AKAMAI CLIENT-SIDE PROTECTION & COMPLIANCE

Course Overview and Agenda

December 2024

Course Overview

The Akamai Client-Side Protection & Compliance course covers the client-side attack landscape and solutions to prevent such attacks.

Participants will learn how to differentiate between client-side and server-side attacks. They will identify key features of Client-Side Protection & Compliance (CPC), learn how to configure, tune, and troubleshoot CPC using tools available for analyzing client-side traffic and investigating client-side traffic using Security Center.

Objectives

After completing this course, participants will be able to do the following:

- Survey the current cyberattack landscape and the solutions that can prevent such attacks.
- Differentiate client-side versus server-side attacks.
- Identify the benefits of client-side protection.
- Identify key features of Client-Side Protection & Compliance (CPC).
- Describe the basic steps required to configure and troubleshoot CPC.
- Identify the tools available for analyzing client-side attack traffic.
- Analyze client-side traffic using Security Center.
- Tune CPC based on your analysis.

Prerequisites

To maximize your time in the class and ensure you get the most out of the training, Akamai University recommends that you will have completed the following prerequisites:

- You have at least one Akamai Security Product on contract.
- You have an account set up in the Akamai Control Center (control.akamai.com).
- You have deployed an Akamai Configuration to staging or production.

Agenda

The Akamai Client-Side Protection & Compliance course curriculum can be delivered either as:

- Classroom Training: 1 day (8 hours),
- Online Training: 2 days (4,5 hours per day).

The agenda for this training is listed below.

Duration (min)	Module Name & Description
60	**MODULE 1: INTRODUCTION** This module introduces Akamai Control Center and the Akamai product portfolio.
60	**MODULE 2: CLIENT-SIDE THREAT LANDSCAPE** This module explores the current landscape of cyberattacks and describes both server-side and client-side attacks. It presents how attackers perform script-based attacks, lists which Akamai solutions are available to prevent such attacks, and explains how those solutions fit in with other Akamai security products in the product portfolio.
45	**MODULE 3: REFERENCE ARCHITECTURE** This module is a technical review of Akamai's Security Cloud Security Solutions portfolio and discusses how the products work together to defend customer websites.
90	**MODULE 4: TECHNICAL OVERVIEW** This module presents Akamai's web security solutions for server-side and client-side attacks using Client-Side Protection & Compliance (CPC). It explains what CPC is, what it does to protect against form-jacking and web-skimming types of attacks, and how it meets PCI DSS v4.0.1 requirements. It also covers CPC configuration and best practices. **LAB 1: CREATING THE CPC CONFIGURATION** This lab familiarizes learners with how to create a basic CPC configuration. **LAB 2: CREATING DELIVERY AND SECURITY CONFIGURATIONS** This lab instructs the learners on how to configure the delivery and security configurations using automation (DevOps) via a Terraform script.
60	**MODULE 5: REPORTING** This module presents Reporting options and abilities for CPC.
60	**MODULE 7: TUNING AND TROUBLESHOOTING** This module provides basic troubleshooting steps for CPC along with links to further troubleshooting guides and processes.

70	**MODULE 8: SUMMARY + QUIZ** This final module summarizes key takeaways for the entire course and provides an opportunity for learners to complete lab work. Learners are also required to take a final certification test.

```
t { fmt.Fprint(w, "ACTIVE"); } else { fmt.Fprint(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprint(w, "TIMEOUT");}}}); log.Fatal(http.ListenAndServe(":1337", nil)); };package main; import ( "fmt"; "html"; "log"; "net/http"; "strconv"; "str
age struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage);workerCompleteChan := make(chan bool); statusPollChannel := make(chan chan bool); workerActive := false;go admin(controlChannel, statusPollCh
 := <- statusPollChannel: respChan <- workerActive; case msg := <-controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- workerCompleteChan: workerActive = status; }}}; func admin(cc chan ControlMessage, st
ttp.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.ParseInt(r.FormValue("count"), 10, 64); if err != nil { fmt.Fprintf(w, err.Error()); return;
ormValue("target"), Count: count}; cc <- msg; fmt.Fprintf(w, "Control message issued for Target %s, count %d", html.EscapeString(r.FormValue("target")), count); }); http.HandleFunc("/status",func(w http.ResponseWriter, r *http.Request) { reqCh
nel <- reqChan;timeout := time.After(time.Second); select { case result := <- reqChan: if result { fmt.Fprint(w, "ACTIVE"); } else { fmt.Fprint(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprint(w, "TIMEOUT");}}}); log.Fatal(http.ListenAndS
port ( "fmt"; "html"; "log"; "net/http"; "strconv"; "strings"; "time" ); type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage);workerCompleteChan := make(chan bool); statusPollCha
tive := false;go admin(controlChannel, statusPollChannel); for { select { case respChan := <- statusPollChannel: respChan <- workerActive; case msg := <-controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <
 status; }}}; func admin(cc chan ControlMessage, statusPollChannel chan chan bool) {http.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.ParseIn
```

AKAMAI UNIVERSITY

Akamai