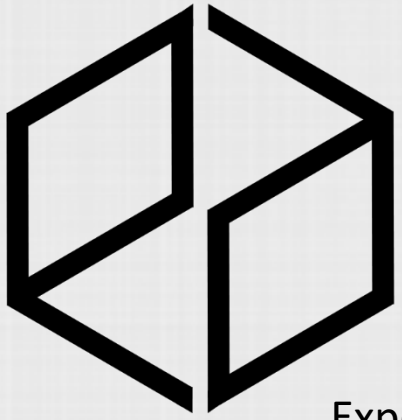
www.pandoralabs.net

PANDORA
SECURITY LABS

Expert advice. Experience advantage.
Proactive Security Solutions Through Cutting-Edge Research.

Web Application Security



PANDORA SECURITY LABS

Expert advice. Experience advantage.
Proactive Security Solutions Through Cutting-Edge Research.
www.pandoralabs.net

We are a Security-as-a-Service company

Providing businesses with on-demand **threat detection & intelligence** resources and capabilities, for **24x7 protection**.

We Make IT Secure

ThreatScout

Security Information & Event Manager

www.pandoralabs.net

WebRanger

Web Application Firewall & CDN

www.webranger.io

& WE ARE COOKING MORE!

We Build Security Software

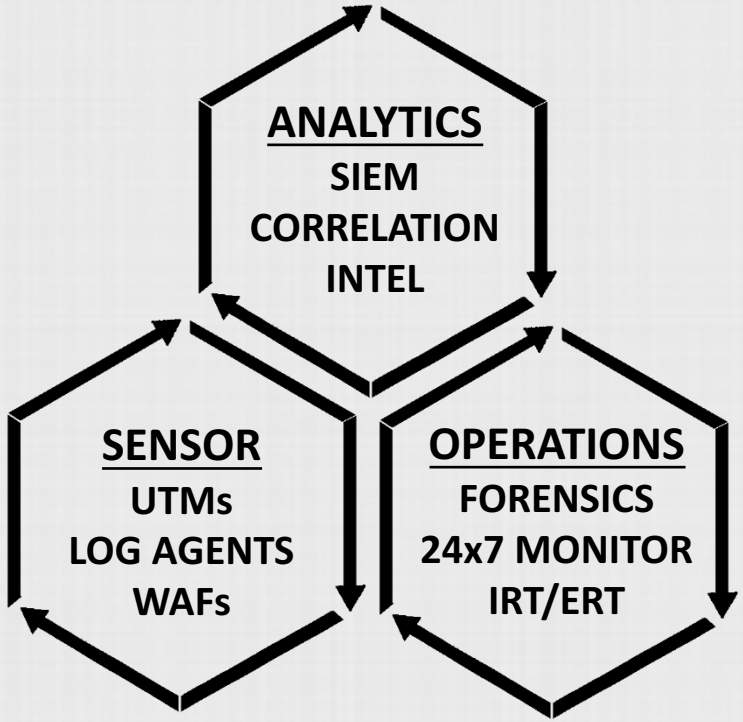
What do we really do?

Our Capabilities

Solutions we created from our capabilities to complement you needs:

- **Defensive Technologies**
- **Offensive Intelligence**
- **Administrative Expertise**

ThreatScout



WebRanger
FORTINET
SOPHOS
pfSense

Pandora SOC

Developing Secure Web Apps

Tips in to developing a secure web application.

Always Use TLS

- Transport Layer Security
- Latest version 1.2
- Ensures that data is encrypted as it travels the wire
- Ensures integrity using message authentication code
- Let’s Encrypt project provides free SSL certificates for TLS
- Use tools like SSL Test by Qualys to verify TLS configuration

Always Use TLS

Let me demo:

Wireshark plaintext

SSL Labs

Never store plaintext passwords

Hash passwords when you store them in the database

Provision your code and database such that the hashing algorithm can be changed

ALWAYS hash with unique salts per record, this prevents rainbow table attacks

Never store plaintext passwords

Let me demo:

Google the plaintext of hash

Hash with salt logic

Use Strong Authentication

Strong authentication (such as tokens, certificates, etc.) provides a higher level of security than username and passwords.

The generalized form of strong authentication is "something you know, something you hold".

Use Strong Authentication

When to use strong authentication:

- For high value transactions
- Where privacy is a strong or legally compelled consideration (such as health records, government records, etc)
- Where audit trails are legally mandated and require a strong association between a person and the audit trail, such as banking applications
- Administrative access for high value or high risk systems

Use Strong Authentication

Best practices:

- Authentication is only as strong as your user management processes
- Use the most appropriate form of authentication suitable for your asset classification
- Re-authenticate the user for high value transactions and access to protected areas (such as changing from user to administrative level access)
- Authenticate the transaction, not the user
- Passwords are trivially broken and are unsuitable for high value systems.

Enforce Good Session Management

Session management is by its nature closely tied to authentication, but this does not mean users should be considered authenticated until the web application has taken positive action to tie a session with a trusted credential or other authentication token.

If possible, tie a session to a specific IP. Force re-authenticate if the IP changes. This is to prevent hijacking and replay attacks.

Enforce session timeouts.

Enforce Good Session Management

Ensure that unauthenticated users does not have any or have minimal privileges only.

Ensure all unprotected pages use as few resources as possible.

Ensure that session tokens are user-unique, non-predictable, and resistant to reverse engineering.

Use Parameterized Queries / Stored Procedures

- Injection happens when data is supplied from one component to another
- Hackers "inject" their code to run instead of yours
  - Example: SQL injection attack String ***query = "SELECT * FROM products WHERE name='" + request.getParameter("id") +"';***
- Code expects a nice parameter in the URL
  - ***http://example.com/products?id=123***
  - Hacker could instead supply this: ***http://example.com/products?id=';+DROP+TABLE+'products';***

Use Parameterized Queries / Stored Procedures

Example:

```
String prodId= request.getParameter(“productId");
String query = "SELECT product_status FROM product_data WHERE product_id = ?
    ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, prodId);
ResultSet results = pstmt.executeQuery( );
```

Sanitize and Validate User Input

Always assume the data is "evil"

ALWAYS sanitize input! (at the BACKEND not front end!)

Encode all user input before using it

Clean up quotes, semi-colons, parentheses, etc.

Sanitize and Validate User Input

Data should be:

- Strongly Typed at all times
- Length Checked and Fields Length Minimized
- Ranged check if numeric
- Unsigned unless required to be signed
- Syntax or grammar should be checked prior to first use or inspection
- Sanitized

Sanitize and Validate User Input

Coding guidelines should use some form of visible tainting on input from the client or untrusted sources, such as third party connectors to make it obvious that the input is unsafe:

taintedPostcode = getParameter(“postCode”);

validation = New Validation();

postCode = validation.isPostcode(taintPostcode);

Sanitize and Validate User Input

Let me demo an old vulnerability: Wordpress

Use Anti-CSRF Tokens

Anti-csrf tokens adds a unique token that must be included with the data submission.

```
<% using(Html.Form("UserProfile", "SubmitUpdate")) { %>
    <%= Html.AntiForgeryToken() %>
   <!-- rest of form goes here -->
<% } %>
```

The output will be something like:

```
<form action="/UserProfile/SubmitUpdate" method="post">
   <input name="__RequestVerificationToken" type="hidden"
   value="saTFWpkKN0BYazFtN6c4YbZAmsEwG0srqlUqqloi/fVgeV2ciIFVmelvzwRZpArs" /> <!-- rest of form goes
   here -->
 </form>
```

Use Anti-CSRF Tokens

```
public class UserProfileController : Controller {
   public ViewResult Edit() { return View();
}
[ValidateAntiForgeryToken]
public ViewResult SubmitUpdate() {
    // Get the user's existing profile data (implementation omitted)
    ProfileData profile = GetLoggedInUserProfile();

      // Update the user object
      profile.EmailAddress = Request.Form["email"];
      profile.FavoriteHobby = Request.Form["hobby"];
      SaveUserProfile(profile);

      ViewData["message"] = "Your profile was updated.";
      return View();
   }
}
```

Log Relevant Data

- **Auditable** – all activities that affect user state or balances are formally tracked
- **Traceable** – it's possible to determine where an activity occurs in all tiers of the application
- **High integrity** – logs cannot be overwritten or tampered by local or remote users
- **Audit logs are legally protected** – protect them

Log Relevant Data

Data from logs can be used to monitor your application

Never log confidential data!

Have an SIEM collect logs and to help you out monitor your applications

Never Disclose Information via Error Messages

- Stack traces show the inner workings of an application
- Do not give attackers clue about your application (ie. Invalid username / password)
- Use generic error messages
- **Do not** send the "**username**" in your password reset emails

Never Disclose Information via Error Messages

Example with Tomcat:

In **CATALINA_HOME**/conf/web.xml , add the following entry.

```
<error-page>
   <exception-type>java.lang.Throwable</exception-type>
   <location>/error.jsp</location>
</error-page>
```

Never Disclose Information via Error Messages

Example in .NET:

In the Web.config file at the application's root, add the following entry.

<configuration> <compilation debug="true"/> </configuration>

Also, consider having a generic error page:

<customErrors mode="On" defaultRedirect="YourErrorPage.htm" />

Never Disclose Information via Error Messages

Let me demo: Joomla

Secure Your Components

- Realities:
  - We did not write the code for every component in our stack
  - We reuse code, components, and libraries
- Use dependency injection tools to manage libraries
  - Maven, NuGet, Cocoa Pods, Npm
- Software should always be kept up to date
- Vulnerability Assessment / Penetration Testing can catch outdated components
- Always check the issue tracker or repository of a library/component before using it

Secure Your Components

- Check your component has vulnerabilities by their Common Vulnerability Enumeration (CVE)
  - https://cve.mitre.org/cve/cve.html

Secure Your Components

cve.mitre.org

Employ Security Testing

Use OWASP Top 10 and OWASP Testing Guide

OWASP Zap

Employ Security Testing

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

Employ Security Testing

Not an OWASP Fanboi?

Employ Security Testing

- CWE/SANS Top 25 Dangerous Software Errors

Employ Security Testing

1. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
3. Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
4. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
5. Missing Authentication for Critical Function
6. Missing Authorization
7. Use of Hard-coded Credentials
8. Missing Encryption of Sensitive Data
9. Unrestricted Upload of File with Dangerous Type
10. Reliance on Untrusted Inputs in a Security Decision
11. Execution with Unnecessary Privileges
12. Cross-Site Request Forgery (CSRF)
13. Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Employ Security Testing

14. Download of Code Without Integrity Check
15. Incorrect Authorization
16. Inclusion of Functionality from Untrusted Control Sphere
17. Incorrect Permission Assignment for Critical Resource
18. Use of Potentially Dangerous Function
19. Use of a Broken or Risky Cryptographic Algorithm
20. Incorrect Calculation of Buffer Size
21. Improper Restriction of Excessive Authentication Attempts
22. URL Redirection to Untrusted Site ('Open Redirect')
23. Uncontrolled Format String
24. Integer Overflow or Wraparound
25. Use of a One-Way Hash without a Salt

www.webranger.io

WebRanger

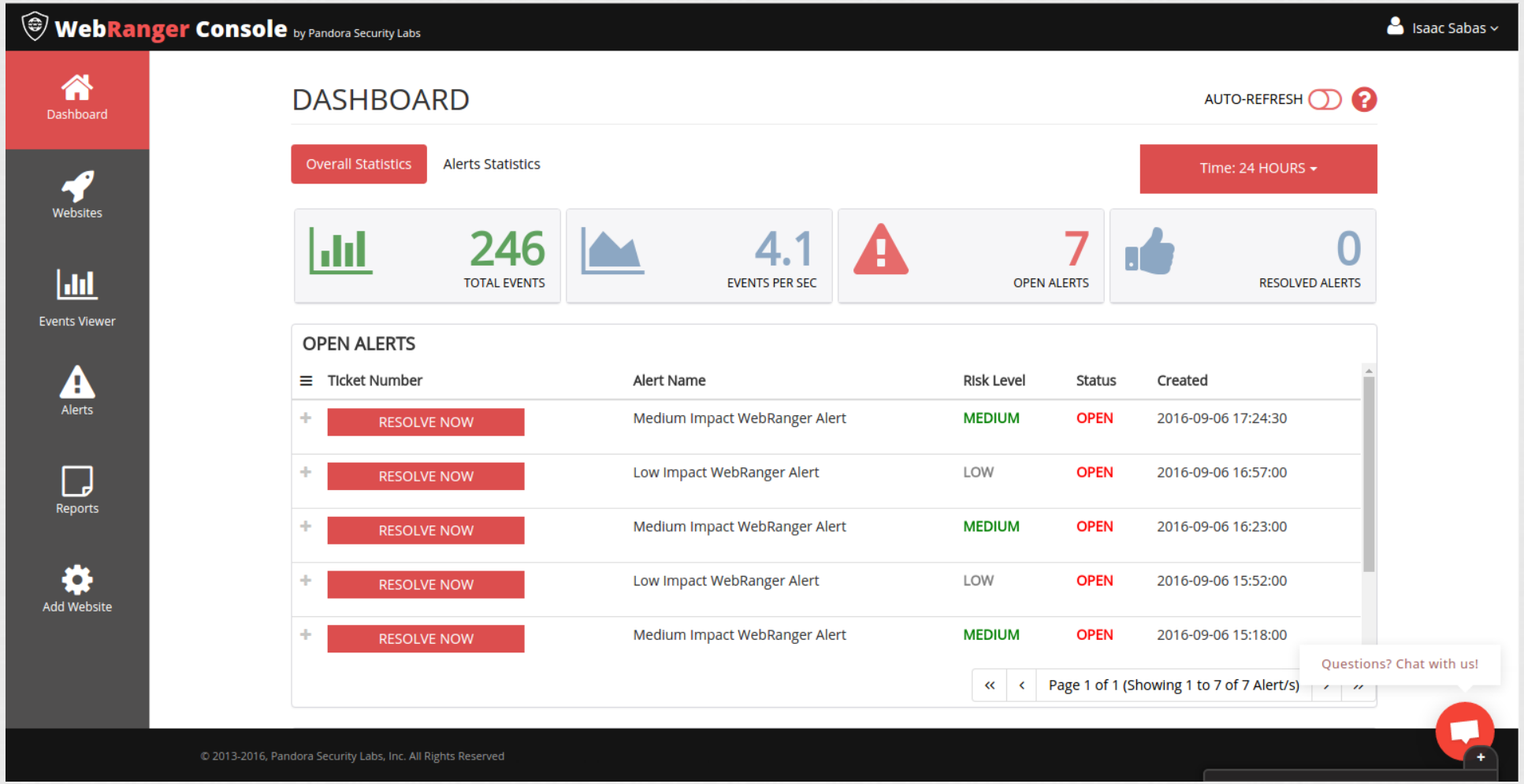
We Ensure Website Security

Securing Your Websites Through **WebRanger**


THREAT
DETECTION &
INTELLIGENCE

1. Awareness is key

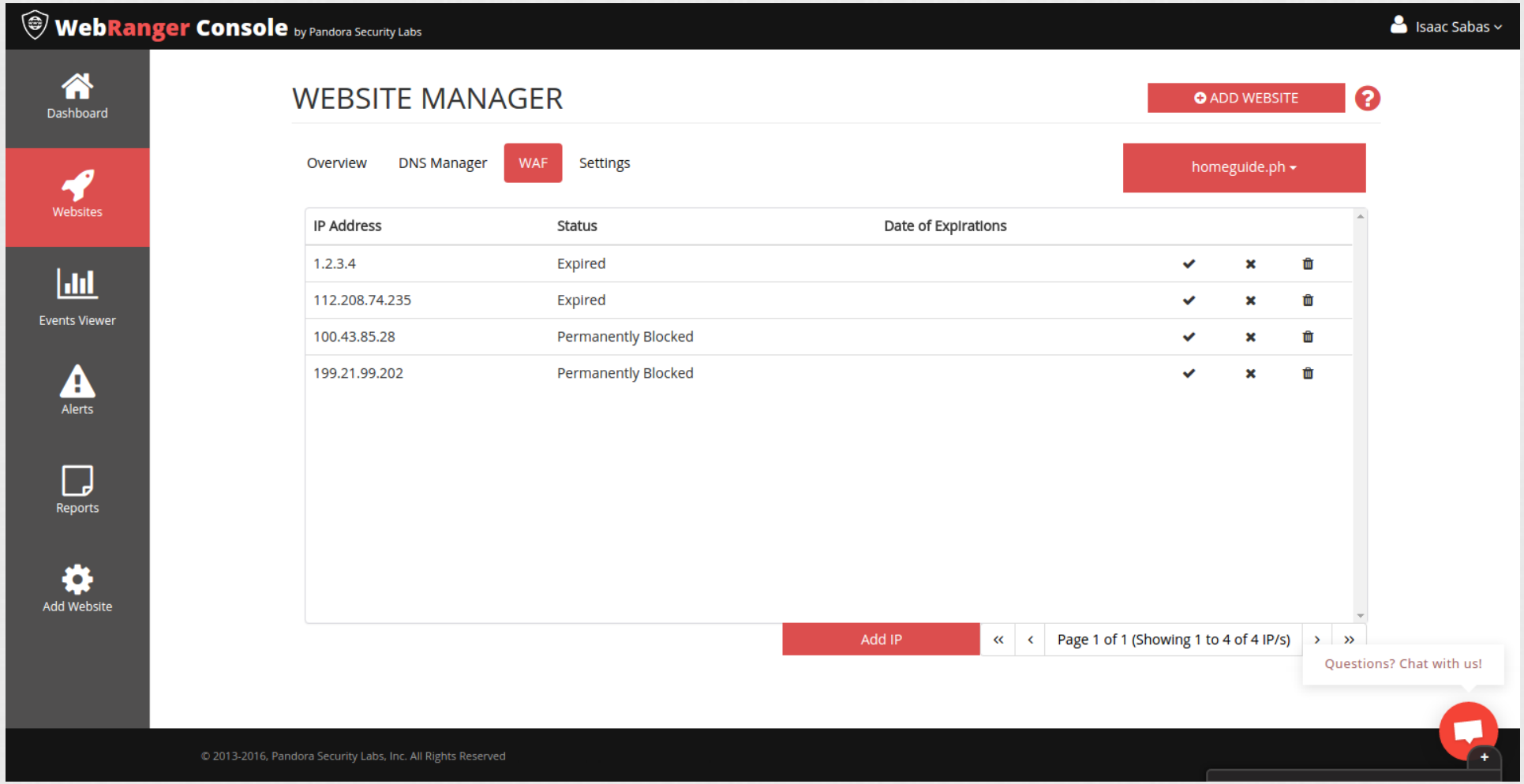
Awareness is the **greatest agent** for change and action.



Awareness is the **greatest agent** for change and action.

2. Access control and performance

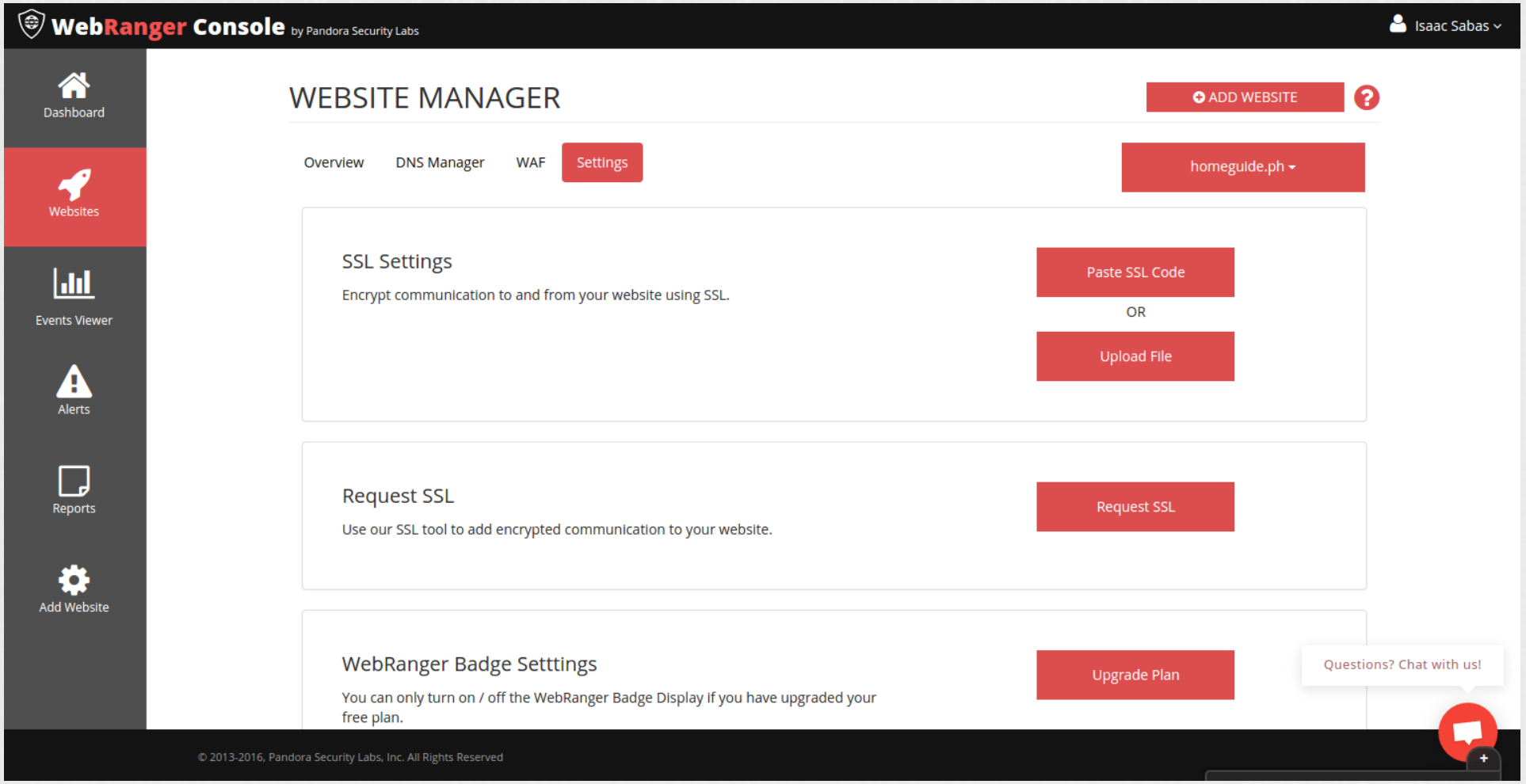
WAF & **CDN** to provide access control and **performance boost** to your site



Web Application Firewall (**WAF**) to **block** threats accessing your website

3. Encrypted communication

Ensuring your users that you are communicating **securely** with them



Get your **FREE** SSL certificate to enable you site to utilize **HTTPS**

WebRanger

Web Application Security

Web Application Security by Pandora Security Labs that protects your web app using all best defensive solutions in 1:

WAF + Threat Analytics + 24x7 Analysts.

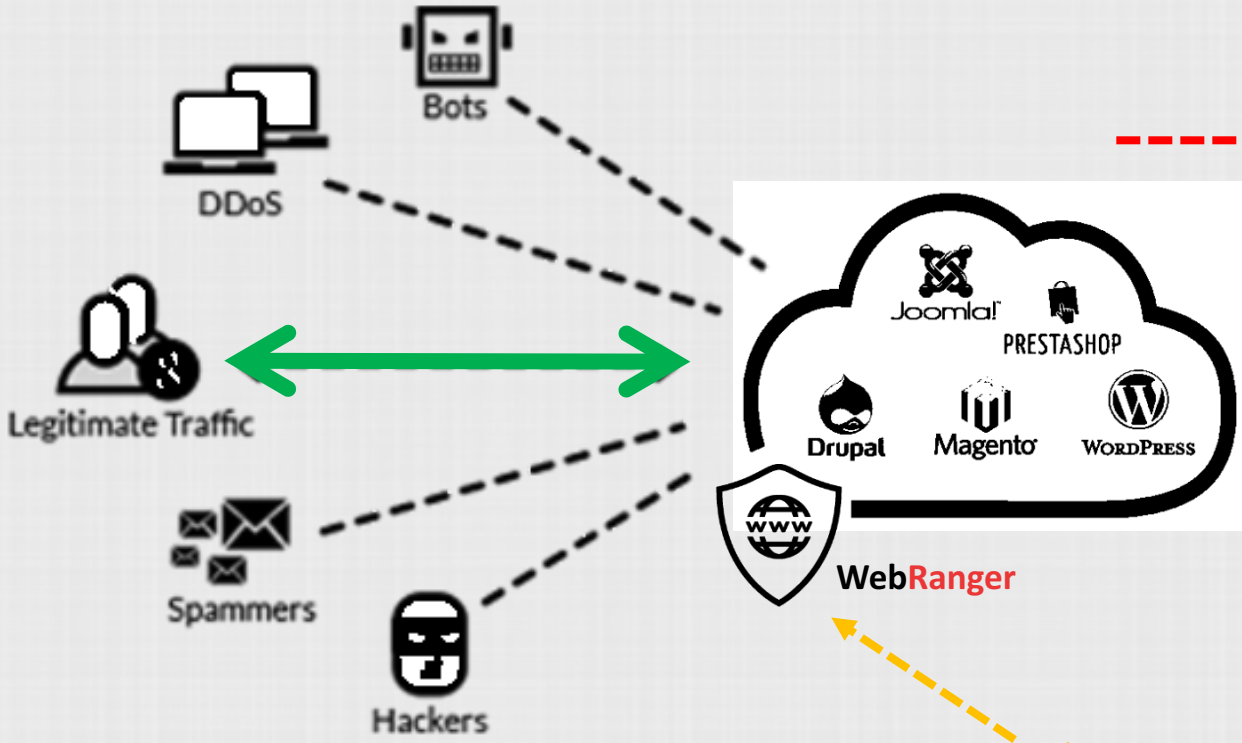
How does WebRanger work to protect your website?

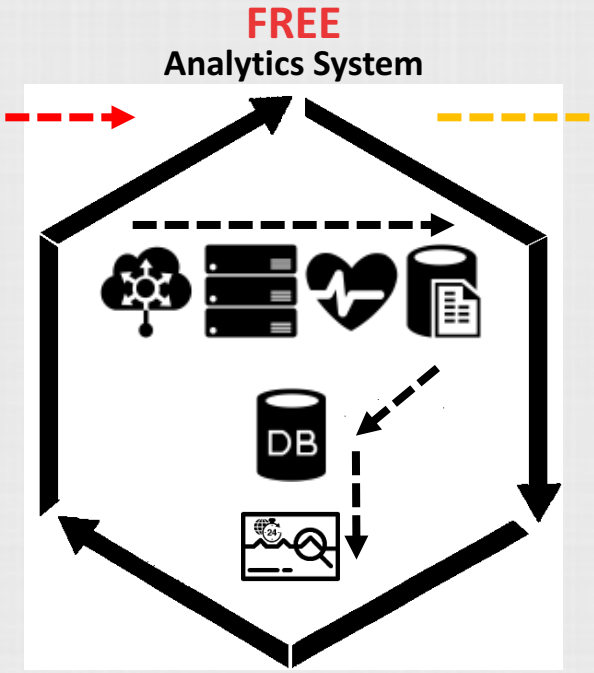
1. Attackers attack your website

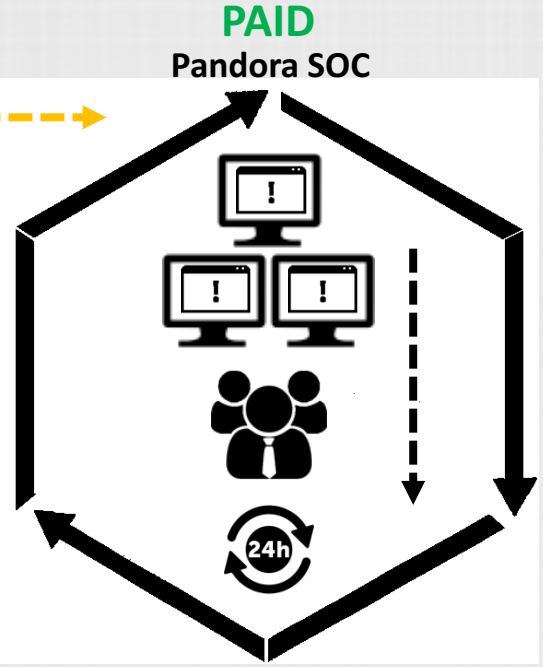
2. WebRanger identifies anomaly and sends data to the analytics system

3. The analytics system correlates data and sends the alerts to SOC

4. The SOC determines if alert is a true alert and informs the client







5. Pandora SOC commands the WAF to block the attack

6. True alerts are then communicated to the client either via phone or email

7. Client views WebRanger Console for the alerts and attacks resolved

8. Attacks with same patterns are blocked

ESSENTIALS	BASIC PLAN	PRO PLAN	BUSINESS PLAN
FREE	**$19**	**$49**	**$149**
Forever	Per Month	Per Month	Per Month
Attack detection ~~& blocking code~~	Attack detection & blocking code	Attack detection & blocking code	Attack detection & blocking code
1 Free Tier / Account	Unlimited Basic Plan / Account	Unlimited Basic Plan / Account	Unlimited Basic Plan / Account
1GB storage	1GB storage	5GB storage	10GB storage
~~24×7 monitoring & incident handling by Pandora SOC~~	24×7 monitoring & incident handling by Pandora SOC	24×7 monitoring & incident handling by Pandora SOC	24×7 monitoring & incident handling by Pandora SOC
~~Email support for alerts~~	Email support for alerts	Email support for alerts	Email support for alerts
~~Phone support for alerts~~	~~Phone support for alerts~~	~~Phone support for alerts~~	Phone support for alerts

Sign up for FREE

WebRanger.io

We Ensure Website Security

www.pandoralabs.net

PANDORA SECURITY LABS

Expert advice. Experience advantage.

Proactive Security Solutions Through Cutting-Edge Research.

Web Application Security