Hello, this is Gasser Ahmed.

Welcome to this presentation on

how to block web form spambots

using the honeypot technique.

So first, let’s talk about web forms. what is a web form?

A web form, also called an HTML form, is an online page that allows for user input.

It is an interactive page that mimics a paper document or form

where users fill out

particular fields

like name, phone number, or

email addresses.

Web forms can be rendered

in modern browsers using

HTML and related

web-oriented languages.

A web form can also contain a

combination of form elements

such as input, checkbox,

submit button, etc.

Alright, so after we knew

what a webform is

let’s talk about spambots and

how they affect web forms.

A spambot is a piece of software

written with the specific

purpose of filling out those web forms

with fake information

that benefits

the spambot author.

Including abusive language,

ads, spam links to malware

and phishing websites

set up by scammers.

Most form spams are

created by bots

which are programmed to

find web forms and fill them out.

When a person clicks on

these spam links, they may be

susceptible to malware downloads

or loss of confidential information.

Spam links may also be posted by bots to generate traffic to shady sites that generate ad revenue

through them.

So how can we

stop a spambot?

In order to do that, we have to think like a programmer writing a spambot.

The simplest of spambots see a form and fill in every field on the form.

So, what’s the solution?

The honeypot technique

in my opinion is

one of the best techniques

to stop a spambot.

However, before I go into details on

what honeypot technique is

and how to implement it I want

to cover two other options that are still in use to prevent spam,

and why you shouldn’t use them.

The first is captcha, which I’m sure you saw it at least once on some websites. And the other is adding a test question to your form.

So captcha is an image that renders text in a not-so-easy-to-read way, also known as

challenge text.

By requiring users to type the challenge text into a text field

it verifies some form of human interaction and intelligence.

So, if what the user enters matches the challenge text, the user is said to have successfully completed

the challenge and

their form submission is

allowed to proceed.

Spam bots, on the other hand, often lack the intelligence to defeat the challenge.

First because the challenge text appears in an image

not html markup, reducing their chances of reading it.

And second, because they’re often

unaware that the form

field attached to the captcha is

looking for a specific entry.

That is, most spam bots fail captchas due to one of these reasons.

Alright, the other option we can use to stop a spambot is adding a test question to your form by implementing a question and answer field.

For example, a signup form may include the following questions

what is smaller, 8 or 9?

or what comes first, E or Y?

or what is the capital letter of “g”?

Humans can easily

answer those questions

whereas spam bots will

not be smart enough.

Once the form is submitted, the answer to the question can be tested.

If it’s correct the form was likely submitted by a human

and can be handled accordingly,

otherwise, a spambot is detected.

So, after we talked about captcha and test question techniques, there’re for sure downsides

for using them.

So please, pause the video,

and try to figure out

why we should avoid using

any of those techniques.

Okay, while both options are easy and help prevent spam

I don’t recommend them because they interfere with the user experience.

Often times they’re frustrating to deal with and motivate users to leave.

A good example for that

would be captchas

that output text too hard

for even humans to read.

For that reason I always

recommend implementing

the least invasive option available which

would be the honeypot technique and that’s what we will be discussing for the rest of this video.

So, a honeypot in terms

of form submissions

means that you are setting up

an extra hidden field

that bots will see and fill out,

but your real human users will not.

It is basically behind the scenes filtering step that protects your forms.

It also doesn't add any additional steps for the real people trying to submit your form like what a captcha does.

So that is, a spambot fills in that field that valid users cannot see, alerting us to their activity.

If the honeypot field is filled in, we can confidently reject the form as a spam.

You can implement this

by adding HTML

and specifically styling

it out using CSS.

It's true though that some sophisticated bots can now read CSS and JavaScript.

However, this is still

an effective method and worth considering if you've got some basic programming skills.

Alright, now after we know what honeypot is, let’s walkthrough its implementation on a simple HTML contact form that doesn’t have honeypot implemented yet

In this walkthrough example

we have a contact form

that consists of the

following 4 elements:

name label

name input

email label

and finally

an email input.

So most of simple bots will search

web form common patterns

like label common names, input id’s, input

common attributes, required fields, etc.

and then the bot will

fill them with fake info.

Also, most common

fields to search

are fields named like

email, phone, or address.

So, lets cheat on that and

create a honeypot field

with the same name as

one of the default fields

and make it look

real with a label

as we don’t want to alert the bot in

any way that this field is special.

So, here we added another email label and input fields.

Next, we need to rename our default fields to something random.

here I renamed name fields to be “nHash” and email ones to “eHash”.

then we need to change the email field type to text instead of email.

By naming the default fields to something random and changing their type to text

the valid fields now begin to look like honeypots to the spambot.

Keep in mind though

that you have to

convert them back to their

proper name on the server side.

Then we need to hide the honeypot field

to keep the valid users from filling it out.

In our form, I hide it with CSS, but you can also hide it with JavaScript.

With CSS, I do it by setting the opacity, height, and width properties to zero. Then I move it to the top

left of the page

by setting top and left

properties also to zero

with an absolute position to make sure it cannot be inspected or detected.

I also added a z-index of -1

that will simply put

that element behind any

other elements in the page

which will make it almost impossible to detect that element.

Another thing to mention, for the CSS class, it needs to be a random word.

In other words, if you call it something like “hide”

then the spambot writer

will pick it out easily.

For that reason,

I called it “nohoney”.

now we finished setting up

our CSS class

and we can add it to

our honeypot HTML element.

and then we will notice the honeypot field is no longer visible in our web form. and it’s ready to attract spambots.

For the last step, on your backend, verify if that honeypot field came filled.

If yes, congrats,

you trapped a spam.

If you prefer, you can do

this check on the client side.

In case of an ajax form, this will avoid use server resources

to compute unuseful data but keep the backend validation anyway.

When you catch a spam, just don’t send the data and do whatever you want with it.

So to wrap up what we did, first, we created a honeypot field with the same name as one of the default fields which was the email field.

then we renamed the

default name and email fields to

"nHash" and "eHash"

respectively.

Then we made the honeypot invisible to real users so they don’t fill it out.

and finally, we verified if the honeypot field came filled.

If yes, then we trap the spam

and we can stop sending the

data to backend and then

stop the form submission.

So as a wrap up, honeypots are awesome because they don’t inconvenience users

like what captcha or test question

techniques do

and they are a valid tool

for avoiding spam bots.

Also, in the context of software

engineering, honeypots improves

the quality of the software not only

by increasing its security

and protection levels

against spambots

but by also providing a more user-friendly experience

than other spam

prevention techniques

that interrupts the form

submission process.

In addition, honeypots are considered as a modern programming practice

which is important to meet

user’s requirements

in a latest and

advanced way.

However, we need to know that

honeypot is just a simple layer

to prevent attacks

in a simple way

and some technologies can identify even these patterns.

So, use all the weapons

you can against spambots.

But I believe that

this simple pattern

can avoid at least 50% of

spams in your webpage.

Thank you for watching!