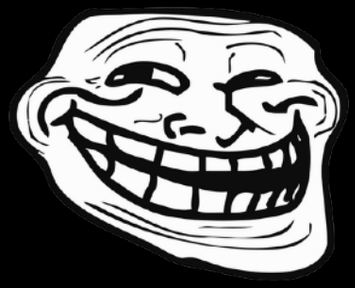
HACKING



Cyber Professionals
are Hackers

BUT

Hackers are not necessarily
Cyber Professionals

Hackers love to…

Break things, understand why it broke and put them back together in a better way

Study low level stuff (C/C++, Assembly, Hardware devices, Network Analysis, etc…)

Satisfy their curiosity by constantly learning and improving their skills

Solve problems by thinking outside of the box

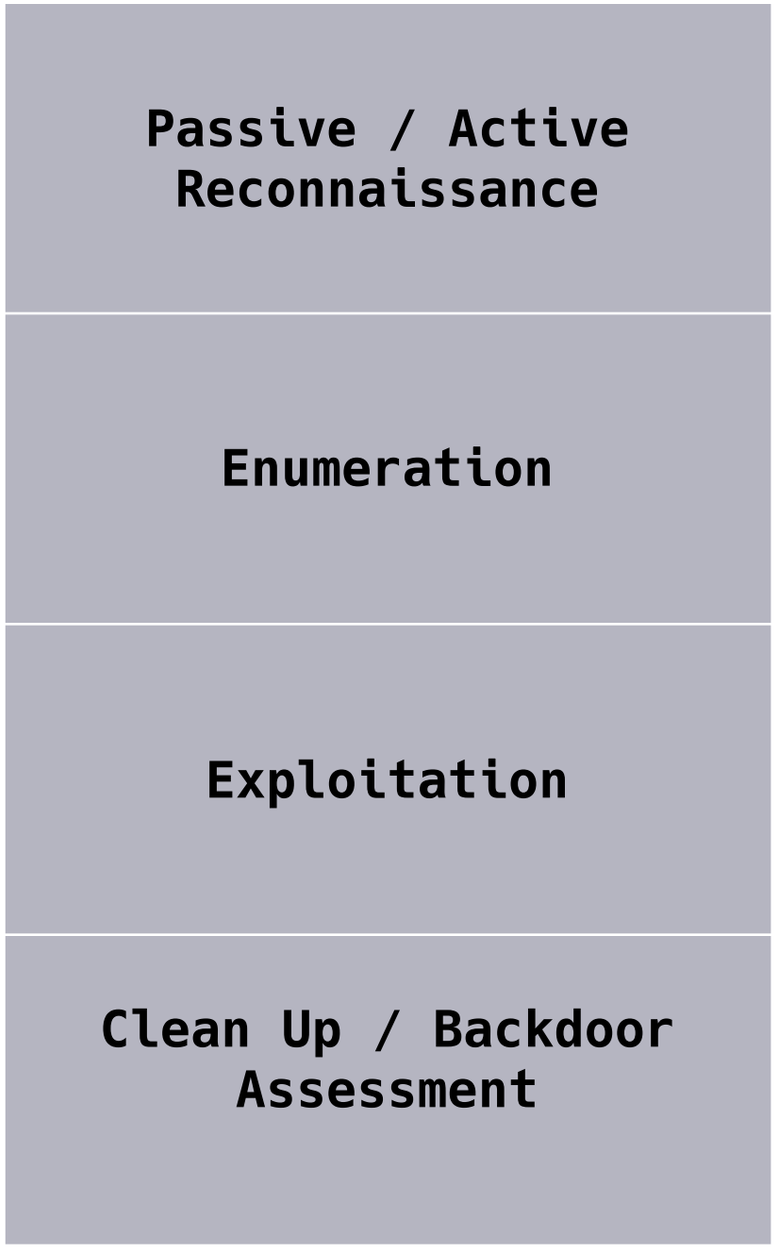
Types Of Hackers

Script Kiddies	White Hats	Gray Hats	Black Hats
Usually very young	Cyber professionals or serious hobbyists	Reformed black hats or cyber pros with a grudge	Motivated by personal gain ($$)
Trying things just for the **lulz**	Motivated by curiosity and knowledge	Motivated by power to some extent	No interest in getting credit for their work
Noisy, no real skills	Reformed black hats	No tolerance for human stupidity	Know how to cover their tracks, live in the shadows
Can do serious damage	Happy to share their knowledge	Hactivists	Can be hired to work along side of the government

Types of Cyber Professionals

Malware Analysts	**Network Pentesters**	**Social Engineers**
Forensic Experts	**Incident Response and Management**	**IoT Devices Researchers**
Exploit Developers	**Researchers**	**Web Application Pentesters**

Hacking Methodology



What does it mean in the WWW

Cross-Site Scripting
YES…

SQLI YES…

but let’s talk about
CSRF

Cookie w/ Session ID
dst_account=5678
amount=30.00

http://banco.com.br

Username/Password

Cookie w/ Session ID

Request

Malicious Response

http://malicioso.com.br

```
<html>
    <body>
        <img width="1" height="1"
src="http://banco.com.br/submitTransference?dst_account=5
678&amount=30.00" />
    </body>
</html>
```

CSRF Characteristics

1. It involves sites that rely on a user's identity
2. It exploits the site's trust in that identity
3. It tricks the user's browser into sending HTTP requests to a target site
4. It involves HTTP requests that have side effects

CSRF Demo

What Can We Do?

Setting up Express to use `csurf`:

```
const express = require(‘express’);
const csrf = require('csurf');

const app = express();

app.use(csrf());

app.use(function(req, res, next){
 // Expose variable to templates via locals
 res.locals.csrftoken = req.csrfToken();
 next();
});
```

Setting a `value` of the `csrf` token in an application's templates:

```
<input type="hidden" name="<i>csrf" value={{csrftoken}} />
```

Note: `{{csrftoken}}` is Handlebars syntax - this will differ slightly in other templating languages.

HELMET

Express.js security with HTTP headers. Latest version: **3.13.0**

Docs Contributors See also npm GitHub

Helmet helps you secure your Express apps by setting various HTTP headers. *It's not a silver bullet*, but it can help!

Quick start

First, run `npm install helmet --save` for your app. Then, in an Express app:

```
var express = require('express')
var helmet = require('helmet')

var app = express()

app.use(helmet())

// ...
```

HTTP Parameter Pollution

1: Send a request with two values for the same key.

```
curl http://example.com:8080/endpoint?name=Itchy&name=Scratchy
```

2: The Express server expects the `name` key to be a `String`, and uses `.toUpperCase()` on it.

```
app.get('/endpoint', function(req, res){
  if(req.query.name){
    res.status(200).send('Hi ' + req.query.name.toUpperCase())
  } else {
    res.status(200).send('Hi');
  }
});
```

The code example assumes that `req.query.name` is a `String` type. But, since there are two arguments with the same name Express returns the results as an `Array`: `['Itchy', 'Scratchy']`. This will throw an `Error` that will crash an Express application.

Regular Expression Denial of Service (ReDos)

Evil Regexes

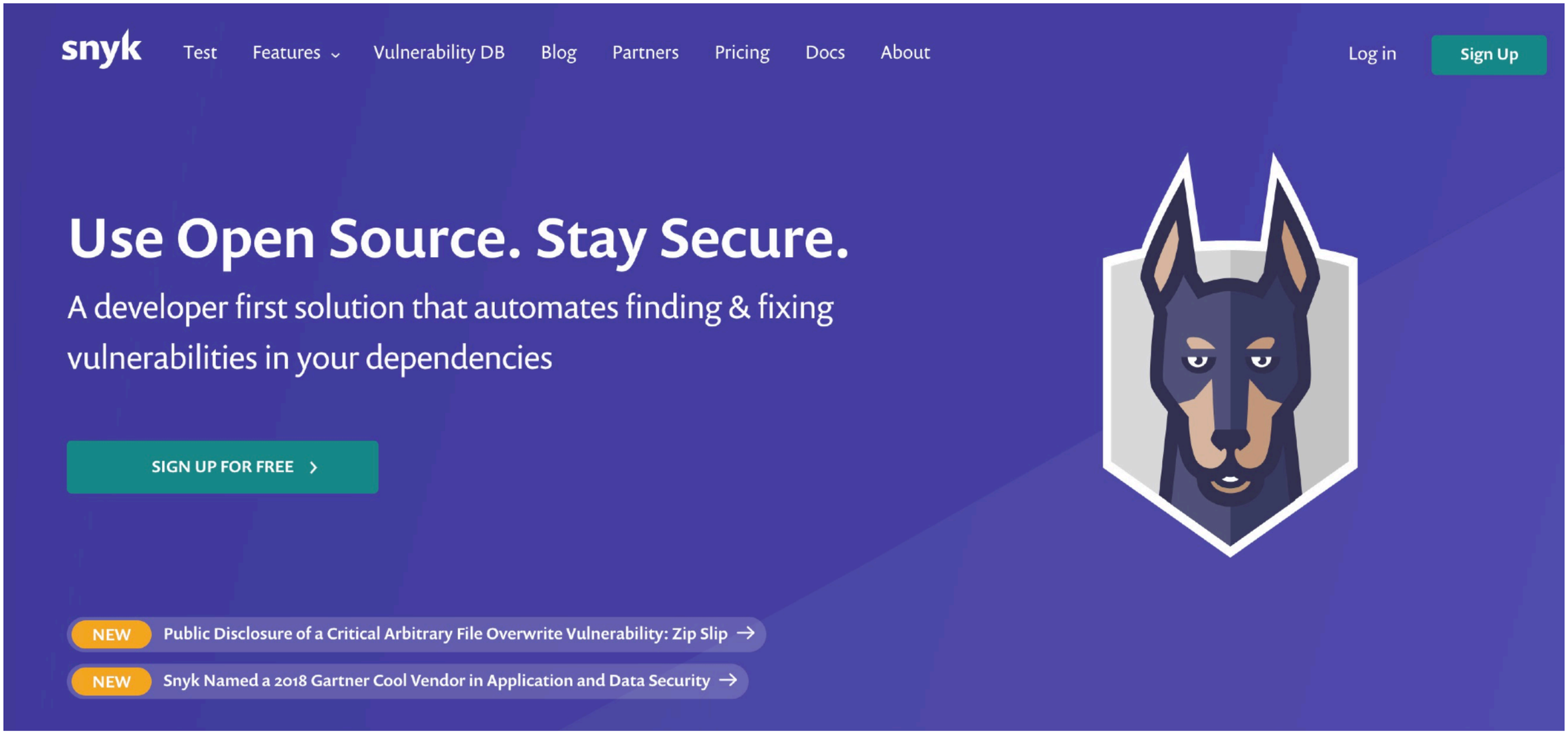
A regex is called "evil" when it can take exponential time when applied to certain non-matching inputs.

Examples of Evil Patterns:

- `(a+)+`
- `([a-zA-Z]+)*`
- `(a|aa)+`
- `(a|a?)+`
- `(.*a){x} | for x > 10`

All the above are susceptible to the input **aaaaaaaaaaaaaaaaaaaaaaaa!** (The minimum input length might change slightly, when using faster or slower machines).

https://snyk.io/



RED

BLUE

Bug Bounty Platforms

bugcrowd OUTHACK THEM ALL™

hackerone

Synack | HACKER-POWERED SECURITY

Benoit Côté-Jodoin (becojo)

107 Reputation - Rank

7

#386807

[flintcms] Account takeover due to blind MongoDB injection in password reset

Share:

State	Resolved (Closed)
Disclosed publicly	August 15, 2018 5:17pm +0300
Reported To	Node.js third-party modules
Asset	flintcms (Source code)
CVE ID	CVE-2018-3783
Weakness	Privilege Escalation
Severity	Critical (9.0)
Participants	
Visibility	Public (Full)

filedescriptor submitted a report to **Twitter**. Mar 28th (2 years ago)

Hi,

I would like to report an issue on Digits web authentication which allows attackers to retrieve the OAuth credential data of an application victims authorized.

Detail

Digits web authentication has strict validation on *host* and *callback_url*. On the server side, the values are compared with the registered domain. However, on the client side, the way parameters are parsed has a wrong assumption. Specifically,
in https://cdn.digits.com/45ed91c4cf9b6bb7465c27574b16910df8a86d2e_1458327827406/javascripts/popup.js

```
return window.location.search.slice(1).split("&").forEach(function(e) {
      var n = e.split("=");
      t[n[0]] = window.unescape(n[1])
  })
```

The above code snippet is responsible to convert query string into parameters, which assumes that the param delimiter has to be ampersand (&). In fact, the server side also accepts semi-colon (;) as param delimiter. For example:

> https://www.digits.com/login?consumer_key=9I4iINIyd0R01qEPEwT9IC6RE%3Bhost=https%3A%2F%2Fwww.periscope.tv

is the same as

> https://www.digits.com/login?consumer_key=9I4iINIyd0R01qEPEwT9IC6RE&host=https%3A%2F%2Fwww.periscope.tv

This creates a problem because from the server's perspective, `a=b;c=d` is two different parameters *a* and *c*, while the client thinks there is only one parameter *a* with value *b;c=d*. Attacker can evade the validation by append `;@attacker.com` in the corresponding param. Such bypass looks like this:

> https://www.digits.com/login?
> consumer_key=9I4iINIyd0R01qEPEwT9IC6RE&host=https%3A%2F%2Fwww.periscope.tv%3B@attacker.com

Shellshock

Not all vulnerabilities come from the applications themselves

Shellshock is a Remote Command Execution vulnerability in bash

It relies in the fact that bash incorrectly executes trailing commands when it **imports a function definition stored into an environment variable**

```
curl -H "user-agent: () { :; }; echo; echo; /bin/bash -c 'cat /etc/passwd'" \
http://localhost:8080/cgi-bin/vulnerable
```

Shellshock Demo

Finding/Publishing Exploits

Exploit-DB

WityCMS 0.6.2 - Cross-Site Request Forgery (Password Change)

EDB-ID: 45127	**Author**: Porhai Eung	**Published**: 2018-08-02
CVE: CVE-2018-14029	**Type**: Webapps	**Platform**: PHP
Aliases: N/A	**Advisory/Source**: N/A	**Tags**: Cross-Site Request Forgery (CSRF)
E-DB Verified:	**Exploit**: Download / View Raw	**Vulnerable App**:

« Previous Exploit

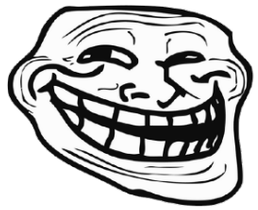
Next Exploit »

```
<!--
# Exploit Title: WityCMS 0.6.2 - Cross-Site Request Forgery (Password Change)
# Vendor Homepage: https://creatiwity.net/witycms
# Software Link: https://github.com/Creatiwity/witycMS/releases/tag/0.6.2
# Exploit Author: Porhai Eung
# Website: http://www.chhaipov.com
# CVE: CVE-2018-14029
# Category: webapps


1. Description

CSRF vulnerability in admin/user/edit in Creatiwity witycMS 0.6.2 allows an attacker to take over a user account by modifying
user's data such as email and password
```

The Magic of Google Dorks



SecuNinja
@secuninja

Follow

google search: intitle:"whoops! there was an error" db_password people are stupid... #securityfail #whoops

11:14 AM - 21 Oct 2017

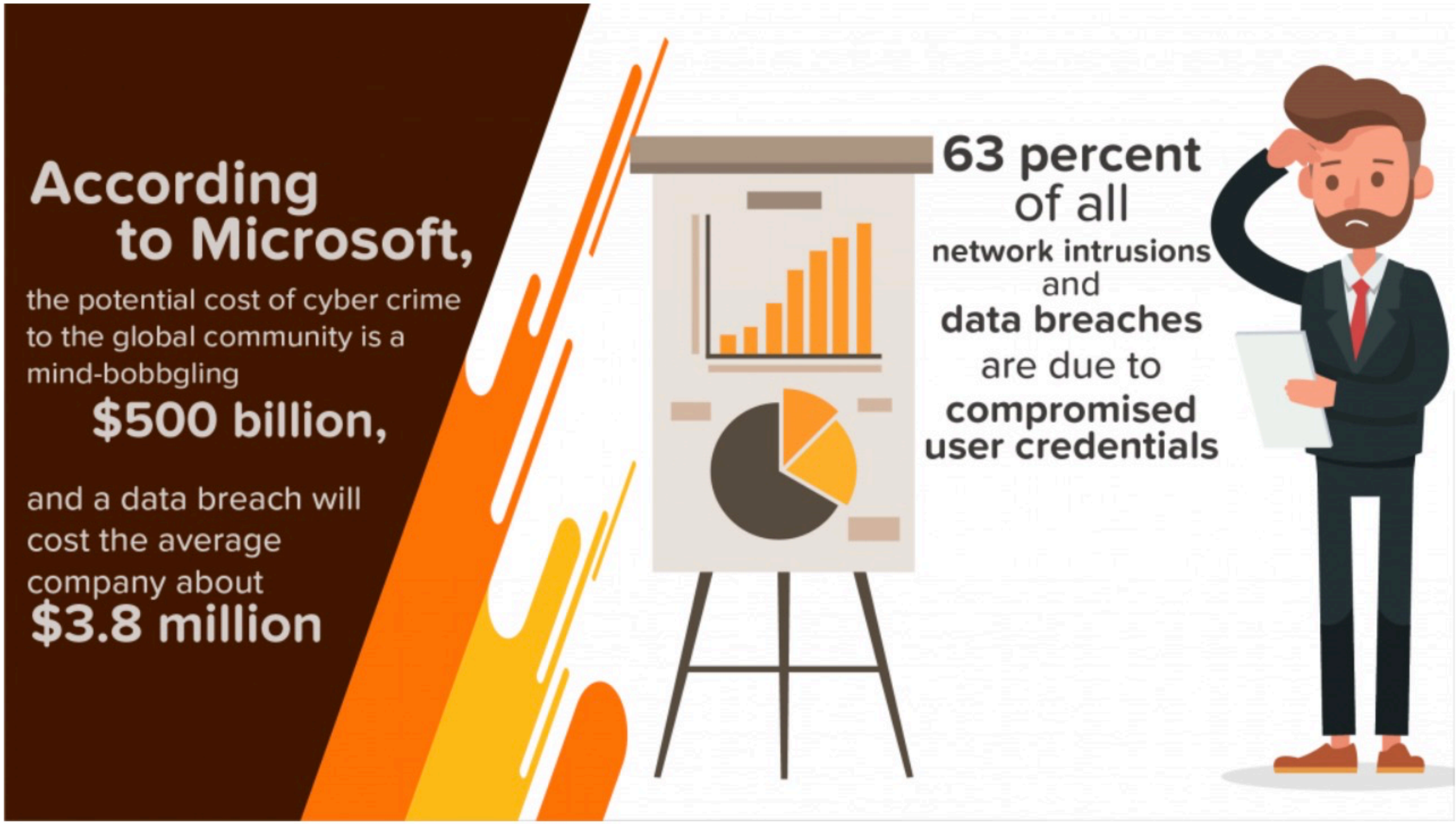
2 Retweets **6** Likes

2 6

Tweet your reply

Reverse Engineering

Why should you care?



77%: Proportion of respondents
in a survey of 2,800 IT professionals
who said their organizations
do not have a formal cybersecurity
incident response plan

“According to the Sonatype 2017 State of Software Supply Chain Report, 80 – 90% of an application is built from 3rd party components that often contain critical vulnerabilities as well. Often dev teams use out-dated versions with known security defects. But even if they are up-to-date, 84% of OpenSource projects (probably mostly smaller ones) do not fix known security defects at all.”

https://blog.secodis.com/2018/01/23/statistics-web-application-security-breaches/

Ressources

Tools	**BurpSuite**	**Wireshark**	**Kali Linux**
Websites	**root-me.org** **hackthis.co.uk**	**vulnhub.com**	**Youtube: Defcon Conferences**
Readings / Courses	**The Web Application Hacker's Handbook v.2**	**OWASP Testing Guide v.4**	**Metasploit Unleashed (OFSEC)**

What we didn’t have time to cover…

- Privacy / Anonymity
- Cybercrime Laws
- Cybercrime Impact on National Security and Infrastructures


Thank You
&&
Happy Hacking