset.

We are sorry for any inconvenience caused.

To complete your webmail email account upgrade, you must reply to this email immediately and provide the information requested below.
****
CONFIRM YOUR EMAIL IDENTITY NOW
E-mail Address:
User Name/ID:
Password:
Re-type Password:
****
Failure to do this will immediately render your email address deactivated from the University Webmail.
****
This E-mail is confidential and privileged. If you are not the intended Recipient please accept our apologies; Please do not Disclose, Copy or Distribute Information in this E-mail or take any action in Reliance on its contents: to do so is strictly prohibited and may be Unlawful.

Please inform us that this Message has gone astray before deleting it.

Thank you for your Co-operation.

Malicious Files and Attachments

Take advantage of:

- Programs that automatically open attachments
- Interfaces that hide extensions yet use them to execute a program
    - trick the user

`love-letter.txt.vbs` *looks like* `love-letter.txt`

`resume.doc.scr` *looks like* `resume.doc`

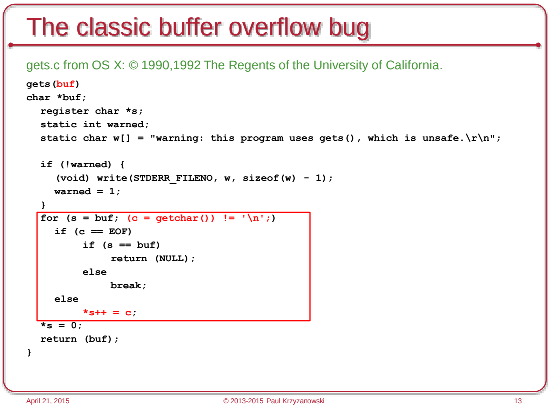
Exploiting bugs

Exploit software bugs

- Most (all) software is buggy
- Big programs have lots of bugs
    - *sendmail*, *wu-ftp*
- some big programs are *setuid* programs
    - *lpr, uucp, sendmail, mount, mkdir, eject*

Common bugs

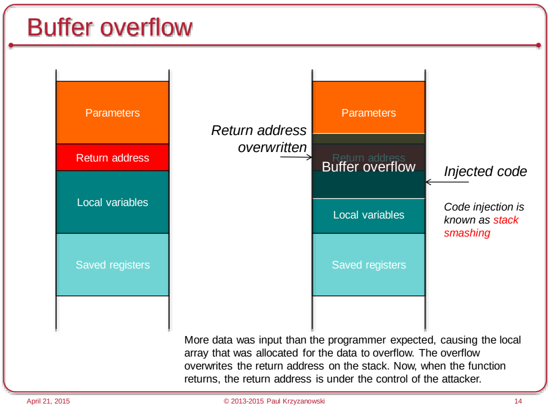
- buffer overflow
  (blindly read data into buffer)
    - e.g., *gets*
- back doors and undocumented options



The classic buffer overflow bug

gets.c from OS X: © 1990,1992 The Regents of the University of California.

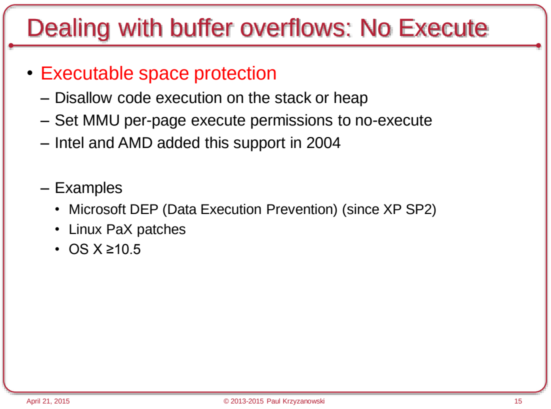
```
gets(buf)
char *buf;
  register char *s;
  static int warned;
  static char w[] = "warning: this program uses gets(), which is unsafe.\r\n";

  if (!warned) {
     (void) write(STDERR_FILENO, w, sizeof(w) - 1);
    warned = 1;
  }
  for (s = buf; (c = getchar()) != '\n';)
    if (c == EOF)
        if (s == buf)
            return (NULL);
        else
            break;
    else
        *s++ = c;
  *s = 0;
  return (buf);
}
```



Buffer overflow

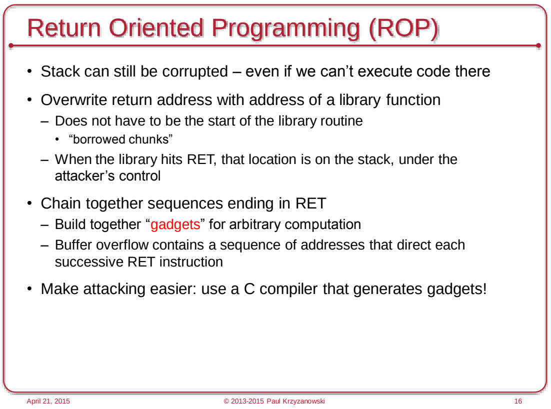
Code injection is known as stack smashing

More data was input than the programmer expected, causing the local array that was allocated for the data to overflow. The overflow overwrites the return address on the stack. Now, when the function returns, the return address is under the control of the attacker.



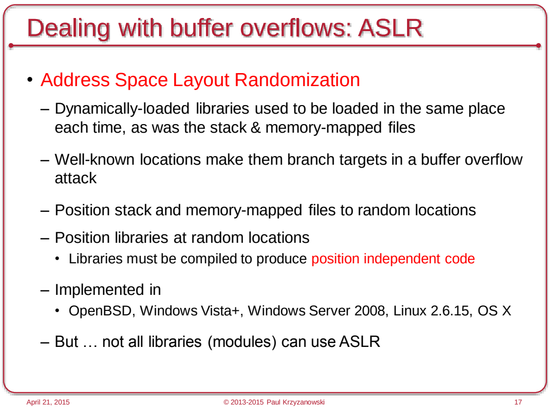
Dealing with buffer overflows: No Execute

- Executable space protection
  - Disallow code execution on the stack or heap
  - Set MMU per-page execute permissions to no-execute
  - Intel and AMD added this support in 2004
  - Examples
    - Microsoft DEP (Data Execution Prevention) (since XP SP2)
    - Linux PaX patches
    - OS X ≥10.5



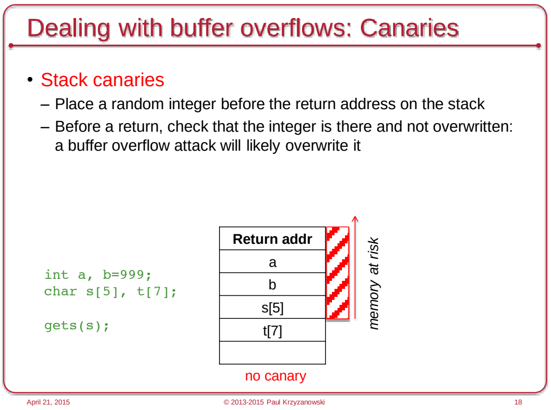
Return Oriented Programming (ROP)

- Stack can still be corrupted – even if we can't execute code there
- Overwrite return address with address of a library function
  - Does not have to be the start of the library routine
    - "borrowed chunks"
  - When the library hits RET, that location is on the stack, under the attacker's control
- Chain together sequences ending in RET
  - Build together "gadgets" for arbitrary computation
  - Buffer overflow contains a sequence of addresses that direct each successive RET instruction
- Make attacking easier: use a C compiler that generates gadgets!



Dealing with buffer overflows: ASLR

- Address Space Layout Randomization
  - Dynamically-loaded libraries used to be loaded in the same place each time, as was the stack & memory-mapped files
  - Well-known locations make them branch targets in a buffer overflow attack
  - Position stack and memory-mapped files to random locations
  - Position libraries at random locations
    - Libraries must be compiled to produce position independent code
  - Implemented in
    - OpenBSD, Windows Vista+, Windows Server 2008, Linux 2.6.15, OS X
  - But … not all libraries (modules) can use ASLR



Dealing with buffer overflows: Canaries

- Stack canaries
  - Place a random integer before the return address on the stack
  - Before a return, check that the integer is there and not overwritten: a buffer overflow attack will likely overwrite it

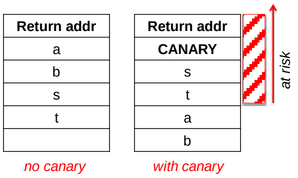
```
int a, b=999;
char s[5], t[7];

gets(s);
```

no canary

Dealing with buffer overflows: Canaries

- Stack canaries
  - Place a random integer before the return address on the stack
  - Before a return, check that the integer is there and not overwritten: a buffer overflow attack will likely overwrite it
  - Allocate arrays into higher memory in the stack so they won’t clobber other automatic (stack-based) variables

```
int a, b=999;
char s[5], t[7];

gets(s);
```



Return addr
a
b
s
t

no canary

Return addr
CANARY
s
t
a
b

with canary

at risk

Virus

- Does not run as a self-contained process
- Code is attached onto another program or script

- File infector
  - primarily a problem on systems without adequate protection mechanisms
- Boot-sector
- Macro (most common form: visual basic scripting)
- Hypervisor
  - intercept traps and privileged instructions from OS

Virus scanning

- Search for a “signature”
  - Bunch of bytes present in a virus that (we hope!) is unique to the virus and not any legitimate code
  - *NOT a cryptographic signature!*

- Some viruses are encrypted
  - Signature is either the code that does the decryption or the scanner must be smart enough to decrypt the virus
  - Crypting service: obfuscates malware to escape virus detectors

- Some viruses mutate to change their code every time they infect another system
  - Run the code through an emulator to detect the mutation

Virus scanning

- You don’t want to scan through hundreds of thousands of files
  - Search in critical places likely to be infected (e.g., \windows\system32 or removable media)

- Passive disk scan vs. active I/O scan … or both
- “Zero-Day Threats”: new virus – signature unknown
- Virus scanning is largely ineffective now
  - Estimates are that only 10-30% of new viruses in the network are detectable
  - Malware detection/prevention has to rely on other mehtods

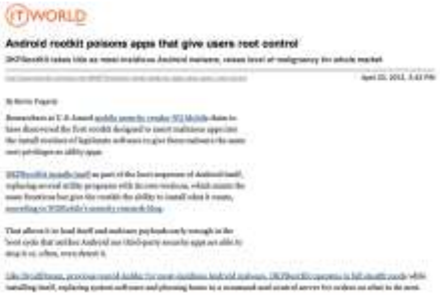
Key loggers

- Record every keystroke
- Windows *hook* mechanism
  - Procedure to intercept message traffic before it reaches a target windows procedure
  - Can be chained
  - Installed via *SetWindowsHookEx*
  - WH_KEYBOARD and WH_MOUSE
    - Capture key up, down events and mouse events
- Hardware loggers

Rootkits

- Replacement commands (or standard shared libraries or OS components) to hide the presence of an intruder
  - *ps, ls, who, netstat, …*
- Hide the presence of a user or additional software (backdoors, key loggers, sniffers)
- Now the OS can no longer be trusted!
- Examples
  - Lenovo Superfish adware (2014)
    - Preinstalled self-signed root certificate
    - Allows anyone on your network to silently intercept HTTPS communications
  - Sony BMG DRM rootkit (October 2005)
    - Creates hidden directory; installs several of its own device drivers; reroutes Windows system calls to its own routines
    - Intercepts kernel-level APIs and disguises its presence with cloaking (hides sys files)
  - Carrier IQ (December 2011)
    - Software for cell phone analytics – designed to be undetectable
    - Installed on Sprint, HTC, Apple (iPhone ≤4), Samsung, BlackBerry, ….

Rootkits



ITWORLD

Android rootkit poisons apps that give users root control

DKFBootKit takes title as most malicious Android malware, raises level of malignancy for whole market

[illegible] April 22, 2012, 3:43 PM

By Kevin Fogarty

Researchers at U.S.-based mobile security vendor NQ Mobile claim to have discovered the first rootkit designed to insert malicious apps into the install routines of legitimate software to give themselves the same root privileges as utility apps.

DKFBootKit installs itself as part of the boot sequence of Android itself, replacing several utility programs with its own versions, which mimic the same functions but give the rootkit the ability to install when it wants, according to NQMobile's security research blog.

That allows it to load itself and malware payloads early enough in the boot cycle that neither Android nor third-party security apps are able to stop it or, often, even detect it.

Like DroidDream, previous record holder for most dangerous Android malware, DKFBootKit operates in full stealth mode while installing itself, replacing system software and phoning home to a command and control server for orders on what to do next.

Dealing With Rootkits

- Hope you don't get one!
- Restrict permissions to modify system files
  - Makes it hard to install a rootkit in the first place
  - … But users often grant permissions during installation
    - And permissions may be needed for drivers
- Signed software and operating system components
  - Microsoft Vista & Windows 7:
    - Requires kernel-mode software to have a digital signature (x64-based systems only)
- Tripwire
  - Software to monitor for changes in files and components in a system

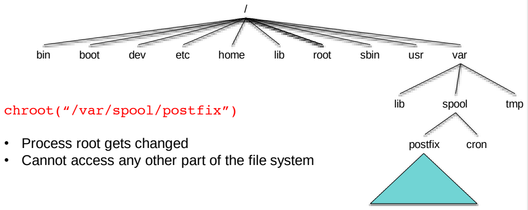
Sandboxing

Restricting what apps can do

- Traditional OS approach to protection
  - Privileges are an attribute of the user
  - Protection mechanisms were designed to place restrictions on *other* users
  - Every process the user runs has the same privileges
    - A program can delete all of <u>your</u> files
      … or upload them to a remote server
- *How do we protect ourselves from malicious applications?*
- Mandatory Access Control (MAC)? Yes … in a way…
  - Traditional forms of MAC (e.g., Bell-LaPadula)
    - Restricts information flow between *<u>classes</u>* of users
    - Doesn't solve the problem
  - Enforces ability of processes to access objects
    - Discretionary Access Control allows users/processes to make policy changes

A basic approach: chroot jail

chroot system call: change the root directory for a process



```
chroot("/var/spool/postfix")
```

- Process root gets changed
- Cannot access any other part of the file system

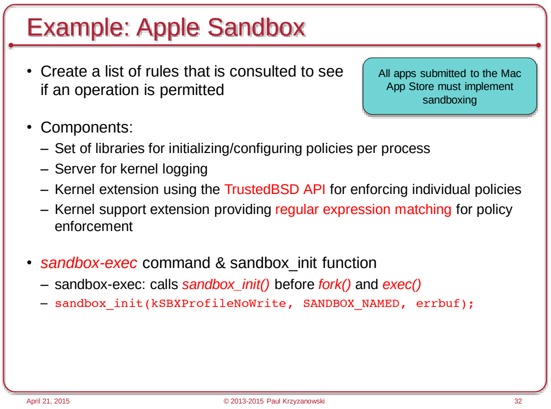
Sandboxing

Per process access control: restrict what applications can do

> For example:
> – *An app shouldn't read /etc/passwd*
> – *An app shouldn't write any files*
> – *An app should not establish a TCP/IP connection*
> – *An app shouldn't access your contacts*

Sandboxing is used in:

- Google Chrome browser
- Microsoft Office 2010 Protected View
- Android Application Sandbox
- Apple iOS sandboxing
- Apple XNU Sandbox framework
- SELinux
- FreeBSD TrustedBSD system (mostly MAC)
- Windows: access control at kernel object level with inherited permissions

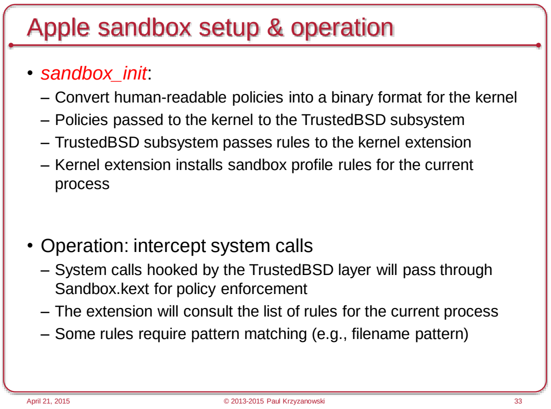


Example: Apple Sandbox

- Create a list of rules that is consulted to see if an operation is permitted

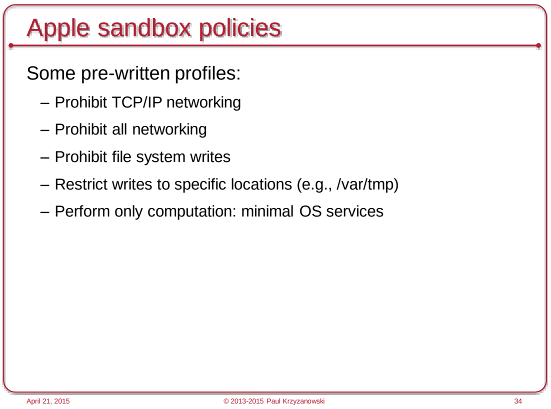
All apps submitted to the Mac App Store must implement sandboxing

- Components:
  - Set of libraries for initializing/configuring policies per process
  - Server for kernel logging
  - Kernel extension using the TrustedBSD API for enforcing individual policies
  - Kernel support extension providing regular expression matching for policy enforcement
- *sandbox-exec* command & sandbox_init function
  - sandbox-exec: calls *sandbox_init()* before *fork()* and *exec()*
  - `sandbox_init(kSBXProfileNoWrite, SANDBOX_NAMED, errbuf);`



Apple sandbox setup & operation

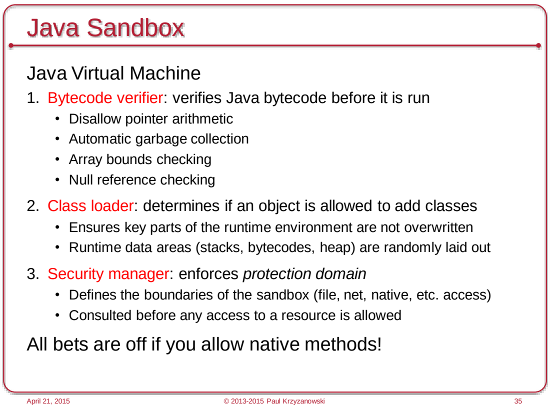
- *sandbox_init*:
  - Convert human-readable policies into a binary format for the kernel
  - Policies passed to the kernel to the TrustedBSD subsystem
  - TrustedBSD subsystem passes rules to the kernel extension
  - Kernel extension installs sandbox profile rules for the current process
- Operation: intercept system calls
  - System calls hooked by the TrustedBSD layer will pass through Sandbox.kext for policy enforcement
  - The extension will consult the list of rules for the current process
  - Some rules require pattern matching (e.g., filename pattern)



Apple sandbox policies

Some pre-written profiles:

- Prohibit TCP/IP networking
- Prohibit all networking
- Prohibit file system writes
- Restrict writes to specific locations (e.g., /var/tmp)
- Perform only computation: minimal OS services

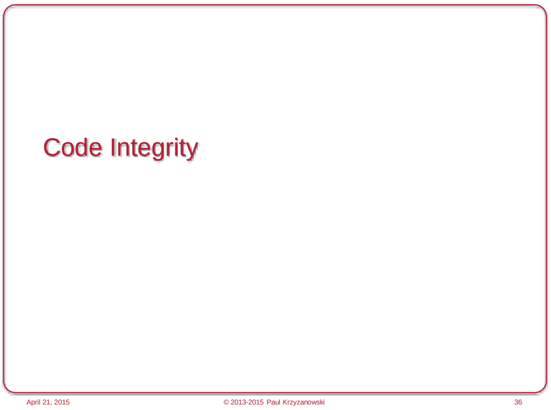


Java Sandbox

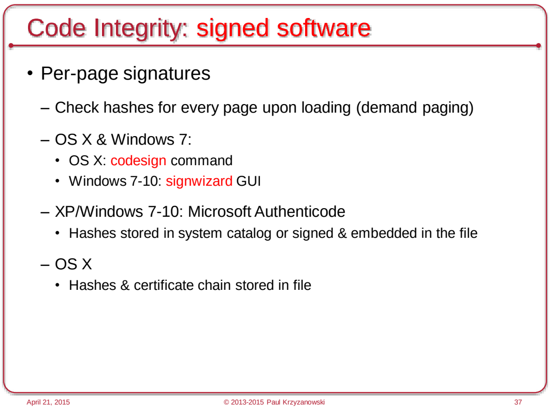
Java Virtual Machine

1. Bytecode verifier: verifies Java bytecode before it is run
   - Disallow pointer arithmetic
   - Automatic garbage collection
   - Array bounds checking
   - Null reference checking
2. Class loader: determines if an object is allowed to add classes
   - Ensures key parts of the runtime environment are not overwritten
   - Runtime data areas (stacks, bytecodes, heap) are randomly laid out
3. Security manager: enforces *protection domain*
   - Defines the boundaries of the sandbox (file, net, native, etc. access)
   - Consulted before any access to a resource is allowed

All bets are off if you allow native methods!

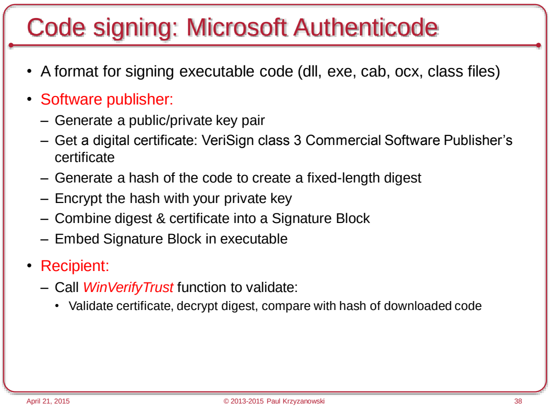


Code Integrity



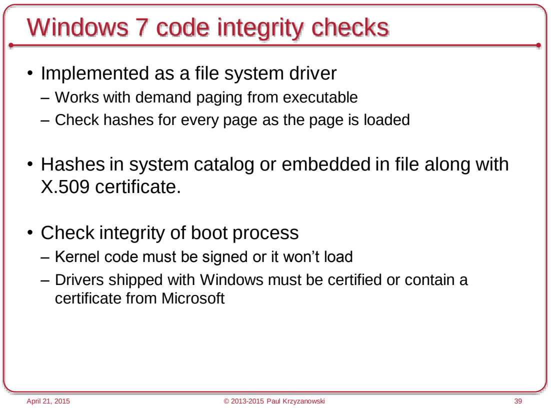
Code Integrity: signed software

- Per-page signatures
  - Check hashes for every page upon loading (demand paging)
  - OS X & Windows 7:
    - OS X: codesign command
    - Windows 7-10: signwizard GUI
  - XP/Windows 7-10: Microsoft Authenticode
    - Hashes stored in system catalog or signed & embedded in the file
  - OS X
    - Hashes & certificate chain stored in file



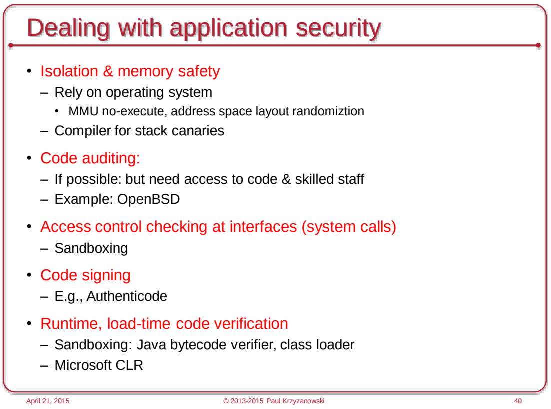
Code signing: Microsoft Authenticode

- A format for signing executable code (dll, exe, cab, ocx, class files)
- Software publisher:
  - Generate a public/private key pair
  - Get a digital certificate: VeriSign class 3 Commercial Software Publisher's certificate
  - Generate a hash of the code to create a fixed-length digest
  - Encrypt the hash with your private key
  - Combine digest & certificate into a Signature Block
  - Embed Signature Block in executable
- Recipient:
  - Call *WinVerifyTrust* function to validate:
    - Validate certificate, decrypt digest, compare with hash of downloaded code



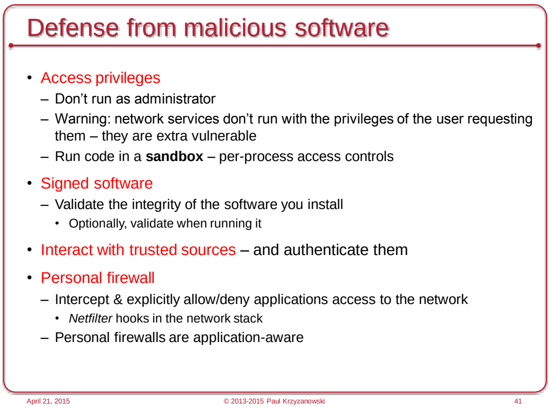
Windows 7 code integrity checks

- Implemented as a file system driver
  - Works with demand paging from executable
  - Check hashes for every page as the page is loaded
- Hashes in system catalog or embedded in file along with X.509 certificate.
- Check integrity of boot process
  - Kernel code must be signed or it won't load
  - Drivers shipped with Windows must be certified or contain a certificate from Microsoft



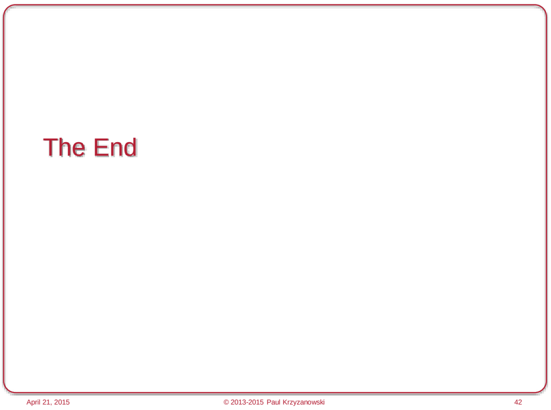
Dealing with application security

- Isolation & memory safety
  - Rely on operating system
    - MMU no-execute, address space layout randomiztion
  - Compiler for stack canaries
- Code auditing:
  - If possible: but need access to code & skilled staff
  - Example: OpenBSD
- Access control checking at interfaces (system calls)
  - Sandboxing
- Code signing
  - E.g., Authenticode
- Runtime, load-time code verification
  - Sandboxing: Java bytecode verifier, class loader
  - Microsoft CLR



Defense from malicious software

- Access privileges
  - Don't run as administrator
  - Warning: network services don't run with the privileges of the user requesting them – they are extra vulnerable
  - Run code in a **sandbox** – per-process access controls
- Signed software
  - Validate the integrity of the software you install
    - Optionally, validate when running it
- Interact with trusted sources – and authenticate them
- Personal firewall
  - Intercept & explicitly allow/deny applications access to the network
    - *Netfilter* hooks in the network stack
  - Personal firewalls are application-aware



The End