CSE331 Computer Security Fundamentals

Recap

Nick Nikiforakis

nick@cs.stonybrook.edu
Low-level
• The browser is a program written in memory-unsafe languages
• Can have plugins written in memory-unsafe languages (Flash, Java, etc.)

Attacks
• Buffer overflows and overreads
  • Stack
  • Heap
• Integer overflows
• Execution of attacker-provided/desired code
  • Shellcode
  • Ret2libc
  • ROP

Web
• The browser is a container and execution environment of web applications

Attacks
• Against the web application
  • Client-side
  • Server-side
• Against the user
• Against the browser
  • Against web login
  • Against the underlying OS

Security basics (Principles / Ideas / Reoccurring mistakes)
What is computer security?

- Everyone has their own definition
  - No single one is perfect
- Computer security deals with protecting data, programs, and systems against intelligent adversaries.

- Safety vs Security
  - What’s the difference between the two?
  - Do they interact?
Why is it hard?

• It is hard to get security right because:
  – Security is hard to test for
    • Testing correctness versus security
  – It requires a deep understanding of all technologies involved in the design and implementation of a system
    • Really hard in large real systems
  – Users are typically the weakest link
  – **Asymmetry** between attack and defense
CIA

• What do we mean by “protection”? 
• Security is about CIA

– **Confidentiality**: Keeping data and resources hidden or protected from unauthorized disclosure

– **Integrity**: Data and Programs are modified in specified and authorized ways. Data integrity and origin integrity.

– **Availability**: Systems and networks are available for use by legitimate users
Definitions

- Computer Security is always related to a specific attacker/threat model
- **Threat**: A potential violation of security by an attacker
- Is this program/website secure? -> Who is the attacker? Who are you defending against?
  - Your bf/gf
  - A script kiddie
  - An opportunistic hacker
  - A hacker collective who has targeted you
  - A government
Definitions

• **Threat model**: A set of possible attacks and attackers that a system tries to protect against.
  – Physical attacks
  – Outsider attacks
  – Insider attacks (In your local network)
  – Attacks over the Internet
  – ...
Policies and mechanisms

- **Security policy**: A statement of what is and what is not allowed
  - E.g. Only computer science students are allowed to enter the lab
- **Security mechanism**: A method, tool, or procedure for enforcing a security policy
  - How do we enforce the above desired policy?

- A security policy can be implemented by a variety of security mechanisms
When faced with an attacker...

• A security mechanism can:
  – Prevent the attack
    • Attack will fail
  – Detect the attack
    • Attack may succeed but we know about it and can thus react
  – Recover from the attack
    • Stop the attack and go back to a “healthy” state
    • System functions correctly even in the presence of attacks
Authentication
Password-based Authentication

• Passwords are key to the process of authentication

  – Authentication is at the heart of security

How do I know you are, who you say your are?

Are you allowed to do the thing you are asking?
User Authentication

• Authentication based on:
  – What you know (password, answer to security question, personal image etc.)
  – What you have (smartcard, hardware token, etc.)
  – Who you are (biometrics)
  – Where you are (IP address, GPS location)
    • Less popular, but smartphones could change this
Attackers

• What is the threat model?
  – Online attacker
    • Tries to login to a service by iteratively trying passwords and looking whether he was successful
  – Offline attacker
    • Stole password database and tries to recover the, hopefully protected, passwords
      – Also known as a “dictionary attack”
  – Against one user
  – Against all/any user
How do attackers use passwords?

• Once a database of credentials is leaked, attackers can use them in multiple ways
  – Extract emails and usernames
    • Chances are that users are reusing the same username/email address in other unrelated services
  – Learn what are the most common passwords that most users use
  – Learn what are the passwords that specific users use

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:alice@gmail.com">alice@gmail.com</a></td>
<td>ilovedogs</td>
</tr>
<tr>
<td><a href="mailto:bob@yahoo.com">bob@yahoo.com</a></td>
<td>Password!</td>
</tr>
<tr>
<td><a href="mailto:eve@outlook.com">eve@outlook.com</a></td>
<td>1q2w3e4r</td>
</tr>
<tr>
<td><a href="mailto:john@stonybrook.edu">john@stonybrook.edu</a></td>
<td>g@rfield1</td>
</tr>
</tbody>
</table>
Credential stuffing

• Attackers build programs that try these credentials against other services
  – These programs act like regular users trying to log in
  – Attackers bet on users reusing their passwords

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:alice@gmail.com">alice@gmail.com</a></td>
<td>ilovedogs</td>
</tr>
<tr>
<td><a href="mailto:bob@yahoo.com">bob@yahoo.com</a></td>
<td>Password!</td>
</tr>
<tr>
<td><a href="mailto:eve@outlook.com">eve@outlook.com</a></td>
<td>1q2w3e4r</td>
</tr>
<tr>
<td><a href="mailto:john@stonybrook.edu">john@stonybrook.edu</a></td>
<td>g@rfield1</td>
</tr>
</tbody>
</table>

supercutecats.com
Password Hashing

- Instead of user password, store $\text{Hash(password)}$
- When user enters a password, compute its hash and compare with the entry in the password file
  - System does not store actual passwords
  - Cannot go from hash to password
    - ... except by guessing the password
- Hash function $H$ must have some properties
  - Given $H(\text{password})$, hard to find any string $X$ such that $H(X)=H(\text{password})$ - why?
Sample Cryptographic hash functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Year of release</th>
<th>Digest size (output size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5 (Media Digest 5)</td>
<td>1992</td>
<td>128-bit</td>
</tr>
<tr>
<td>SHA-1 (Secure Hash Algorithm 1)</td>
<td>1995</td>
<td>160-bit</td>
</tr>
<tr>
<td>SHA-256 (Part of the SHA-2 family)</td>
<td>2001</td>
<td>256-bit</td>
</tr>
</tbody>
</table>

MD5(“helloworld”) = d73b04b0e696b0945283defa3eee4538
SHA-1(“helloworld”) = e7509a8c032f3bc2a8df1df476f8ef03436185fa
SHA-256(“helloworld”) = 8cd07f3a5ff98f2a78cfc366c13fb123eb8d29c1ca37c79df190425d5b9e424d
Examples

SHA1(“mysecretpassword”) = 08cd923367890009657eab812753379bdb321eeb
SHA1(“mysecretpassword”) = 0c894b9cd0fef7d1ccfe0729d5ff7af9509731ed
SHA1(“mysecretpassword”) = 27c2d31b648cf7773032d1a06c8ee610c3f5b32c

Small changes in input

Large differences in output

This is called the “avalanche” effect
Hashing vs. Encryption

• Hashing is one-way. There is no “uh-hashing”!
  – A ciphertext can be decrypted with a decryption key... hashes have no equivalent of “decryption”

• Hash(x) looks “random”, but can be compared for equality with Hash(x’)
  – Hash the same input twice → same hash value

• Cryptographic hashes are also known as “cryptographic checksums” or “message digests”
Salting

• Instead of just hashing the user’s password, hash the user’s password when concatenated with a per-user random value

<table>
<thead>
<tr>
<th>Username</th>
<th>Salt</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>nick</td>
<td>199654</td>
<td>a4de0a7f8905a0e3832cca84eff4f76a465c85a3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA1(“199654mysecretpassword”)</td>
</tr>
<tr>
<td>nick</td>
<td>08cd923367890009657eab812753379bdb321eeb</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA1(“mysecretpassword”)</td>
</tr>
</tbody>
</table>
Hash stretching

• Why restrict ourselves to only one hash operation?

• If we perform multiple hashing rounds:
  – An attacker would need significantly more resources per cracking attempt
  – A server can still cope with the increased load because users are not authenticating all at the same time

• Standardized multi-round hashing algorithms
  – PBKDF2, brypt, scrypt
PBKDF2 + HMAC-SHA-256

PBKDF2 using XOR to combine 10,000 successive HMAC-SHA-256 outputs into a final hash

Image source: https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/
Replay attacks and possible solutions

• The standard, password-based authentication is vulnerable to **replay attacks**
  – A network attacker can see the password in traffic, and then later reuse to authenticate as the victim

• We can encrypt the entire channel to protect against this (explore this later in class) but we can also tackle it with **one-time passwords (OTP)**
Challenge-Response

Why is this better than the password over a network?
Something you have

• Things one can have
  – Access to your smartphone
    • Has gained a lot of traction recently due to popular web applications (Gmail, Twitter, etc.) supporting it
  – A bank card
  – A security token
    • A piece of hardware containing crypto that either generates one-time passwords or does a challenge-response protocol
  – A badge

• Problems
  – Stolen / forgotten / lost / duplicated
    • Higher cost to change than passwords
  – Cost of user education and support
Something you are

• **Biometrics**
  – Fingerprints
  – Palms
  – Face
  – Iris/Retina scanning
  – Voice
  – How you walk? How you type? How you swipe?
    • Research in continuous authentication

• **Benefits**
  – Nothing to remember
  – Passive (nothing to type, always carrying them around)
  – Can’t share
  – Can be fairly unique
Access control
### Access Control Matrix Description

- **Objects** $O = \{ o_1, ..., o_m \}$  
  - All protected entities.
- **Subjects** $S = \{ s_1, ..., s_n \}$  
  - Active entities, $S \subseteq O$
- **Rights** $R = \{ r_1, ..., r_k \}$
- **Entries** $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, ..., r_y \}$ means subject $s_i$ has rights $r_x, ..., r_y$ over object $o_j$
Access Control Lists (ACLs)

• Implement ACM by column.
  – Access control by object.

• Example: UNIX ACLs
  – Short “rwx” user/group/other.
  – Long POSIX ACLs.

• ACL is stored “close” to the object

<table>
<thead>
<tr>
<th>User</th>
<th>audit data</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>rw</td>
</tr>
<tr>
<td>alice</td>
<td>r</td>
</tr>
<tr>
<td>bob</td>
<td></td>
</tr>
</tbody>
</table>
Capabilities

- Implement ACM by row, instead of by column
- Access Control associated with subject.

Example: UNIX file descriptors
- System checks ACL on file open, returns $fd$.
- Process subsequently uses $fd$ to read and write file.
- If ACL changes, process still has access via $fd$.

<table>
<thead>
<tr>
<th>User</th>
<th>ls</th>
<th>homedir</th>
<th>rootdir</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>rx</td>
<td>rw</td>
<td>r</td>
</tr>
</tbody>
</table>
Mandatory Access Control

- Mandatory Access Control (MAC) is a more restrictive scheme that does not allow users to define permissions on files, regardless of ownership. Instead, security decisions are made by a central policy administrator.

  - Each security rule consists of a **subject**, which represents the party attempting to gain access, an **object**, referring to the resource being accessed, and a series of permissions that define the extent to which that resource can be accessed.
3 Formal MAC Examples

• Bell La Padula model for the confidentiality of data
• Biba model for the integrity of data
• Chinese wall model for protecting the confidentiality (and integrity) of data from conflict of interest
Role-Based Access Control

• The role-based access control (RBAC) model can be viewed as an evolution of the notion of group-based permissions in file systems.
  – Non-DAC model
  – “Middle of the road” between MAC and DAC

• An RBAC system is defined with respect to an organization, such as company, a set of resources, such as documents, print services, and network services, and a set of users, such as employees, suppliers, and customers.
Visualizing Role Hierarchy
Attribute-based Access Control

- Attribute-based Access Control (ABAC) is more recent than the access control schemes we have looked at so far
- Access rights are granted to users through policies that combine many attributes together
- Attributes can be based:
  - On the subject trying to perform the access
  - On the object being accessed
  - On the environment over which the access is happening
ABAC Scheme

Image source: http://www.axiomatics.com/attribute-based-access-control.html
Covert channels
Covert channels

- Covert channels are used as a means of communication between two processes.
- The processes may be:
  - Authorized to communicate, but not in the way they actually are
  - Prohibited from communicating
- One process is a Trojan
  - Transmits data in a covert manner
- The other process is a Spy
  - Receives the data sent by the Trojan
Criteria for the presence of covert channels

**Covert Storage Channel**

- Sender and receiver have access to some attribute of a shared object
- Sender must be able to modify the attribute
- Receiver must be able to view that attribute
- There must be a mechanism for initiating both processes and sequencing their actions to the shared resource

**Covert Timing Channel**

- Sender and receiver have access to some attribute of a shared object
- Both have access to a time reference (e.g. real-time clock)
- The sender must be able to control the timing of the detection of a change in the shared attribute
- There must be a mechanism for initiating both processes and sequencing their actions to the shared resource
General Characteristics

• Characteristics of a covert channel:
  – Existence: Is there a channel present that can be abused?
  – Bandwidth: How many bits per second can be transmitted?
  – Noise-levels: How much noise is there on this channel?

• Once we find one:
  – Eliminate it
  – Reduce Bandwidth / Increase Noise
  – Monitor for exploitation
Network channels

- **Definition:** Mechanisms for sending information without the knowledge of the network administrator or other users.

- **Key idea:** Exploit new/existing outgoing communication channel to covertly leak information

- The Trojan is local but the Spy is remote

- As with local channels, there is a very wide range of available techniques
Software Security
X86 Processor

• Most common processor type in desktop/laptop/server environments

• X86 Instruction set
  – The CPU’s language
  – Operation <destination>, <source>

• Native programs (C, C++, etc.) are eventually compiled to x86 instructions

• 32bit, 64bit
GDB

- GDB: Gnu Project Debugger
- Start/Stop/Pause an executing program
- Investigate/Alter
  - Registers
  - Memory locations
  - Call functions
  - ...

- 44
## Process Memory Layout

<table>
<thead>
<tr>
<th>high mem</th>
<th>low mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv, env</td>
<td>text</td>
</tr>
<tr>
<td>stack</td>
<td></td>
</tr>
<tr>
<td>heap</td>
<td></td>
</tr>
<tr>
<td>bss</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
</tbody>
</table>

### Notes

- **Argv/Env**: CLI args and environment
- **Stack**: generally grows downwards
- **Heap**: generally grows upwards
- **BSS**: uninitialized global data
- **Data**: initialized global data
- **Text**: read-only program code
b() {
  ... 
}

a() {
  b();
}

main() {
  a();
}
What is a Stack Frame?

Block of stack data for one procedure call.

Frame pointer (FP) points to frame:

- Use offsets to find local variables.
- SP continually moves with push/pops.
- FP only moves on function call/return.
- Intel CPUs use ebp register for FP.
## Stack at Function Start

<table>
<thead>
<tr>
<th>old stack frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter #N</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td>parameter #1</td>
</tr>
<tr>
<td>return address</td>
</tr>
<tr>
<td>old FP</td>
</tr>
<tr>
<td>Space for local vars</td>
</tr>
<tr>
<td>Space for local vars</td>
</tr>
</tbody>
</table>

- EBP (Base Pointer)
- ESP (Stack Pointer)
Stack based buffer overflow

```
f0:
    ...
    call f1
    ...
```

```
f1:
    buffer[]
    overflow()
    ...
```
Stack based buffer overflow

f0:
  ...
  call f1
  ...

f1:
  buffer[]
  overflow()
  ...

Stack

Return address f0
Saved Frame Ptr f0
Local variables f0

Arguments f1
Return address f1
Saved Frame Ptr f1

Space for buffer
Injected Code

Stack based buffer overflow

Stack

Return address f0
Saved Frame Ptr f0
Local variables f0
Arguments f1
Overwritten address
Overwritten address
Injected Code

f0:

... call f1
...

f1:

buffer[]
overflow()
...

IP
FP
SP
Stack canaries

```
f0:
  ...
  call f1
  ...
```

```
f1:
  buffer[]
  overflow()
  ...
```
Stack based buffer overflow

f0:
  ...
call f1
  ...

f1:
  buffer[]
  overflow()
  ...

Stack:
- Return address f0
- Canary
- Saved Frame Ptr f0
- Arguments f1
- Return address f1
- Canary
- Saved Frame Ptr f1

IP
FP
SP
Stack based buffer overflow

f0:
  ...
  call f1
  ...

f1:
  buffer[]
  overflow()
  ...

Stack

Return address f0
Saved Frame Ptr f0
Canary
Arguments f1
Overwritten address
Canary
Frame-pointer overwriting

• Canary is protecting the return address, so we can’t overflow into that

• But what about the saved-frame pointer?
Frame-pointer overwriting

- Canary is protecting the return address, so we can’t overflow into that.

- But what about the saved-frame pointer?
Indirect-pointer overwriting

- If there are pointers present between the overflowed buffer and the canary, these can also be abused.
- Pointers can be used to write memory (bypass canary).
- Read-out sensitive memory.
Data-only attacks

• Non-control variables can also lead to exploitation.

• Examples:
  – Filenames
  – Filepaths
  – Passwords
  – Integers controlling sensitive operations
Heap based buffer overflow

• If a program contains a buffer overflow vulnerability for a buffer allocated on the heap, there is no return address nearby.

• So attacking a heap based vulnerability requires the attacker to overwrite other code pointers.

• We look at two examples:
  – Overwriting a function pointer
  – Overwriting heap metadata
Heap management in dlmalloc

Dlmalloc maintains a doubly linked list of free chunks

When chunk c gets unlinked, c’s backward pointer is written to *(forward pointer+12)

Or: green value is written 12 bytes above where red value points
Exploiting a buffer overrun

Green value is written 12 bytes above where red value points

A buffer overrun in d can overwrite the red and green values

• Make Green point to injected code
• Make Red point 12 bytes below a function return address
Exploiting a buffer overrun

Top Heap grows with brk()

Green value is written 12 bytes above where red value points

Net result is that the return address points to the injected code
Format-string vulnerabilities

• Attacker is able to retrieve information from memory by constructing format-strings that are then fed into printf and printf-like functions

Nick
  Welcome user Nick
Nick %d
  Welcome user: Nick -6037020
Nick %x %x %x %x
  Welcome user: Nick ff9bd6b4 f7771ac0 8048230 ff9bd6a8
Where Does an Integer Overflow Matter?

- Allocating spaces using calculation.
- Calculating indexes into arrays
- Checking whether an overflow could occur
Non-executable data

• Direct code injection attacks at some point execute data
  – Most programs never need to do this

• Hence, a simple countermeasure is to mark data memory (stack, heap, ...) as non-executable
  – Write-XOR-Execute, DEP

• This counters direct code injection
  – In principle, this countermeasure may also break certain legacy applications
Randomization of Memory Address Space

If you don’t know where a buffer/function is, then your exploit is unreliable.

ASLR (Address Space Layout Randomization)
- Stack location.
- Shared library locations.
- Heap location.

PIE: Position Independent Executable
- Default format: binary compiled to work at an address selected when program was compiled.
- Gcc can compile binaries to be freely relocatable throughout address space.
  - gcc flags: -fpie -pie
  - Program loaded at different address for each invocation.
Return-into-libc: overview

Stack

- Params for f1
- Return addr
- Params for f2
- Return addr
- Params for f3
- Return addr

Code Memory

- f1
  - return
- f2
  - return
- f3
  - return

SP

IP
Return-into-libc: overview

Stack
- Params for f1
- Return addr
- Params for f2
- Return addr
- Params for f3
- Return addr

Code Memory
- f1
- return
- f2
- return
- f3
- return

IP
Return-into-libc: overview
Return-into-libc: overview

Stack

- Params for f1
- Return addr
- Params for f2
- Return addr
- Params for f3
- Return addr

Code Memory

- f1
- .
- .
- return
- f2
- .
- .
- return
- f3
- return
- .
- return
Return-into-libc: overview

Stack

Params for f1

Return addr

Params for f2

Return addr

Code Memory

f1

.. return

f2

.. return

f3

return

.. return
Return-into-libc: overview

Stack
- Params for f1
- Return addr
- Params for f2
- Return addr

Code Memory
- f1
  - return
- f2
  - return
- f3
  - return
  - return
  - return

SP

IP
Return-into-libc: overview

Stack

- Params for f1
- Return addr

Code Memory

- f1
  - return
- f2
  - return
- f3
  - return
Return Oriented Programming

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80abdea0</td>
<td>EAX = SMTH</td>
</tr>
<tr>
<td>0x309</td>
<td>EBX = SMTH</td>
</tr>
<tr>
<td>0x80345677</td>
<td>ECX = SMTH</td>
</tr>
<tr>
<td>&amp;&quot;/tmp/lala&quot;</td>
<td></td>
</tr>
<tr>
<td>0x80abddaa</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0x80abcdee</td>
<td></td>
</tr>
<tr>
<td>0x80345678</td>
<td>Ret;</td>
</tr>
<tr>
<td>0x08abcdee</td>
<td>Ret;</td>
</tr>
<tr>
<td>0x08abcdef</td>
<td>Ret;</td>
</tr>
<tr>
<td>0x80abddaa</td>
<td>Ret;</td>
</tr>
<tr>
<td>0x80abdea0</td>
<td>Int 0x80;</td>
</tr>
</tbody>
</table>

ESP
Return Oriented Programming

EAX = SMTH
EBX = SMTH
ECX = SMTH

0x80abdea0
0x309
0x80345677
"/tmp/lala"
0x80abddaa
8
0x80abcdee

0x80345677: pop $ecx;
0x80345678: ret;
...
0x80abcdee: pop $eax;
0x80abcdef : ret;
...
0x80abddaa: pop $ebx;
0x80abddab: ret;
...
0x80abdea0: int 0x80;
...
Return Oriented Programming

<table>
<thead>
<tr>
<th>High</th>
<th>Low</th>
<th>EIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x80abdea0</td>
<td>0x80abddaa</td>
<td>0x80abddab: ret;</td>
</tr>
<tr>
<td>0x309</td>
<td>8</td>
<td>0x80abcdee: pop $eax;</td>
</tr>
<tr>
<td>0x80345677</td>
<td>&amp;”/tmp/lala”</td>
<td>0x08abcdef : ret;</td>
</tr>
<tr>
<td></td>
<td>0x80abddaa</td>
<td>0x80abddaa: pop $ebx;</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0x80abdde0: int 0x80;</td>
</tr>
</tbody>
</table>

EAX = 8
EBX = SMTH
ECX = SMTH

0x80abdea0: int 0x80;
Return Oriented Programming

EAX = 8
EBX = SMTH
ECX = SMTH

...
Return Oriented Programming

EAX = 8
EBX = &"/tmp..."
ECX = SMTH

0x80abdea0
0x309
0x80345677
&"/tmp/lala"
0x80abddaa
8
0x80abcdee

...
## Return Oriented Programming

### Stack Contents

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80abdea0</td>
<td>...</td>
</tr>
<tr>
<td>0x309</td>
<td></td>
</tr>
<tr>
<td>0x80345677</td>
<td></td>
</tr>
<tr>
<td>&amp;&quot;/tmp/lala&quot;</td>
<td></td>
</tr>
<tr>
<td>0x80abddaa</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0x80abcdee</td>
<td></td>
</tr>
</tbody>
</table>

### Code Examples

- `0x80abdea0: int 0x80;`
- `0x80abddaa: pop $ebx; 0x80abddab: ret;`
- `0x08abcdee: pop $eax; 0x08abcdef : ret;`
- `0x08abcdef: ret;`
- `0x80345677: pop $ecx; 0x80345678: ret;`
- `0x80345677: pop $ecx; 0x80345678: ret;`
- `0x80abdea0: int 0x80;`

### Registers

- **EAX = 8**
- **EBX = &"/tmp...”**
- **ECX = SMTH**
Return Oriented Programming

EAX = 8
EBX = &"/tmp..."
ECX = 0x309

```
0x80abdea0
0x309
0x80345677
&"/tmp/lala"
0x80abddaa
8
0x80abcdee
```

```
ESP
```

```
High
...
...
...
0x80abdea0
0x309
0x80345677
&"/tmp/lala"
0x80abddaa
8
0x80abcdee
```

```
EIP
```

```
Low
...
...
...
0x80345677: pop $ecx;
0x80345678: ret;
...
0x08abcdee: pop $eax;
0x08abcdef : ret;
...
0x80abddaa: pop $ebx;
0x80abddab: ret;
...
0x80abdea0: int 0x80;
...
## Return Oriented Programming

<table>
<thead>
<tr>
<th>ESP</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0x80abdea0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x309</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x80345677</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;&quot;/tmp/lala&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x80abddaa</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x80abcdee</td>
</tr>
</tbody>
</table>

EAX = 8  
EBX = &"/tmp..."  
ECX = 0x309  

```
0x80345677: pop $ecx;  
0x80345678: ret;
```

```
0x80abcdee: pop $eax;  
0x80abcdef : ret;
```

```
0x80abddaa: pop $ebx;  
0x80abddab: ret;
```

```
0x80abdea0: int 0x80;  
...```
Dangling pointers

• A dangling pointer is a pointer that does not point to a valid object of the appropriate type

• Dangling pointers typically occur when memory is freed but not all pointers that used to point to that memory, are invalidated
int dangling() {
    char *p1;
    char c;
    ...
    p1 = malloc(42);
    ...
    free(p1);
    ...
    if (...){
        if (...){
            c = *p1;
        }
    }
}
Principles of Secure Design
Step back

• Let’s look at categories of Security Flaws

• **Architectural/Design-level flaws**: Security issues that the original design of the program did not consider or did not solve correctly

• **Implementation Flaws**: Errors made in coding a design

• **Operational flaws**: Problems arising from how the software is installed and/or configured
**Attack Surface**

**Attack surface**: the set of ways an application can be attacked.

Used to measure attackability of a system.

- The larger the attack surface of a system, the more likely an attacker is to exploit its vulnerabilities and the more damage is likely to result from attack.

- Compare to measuring vulnerability by counting number of reported security bugs.

- Both are useful measures of security, but have very different meanings.
Secure Design Principles

1. Least Privilege
2. Fail-Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation of Privilege
7. Defense in depth
8. Least Common Mechanism
9. Psychological Acceptability
Secure Design Principles

1. Least Privilege
2. Fail-Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation of Privilege
7. Defense in depth
8. Least Common Mechanism
9. Psychological Acceptability
(Very) basics of cryptography
Cryptography

• Cryptography comes from the Greek words «κρυπτό» (secret) and «γραφή» (writing)
  – Mangling information in a way that allows unmangling by the intended recipients

• Common Uses of Cryptography:
  – Protect the confidentiality of information in transit (or in storage)
  – Protect the integrity of information (even if that is public)
  – Authenticate a user
  – Non-repudiation: A sender cannot send a message and then deny it
Cryptography

• A typical cryptographic system involves an algorithm and one (or more) keys
  – If a key is compromised you just change the key, you don’t have to come up with a new algorithm

• Kerckhoffs’ Principle: “A cryptosystem should be secure even if everything about the system, except the key, is public knowledge”
  – The security does not depend on the attacker not knowing which algorithm we are using
Types

• Types of cryptographic functions:
  – Symmetric Key Cryptography (also known as Secret Key Cryptography)
  – Asymmetric Key Cryptography (also known as Public Key Cryptography)
  – Hash functions
    • We looked at some characteristics of those when we talked about authentication and storing passwords
One-Time Pad (Vernam Cipher)

Cipher achieves **perfect secrecy** if and only if there are as many possible keys as possible plaintexts, and every key is equally likely  
(Claude Shannon, 1949)
Encrypting a Large Message

• So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size

• **Electronic Code Book (ECB) mode**
  – Split plaintext into blocks, encrypt each one separately using the block cipher

• **Cipher Block Chaining (CBC) mode**
  – Split plaintext into blocks, XOR each block with the result of encrypting previous blocks

• Also various counter modes, feedback modes, etc.
Applications of Public-Key Crypto

• Encryption for confidentiality
  – Anyone can encrypt a message
    • With symmetric crypto, must know the secret key to encrypt
  – Only someone who knows the private key can decrypt
  – Secret keys are only stored in one place

• Digital signatures for authentication
  – Only someone who knows the private key can sign
  – Everyone who knows the public key can validate the signature

• Session key establishment
  – Exchange messages to create a secret session key
  – Then switch to symmetric cryptography (why?)
Malware
Malware, short for malicious software, is software designed to gain access to confidential information, disrupt computer operations, and/or gain access to private computer systems. Malware can be classified by how it infects systems:

- Trojan Horses
- Viruses
- Worms

Or by what assets it targets:

- Ransomware
- Information stealers
- Spyware and adware
- Backdoors
- Rootkits
- Botnets
Virus

A computer virus is a type of malware that, when executed, replicates by inserting copies of itself (possibly modified) into other files. This process is called infecting.
Worms

Copies self from one computer to another

Self-replicating: No user action required unlike virus or Trojan horse programs.

Spreads via network protocols

ex: SMTP (email), fingerd, MS SQL
Generic malware

• In recent years, attackers have moved away from viruses and worms and more towards malicious executables which just attempt to gain money from each installation
  – Steal user’s credentials
  – Use the user’s Internet connection for sending SPAM
  – Use the user’s CPU for mining cryptographic currencies
  – Use extortion and ask for money from the victim user
Ransomware

• Ransomware takes a user’s files hostage and only releases them if the user pays “ransom”

• Evolution
  – Fake AVs
    • Pretending that your machine is infected and asking for money
  – Locker malware
    • Making it very hard to utilize your computer but leaving files intact
  – Crypto-ransomware
    • Using unbreakable encryption to encrypt all your files
    • If done correctly, there is no bypassing possible
Malware Handling Process

1. Static Analysis (for signatures)
2. Dynamic Analysis (for signatures)
3. Reverse Engineering (if needed)
4. Signature Creation
5. Signature Deployment
A botnet is

- A network of compromised machines
- that can be controlled remotely
- used for malicious activities.
C&C Structure

Centralized

- All bots connect to a single C&C server.
- Single point of failure that can take down botnet.

Decentralized (peer-to-peer)

- Every bot is also a C&C server.
- Botmaster can administrate from any bot.

Hybrid

- Every bot is a C&C server.
- Also have separate C&C server.
Protecting C&C

1. Bulletproof Hosting
   - ISP that permits criminal activity, spamming, c&c.

2. Dynamic DNS
   - C&C changes IP addresses every few minutes.
   - Uses DDNS service to point to changing IPs.

3. Fast Fluxing
   - DNS requests resolve to set of flux agent IPs.
   - Flux agents redirect traffic to C&C server.
SSL/TLS
What Is SSL / TLS?

• Secure Sockets Layer and Transport Layer Security protocols
  – Same protocol design, different crypto algorithms
• De facto standard for Internet security
  – “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
• Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc.
Handshake Protocol Structure

- ClientHello
- ServerHello, [Certificate], [ServerKeyExchange], [CertificateRequest], ServerHelloDone
- [Certificate], ClientKeyExchange, [CertificateVerify]
- switch to negotiated cipher
- Finished

Switch to negotiated cipher

Record of all sent and received handshake messages
HTTPS and Its Adversary Model

- HTTPS: end-to-end secure protocol for Web
- Designed to be secure against network attackers, including man-in-the-middle (MITM) attacks

- HTTPS provides encryption, authentication (usually for server only), and integrity checking
Mixed Content and Network Attacks

- Banks: after login, all content served over HTTPS
- Developer error: somewhere on bank site write
  ```html
  <script src=http://www.site.com/script.js> </script>
  ```
  - Active network attacker can now hijack any session (how?)
- Better way to include content:
  ```html
  <script src=/www.site.com/script.js> </script>
  ```
  - Served over the same protocol as embedding page
HTTP → HTTPS and Back

• Typical (bad) pattern: HTTPS upgrade
  – Come to site over HTTP, redirect to HTTPS for login
  – Browse site over HTTP, redirect to HTTPS for checkout

• sslstrip: network attacker downgrades connection
  – Rewrite <a href=https://…> to <a href=http://…>
  – Redirect Location: https://... to Location: http://...
  – Rewrite <form action=https://... > to <form action=http://...>

Can the server detect this attack?
CA Hierarchy

• Browsers, operating systems, etc. have trusted root certificate authorities
  – Firefox includes over 100 certificates of trusted root CAs

• A Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  – Certificate “chain of trust”
    • \( \text{sig}_{\text{Verisign}}(\text{“SBU”}, \text{PK}_{\text{SB}}), \text{sig}_{\text{SB}}(\text{“Nick N.”}, \text{PK}_{\text{Nick}}) \)

• CA is responsible for verifying the identities of certificate requestors, domain ownership
Certificate Hierarchy

What power do they have?

Who trusts their certificates?
Certificate Transparency

• Experimental open standard for monitoring and auditing digital certificates
  – This was put in place after many incidents where certificate authorities were found to be issuing certificates that they were not supposed to

• A CA that participates in Certificate Transparency adds to a global append-only log an entry for each new certificate that it issues
  – The idea is that others can monitor this log and sound an alarm when domain certificates were created without the knowledge of the appropriate domain owners

• Google Chrome requires Certificate Transparency for all certificates issued after April 2018
Economics of SSL

• This was the only way of doing things up until 2016 when Let’s Encrypt became publicly available

• Let’s Encrypt is a certificate authority that provides free domain-verified certificates
  – They were not the first cost-free provider but they had the backing and PR of large companies
  – They provided software that makes the fetching and installation of a certificate, essentially automatic
Everyone uses Let’s Encrypt...including the bad guys

• Since everyone can now have an SSL certificate, so can the bad guys
  – https://www.totallypaypal.com

• Websites with SSL have been traditionally thought of as “secure” and “safe” so phishing attacks may work better on SSL websites
To summarize the differences between the two

HTTPS & SSL doesn't mean "trust this." It means "this is private." You may be having a private conversation with Satan.
HTTPS with interception

The interception proxy can be:
- A piece of software on the client endpoint
- A network device that intercepts the client’s traffic before it reaches the outside world
How the web works
Browser and Network

Browser

OS

Hardware

Network

request

reply

website
Step 1
Question: what is the IP Address of some-webserver.com? Please reply to My IP Address

Step 2
Question: where can I find the IP Address of some-webserver.com?

Step 3
Answer: I don't know but .com NameSpace should have the answer

Step 4
Question: What is the IP Address of some-webserver.com?

Step 5
Answer: Primary DNS Server of some-webserver.com knows it.

Step 6
Question: What is the IP Address of some-webserver.com?

Step 7
Answer: Here is the IP Address of some-webserver.com.

Step 8
Answer: Here is the IP Address of some-webserver.com.
The format of URLs is the following:

- Items in brackets are optional

scheme://[username:password@]hostname[:port] [/path/to/resource][?query_string][#fragment]
HTTP Requests

• A request has the form:

```html
<METHOD> /path/to/resource?query_string HTTP/1.1
<header>*

<BODY>
```

• HTTP supports a variety of methods, with two being the most popular ones:
  – GET: intended for information retrieval
    • Typically the BODY is empty
  – POST: intended for submitting information
    • Typically the BODY contains the submitted information
HTTP Responses

• A response has the form

   HTTP/1.1 <STATUS CODE> <STATUS MESSAGE>
   <header>*
   <BODY>

• Important response codes:
  – 2XX: Success, e.g. 200 OK
  – 3XX: Redirection, e.g. 301 Moved Permanently
  – 4XX: Client side error, e.g. 404 Not Found
  – 5XX: Server side error, e.g. 500 Internal Server Error
No state

• HTTP is, by design, stateless
  – There’s nothing baked in the protocol to identify one request as part of sequence of other requests

• You can try to do it by IP address, but that’s not going to work well
  – Network Address Translation
  – DHCP
  – Mobile devices
Web-Application-level State

• The de facto way of having state in modern web applications is through the use of sessions transferred over **HTTP cookies**
  – Other techniques, such as HTTP Basic Auth, are not popular so we will not cover them in this course

• **Cookie**: A small piece of data sent by the webserver to browsers, which the browsers are to include to all their subsequent requests to that server.
**Cookie-based Session Management**

**Browser**

Request `example.com/index.html`

Response `Set-Cookie: SID=12345`

Request `example.com/login.html`

Cookie: `SID=12345`

Response

Request `example.com/login.php`

Cookie: `SID=12345`

**Server**

Session Store

`12345` auth: false
Cookie-based Session Management

Browser

Request `example.com/index.html`
Response `Set-Cookie: SID=12345`

Request `example.com/login.html`
Cookie: SID=12345
Response

Request `example.com/login.php`
Cookie: SID=12345
Response

Cookie Jar
`example.com
SID=12345`

Session Store
12345
auth: true
user: Bob
JavaScript

• “The world’s most misunderstood programming language”

• Language executed by the Web browser
  – Scripts are embedded in webpages
  – Can run before HTML is loaded, before page is viewed, while it is being viewed, or when leaving the page

• Used to implement “active” webpages and Web applications

• A potentially malicious webpage gets to execute some code on user’s machine
Content in the Browser

• Origin-based separation of documents
  – Naturally enforced by the Same-Origin Policy
  – Allows you to separate sensitive parts and non-sensitive parts
  – Prevents unintended sharing of information
  – Prevents escalation of successful attack

**Origin**
The triple <scheme, host, port> derived from the document’s URL. For http://example.org/forum/, the origin is <http, example.org, 80>

**Same-Origin Policy**
Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted
Web security: Client-side
<table>
<thead>
<tr>
<th>OWASP Top 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 – Injection</td>
</tr>
<tr>
<td>A2 – Broken Auth and Session Management</td>
</tr>
<tr>
<td>A3 – Cross-site Scripting</td>
</tr>
<tr>
<td>A4 – Insecure Direct Object References</td>
</tr>
<tr>
<td>A5 – Security misconfiguration</td>
</tr>
<tr>
<td>A6 – Sensitive Data Exposure</td>
</tr>
<tr>
<td>A7 – Missing function level access control</td>
</tr>
<tr>
<td>A8 – Cross-site Request Forgery</td>
</tr>
<tr>
<td>A9 – Using components with kn. vulnerabilities</td>
</tr>
<tr>
<td>A10 – Unvalidated redirects and Forwards</td>
</tr>
</tbody>
</table>
Cross-Site Scripting (XSS)

- Different types of script injection
  - **Persistent**: stored data used in the response
  - **Reflected**: part of the URI used in the response
  - **DOM-based**: data used by client-side scripts

**Reflected XSS**

http://www.example.com/search?q=<script>alert('XSS');</script>

<h1>You searched for<script>alert('XSS');</script></h1>
Content Security Policy

• Detect and mitigate certain types of attacks, mainly XSS
• The policy is delivered by a website to a browser through an HTTP header

Content-Security-Policy: policy

• Through CSP, websites can list a series of sources that are trusted for remote content, for the current page
  – JavaScript
  – Iframes
  – CSS
• Anything not on the list, is denied
Remote JavaScript libraries

The code coming from foo.com will be incorporated in mybank.com, as if the code was developed and present on the servers of mybank.com
Cross-site Request Forgery (CSRF)

• Is an attack where the attacker tricks the browser into injecting a request into an authenticated session
  – E.g. by means of scripting
  – E.g. by means of remote resource inclusion

• Attacker can perform requests/operations in the name of the user
Cross-site Request Forgery
Session Fixation

https://www.owasp.org/index.php/Session_fixation
Mitigating Session Hijacking

- Protecting session cookies
  - Deploy application over TLS only
  - Secure: prevents cleartext transmission
  - HttpOnly: prevents script access

**Correct Set-Cookie header**

```
Set-Cookie: SID=123abc; Secure; HttpOnly
```
HSTS

- Use full-site SSL in combination with Secure cookie and HTTP-only Cookie

- HSTS: HTTP Strict Transport Security
  - Force the browser to always contact the server over an encrypted channel, regardless of what the user asks

HTTP Header

```
Strict-Transport-Security: max-age=31536000
```
Win a free iphone!

Just click on red and green!

Quick while the offer lasts!
Web security: Server-side
## OWASP Top 10

<table>
<thead>
<tr>
<th>A1</th>
<th>Injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>Broken Auth and Session Management</td>
</tr>
<tr>
<td>A3</td>
<td>Cross-site Scripting</td>
</tr>
<tr>
<td>A4</td>
<td>Insecure Direct Object References</td>
</tr>
<tr>
<td>A5</td>
<td>Security misconfiguration</td>
</tr>
<tr>
<td>A6</td>
<td>Sensitive Data Exposure</td>
</tr>
<tr>
<td>A7</td>
<td>Missing function level access control</td>
</tr>
<tr>
<td>A8</td>
<td>Cross-site Request Forgery</td>
</tr>
<tr>
<td>A9</td>
<td>Using components with known vulnerabilities</td>
</tr>
<tr>
<td>A10</td>
<td>Unvalidated redirects and Forwards</td>
</tr>
</tbody>
</table>
What if we, as the script programmers, blacklist the semicolon? Will it solve our problems?
Remote Command Execution

• These were examples of a vulnerability belonging to the class of “Remote Command Execution”

• One of the more deadly attacks
  – Attacker has foothold on server
  – This foothold can be used to (among others): steal data, connect to other machines internal to the network, or try to become root
Local File Inclusion

Intended use:

Unintended use:
• http://example.com/index.php?f=../../../../../etc/passwd

So now an attacker can read text files located on the system which are readable by the web server process.
Remote File Inclusion

Intended use:
• http://example.com/index.php?f=contactus

Unintended use:

So now an attacker can also run arbitrary PHP commands on the server
SQL Injection

What if an attacker submits:

- **Username**: administrator’--
- **Password**: doesnotmatter

Then the query becomes

```sql
SELECT * from members where
username = 'administrator'--' and password = 'doesnotmatter'
```

Comment operator for some SQL databases
Prepared statements

mysql_query("SELECT * from members where username = '{$user' and password = '{$pass'’");

Now becomes

$stmt = $conn->prepare("SELECT * from members where username=? and password=?");
$stmt->bind_param("ss", username,password);
$stmt->execute();
$res = $stmt->get_result();
Injection (HTTP Parameter Pollution)

• The reason why HPI exists is because the HTTP specification is not explicit about what servers should do when they receive a request having two parameters with the same name.

• http://www.example.com?foo=1&bar=yes&foo=2

• At the server-side, what should the value of the foo variable be?
  – 1, 2, “12”, [1,2]?

• It turns out that different combinations of web server/server-side languages do different things.
Injection (HTTP Parameter Pollution)

1. **Attacker to public-facing webserver**
   - POST /bank/52/Default.aspx HTTP/1.1
   - Content-Type: application/x-www-form-urlencoded
   - Host: acmebank.com
   - Content-length: 65
   - fromAcc=123456&amount=1000&toAcc=654321&Submit=Submit

2. **Public-facing webserver to internal webserver**
   - POST /doTransfer.asp HTTP/1.1
   - Host: internal.acmebank.com
   - Content-length: 72
   - FROM=123456&clearedfunds=false&AMOUNT=1000&TO=654321
Injection (HTTP Parameter Pollution)

Attacker to public-facing webserver

1

POST /bank/52/Default.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: acmebank.com
Content-length: 65

fromAcc=123456%26clearedfunds=true&amount=1000&toAcc=654321&Submit=Submit

Public-facing webserver to internal webserver

2

POST /doTransfer.asp HTTP/1.1
Host: internal.acmebank.com
Content-length: 72

FROM=123456&clearedfunds=true&clearedfunds=false&AMOUNT=1000&TO=654321
Insecure Direct Object References

• Imagine you login to your bank account, and there’s a link that allows you to see last month’s statement:

• In insecure direct object reference, the application exposes a direct reference to an object which can allow attacker to bypass authorization, e.g.,
  • http://bank.com/get_last_statement.php?u=976654
  • http://bank.com/get_last_statement.php?u=976653
  • http://bank.com/get_last_statement.php?u=976652
Logic Vulnerabilities

• HTTP parameter tampering vulnerabilities are a subset of logic vulnerabilities in web applications

• Logic vulnerabilities typically rely on breaking assumptions made by architects and developers

• Assumptions
  – Step 2 can only be performed after Step 1
  – The web application controls the navigation steps
  – Users cannot change parameters that they cannot see
  – Etc.