Think your apps are secure in the cloud?
Learn how to safeguard your software

Sean Auriti
Tara Williams

November 02, 2017

The cloud is not secure. Developers know through experience that software and servers get hacked thanks to bad code and insecure Internet connections to remote servers and data centers. You can compare using the cloud to having valuables that are locked up in a suitcase in a parking lot. By nature, both are susceptible to attacks. In this article, we explain the reasons why the cloud is not secure, but we also examine ways to mitigate any insecurities by illustrating some Django examples through the inception of BugHeist.

"There is a lot of flawed JavaScript, PHP, and Python code that create holes into which hackers inject malicious JavaScript and SQL. Delivered through email attachments, links and browsers, this bad code crawls into cloud-based databases where hackers can execute remote commands to steal proprietary data."

Why your apps might be weak

Many software developers and security professionals point to the lack of secure software that is built into the Software Development Life Cycle (SDLC) as a huge reason why there are so many software vulnerabilities. Add tight deployment deadlines, and you have the culprits that contribute to the epidemic of bugs that infiltrate internet-connected platforms and devices that consumers and companies use, which exist in computer networks, data centers, cloud computing platforms, and mobile applications.
There is a lot of flawed JavaScript, PHP, and Python code that create holes into which hackers inject malicious JavaScript and SQL. Delivered through email attachments, links and browsers, this bad code crawls into cloud-based databases where hackers can execute remote commands to steal proprietary data.

Many developers fail to secure their data by forgoing Secure Sockets Layer (SSL) certification and validation in their application development practices to release an application as soon as possible. The SSL protocol includes a certificate that ensures files that are housed on your web server are encrypted when they are sent back and forth over the Internet to visitors' browsers by using Hypertext Transfer Protocol Secure (HTTPS). Without HTTPS and SSL protection, hackers can conduct man-in-the-middle attacks by intercepting and monitoring data packets through the routers in their LAN and WAN networks.

They can also hijack data through an Address Resolution Protocol (ARP) spoof. This works when hackers search for specific IP and Media Access Control (MAC) addresses on a local network. Once they identify their targets' IP and MAC addresses, ARP will grab and impersonate them sending data packets to unsuspecting users from their routers.

Over the last several years, APIs became integral to application development. They add features and functionality to websites and mobile applications automatically, but they are also insecure. Do you test the third-party APIs you added to your scripts to check whether they allow bad actors in your software environment? Many APIs fail to include methods like OAuth and two-factor authentication to safeguard data traveling to and from cloud-based servers and data centers. OAuth uses tokens to validate and secure data between the client and server, and it's open source.

Many applications compromise user data as a result of an organization's flawed software security policy. Last year, we created a collaborative work group through a third-party cloud collaboration tool for Alpha One Labs, a Brooklyn maker space where we created prototypes for an LCD hat, laser scrolling sign, 3D POV display, smart recycling bin and a robotic food exchange. The email addresses in this work group were exposed publicly and getting scraped and spammed. The cloud company was contacted about this issue, but it took approximately 2 to 3 months before the company created a feature that secured the addresses and made them private.

Frustrated with this cloud company and various other organizations that failed to address their software security issues (allowing them to remain active for weeks or months after being notified), an application that uses Django called BugHeist was born. It rewards individuals for reporting and fixing software and hardware bugs that they find on the Internet, which range from security weaknesses to design, performance and functionality issues.
Why Django?

We considered a number of frameworks/platforms when he had the idea of building BugHeist, including C++, Java, and PHP. After we analyzed these frameworks, we decided that Django would make the most sense since it protects data from the most common software-hacking attacks by embedding secure code in its framework.

Since BugHeist's launch in July 2016, the site has helped approximately 100 companies identify and address over 350 software bugs and issues. For more details about Django's open source framework, visit djangoproject.com. To learn more about BugHeist's open source platform, you can visit the GitHub repository at https://github.com/owasp/blt.

Let's look at some specific Django examples regarding cloud security issues. Django enables you to activate software that will protect your site from the following security problems: Cross-site request forgery, cross-site scripting, SQL injection, clickjacking, insecure SSL/HTTPS, and unprotected host header validation. We'll go over these security risks in detail below.

Read the article "Prevent cross-site request forgery: Know the hidden danger in your browser tabs."

Cross-site request forgery (CSRF)

CSRF happens when a browser that contains malicious code steals data from a web server and sends it to a fake site. Another instance can be a website that houses an A frame, image tag, or when other bad code triggers an insecure HTTP request to a fraudulent website.

Because the initial browser request was approved by the website's HTTPS/SSL protocol with embedded authentication tokens, the web server verifies and executes the malicious codes' commands as if it was the user's intentional browser request. In reality, the malicious code altered the original code in the browser's initial request, engineering it to steal unauthorized data while it masquerades as the unsuspecting user.

Activating Django's CSRF middleware, ensures that forms with POST elements are free from code that create requests to steal unauthorized data from your site.

Here is the sample code that protects BugHeist from cross-site request forgery. It is included by default in Django's Middleware Classes section:
django.middleware.csrf.CsrfViewMiddleware

To ensure that attackers don't steal proprietary information through code that's injected into BugHeist's "Post" HTML and PHP/Python, we coded the following CSRF Token tag:

```html
<form class="form" action="/issue/" method="post" enctype="multipart/form-data">
  {% csrf_token %}
</form>
```

More resources on XSS

Read the OWASP page on XSS here.
Cross-site scripting (XSS)

Hackers inject malicious code into websites and mobile applications by using this method of cross-site scripting. This code is embedded into an unsuspecting user’s browser, which retrieves unauthorized data and sends it to suspicious parties.

Stored XSS is when the malicious code scripts are housed on the website servers and is executed upon a browser’s request to the back end. When a user sends a browser request that is embedded with scripts that steal code from servers, this is called Reflected XSS.

By default, Django escapes or protects variables in your software environment, unless they are marked explicitly as "safe."

Check out our library for more articles on SQL injection here.

SQL injection

If developers fail to sanitize their input code when they develop applications for a SQL database, it gives individuals the perfect vector to grab proprietary data from the back end.

Don't be a victim of ransomware. Take the steps to protect yourself.

Hackers simply look for a vulnerable location in the code and insert SQL queries that return data they can sell on the Dark Web or use to blackmail organizations, like hospitals or government agencies.

When you use Django, you have SQL injection defenses through built-in Object Relational Mapping (ORM). Another software layer that uses object-oriented programming instead of SQL code, ORM gives you a method to access, write, and update secure queries over and over again. Because it allows you to write queries in the programming languages with which you are familiar, ORM helps speed up back-end updates.

Clickjacking

In this scenario, users click buttons or links on sites and mobile applications that disguise malicious code that redirect their actions without their knowledge.

Django automatically detects and protects websites from suspicious iframe requests through a middleware option that prevents browsers from rendering scripts that contain this bad code. It also features PBKDF2 and bcrypt, encryption standards that cannot get cracked by using rainbow tables with decrypted password hashes. PBKDF2 and bcrypt also take a significant amount of time to compute or create since they use a salt function to further randomize data, making it challenging for hackers to brute force them.

Find further reading on SSL and HTTPS.
Insecure SSL/HTTPS

Website and mobile application servers that don't have HTTPS and SSL architecture invite hacking attacks. Without this protocol, users' browsers will not be able to confirm a secure connection between it and the server because they will not be able to verify the site's domain name, organization that owns the site, and other credentials that are encrypted in a certificate that is issued by a third-party certificate authority.

Unprotected host header validation

To protect your website or mobile application, all of your connections should feature a HTTPS header in every page to which your site connects. HTTPS is part of an Internet security protocol that incorporates an HTTP Transport Security (HSTS) policy. This policy enables developers to register their certificates with browsers so that they are preloaded with HTTPS credentials they can validate on demand. Also, any HTTP request sent to your web application will be converted into an HTTPS connection with Django.

The following code shows the SQL and JavaScript code that protects the database from SQL injection and insecure HTTP/SSL requests in Django:

```python
if 'DATABASE_URL' in os.environ:
    DEBUG = False
    EMAIL_HOST = 'smtp.sendgrid.net'
    EMAIL_HOST_USER = os.environ.get('SENDGRID_USERNAME', 'blank')
    EMAIL_HOST_PASSWORD = os.environ.get('SENDGRID_PASSWORD', 'blank')
    EMAIL_PORT = 587
    EMAIL_USE_TLS = True
    If not TESTING:
        SECURE_SSL_REDIRECT = True
    GS_ACCESS_KEY_ID = os.environ.get('GS_ACCESS_KEY_ID', 'blank')
    GS_SECRET_ACCESS_KEY = os.environ.get('GS_SECRET_ACCESS_KEY', 'blank')
    GS_BUCKET_NAME = 'bhfiles'
    DEFAULT_FILE_STORAGE = 'storages.backends.gs.GSBotoStorage'
    GS_FILE_OVERWRITE = False
    GS_QUERYSTRING_AUTH = False
    MEDIA_URL = "https://bhfiles.storage.googleapis.com/"
    ROLLBAR = {
        'access_token': os.environ.get('ROLLBAR_ACCESS_TOKEN', 'blank'),
        'environment': 'development' if DEBUG else 'production',
        'root': BASE_DIR,
        'exception_level_filters': [
            (Http404, 'warning')
        ]
    }
    import rollbar
    rollbar.init(**ROLLBAR)
    # local dev needs to set SMTP backend or fail at startup
    if DEBUG:
        EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
    ACCOUNT_EMAIL_REQUIRED = True
    ACCOUNT_USERNAME_REQUIRED = True
    ACCOUNT_EMAIL_VERIFICATION = "optional"
    # Honor the 'X-Forwarded-Proto' header for request.is_secure()
    SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```
Become a better application security developer

Developers continue to create more web applications thanks to the demands of the market, and the fact that cloud-connected applications on a multitude of devices are integrated in individuals' work and home lives, which makes software security crucial.

Here are some tips for developers who want to improve their software security skills:

- Participate in bug bounties where you share the software and hardware bugs you find on websites, mobile applications and computer networks for cash that come from companies that sponsor these events.
- Work with other developers and product managers to develop an application security policy for your organization.
- Compete in ethical hackathons and Capture the Flag (CTF) events.
- Contribute your programming skills to an open source project.
- Attend industry conferences that are organized by OWASP and other organizations where the latest secure software developments are discussed.
- Join in-person and online events and forums where best practices in secure coding practices are shared.
- Study application security through online and in-person courses offered by organizations like the SANS Institute, The International Information System Security Certification Consortium (ISC)², the International Council of E-Commerce Consultants (EC-Council) and other software industry associations.
Related topics

- Participate in reporting bugs to BugHeist
- Find more articles on Django
- Visit the Django site
- Interact with other security-minded pros

© Copyright IBM Corporation 2017

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)