Slide 1

Hello, this is Gasser Ahmed, [pause] welcome to this presentation on how to block web form spambots using the honeypot technique.

Slide 2

So first, let’s talk about web forms [pause] what is a web form? [space] 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 or phone number. [pause] Web forms can be rendered in modern browsers using HTML and related web-oriented languages [pause] A web form contains a combination of form elements such as input [space], checkbox [space], submit button [space], etc.

Slide 3

Alright, so after we knew what a webform is, let’s talk about spambots and how they affect web forms [pause][space] 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. [pause][space] Most form spams are created by bots which are programmed to find web forms and fill them out. [pause][space] When a person clicks on these spam links, they may be susceptible to malware downloads or loss of confidential information. [pause][space] Spam links may also be posted by bots to generate traffic to shady sites that generate ad revenue through them.

Slide 4

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. [pause] So, what’s the solution? [pause][space] The honeypot technique in my opinion is one of the best techniques to stop a spambot. [pause] However, before I go into detail 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.[pause][space] the first is captcha, which I’m sure you saw it at least once on some websites [pause][space] and the other is adding a test question to your form.

Slide 5

So captcha is an image [space] that renders text in a not-so-easy-to-read way, also known as *challenge text*. [pause][space] By requiring users to type the challenge text into a text field, it verifies some form of human interaction and intelligence. [pause] 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. [pause][space] Spam bots, on the other hand, often lack the intelligence to defeat the challenge.[pause] First because the challenge text appears in an image, not html markup, reducing their chances of reading it. [pause] And second, because they're often unaware that the form field attached to the captcha is looking for a specific entry. [pause] That is, most spam bots fail captchas due to one of these reasons.

Slide 6

Alright, the other option we can use to stop a spambot [pause] 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: [pause][space] *What is smaller, 8 or 9?* [pause][space] or What comes first, E or Y?[pause][space] or [pause][space] What is the capital letter of "g"? [pause] 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 [pause] 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.

Slide 7

[pause 5 seconds] okay, while both options are easy and help prevent spam, I don't recommend them [pause] because they interfere with the user experience. [pause] Often times they're frustrating to deal with and motivate users to leave. [pause][space] A good example for that [pause] would be captchas that output text too hard [click] for even humans to read. [pause] For that reason I always recommend implementing the least invasive option available which would be the honeypot technique [pause] and that's what we will be discussing for the rest of this video.

Slide 8

So a honey-pot in terms of form submissions means that you're [space] setting up an extra hidden field that bots will see and fill out, but *your* real human users will not. [pause] It's basically behind the scenes filtering step that protects your forms. [pause] It also doesn't add any additional steps for the real people trying to submit your form like what a captcha does. [pause][space] So that is, a spambot fills in that field that valid users can not see, alerting us to their activity. [pause][space] If the honeypot field is filled in, we can confidently reject the form as a spam. [pause] You can implement this by adding HTML and specifically styling it out using CSS. [pause] It's true though that some sophisticated bots can now read CSS and JavaScript. [pause] However, this is still an effective method and worth considering if you've got some basic programming skills.

Slide 9

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 [pause][space] In this walkthrough example we have a [space] contact form that consists of the following 4 elements: [space] name label [space] name input [space] email label and finally [space] email input.

Slide 10

So most of simple bots will search web form common patterns, like [space] label common names, [space] input id's, input common attributes, required fields, etc, and then the bot will fill them with fake info. [pause] Also, the most common fields to search are fields [space] named like email, phone, or address.

Slide 11

So, lets cheat on that and create a honeypot field with the same name [space] as one of the default fields and make it look real with a label [pause] as we don't want to alert the bot in any way that this field is special. [pause] So here we added another email label and input fields.

Slide 12

Next, we need to rename our default fields to something random [pause] here I [space] renamed name fields to be "nHash" and [space] email ones to "eHash" [pause] then we need to change the email field type to text [space] instead of email. [pause] 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. [pause] Keep in mind though that you have to convert them back to their proper name on the server side.

Slide 13

Then we need to hide the honeypot field to keep the valid users from filling it out. In our form, I hide it with [space] CSS, but you can also hide it with JavaScript. [pause] With CSS, I do it by setting the [space] opacity, [space] height, and [space] width properties to zero then I move it to the top left of the page by setting [space] top and [space] left properties also to zero with an [space] absolute position to make sure it can not be inspected or detected. I also added a [space] z-index of -1 [pause] that will simply put that element behind any other elements in the page [pause] which will make it almost impossible to detect that element. [pause] Another thing to mention, [pause] for the CSS class, it needs to be a random word. In other words, if you call it something like [pause] "hide"[pause], then the spambot writer will pick it out easily. For that reason, I called it [pause][space] "nohoney". [pause] now we finished setting up our CSS class and we can add it to our honeypot HTML element. [pause][space] and then we will notice the honeypot field is no longer visible in our web form and it's ready to attract spambots.

Slide 14

For the last step, [pause] on your backend [pause] verify if that honeypot field came filled. If yes, congrats, you trapped a spam. [pause] If you prefer, you can do this check on the client side. [pause] in case of an ajax form, this will avoid use server resources to compute unuseful data but keep the backend validation anyway. [pause] When you [space] catch a spam, just don't send [space] the data and do whatever you want with it.

Slide 15

So to wrap up what we did, first, we [space] created a honeypot field with the same name as one of the default fields which was the email field, [pause][space] then we renamed the default name and email fields to nHash and eHash respectively. [then][space] then we made the honeypot invisible to real users so they don't fill it out [pause][space] 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.

Slide 16

So as a wrap up, honeypots are awesome because they [space] don't inconvenience users like what captcha or test question techniques do, and they are a valid tool for avoiding spam bots. [pause] Also, in the context of software engineering, [space] 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 [pause] In addition, honeypots are considered as a [space] modern programming practice which is important to meet user's requirements in a latest and advanced way [pause] However, we need to know that honeypot is just a simple layer to prevent attacks in a simple way, [pause] and some technologies can identify even this patterns, [pause] so [space] 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.[pause] Thank you for watching!