MITIGATING BOTNET-BASED DDOS ATTACKS AGAINST WEB SERVERS

by

Peter L. Djalaliev

B.Sc., Washington and Lee University, 2005

M.Sc., University of Pittsburgh, 2009

Submitted to the Graduate Faculty of

the Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2013

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES, DEPARTMENT OF COMPUTER SCIENCE

This dissertation was presented

by

Peter L. Djalaliev

It was defended on

July 15, 2013

and approved by

Dr. Adam Lee, Assistant Professor, Department of Computer Science

Dr. Youtao Zhang, Associate Professor, Department of Computer Science

Dr. Daniel Mosse, Professor, Department of Computer Science

Dr. Prashant Krishnamurthy, Associate Professor, School of Information Sciences

Dissertation Director: Dr. Adam Lee, Assistant Professor, Department of Computer Science

MITIGATING BOTNET-BASED DDOS ATTACKS AGAINST WEB SERVERS

Peter L. Djalaliev, PhD

University of Pittsburgh, 2013

Distributed denial-of-service (DDoS) attacks have become wide-spread on the Internet. They continuously target retail merchants, financial companies and government institutions, disrupting the availability of their online resources and causing millions of dollars of financial losses. Software vulnerabilities and proliferation of malware have helped create a class of application-level DDoS attacks using networks of compromised hosts (botnets). In a botnet-based DDoS attack, an attacker orders large numbers of bots to send seemingly regular HTTP and HTTPS requests to a web server, so as to deplete the server's CPU, disk, or memory capacity.

Researchers have proposed client authentication mechanisms, such as CAPTCHA puzzles, to distinguish bot traffic from legitimate client activity and discard bot-originated packets. However, CAPTCHA authentication is vulnerable to denial-of-service and artificial intelligence attacks. This dissertation proposes that clients instead use hardware tokens to authenticate in a federated authentication environment. The federated authentication solution must resist both man-in-the-middle and denial-of-service attacks. The proposed system architecture uses the Kerberos protocol to satisfy both requirements. This work proposes novel extensions to Kerberos to make it more suitable for generic web authentication.

A server could verify client credentials and blacklist repeated offenders. Traffic from blacklisted clients, however, still traverses the server's network stack and consumes server resources. This work proposes Sentinel, a dedicated front-end network device that intercepts server-bound traffic, verifies authentication credentials and filters blacklisted traffic before it reaches the server. Using a front-end device also allows transparently deploying hardware acceleration using network co-processors. Network co-processors can discard blacklisted traffic at the hardware level before it wastes front-end host resources.

We implement the proposed system architecture by integrating existing software applications and libraries. We validate the system implementation by evaluating its performance under DDoS attacks consisting of floods of HTTP and HTTPS requests.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

PREFACE

There are many people I owe more gratitude than I can express. First, nothing good can be accomplished without wise guidance and leadership, so I would like to thank my advisor Dr. Adam Lee and my previous advisor Dr. José Brustoloni. They have been great teachers, mentors and role models for me. I would also like to thank my lovely wife Lory, whom I met during my first week as a Ph.D student. We did the whole ride through our doctorate studies together, hand in hand, and what a ride that was! She has always been there for me during good and bad times. I would like to thank my mother Donka for raising me with the attitude that I can accomplish anything I want if I really pursue it. My deepest gratitude also goes to the rest of my family for being there for me and with me. I would like to thank my friends at the University of Pittsburgh and outside for being there and for the great times we have had. Most of all, I would like to thank God for letting His light shine on us.

1.0 INTRODUCTION

The emergence of the World Wide Web has led to the rise of different types of online business transactions. The Web has become a media for retail merchants to conduct business in large, geographically distributed markets of consumers. Web technologies have provided 1) businesses with low costs of entry to vast markets of potential customers and 2) consumers with convenient and quick access to a large number of merchants from their home computers. Retailers store consumer goods in centralized warehouses and sell them through online stores, thus reducing or eliminating the costs of operating physical stores and large distribution networks to supply those stores. Consumers use the Web to research, compare prices of, and purchase a variety of consumer products and professional services, such as electronics, household items and online tax form assistance.

The Web has also enabled leaner, more effective business operations. It allows companies to outsource operations, such as manufacturing and accounting, and ensure tight integration between local and outsourced activities. New paradigms, such as cloud computing and "software as a service" (SaaS), have enabled businesses to outsource IT infrastructure and software applications to external providers, reducing the costs of acquiring and maintaining hardware and software resources.

The utility of the Web, however, depends on the continuous, on-demand availability of online resources. Online storefronts must be available and responsive at the time users access them. According to a 2009 survey, 47% of consumers expect to wait no more than 2 seconds for a web page to load and 40% would abandon the site if the page does not load within 3 seconds. 79% of consumers state that they would not buy online from a merchant again after a dissatisfactory online shopping experience and 27% say that they are also less likely to buy from the merchant's physical retail stores [25]. Within a business organization, employees must be able to accesses IT resources on demand. On average, businesses lose between \$84,000 and \$108,000 for every hour of IT system downtime [40].

1.1 DISTRIBUTED DENIAL-OF-SERVICE ATTACKS

Distributed denial-of-service (DDoS) attacks are a widespread threat to web servers and aim to disrupt the availability of web sites. In such an attack, an attacker orders multiple hosts to send malicious traffic toward a server, so as to deplete network bandwidth or server CPU, disk, or memory resources. A 2012 study among network operators reports that almost half experienced infrastructure outages due to DDoS attacks and 67% of them experienced attacks of more than 6 hours [27]. Web site downtime and brand reputation damage due to DDoS attack are costly. 70% of online retailers report losses of more than \$100,000 per hour and 80% of financial services companies report losses of \$10,000 per hour while under a DDoS attack [28].

DDoS attacks increasingly use *botnets* to generate floods of seemingly normal requests that mimic flash crowds, i.e natural peaks in client traffic. Botnets are networks of compromised hosts, *bots*, controlled by a *botmaster* [116]. The hosts are usually infected by malware that spreads like a virus (e.g., as e-mail attachments, downloadable files, or smartphone apps [100]) or a worm (e.g., by exploiting server applications with known vulnerabilities). Once the malware infects a host, it downloads the bot executable from a known URL. Contemporary botnets also recruit new bots through drive-by downloads [113] from compromised well-known web sites. A botmaster gains access to a well-known web server and injects into its documents code snippets, such as zero-pixel HTML IFRAMEs [112]. The IFRAME points to a botmaster-controlled web server with malicious code, which exploits known browser vulnerabilities and infects the client host. The Storm botnet in 2007 reportedly injected malicious IFRAMEs into a U.S. Republican Party web site [89]. Other infected well-known web sites include media web site USAToday.com and online retailers Walmart.com and Target.com [101].

Researchers have estimated that botnets control hundreds of thousands or even millions of hosts, e.g. Storm [129] with 400,000 hosts, Coreflood [66] with 2,000,000 hosts, Citadel [71] with 5,000,000 and Conficker [98] with 10,000,000 hosts. Bots receive commands from the botmaster through some command and control (C&C) infrastructure. Most bots use Internet Relay Chat (IRC) or HTTP servers for C&C, but peer-to-peer (P2P) networks, instant messaging, and Twitter are also used. Botmasters often use layers of indirection to obfuscate the identities of C&C servers [23] and the size of botnets [115].

Botnets are operated for profit and are available for hire [79]. Attackers use botnet-based DDoS attacks for online criminal activities, such as extorting commercial web sites [99] and disturbing the business of online competitors [65, 24]. Political opponents also employ DDoS attacks in cyber warfare. Opt-in botnets are established when users intentionally install software to partake in cyber attacks as a form of civil disobedience [121, 110, 111]. In 2006, anti-Russian actions by the Estonian government triggered DDoS attacks against Estonian government, financial and media web sites [60]. In recent years, some DDoS attacks had ideological motives. In 2012 and 2013, BroBot botnet was used to launch more than 200 separate DDoS attacks against U.S. banking institutions to protest a YouTube movie trailer deemed offensive to Muslims and demand its removal [92].

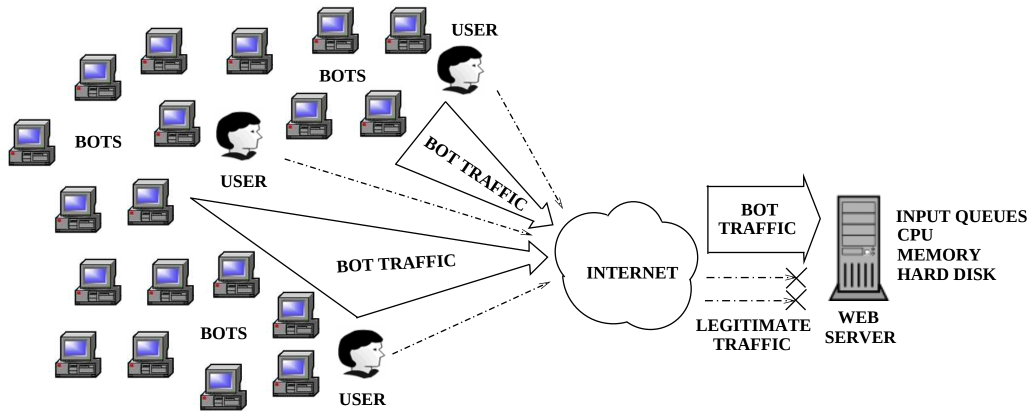


Figure 1: Botnet-based DDoS attacks

Botnets can target and disrupt the availability of web sites by generating application-level DDoS attacks consisting of floods of HTTP and HTTPS (HTTP over TLS) requests. During such an attack, bot-originated traffic crowds out legitimate client requests because botnets can generate requests at rates several magnitudes higher than legitimate clients (Fig. 1). Client and bot requests traverse the Internet and are received by the web server. If the web server's queues and buffers can hold more packets, it enqueues packets for processing. If not, it drops packets until resources become available. Processing a packet may require CPU resources, memory, and disk accesses. A flood of requests depletes the server's queueing capacity, causing both bot and client requests to experience long queuing delays and high packet drop rates. Bots typically flood servers without waiting for responses. Legitimate users, however, experience high and variable response times, causing them to give up or access a different web site.

TLS-enabled web servers are even more vulnerable to denial-of-service attacks than HTTP-only servers. Transport Layer Security (TLS) is a cryptographic protocol widely used to provide session-layer security over the Internet. Botnets can target the CPU capacity of TLS servers with only a modest number of simultaneous connections because TLS is asymmetric and can consume up to 15 times more resources on the server than on the client [30]. Servicing connections at high rates depletes the CPU capacity of a TLS server because an attacker can require the server to execute a CPU-intensive private key operation during each TLS handshake. TLS reduces the throughput of a web server by a factor of 3.4 to 9 times [59], making TLS servers especially vulnerable to distributed denial-of-service attacks. Major certificate authorities and Internet companies will transition to SSL server certificate with 2048-bit keys by the end of 2013 [32, 34, 128]. This will make TLS servers more vulnerable to DDoS attacks because 2048-bit RSA private key operations typically require 4-7 times more CPU cycles than their 1024-bit counterparts [36].

Existing work has not provided effective solutions for mitigating or preventing large application-level DDoS attacks. Research work and commercial solutions provide network-based defenses against attacks that aim to deplete resources at the network and transport level [1, 45]. However, network-based defenses are unable to detect application-level attacks that mimic legitimate client activity. Researchers have also proposed CAPTCHA puzzles [131], but CAPTCHAs are vulnerable to algorithmic artificial intelligence attacks [20], computationally intensive to generate and may be outsourced to online CAPTCHA-solving services [42] or Amazon's Mechanical Turk [51].

TLS servers can use commercially-deployed hardware SSL accelerators that can process up to 44,000 connections per second for 1024-bit keys and 24,000 connections per second for 2048-bit keys [36]. Botnets formed of millions of hosts have sufficient resources to generate attacks that overload the hardware accelerators. Researchers have proposed using client puzzles [86, 119, 61] to make TLS connections computationally expensive for clients, but botnets can flood a TLS server with only a few connections from each bot.

1.2 THESIS STATEMENT AND CONTRIBUTIONS

This thesis contributes novel solutions to the problem of mitigating distributed denial-of-service attacks against web servers. My goal is to provide sufficient evidence to satisfy the following thesis statement:

I claim that federated client authentication using hardware tokens can effectively protect HTTP and HTTPS servers against botnet-based DDoS attacks, and that hardware-accelerated server front-ends can transparently and scalably facilitate such authentication.

This dissertation presents the design and implementation of a system for mitigating DDoS attacks against web servers. I evaluate the system's security and performance to support the claims above. Fig. 2 presents an overview of the system's architecture. Different aspects of the architecture outline the contributions of this dissertation, as labeled in Fig. 2 and described below:

1. Clients authenticate with hardware tokens to prove that they are human users. They include proof of such authentication with web requests to make the requests distinguishable from bot traffic. Hardware token authentication proves the existence of human users because it is infeasible for bots to possess tokens or obtain token codes at sufficiently high rates to generate floods of requests. This dissertation shows that authentication using hardware tokens is more secure and has better performance than CAPTCHA authentication.

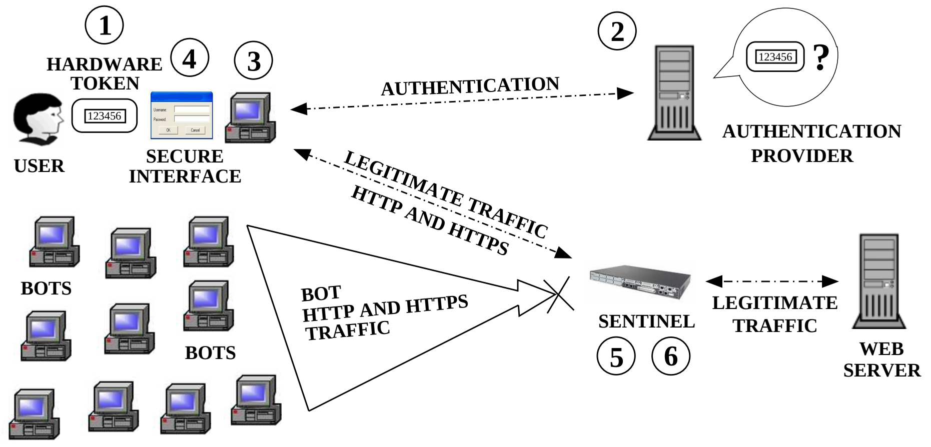


Figure 2: Overview of system architecture

2. Clients obtain Kerberos federated authentication credentials as proof of hardware token authentication and use them to access web servers under DDoS attack. Federated authentication providers are independent from, but trusted by web servers. Without federated authentication, users must possess an carry a hardware token for each web server they access. Federated authentication allows users to possess only one or a few tokens, making hardware token authentication more practical for open systems, such as the Web.

3. The client authenticates using a single authentication request and little client configuration. A Kerberos V5 protocol client must maintain configuration to discover the authentication paths to all desired servers and send multiple authentication request to each entity on the path. This has made Kerberos unsuitable for the Web, where clients with minimal configuration access arbitrary web servers. This dissertation extends the Kerberos architecture to make Kerberos authentication more suitable for the web. The extension enables a client with minimal configuration to authenticate using only the web server's domain name and a single authentication request.

4. Users enter their authentication credentials through a user interface resilient to spoofing attacks. This work integrates the Kerberos protocol with the Identity Metasystem authentication framework [53]. The integration improves the security and usability of

using Kerberos for user authentication on the Web.

5. A hardware-accelerated Sentinel device intercepts all server-bound HTTP traffic, admits requests with valid federated authentication credentials and drops bot-originated traffic. Blacklisting bots and dropping bot-originated packets on the hardware level allows Sentinel to handle larger DDoS attacks. Sentinel prevents bots from spoofing IP addresses to bypass blacklisting.
6. Sentinel also verifies user credentials in TLS connection requests. This work proposes a TLS handshake extension that allows Sentinel to handle up to 39,000 TLS connections/second without blacklisting. Sentinel prevents bots from sending HTTPS requests to web servers by filtering out TLS requests. Existing SSL/TLS accelerators can offload the processing of TLS handshakes at higher rates, but do not block bot-originated HTTPS requests.

The rest of this chapter describes each contribution in further detail.

Hardware token authentication I propose the use of hardware tokens, such as RSA SecurIDs [118] or PayPal security keys [11], to authenticate clients before accessing a web server under a denial-of-service attack. A web server verifies hardware token authentication credentials to distinguish client from bot requests. Tokens use secret seeds and a pseudo-random number generation algorithm to generate token codes at fixed intervals of time, such as 30 or 60 seconds. A user presents the current code from the token's LCD screen and the server verifies the code within an acceptable clock skew. Hardware token authentication is resilient to DDoS attacks because it does not require the web server to perform resource-consuming computations for unauthenticated client requests. It is also resilient to algorithmic attacks because solving a challenge requires the possession of a hardware token. Botmasters cannot utilize hired workers or online services to read and enter token codes because tokens generate codes at fixed, low rates. Launching a DDoS attack of sufficiently high rate requires the botmaster to purchase a large number of hardware tokens, which substantially increases the cost and complexity of the attack.

Federated authentication environments Commercial solutions for hardware token authentication on the Web exist and are already used by financial and medical institutions, but these solutions are not appropriate for mitigating denial-of-service attacks against generic web servers. A study has shown that web users have an average of 25 accounts with web sites requires authentication[72]. It is unrealistic that a user can carry a token for each web site under DDoS attack.

To mitigate this problem, I propose that clients use hardware tokens within a federated authentication environment. In this scenario, a user registers with one or a few *authentication providers* and receives hardware tokens from them. For example, Google acts as an authentication provider for the OpenID federated authentication protocol [16] and can require its OpenID users to authenticate with a hardware token instead of a username/password pair. When the user accesses a web server under a DDoS attack, the web site prompts them to authenticate with an authentication provider using a hardware token. The provider authenticates the user and issues an authentication token to access the web site. Federated authentication reduces the number of hardware tokens per user, making them more suitable for authentication on the Web. For example, multiple web servers may trust Google to verify the human nature of its users. Each Google user needs to possess only a singe hardware token issued by Google instead of a single hardware token for each desired web server.

Researchers have proposed multiple schemes for federated authentication. Many of them, however, are vulnerable to man-in-the-middle (MITM) attacks that allow an attacker can capture valid user credentials. Resisting MITM attacks is crucial for federated authentication schemes because capturing a single set of user credentials allows an attacker to gain access to multiple web servers.

Public key cryptography, e.g., as used in the TLS protocol, can be used to prevent MITM attacks by authenticating the source of each message. However, public key cryptography is resource consuming and makes parties vulnerable to DDoS attacks. A federated authentication solution must resist both MITM and DDoS attacks to be useful for DDoS attack mitigation. This dissertation work builds upon the Kerberos authentication protocol because it enables federated authentication, uses efficient symmetric encryption algorithms, and is resilient to MITM attacks. Chapter 7 discusses the resilience of Kerberos authentication providers to DDoS attacks.

Improved Kerberos authentication for the Web X.509 certificate-based authentication, the prevalent authentication mechanism for Web communication, relies on well-educated users and is vulnerable to phishing and man-in-the-middle attacks. Kerberos provides mutual authentication and is resilient to both attacks, but the amount of client configuration required prevents it from becoming widely adopted on the Web. A Kerberos client must iteratively traverse an authentication path of multiple authentication providers before obtaining credentials for the desired web server. The client must be pre-configured to discover the address of every authentication provider between the client and server. Such configuration is impractical for generic web clients.

This dissertation work contributes an improved mechanism for Kerberos authentication on the Web. I propose that a client offloads Kerberos authentication to the client's authentication provider. The client requests a ticket from the authentication provider, which discovers the authentication path, traverses it and returns to the client a service ticket for the desired web server. The structure and length of the authentication path is transparent to client, which substantially reduces client configuration.

Enhanced user interface for Kerberos authentication Authentication protocols for the Web are vulnerable to interface spoofing attacks. The web browser queries users for authentication credentials through dialog windows or other graphical interface elements. Malicious web scripts and custom-crafted web pages, however, can closely imitate the user interface expected by the user. Resilience to spoofing attacks is crucial in a federated authentication environment because the attacker can access multiple service providers using a single set of stolen credentials.

The Identity Metasystem authentication framework can enhance authentication protocols with easy-to-understand user interface resilient to spoofing attacks. The existing framework, however, uses computationally expensive public-key cryptographic operations, which makes the authentication provider vulnerable to denial-of-service attacks. This dissertation work contributes an extension to the Identity Metasystem framework to integrate it with the Kerberos protocol. The integrated protocol makes Kerberos credentials transparent to users and is resilient to both spoofing and denial-of-service attacks.

Hardware-accelerated front ends This dissertation designs and implements Sentinel, a transparent server front end, that intercepts server-bound traffic, verifies client credentials and drops traffic from blacklisted clients before it reaches the server. Front end devices enable the use of hardware acceleration without server modifications. Sentinel avoids wasting resources on traffic from blacklisted clients by discarding bot-originated traffic on a network acceleration card.

Sentinel can verify client credentials in server-bound HTTP requests and respond with authentication challenges to requests without credentials. This dissertation provides mechanisms for web browsers to interpret authentication challenges, encode the Kerberos credentials into HTTP cookies, and transmit the cookies with subsequent HTTP requests. Sentinel verifies the Kerberos credentials before allowing the requests to reach the server. Generating authentication challenges makes Sentinel vulnerable to DDoS attacks during a short window of time, but the attack is mitigated once Sentinel starts blacklisting clients. The performance results in Chapter 6 shows that even during that short attack window, Sentinel performs slightly better than a standard web server.

Mitigating HTTPS-based DDoS attacks Verifying credentials in HTTP cookies is not suitable for checking credentials in HTTPS requests because cookies are always transmitted encrypted over TLS. Checking client credentials in HTTPS requests requires the server-side HTTPS front end to decrypt and possibly re-encrypt all server-bound HTTPS traffic. Continuous decryption and re-encryption is expensive because TLS clients and servers may negotiate CPU-intensive symmetric encryption algorithms, such as 3DES. HTTPS server front end devices also must not terminate TLS connections because TLS is a stateful protocol and HTTPS proxies must maintain state for all terminated connections. Requiring an HTTPS front end to decrypt HTTP traffic or maintain TLS connection state makes the front end itself vulnerable to DDoS attacks.

Checking credentials in HTTS requests also does not effectively mitigate DDoS attacks. HTTPS requests are sent only after the end of the TLS handshake, during which the server must perform expensive public-key cryptographic operations. A botmaster can flood a server by establishing thousands of TLS connections without transmitting any HTTP traffic over TLS. It is not sufficient for an HTTPS server to offload TLS handshake processing to SSL/TLS accelerators. Such accelerators can process high rates of TLS handshakes, but do not filter the bot-originated HTTP requests inside the TLS tunnels and the attacker can still flood the web server with HTTP requests.

HTTPS server front ends can prevent this attack by verifying client credentials in the beginning of each TLS handshake. I propose that clients encode Kerberos credentials in backwards-compatible TLS extensions [46]. They extend the functionality of TLS Client Hello messages and are transmitted unencrypted, so an HTTPS server front end can verify the credentials without carrying out expensive cryptographic operations. Verifying credentials during the TLS handshake prevents attackers from sending HTTP requests inside TLS tunnels because it prevents them from establishing such tunnels. It is acceptable to transmit the Kerberos credentials without TLS encryption because the Kerberos protocol already encrypts Kerberos credentials internally.

1.3 ROADMAP

The rest of this dissertation is organized as follows. Chapter 2 presents background information on network protocols and analyzes their resilience to MITM and DDoS attacks. Chapter 3 presents the design and implementation of Sentinel, a network device for mitigating DDoS attacks. Chapters 4 and 5 describe the design and implementation of a system that uses hardware token authentication in a federated setting distinguishing between human clients and bots. Chapters 6 and 7 present the performance and security analysis of the system. Chapter 8 discusses related work and alternative approaches. Chapter 9 concludes the dissertation.

2.0 SYSTEM MODEL AND BACKGROUND

This chapter sets the stage for the rest of this dissertation by describing the system model and presenting background information on existing federated authentication protocols. Section 2.1 discusses the system model for this dissertation. It describes the type of DDoS attacks addressed in this dissertation—HTTP and TLS floods that consume web servers' CPU resources—and the assumptions made about web clients, web servers and botnets. The rest of the chapter presents background information about federated authentication. Many federated authentication schemes use the HTTP and TLS protocols for message passing. Section 2.2 and Section 2.3 describe HTTP and TLS, respectively. Section 2.4 discusses federated identities and existing federation authentication protocols: OpenID, SAML, Passport, Identity Metasystem, and Kerberos. I discuss the resilience of each protocol to man-in-the-middle (MITM) and denial-of-service (DoS) attacks. Section 2.5 concludes this chapter.

2.1 SYSTEM MODEL

The system model for this dissertation consists of legitimate clients accessing web servers under a botnet-based DDoS attack (Fig. 3). The client hosts are general-purpose computers, such as desktop towers or laptops, running general-purpose operating systems, such as Linux, Windows or MacOS. Users access web sites hosted on the web servers via web browsers installed on the clients. The browsers, such as Mozilla Firefox, Internet Explorer, or Google Chrome, are extensible via standard add-on or extension modules.

The web servers are in the backend of a web server farm with network infrastructure, e.g., routers, firewalls, and load balancers, deployed in front of them. This system model

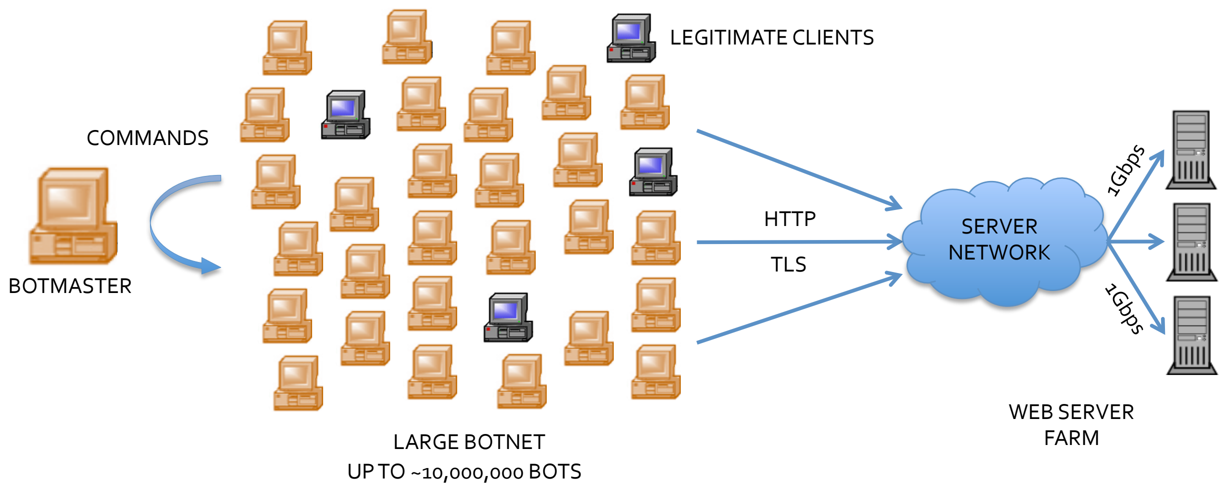


Figure 3: System Model

assumes 1Gbps full-duplex network links inside the server farm network. The web server hosts are generic server-class computers running server-class operating systems and generic web server applications, e.g., Apache or nginx, configured to service both HTTP and TLS requests.

The botnet generating the DDoS attack is a collection of compromised general-purpose computers (bots) that receive commands from a botmaster to generate floods of HTTP or TLS requests against the web server. This system model considers HTTP and TLS floods that deplete the web servers' CPU capacity before depleting the server farm's network bandwidth. This dissertation does not address DDoS attacks that target the web server farm's network bandwidth. The performance results in Section 6.6 show that a single bot HTTP connection consumes 9166 bits of network bandwidth on the average. If a 1Gbps full-duplex network link in the server farm has 2Gbps bi-directional capacity (1Gbps in each direction), the botnet needs to generate at most 218,198 HTTP connections/second to congest the link. If each bot generates one request per second, this requires a botnet of at most 218,198 bots. This dissertation's system model does not consider botnets larger than 220,000 bots because it does not consider DDoS attacks that congest the web server farm's network links.

2.2 HYPERTEXT TRANSFER PROTOCOL

The Hypertext Transfer Protocol (HTTP) is the de facto standard for client-server communication on the Web. It allows a client to retrieve resources (e.g. HTML documents and images) over a TCP connection. In HTTP 1.0 [44], every HTTP request uses a separate TCP socket connection. HTTP 1.1 [70] uses persistent TCP connections to carry multiple requests. It also allows pipelining, i.e clients can transmit back-to-back HTTP requests without waiting for a server response. HTTP is a stateless protocol, but servers may establish stateful sessions using HTTP cookies. A cookie is a server-issued token that the client sends with all HTTP requests to the server. The server uses the token to identify which session a request belongs to.

HTTP is vulnerable to both MITM and DDoS attacks. An HTTP message recipient cannot authenticate the sender, which allows a man-in-the-middle to intercept, and inject HTTP messages. Processing HTTP requests may also require the server to fetch files from memory, disk storage or backend servers, as well as to generate dynamic content. Therefore, DDoS attack using floods of HTTP requests can deplete the server's CPU, memory or disk access capacity.

2.3 TRANSPORT LAYER SECURITY

The Transport Layer Security (TLS) protocol [62] and its predecessor, Secure Sockets Layer (SSL) [73], provide a secure communication layer for application-level protocols such as HTTP. These protocols prevent eavesdropping, tampering and forgery of data packets and provide mutual authentication where the client verifies the identify of the server and the server optionally verifies the identity of the client.

TLS uses a handshake protocol to establish new sessions. The components of a handshake include:

- Security parameter negotiation: the client and server negotiate security parameters such as TLS protocol version, cipher suite and an optional compression method. The cipher suite specifies algorithms for mutual authentication, key exchange, encryption and message authentication.
- Mutual authentication: TLS uses public/private key pairs and X.509 [81] certificates for server and (optionally) client authentication.
- Master secret computation: the TLS endpoints use a key exchange algorithm, such as RSA [19], Diffie-Hellman [63], or Elliptic Curve Diffie Hellman [62] to agree on a shared pre-master secret (PMS) for the new session. They combine the PMS with random nonces to compute a shared master secret, from which they derive session keys for symmetric encryption and message authentication.

There are three key exchange algorithms commonly used during the TLS handshake:

- RSA [19]: the client generates a random PMS and encrypts it with the server's RSA public key. The server decrypts the PMS using its private key.
- Diffie-Hellman (DH) [63]: the client and server exchange DH parameters and use them to calculate the PMS.
- Elliptic Curve Diffie-Hellman (ECDH) [62]: the client and server exchange ECDH parameters and use them to calculate the PMS.

The RSA and Diffie-Hellman algorithms require the server to perform a modular exponentiation operation using a large modulus. Modular exponentiation implemented using the repeated squaring algorithm [94] requires $O(n^3)$ bit operations, where n is the modulus size. Developments in prime number factorization algorithms [39] have enabled attacks against RSA encryption with 700-bit moduli. NIST currently recommends 2048-bit modulus size [43]. Certificate authorities and Internet companies already started transitioning SSL server certificates to 2048-bit keys [32, 34, 128]. TLS servers are vulnerable to DDoS attacks, because a botnet can exploit this property and make TLS servers unresponsive using floods of thousands of TLS connection requests.

The TLS protocol has built-in mechanisms to protect from MITM attacks. The TLS server certificate must be issued by a trusted CA and its common name attribute must match the server's domain name. These conditions prevent MITM attacks, assuming that an attacker cannot compromise the server's private key, gain control over the server's web domain, or interfere with the client's DNS queries. Most web browsers, however, allow users to continue a connection even in the case of a domain name mismatch. Users often do not understand certificate-related error messages and dismiss them [38]. Some browsers, such as Mozilla Firefox, have made it harder for users to dismiss such errors, but the problem has persisted. A Firefox bug report in 2008 describes a user getting "invalid certificate" error messages when connecting to secure web servers, such as PayPal.com, Facebook and MySpace [22]. Analyzing the supposed PayPal.com TLS certificate revealed a possible MITM attack: the certificate was self-signed, claimed to come from PayPal.com and was different from the actual TLS certificate used by PayPal.com at that time. However, the Firefox user dismissed all certificate error messages, possibly becoming a victim of a MITM attack.

2.4 FEDERATED AUTHENTICATION

Federated identities aim to solve the problem of users who need separate identities to access resources belonging to different administrative domains. Without federated identities, each administrative domain is responsible to establish users' identities before allowing them to access resources within the domain. Federated identities allow an *identity provider* A to establish the user's identity and make *assertions* about that identity to *relying parties* B and C (Fig. 4). An assertion makes a statement about a single aspect of the user's identity, e.g., "the user's driver's license number is 123-45-6789" or "the user's date of birth is Jan 1, 1970". Relying party B may ask a user Alice for her driver's license number. Alice authenticates with her identity provider A, requests an *authentication token* containing an assertion about her driver's license number, and sends the token to B. B need not authenticate Alice's identity if it trusts A and is satisfied with the assertion provided. A different relying party C may request Alice to provide her date of birth. Alice's identity provider issues a new

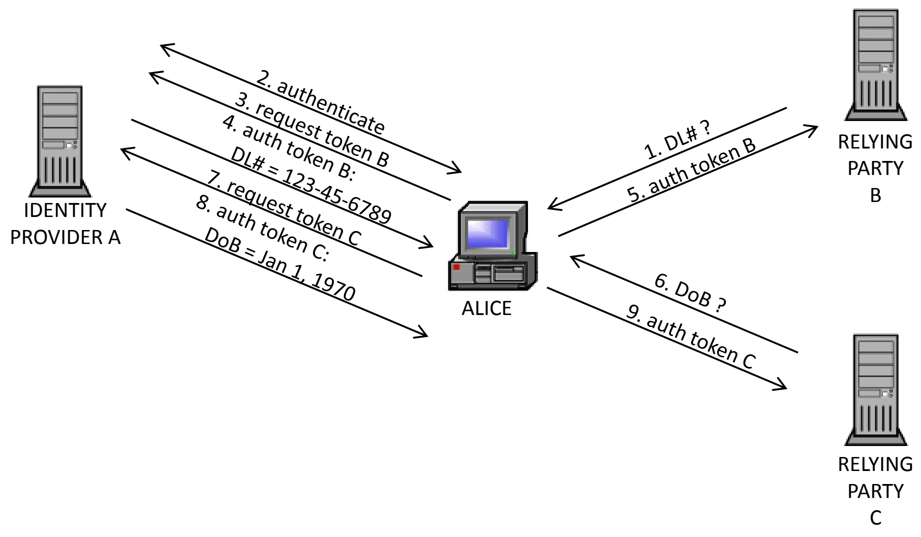


Figure 4: Federated identities and authentication

authentication token, which Alice sends to C. The identity provider may provide *single sign-on*, in which case it recognizes that Alice authenticated recently and does not require her to re-authenticate.

The rest of this section discusses multiple federated authentication protocols and their resilience to DDoS, MITM, and phishing attacks. Resilience to DDoS attacks is critical because requiring users to authenticate with a federated authentication provider before accessing a web server under a DDoS attack makes the authentication provider itself a potential target of the DDoS attack. Resilience to MITM and phishing attacks is important because a DDoS attacker can bypass the authentication requirement if they can steal federated authentication tokens at sufficiently high rates. This section describes the OpenID [16], Passport [122] and SAML [55] federated authentication schemes, which use redirect mechanisms vulnerable to MITM and phishing attacks. The Identity Metasystem framework [54] complements these protocols and makes them resistant to MITM and phishing attacks, but its usage of public key cryptography makes it vulnerable to denial-of-service attacks. The section concludes with the description of the Kerberos [109] protocol and its strengths against MITM, phishing, and DDoS attacks.

2.4.1 OPENID

OpenID [16] is an identity management scheme that provides federated authentication and single sign-on. An OpenID user sends to a relying party a URL that identifies the user's identity provider (Fig. 5). The relying party extracts the provider's address from the URL and optionally negotiates a secret HMAC key using Diffie-Hellman key agreement. The relying party redirects the user to the identity provider, passing along requests for the identity assertions it requires. For example, the relying party may require the user's age and zip code. The user authenticates with and authorizes the identity provider to release the requested identity information. The provider redirects the user back to the relying party, passing along the identity assertions possibly protected with the HMAC key [21]. OpenID identity providers allow single sign-on by keeping users logged in and issuing new identity assertions without re-authentication.

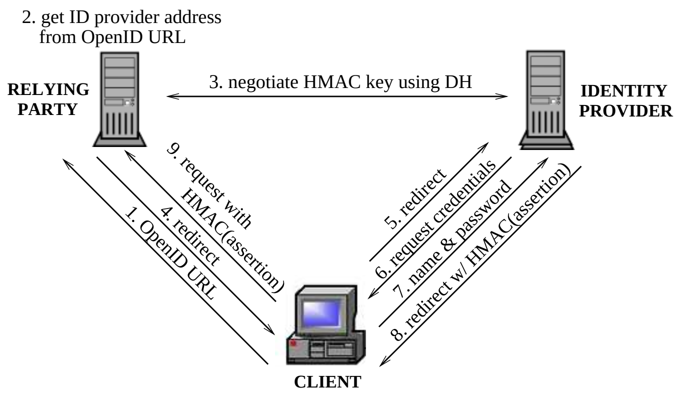

2. get ID provider address from OpenID URL
RELYING PARTY
3. negotiate HMAC key using DH
IDENTITY PROVIDER
9. request with HMAC(assertion)
4. redirect
1. OpenID URL
5. redirect
6. request credentials
7. name & password
8. redirect w/ HMAC(assertion)
CLIENT

Figure 5: OpenID 2.0 message exchange

OpenID is vulnerable to phishing and MITM attacks [127]. A malicious relying party can redirect the client to a fake identity provider, which mimics the real provider, and obtain the user's authentication credentials. OpenID is also vulnerable to man-in-the-middle attacks. The protocol specification does not authenticate the redirect message in step 4. A man-in-the-middle can modify the redirect address and forward the user to a fake provider.

There are numerous OpenID extensions that integrate OpenID with the OAuth 2.0 protocol [31]. Facebook uses OAuth 2.0 in its Facebook Connect protocol [26]. The OpenID

Connect protocol is used by PayPal [9] and combines OAuth 2.0 and OpenID 2.0 to provide third-party applications with single sign-on and access to user information. Google [33] also provides an integrated protocol for authentication to Google APIs. OAuth 2.0 is vulnerable to attacks because the standard leaves many details up to the implementation [77]. Researchers have exploited the protocol's vulnerabilities and demonstrated ways to acquire user credentials for Google and Facebook [132]. All OpenID OAuth 2.0 extension designs assume that authentication is performed inside an SSL tunnel, making the protocols vulnerable to denial-of-service attacks.

2.4.2 PASSPORT

The Passport protocol [122] uses a centralized login server to authenticate users. A user requests to login to a relying party, which redirects him to a Passport server. The user authenticates and is redirected back to the relying party along with an authentication token encrypted using a secret key shared between the relying party and the Passport server. The relying party decrypts the token and sends an encrypted cookie to the client browser. Passport provides single sign-on by allowing the client to present the cookie to the relying party again and receive service without re-authentication.

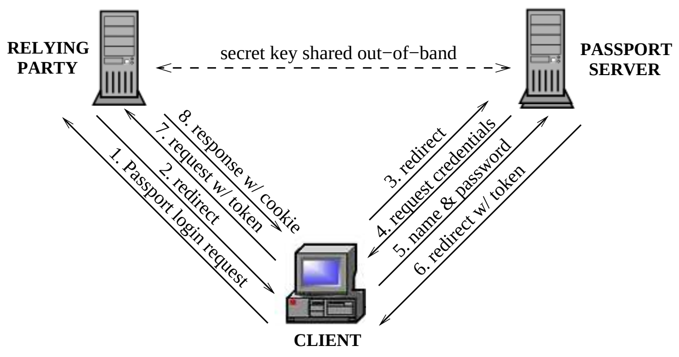


Figure 6: Passport message exchange

The Passport protocol is vulnerable to MITM and phishing attacks, where an attacker acting as a MITM or a malicious relying party redirects the client to a malicious Passport server [95]. Using TLS prevents this attack, but makes relying parties vulnerable to DDoS attacks. The Windows Live ID framework [17] has superseded Passport. The Windows Live ID Web Authentication scheme reports using the same message exchange as the Passport protocol [18].

2.4.3 SECURITY ASSERTION MARKUP LANGUAGE

The Security Assertion Markup Language [55] is an XML-based standard for exchanging authentication and authorization information between security domains. A SAML client requests a page from a relying party, which redirect the client to authenticate with an authentication server. The authentication server generates a set of identity assertions and issues a SAML token identifying the set. The client sends a second request to the relying party and includes the token. The relying party requests the identity assertions associated with the token and authorizes the user. SAML provides single sign-on by allowing a client to request service multiple times using the same artifact.

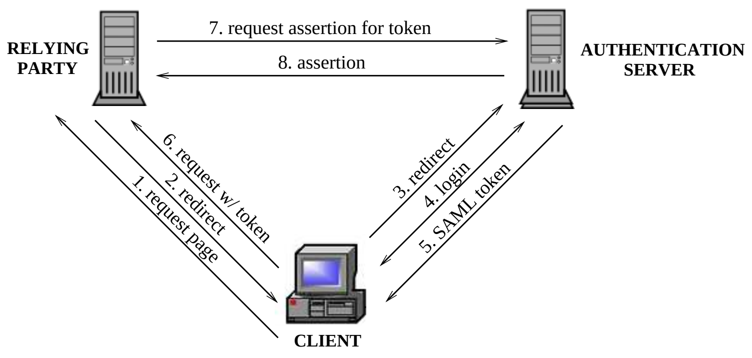


Figure 7: SAML message exchange

Groß [75] shows that the SAML protocol is vulnerable to man-in-the-middle attacks. An attacker can modify the redirect address in step 2 to forward the client to a fake authentication server. An attacker can also exploit SAML's encoding of the token. The authentication server in step 5 redirects the client to the relying party URL and encodes the token in the URL's query string. A man-in-the-middle can intercept the assertion request in step 7, causing the relying party to return an error message to the client. The attacker can then intercept the next client's HTTP request and extract the unused SAML token from the HTTP Referer field. This attack is possible because SAML authentication server encodes the token into the redirect URL. The HTTP 1.1 specification [70] discourages encoding sensitive information into a URL query string.

2.4.4 IDENTITY METASYSTEM

The Identity Metasystem (IdM) [54] is a framework for identity management and federated authentication. Each user has a number of digital identities stored as information cards (InfoCards), e.g digital driver's license or credit card. Each InfoCard is issued by an identity provider and makes a number of claims about the user. For example a bank may issue a InfoCard that contains credit card information, while the local motor vehicle administration may issue a driver's license InfoCard that contains claims about the user's age and address. The IdM framework allows a user to manage InfoCards and the release of the identity information in them.

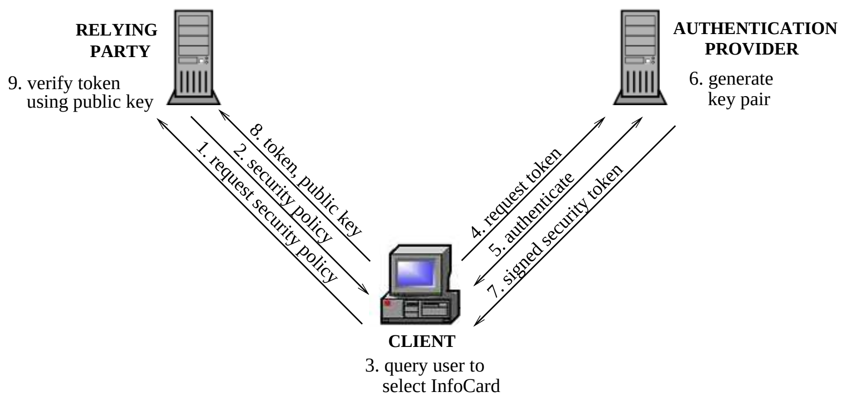


Figure 8: Identity Metasystem message exchange

An IdM client initiates authentication by requesting the relying party's security policy, which specifies the required identity information. The client application selects the InfoCards that contain claims for the required information and queries the user to select one.

The application then queries the user to authenticate with the InfoCard's identity provider. The provider generates a public/private key pair, generates a token containing the required identity information assertions, and signs the token with the new private key. The client presents the token and the public key to the relying party, which uses the public key to authenticate the token and verify its integrity.

The Identity Metasystem does not provide federated authentication and single sign-on, but serves as a framework for federated authentication schemes, such as OpenID, SAML or Windows Live ID. Each authentication scheme specifies the format and handling of the IdM authentication token. The usage of a client application and InfoCards eliminates the redirect messages in the OpenID, Passport, and SAML authentication protocols. This makes OpenID and Passport resistant to man-in-the-middle and phishing attacks.

However, IdM uses public key cryptography, which may make relying parties vulnerable to denial-of-service. A bot master can request security policies and flood the relying party with fake tokens signed with random bit strings instead of private keys. The relying party has to perform a public key operation before it can detect that a token is fake. The relying party can filter repeated offenders by IP address, but a large botnet commands enough bots to launch a DDoS attack using only a few requests per bot.

2.4.5 KERBEROS

The federated authentication protocols above are all vulnerable to either man-in-the-middle or distributed denial-of-service attacks. The Kerberos protocol, however, is resilient to both types of attacks. A Kerberos client sends a request to a ticket-granting server, which returns a *ticket-granting ticket* encrypted with the user's password (Fig. 9). The client uses the ticket to request one or more *service-granting tickets* to access servers. The service-granting ticket contains a new session key for communicating with a server, encrypted with a secret key shared between the server and the ticket-granting server.

A Kerberos ticket-granting server is the principal authentication and key distribution authority for a Kerberos realm, a set of clients and servers. Kerberos provides federated

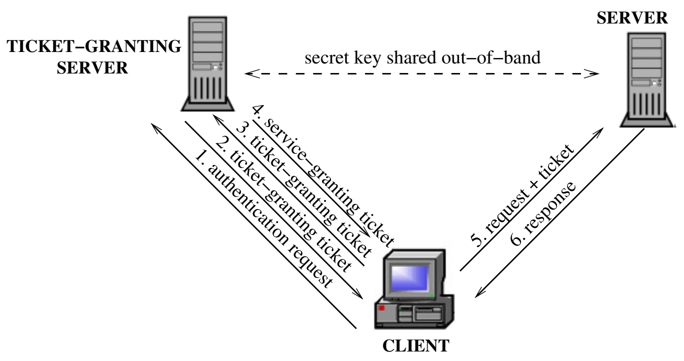

TICKET–GRANTING SERVER
SERVER
secret key shared out–of–band
4. service–granting ticket
3. ticket–granting ticket
2. ticket–granting ticket
1. authentication request
5. request + ticket
6. response
CLIENT

Figure 9: Kerberos message exchange

authentication by allowing a client to authenticate in one realm and access a server in another. The ticket-granting servers in the two realms communicate using a shared secret key. Kerberos v4 requires every pair of realms to have a shared key. Kerberos v5 organizes realms in a tree structure to reduce the number of shared keys.

The Kerberos protocol is resilient to denial-of-service and man-in-the-middle attacks. Kerberos uses symmetric key cryptography to encrypt tickets. Symmetric encryption algorithms are more efficient than public key encryption and make servers more resilient to denial-of-service attacks. Kerberos also prevents man-in-the-middle and replay attacks. A man-in-the-middle cannot sniff or modify security-sensitive information because Kerberos tickets are encrypted with secret keys at all times.

Kerberos uses *authenticators* to prevent replay attacks. When a client receives a ticket with a new session key, it also receives the session key encrypted with the user's password. The client decrypts the key and uses it to encrypt an authenticator containing session information, such as client and server addresses. The server receives the Kerberos ticket and authenticator, decrypts the ticket and uses the session key inside to verify the authenticator. Authenticators prevent attackers from replaying stolen Kerberos tickets.

Kerberos authentication is supported by popular web browsers, e.g., Mozilla Firefox and Internet Explorer, and web servers, e.g., Apache and Microsoft Internet Information Server. However, Kerberos has not been widely adopted for authentication on the Web. Web servers assume that they belong to a Kerberos realm and secret keys are negotiated out-of-band. However, the Web does not have a centralized key distribution authority. A Kerberos client must also be configured to discover and traverse the authentication path to a desired server. For example, the client must be able to locate the address of every intermediate KDC on the authentication path. This is unsuitable for web clients, which have minimal configuration and communicate with arbitrary web servers.

2.5 CONCLUSION

This chapter presents the system model and background information about federated authentication to set the stage for the rest of this dissertation. The descriptions of the HTTP and TLS protocols in Section 2.2 and 2.3 provide the foundation for discussing authenticated HTTP and TLS requests in Sections 3.2, 4.3, and 5.3. The Kerberos protocol presented in Section 2.4 is the backbone of the federated authentication architecture proposed in this dissertation. Sections 4.2 and 5.2 present proposed Kerberos extensions and Section 5.1 discusses the integration of Kerberos with hardware token authentication. The Identity Metasystem framework presented in Section 2.4.4 is used in Sections 4.4 and 5.5 to provide a user interface for Kerberos authentication.

3.0 SENTINEL

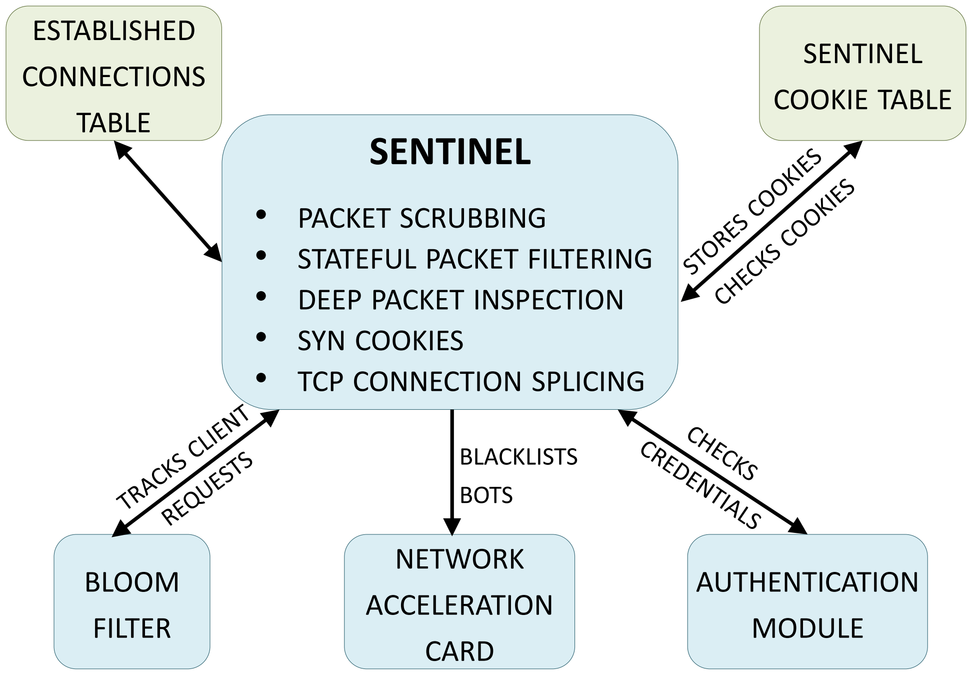


Figure 10: Overview of Sentinel

This chapter describes the design and implementation of Sentinel [64], the front-end network device I propose to offload bot-based DDoS attacks from web servers. Sentinel mitigates DDoS attacks by identifying and dropping bot traffic. Sentinel operates as a transparent proxy, which splices TCP connections, performs packet scrubbing, stateful packet filtering, and deep packet inspection (Fig. 10) with the help of an *established connections table*. It uses pluggable *authentication modules* to check client requests for authentication credentials and to return authentication challenges. This chapter describes Sentinel's design and implementation using CAPTCHA authentication. Sentinel issues authentication cookies to authenticated clients and checks for an attached cookie before passing client requests through to the web server. Sentinel tracks the number of unauthenticated client requests in a Bloom filter and marks clients as bots once the number reaches a pre-defined threshold. Sentinel uses a network acceleration card to mark client IP addresses as bots and discard bot-originated traffic in hardware.

The rest of this chapter is organized as follows. Section 3.1 introduces Sentinel's modes of operation and Section 3.2 discusses its extensible authentication framework. Section 3.3 describes the techniques that Sentinel uses to avoid committing resources to unauthenticated clients. Sections 3.4 and 3.5 discuss client blacklisting and deep packet inspection. Sentinel's implementation is described in Section 3.6.

3.1 MODES OF OPERATION

Sentinel has two modes of operation, *normal* and *suspected_attack* for each server behind it. It switches between them based on server load estimates and two configuration parameters, L_{normal} and L_{attack}, where $L_{normal} < L_{attack}$. To obtain such estimates, Sentinel periodically sends an HTTP request and measures the response time of each server. Sentinel estimates the load on server i as an exponentially-weighted moving average L_i of i's measured response times. At Sentinel startup, each server is in normal mode. When $L_i \geq L_{attack}$, Sentinel puts server i in *suspected_ attack* mode. As Sentinel mitigates the attack against the server, L_i decreases and eventually $L_{normal} < L_i < L_{attack}$. When L_i decreases further and $L_i \leq L_{normal}$, Sentinel places server i again in *normal* mode.

3.2 EXTENSIBLE IN-NETWORK CLIENT AUTHENTICATION

In *suspected_attack* mode, Sentinel authenticates clients to filter out requests from bots. It gives Sentinel cookies to authenticated clients – a blob of random data and a cookie creation timestamp encrypted with a secret Sentinel symmetric key and stored in a Sentinel internal cookie table. If a request arrives without such a cookie and without a response to an authentication challenge, Sentinel replies with an authentication challenge. If a request arrives with a valid response to an authentication challenge, Sentinel replies with a cryptographic HTTP cookie that the client will include in future requests. Sentinel only forwards requests with such a cookie to servers. On the other hand, in *normal* mode, Sentinel does not challenge clients, and forwards all requests without checking for cookies.

The formats of the authentication challenge and response depend on the authentication method used. The challenge may contain a nonce to prevent replay. To facilitate use of future authentication methods, Sentinel separates in a specific module the functions for generating an authentication challenge and verifying an authentication response according to a particular method. The module also contains an initialization function that loads and initializes an authentication table.

Note that the client's IP address is insufficient to distinguish authenticated clients: many clients inside home and enterprise networks reach web servers via middleboxes, such as routers and web proxies [56], that map multiple internal, private IP addresses to a single public IP address. Consequently, legitimate clients may share the same IP address with bots. Cookies permit distinguishing authenticated clients from other hosts that may exist behind the same middlebox. An attacker may also spoof a legitimate client's IP address and attempt to impersonate the legitimate client. Sentinel prevents spoofing attacks using SYN cookies (Section 3.3).

A botmaster might have a human obtain a cookie from Sentinel and then distribute copies of the cookie to a large number of bots. To thwart such collusions, Sentinel maintains in an **HTTP cookie table (HCT)** the status of each cookie, including expiration time and number of outstanding requests. Sentinel can be configured to (a) allow the use of a cookie only in requests from a particular IP address, and (b) drop a request containing a cookie if the request would cause the cookie's number of outstanding requests to exceed *cookie_req_lim*. Option (a) is undesirable if clients may use mobile IP, while option (b) requires Sentinel to analyze not only requests from clients but also responses from servers. Sentinel uses option (b) by default.

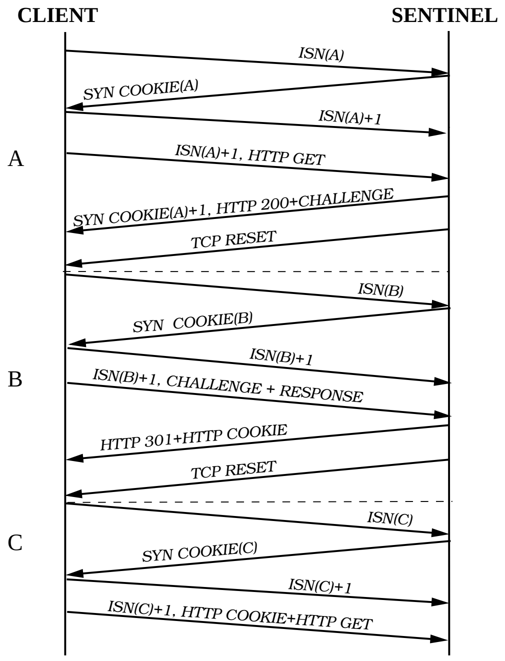


Figure 11: SYN cookies and asynchronous client authentication

3.3 COMMITTING RESOURCES TO UNAUTHENTICATED CLIENTS

Sentinel combines three techniques to avoid committing to unauthenticated clients Sentinel's memory and any of the protected servers' resources.

First, when Sentinel receives a client request for a new connection to a protected server, if the client is not blacklisted, Sentinel replies on behalf of the server with a standard SYN cookie [45]. That is, Sentinel computes a cryptographic function of the client's and server's IP addresses and port numbers, the client's proposed maximum segment size, the current time, and a secret. Sentinel uses the result as the initial sequence number (ISN) it sends to the client. Sentinel does not store any values related to the client's connection request. When the client replies, its acknowledgment number should equal Sentinel's ISN plus one. Sentinel verifies this property cryptographically and recovers from the client's reply (instead of a table) the TCP state needed for the connection with the client. Sentinel may offload this function to the network interface card for hardware acceleration.

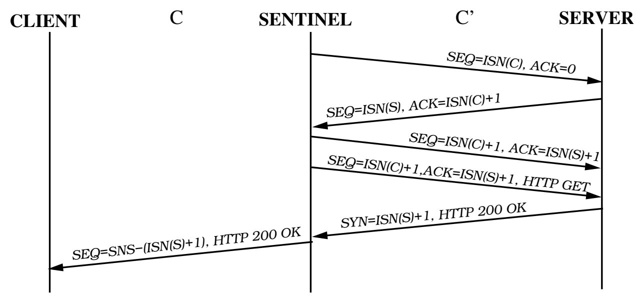

CLIENT
C
SENTINEL
C'
SERVER
SEQ=ISN(C), ACK=0
SEQ=ISN(S), ACK=ISN(C)+1
SEQ=ISN(C)+1, ACK=ISN(S)+1
SEQ=ISN(C)+1,ACK=ISN(S)+1, HTTP GET
SYN=ISN(S)+1, HTTP 200 OK
SEQ=SNS−(ISN(S)+1), HTTP 200 OK

Figure 12: TCP connection splicing

Second, if Sentinel needs to authenticate clients (*suspected_attack* mode), it does so asynchronously, as illustrated in Fig. 11. When Sentinel receives a request from an unauthenticated client, Sentinel uses the first connection (A) to convey Sentinel's authentication challenge to the client. To avoid holding TCP state, Sentinel forcibly closes this connection. Using a new connection (B), the client presents its authentication response to Sentinel. If the response is valid, Sentinel redirects the client to the URL the client originally requested (HTTP 301 status code), provides the client an HTTP cookie, and again forcibly closes the connection (the client always starts a new connection after receiving status code 301). Finally, using another connection (C), the client resends its original request, but now with the Sentinel cookie.

Third, when Sentinel receives a request with a valid Sentinel cookie on a connection (C) (Fig. 11), Sentinel opens another connection (C') with the protected server (Fig. 12). Sentinel then splices together the client and server connections (C) and (C'). The client's and Sentinel's ISNs in C are respectively ISN_c and $ISN_{sc} = SYN\_cookie$. Sentinel's and the server's ISNs in C' are respectively $ISN_{ss} = ISN_c$ and ISN_s. To splice the two connections C and C' together, Sentinel computes C's *sequence number shift (SNS)*: $SNS = SYN\_cookie - ISN_s$. Sentinel allocates for the spliced connections an entry in Sentinel's **established connection table (ECT)**, and stores C's SNS in it.

ECT is a hash table indexed by the client's and server's IP addresses and port numbers. When Sentinel receives from a client a packet with acknowledgement matching an ECT entry, Sentinel adjusts the acknowledgement number by SNS and forwards the packet to the server, without checking for cookies. Conversely, when Sentinel receives a packet from the server, Sentinel adjusts the sequence number by SNS and forwards it to the client.

When the HTTP cookie used to create an ECT entry expires or Sentinel has not observed any valid packet in the respective connections for a period longer than a threshold, Sentinel forcibly closes the connections and destroys the ECT entry.

3.4 COUNTING UNAUTHENTICATED REQUESTS AND BLACKLISTING

Attackers might frivolously trigger authentications to exhaust Sentinel CPU cycles or network bandwidth, thus causing denial of service to legitimate users despite the defenses described in the previous subsection. To avoid such attacks, Sentinel keeps track of how many authentication challenges it has issued to each client IP address. When the number of consecutive HTTP requests without a valid Sentinel cookie exceeds the threshold, the client will be blacklisted as a bot. Sentinel stores this information in a **Challenge Count Bloom Filter (CCBF)**. CCBF contains N cells, each with a b-bit counter num_chal_i. Each IP address corresponds to k of those cells, selected by k hash functions. Cell i is said to be *saturated* if $num\_chal_i = S$, where $S = 2^b - 1$. For example, the CCBF can contain $N = 1048576$ cells, each with a $b = 5$-bit num_chal_i counter. Each IP address may go through $k = 2$ different hash functions and map to two different cells. A cell is saturated when its num_chal_i counter reaches $S = 31$. If all two cells corresponding to an IP address have $num\_chal_i = 31$, the client behind that IP address with very high probability has sent 31 consecutive HTTP requests without a valid cookie and should be blacklisted as a bot.

In *suspected_attack* mode, when Sentinel issues an authentication challenge to a client, Sentinel increments the corresponding CCBF counters by $\min(1, S - num\_chal_i)$. Conversely, when Sentinel receives a valid response to an authentication challenge, Sentinel decrements the corresponding CCBF counters by $\min(\eta, num\_chal_i)$. A value $\eta > 1$ is used to avoid long-term error accumulation.

If all CCBF counters corresponding to an IP address are saturated, Sentinel concludes that the IP address harbors a bot, because a human would have given up before making so many authentication errors. Sentinel *blacklists* the IP address and does not consider future packets from it. Sentinel may offload this filtering rule to a network acceleration card. Chapter 6 presents the performance evaluation of Sentinel before blacklisting and after blacklisting. The experimental results show that Sentinel can process up to 19,000 HTTP requests/second before blacklisting. Sentinel takes roughly 10 seconds to identify and blacklist the bot IP addresses. The DDoS attack is then reduced to a flood of SYN packets, which are discarded on the network acceleration card.

3.4.1 BLOOM FILTER FALSE POSITIVES

Due to the nature of Bloom filters, the mechanism for bot detection is resistant to false negatives, i.e. Sentinel will always mark clients with $S = 31$ unauthenticated requests as bots. Attackers can not spoof IP addresses to blacklisting because Sentinel verifies the SYN cookie during the TCP handshake. When a bot sends 31 unauthenticated requests, Sentinel saturates all $k = 2$ Bloom filter counters. When Sentinel looks up the bot IP address, all 2 counters are saturated because the Sentinel decrements them only when it detects the presence of a human user at the client host. Sentinel does not increment already saturated counters from and thus prevents Bloom filter counters from overflowing.

The Sentinel's Bloom filter lookup function, however, is susceptible to false positives, where hash function collisions can saturate all $k = 2$ counters of a legitimate client and mark it as a bot. The probability of a lookup false positive in a binary, non-counting Bloom filter is equal to the probability that all k bits corresponding to the queried element are set to 1 after inserting n elements: $(1 - e^{\frac{-kn}{N}})^k$ [49]. The probability of a lookup false positive in Sentinel's Bloom filter is equal to the probability of all k bits being saturated after blacklisting n IP addresses: $(1 - e^{\frac{-kn}{N^{31}}})^k$. For $k = 2$ and $N = 1048576$, the probability remains close to 0% even after blacklisting millions of IP addresses.

3.4.2 BLACKLISTING

Sentinel blacklisting is similar to standard firewall rules that filter out unauthorized and malicious traffic. Section 8.7 discusses firewalls as an alternative to Sentinel for discarding blacklisted traffic. Sentinel blacklisting is also similar to the Spamhaus Block List (SBL) maintained by the Spamhaus project [13], which blacklists IP addresses determined to belong to spam sources or spam servers. Spamhaus does not recommend SBL users to accept e-mail traffic from blacklisted IP addresses. The responsibility of removing an IP address from the SBL falls entirely on the owner of the blacklisted IP address. SBL blacklist entries are removed after a pre-set timeout, which may range between one day and one year. In comparison, Sentinel blacklists client IP addresses determined to belong to bots. The current Sentinel design assumes that the web server farm behind Sentinel consists of servers belonging to the same organization, e.g., a Google data storage server farm. After blacklisting an IP address as a response to a DDoS attack against a single web server, Sentinel drops traffic from the blacklisted client to all web servers behind Sentinel. Sentinel can be extended to ban a client IP from accessing only a single server by using both the client IP and server IP addresses in the Bloom filter and the filtering rule. This flexibility may be beneficial for mitigating DDoS attacks against server farms, where each server belongs to a different organization.

Sentinel periodically rehabilitates blacklisted IP addresses. During each time period $T_i = 10\ \mathit{seconds}$, Sentinel also measures the number of requests that hit or miss the blacklist, respectively $B_{h,i}$ and $B_{m,i}$, and calculates the average server loads $average(L_i)$. Sentinel continuously collects per-server historical data consisting of the number of blacklist misses $B_{m,i}$ and the average server load $average(L_i)$. It uses the historical data to continuously update a regressional model that estimates the *critical number* of blacklist misses $B_{c,i}$ corresponding to response time $average(L_i) = L_{normal}$. At the end of each time period T_i, if $average(L_i) \leq L_{normal}$ and $\forall i : B_{m,i} + B_{h,i} \leq B_{c,i}$, Sentinel clears the blacklist, rehabilitating all IP addresses. It is the user's or system administrator's responsibility to disinfect the client host and remove the bot malicious executable to prevent being blacklisted again.

3.5 PACKET FILTERING, SCRUBBING, AND DEEP PACKET INSPECTION

Sentinel gleans from observed packets the state of each direction of a connection, including sequence and acknowledgment numbers and receive window size. Sentinel records this information in its ECT and uses it for stateful packet filtering [130], which defines acceptable ranges for TCP and IP header fields throughout a TCP connection and drops packets with header field values that are inconsistent with the current state of the respective connection.

TCP and IP allow packets, including client requests, to be split into multiple segments and fragments. Because reassembly occurs only at the destination, attackers may intentionally split packets to make it difficult for network-based defenses to identify patterns. For example, an attacker can split a field containing a value that network intrusion detection systems (NIDSs) might use as a signature into two or more fragments. Attackers can also intentionally make fragments overlap, such that even if a NIDS can detect patterns across fragments, the NIDS may miss a signature because the NIDS resolves the overlap differently from the destination. Alternatively, attackers can intentionally omit one or more fragments of each packet so as to fill reassembly memory, causing denial of service.

To efficiently deal with such fragmentation attacks, Sentinel selectively performs packet scrubbing [78]. Until Sentinel has all segments and fragments in a request's application-layer header, Sentinel timestamps and enqueues them by sequence number and offset. If contiguous segments or fragments overlap, Sentinel trims the older one before enqueueing the new one. Sentinel also sets a timeout when it receives the first segment or fragment of the application-layer header. If some segment or fragment is still missing when the timeout expires, Sentinel drops all the enqueued segments and fragments of that connection.

When a request's entire application-layer header is available, Sentinel analyzes it in place (deep packet inspection without copying). Sentinel uses the Knuth-Morris-Pratt (KMP) algorithm [93] to search for keywords and patterns in the header. The algorithm runs in *O(http_header_length + pattern_length)* time, enabling high throughput. Analysis also determines the sequence number of the next application-layer header.

After analysis of an application-layer header, Sentinel forwards any segments or fragments with ending sequence number lower than the beginning of the next application-layer header (i.e., Sentinel does not scrub the application-layer payload). If analysis does not reveal the sequence number of the next application-layer header then, after analysis of the current header, Sentinel can be configured to forward packets without scrubbing or further analysis until the connection is closed. The latter case corresponds to legacy HTTP 1.0 without persistent connections or pipelining [70], i.e., with only one request per connection.

3.6 IMPLEMENTATION

Sentinel could be implemented in the kernel network stack above the network device driver layer [64]. However, it would still use CPU cycles and buffer space to process and drop blacklisted bot traffic. In this dissertation, Sentinel is implemented using hardware acceleration. Packets from blacklisted clients are discarded by a network acceleration card without using significant host CPU resources. Thus, Sentinel can withstand attacks of much larger scale.

3.6.1 NETRONOME NFE-I8000

The Netronome Flow Engine (NFE) I8000 platform is a network acceleration card that contains an Intel IXP2855 network processor unit (NPU) and on-card memory (Fig. 13). The NPU comprises an XScale microprocessor and sixteen programmable microengines for on-card packet classification. The NFE platform provides up to 768 MB of RDRAM, 40 MB of QDR SRAM and Ternary Content Addressable Memory (TCAM) space. The card has four SFP network ports and is connected to the host through a PCI Express bus with four active lanes providing 2GB/s aggregate capacity.

The NPU processes Ethernet frames arriving at the NFE ports by matching them consecutively against the NFE's flow and rules tables. The flow table contains actions for existing connections. If a packet does not belong to an existing flow, it is matched against the rules table, which specifies actions for packets from future flows. If the NPU finds a match, it copies the rule back to the flow table (to process further packets from the same flow) and

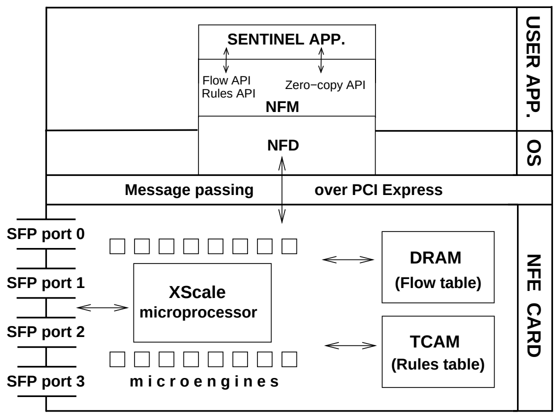


Figure 13: Structure of the NFE Platform

executes the specified action. The NPU can execute one of the following actions on a frame: drop, reject (drop and send TCP reset message), pass (forward to all NFE ports) or forward up to the host for further analysis.

The rules table stored in the TCAM memory of Sentinel's NFE I8000 acceleration card has space for up to 32,768 filtering rules. The key for rules table lookup operations can contain the source IP address, destination IP address and other packet header fields. The rule key can match network flows against an IP address subnet instead of a specific IP address, which allows a single rule to match network flows against multiple IP addresses. Netronome's new-generation NFE-3240 network acceleration card has an expanded TCAM memory and can contain up to 262,144 rules.

3.6.1.1 SOFTWARE APIs

The NFE platform comes with two software packages. The **Network Flow Driver (NFD)** is a set of Linux kernel modules and userspace libraries providing an API for low-level communication with the NFE over the PCI Express bus (see Fig. 13). The **Network Flow Manager (NFM)** uses NFD to provide high-level APIs for application-level packet classification:

- the **zero-copy packet access API** allows NFM applications to access packets that the NFE NPU forwards to the host. The API provides high performance by using zero-copy

DMA mapping to make frame buffers available to userspace applications.

- the **flow and rules APIs** allow NFM applications to make decisions about packet classification on the application level and pass them to the NFE platform to enforce. The flow API provides interaction with the NFE flow table (existing flows) and the rules API is used to populate the NFE rules table (future flows). The NFM rules have two components. The rule key contains slots for packet header fields, such as IP addresses and TCP port numbers. If a field is set into the rule key with a desired value, e.g, TCP port = 80, the field is checked when matching the rule against incoming packets on the NFE card. If no value for that field is specified in the rule key, a wildcard value is assumed and the field is not checked during matching. Each NFM rule also contains an action to take if the packet matches the rule, such as to drop the packet silently, drop the packet and respond with a TCP RST packet, pass the packet through the NFE card without further action, or send the packet up to the user application for further processing.

Sentinel's NFM implementation uses the zero-copy packet access API to allocate and deallocate packet buffers and to send and receive frames through the NFE card. In suspected attack mode, when a client is blacklisted, Sentinel uses the NFM rule and flow APIs to instruct the NFE NPU to drop packets from the client's IP address in existing and future flows. Dropping malicious packets in the NFE card significantly reduces the number of packets reaching the host and the resources used per bot connection.

3.6.2 CAPTCHA AUTHENTICATION MODULE

Sentinel uses CAPTCHAs to distinguish bots from human clients. CAPTCHA images with distorted text containing each approximately 2,000 bytes are stored in the authentication table. Each table entry also contains fields with the respective solution, the time when the puzzle was sent to a client, and whether a solution has been received. The module prepares the challenge as two back-to-back packets. The module rejects solutions received too long after the challenge expires, or if another solution has already solved the challenge. More details on CAPTCHA generation are presented in Section 6.4.

4.0 HARDWARE TOKEN AUTHENTICATION IN A FEDERATED SETTING

The Sentinel front-end device presented in Chapter 3 used CAPTCHA authentication to distinguish between legitimate human users and bots. However, to mitigate botnet-based DDoS attacks, Sentinel must either dynamically generate CAPTCHA puzzles at the same rate as the botnet's request rate, or serve the same puzzle in response to multiple bot requests. Dynamically generating CAPTCHA puzzles is computationally intensive and can deplete Sentinel's CPU resources, while reusing puzzles makes Sentinel vulnerable to attacks using human CAPTCHA solvers.

This chapter presents a new system architecture based on Sentinel that uses hardware token authentication in a federated setting to more efficiently tell human users and bots apart. The architecture uses Kerberos federated authentication to reduce the number of hardware tokens a user needs and extends Kerberos to make it more suitable for light-weight web clients. An extended version of Sentinel leverages its deep packet inspection capabilities to efficiently verify Kerberos authentication credentials. On the client side, Kerberos and hardware token authentication must be unified into an intuitive, secure user interface for entering authentication credentials. The architecture described in this chapter uses the Identity Metasystem framework and information cards to present an integrated, intuitive user interface with built-in security features, such as resilience to spoofing attacks.

Fig. 14 shows an overview of the system architecture. A web client sends web requests over HTTP or TLS to a web server. An extended version of the Sentinel front-end appliance intercepts the client requests, checks for authentication credentials, and challenges the client to authenticate with a hardware token before accessing the web server. The web client sends an authentication request with the user's hardware token code to the client's Kerberos

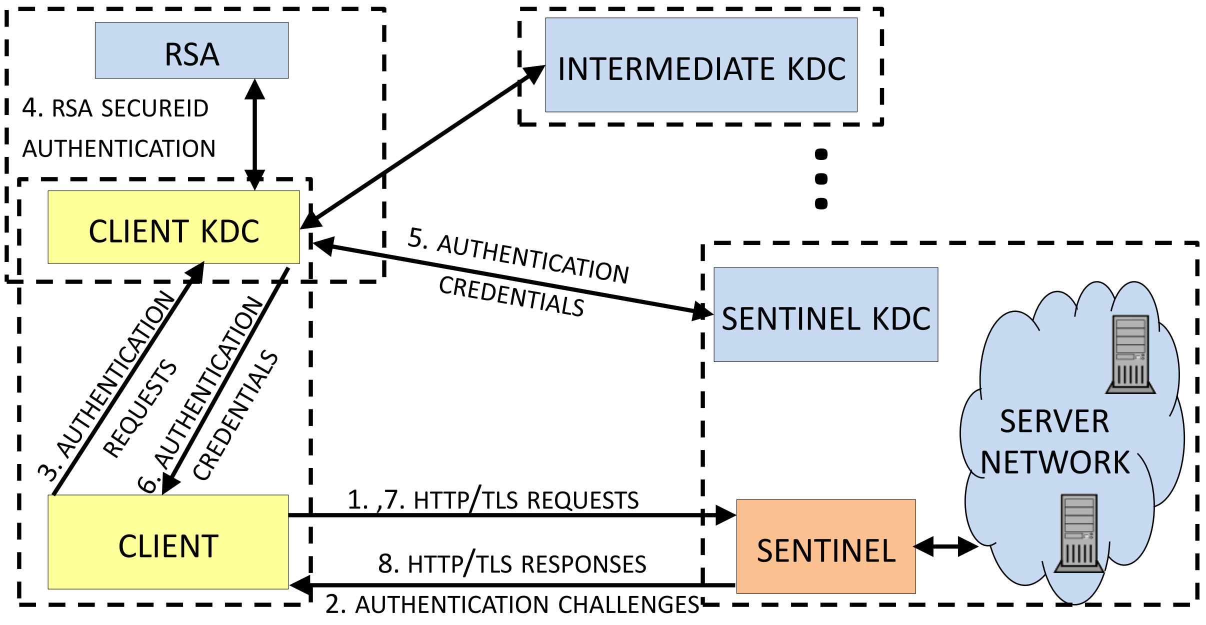

RSA
4. RSA SECUREID AUTHENTICATION
INTERMEDIATE KDC
CLIENT KDC
5. AUTHENTICATION CREDENTIALS
SENTINEL KDC
SERVER NETWORK
3. AUTHENTICATION REQUESTS
6. AUTHENTICATION CREDENTIALS
CLIENT
1. ,7. HTTP/TLS REQUESTS
SENTINEL
8. HTTP/TLS RESPONSES
2. AUTHENTICATION CHALLENGES

Figure 14: System architecture overview – dashed lines define trust domains.

KDC. The KDC passes the token code to an RSA authentication server for verification. Upon success, the client's KDC traverses the path of Kerberos realms to Sentinel's KDC, obtains Kerberos authentication credentials from Sentinel's KDC, and returns the credentials to the client. The client sends the credentials to the web server with subsequent HTTP and TLS connections. Sentinel intercepts the requests, verifies the credentials and passes the requests to the web server.

The orange box in Fig 14 represents the novel component introduced by this architecture: the Sentinel front end device. The yellow boxes represent existing software applications extended in this architecture: the client's browser and the client's KDC. The blue boxes in Fig. 14 represent existing, unmodified software applications: the RSA Authentication Manager, the intermediate Kerberos KDC, Sentinel's KDC, the web servers, and the web server farm network infrastructure. The dotted lines show the trust domains in the system.

The extensions to the software applications and network protocols proposed in this architecture are minimally intrusive. The client's browser is naturally extensible through browser add-ons. The HTTP and TLS protocols used by the web client, server, and Sentinel are also naturally extensible via opaque HTTP cookies and TLS extensions, respectively. The extension to the client's KDC is implemented by adding functionality to the existing KDC application and adding a separate proxy ticket service running on the same host. The system architecture proposed here does not require the modification of server network infrastructure, such as firewalls and routers. The addition of Sentinel is transparent to web clients, web servers and network infrastructure.

The rest of this chapter is organized as follows. Section 4.1 presents the hardware token authentication mechanism that distinguishes between human users and bots. Section 4.2 discusses the proposed Kerberos protocol extensions. Section 4.3 introduces the Sentinel extensions for processing and verifying Kerberos authentication credentials. Section 4.4 describes the integration with the information card client-side user interface.

4.1 HARDWARE TOKEN AUTHENTICATION

Hardware tokens, such as RSA SecurIDs, are small form factor hardware devices that use an internal clock and a pseudo-random number generator (PRNG) to generate one-time passwords (token codes) over fixed intervals of time. The PRNG is seeded with a factory-encoded secret key. The token uses an internal battery to power itself and displays the one-time passwords on an LCD display. When authenticating, a user reads the current token code and provides it to an authentication server, such as the RSA Authentication Manager [35]. The server uses the same PRNG and seed key to determine the expected token code at this time and verifies that that the two codes match.

Hardware token authentication is suitable for identifying traffic from automated bots because it is infeasible for bots to obtain one-time passwords from hardware tokens. A botmaster can stage DDoS attacks by employing people to manually enter hardware token codes. However, RSA SecurIDs release new one-time passwords over fixed intervals of one minute, which implies that the botmaster must obtain a large number of tokens to sustain the high rate of token codes needed to carry out a successful DDoS attack. It is infeasible for an attacker to obtain a large number of tokens because each token is registered at an authentication provider and issued only to a legitimate user. A rogue authentication provider may collude with the botmaster and issue hardware tokens for bots. Sentinel can detect this anomaly and revoke trust from the rogue provider. Chapter 7 discusses rogue SecurID authentication providers in greater detail.

4.2 KERBEROS FEDERATED AUTHENTICATION

Unless used in a federated authentication environment, hardware token authentication requires users to have a single hardware token per web server they wish to access. Federated authentication enables multiple web servers to offload authentication to a third-party authentication provider. When using hardware token authentication integrated into a federated setting, users authenticate with a single hardware token to a single authentication provider in order to access multiple web servers. In this dissertation, I propose integrating hardware token authentication with the Kerberos V5 authentication protocol.

Kerberos V5 is a government-approved [50], efficient protocol for mutual authentication. Every Kerberos client or server is known as a *principal* and belongs to an administrative domain known as a *realm*. Every Kerberos realm has a *key distribution center (KDC)* that lets clients authenticate to access servers. Kerberos V5 realms may be organized in a tree hierarchy and traversing the tree edges between two realms forms an authentication path between them. Conventionally, Kerberos realms are mapped to DNS domains [109]. Fig. 15 shows a possible authentication path between two DNS nodes – alice.cs.pitt.edu and webserver.accounting.pnc.com.

The Kerberos V5 protocol for inter-realm authentication [109] uses iterative authentication requests (Fig. 16). A client C in realm R_1 contacts its local KDC K_1 to obtain a service ticket for server S in R_n. R_1 and R_n may be connected through an arbitrary number of intermediate realms that form the authentication path between C and S. K_1 extracts from its configuration file the next-hop realm R_2 on the authentication path to S and issues to

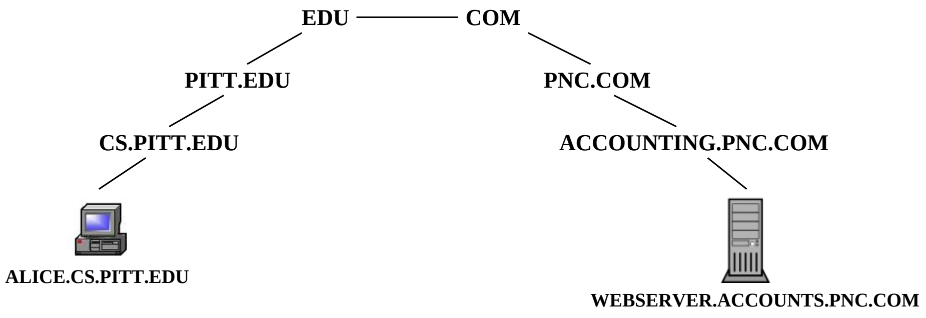


Figure 15: Sample authentication path between Kerberos realms

C a ticket-granting ticket (TGT) for R_2's KDC K_2. C iteratively sends a request to each realm's KDC on the authentication path until it receives a service ticket for S from the KDC K_n in R_n. C stores received tickets in its local credential cache.

Traditional Kerberos inter-realm authentication requires C to know the locations of all KDCs on the auhentication path and to reveal its Kerberos identity to the server. This is unsuitable for generic web browsing, where light-weight clients often visit web sites without authenticating with them. Sections 4.2.2, 4.2.3 and 4.2.4 propose Kerberos extensions to enable light-weight clients to browse web sites without revealing their Kerberos identities.

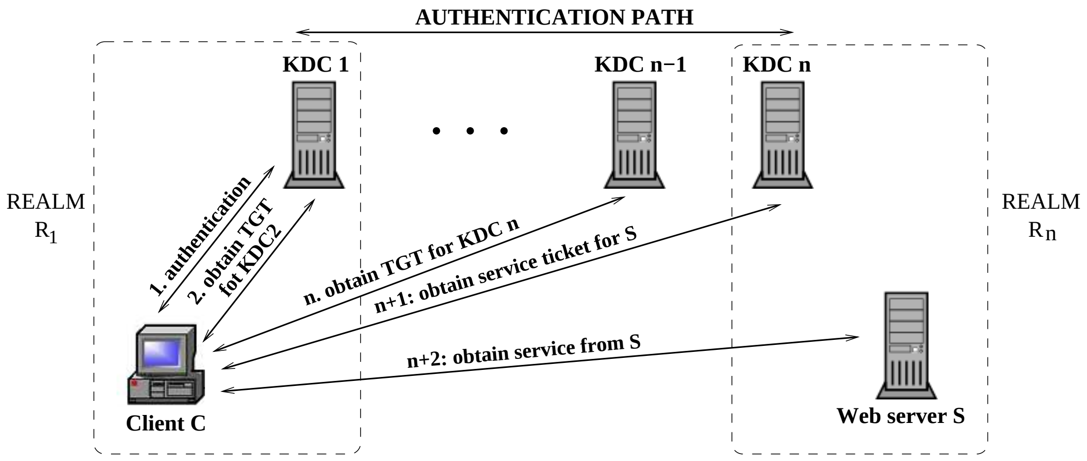


Figure 16: Original Kerberos inter-realm authentication

4.2.1 INTEGRATING HARDWARE TOKEN AUTHENTICATION INTO KERBEROS

This dissertation extends the Kerberos V5 protocol to include hardware token authentication as a single-use authentication mechanism (SAM) [80]. The SAM framework allows single-use authentication mechanisms, such as RSA SecurID one-time passwords, to be used as Kerberos V5 pre-authentication methods. Kerberos clients and key distribution centers (KDCs) use the pre-authentication data (PADATA) fields in authentication requests, responses and error message to negotiate the type of authentication required.

If a Kerberos client sends an authentication request (AS-REQ) without sufficient authentication credentials, the KDC responds with an error message containing a SAM challenge (Fig. 17). The challenge contains the type of SAM required, a nonce and a checksum generated using the client's symmetric key K_c derived from the user's Kerberos password.

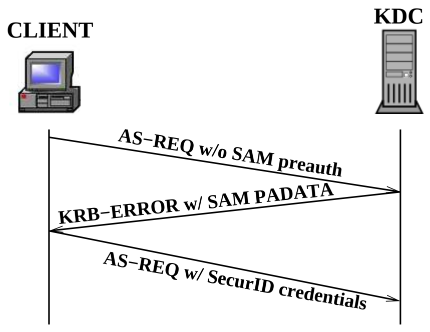


Figure 17: Kerberos pre-authentication with hardware tokens

The client uses K_c to verify the SAM challenge checksum, extracts the SAM authentication type and queries the user for his/her RSA SecurID PIN and token code. The client then encrypts the credentials and KDC's nonce with K_c and includes the encrypted data in a SAM response within a new AS-REQ message. The KDC decrypts the encrypted data with the client's key, verifies the nonce and contacts an RSA authentication server to verify the SecurID credentials.

4.2.2 OFFLOADED KERBEROS AUTHENTICATION REQUESTS

Despite the need for secure and efficient authentication, Kerberos has not become widely deployed on the Web. This is likely because Kerberos is not well-suited for allowing generic web clients with minimal client configuration to easily authenticate with arbitrary web servers. Web clients must be aware of the structure of Kerberos realms and must be pre-configured with the addresses of the KDCs of all realms on the authentication path to the desired web server. Clients must iteratively authenticate with each KDC before accessing the web server. Kerberos does not allow caching of Kerberos tickets between clients. If a second client in the same realm wishes to access the same web server, it needs to iteratively authenticate with all KDCs again. Iterative authentication adds redundant network communication, extra load on Kerberos KDCs and extra latency to client web requests.

In this dissertation, we introduce offloaded Kerberos authentication requests that better utilize cached tickets and require less client configuration (Fig. 18). In this example, C requests a service ticket for S from the local KDC K_1. K_1 traverses the authentication path to obtain a proxy ticket for S. Kerberos proxy tickets allow K_1 to delegate to C access rights for the HTTP service at S. K_n binds the proxy ticket to C's IP address. K_1 returns the ticket to C, which uses it to access S.

If a client C_2 in the same realm R_1 wants to authenticate with S, K_1 does not need to traverse the whole authentication path. It uses the cached TGT from K_{n-1} to request from K_n another proxy ticket bound to C_2's IP address. Reusing cached tickets for multiple clients in the same realm reduces the number of messages and the overhead on the KDCs for realms R_2–R_{n-1}.

Offloaded Kerberos authentication preserves the security semantics of the existing Kerberos protocol. Caching and reusing proxy ticket-granting tickets at the client's KDC is similar to caching and reusing TGTs by clients the existing Kerberos protocol. The cached proxy TGTs remain valid and can be reused until their expiration time. The availability of the client's KDC and its resilience to DDoS attacks is discussed in Chapter 7.

Kerberos clients using offloaded authentication without TGT caching experience similar

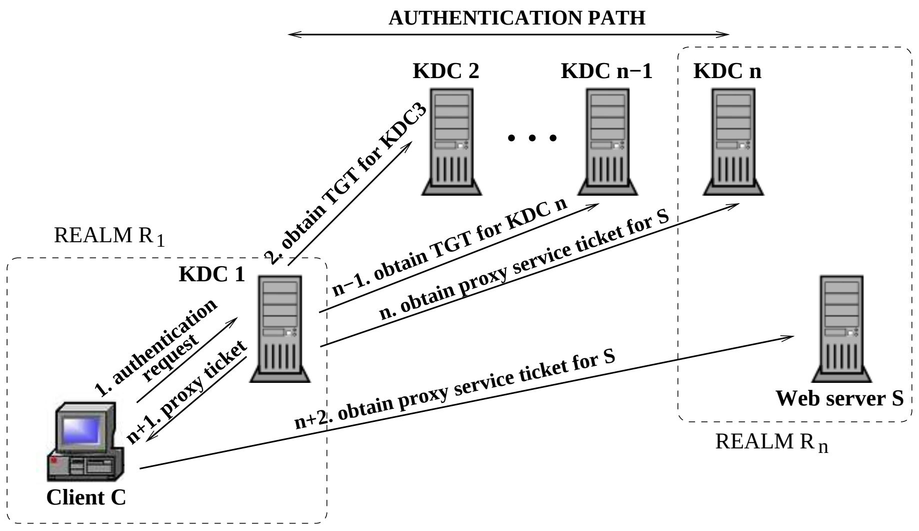

AUTHENTICATION PATH
KDC 2
KDC n−1
KDC n
REALM R 1
KDC 1
2. obtain TGT for KDC3
n−1. obtain TGT for KDC n
n. obtain proxy service ticket for S
1. authentication request
n+1. proxy ticket
n+2. obtain proxy service ticket for S
Web server S
REALM R n
Client C

Figure 18: Kerberos cross-realm authentication with offloaded requests

latency to clients using iterative authentication because a host (KDC or client) from the client's realm must traverse the authentication path to the server and obtain a ticket from each KDC. Offloaded authentication with TGT caching reduces client request latency when TGTs for that authentication path have previously been obtained and cached.

4.2.3 KERBEROS CLIENT CONFIGURATION

Offloaded Kerberos authentication requests also reduce the amount of Kerberos client configuration. Clients are not aware of intermediate KDCs on the authentication path to a server, so they do not need the addresses of those KDCs. The existing server principal name canonicalization Kerberos extension [114] further reduces client configuration by allowing clients to request a ticket for a server S without knowing its Kerberos realm or canonical name. For example, a client can request a service ticket for http/google.com without knowing Google's Kerberos realm. This dissertation uses the Kerberos principal name canonicalization extension.

4.2.4 KERBEROS CLIENT ANONYMITY

Offloaded Kerberos authentication enables clients to authenticate with web servers using *anonymous Kerberos tickets* [135] without explicitly revealing their Kerberos identities. A client must reveal its identity to the local KDC to authenticate, but the KDC does not reveal the identity to intermediate KDCs. Anonymous Kerberos tickets [135] allow users to obtain TGTs and service tickets for an anonymous principal $anonymous@R_1$. If a web server trusts a KDC to sufficiently authenticate and authorize a client, the server need not know the client's Kerberos identity. The proxy service ticket returned to the client is bound to the client's IP address to prevent attackers from stealing the Kerberos ticket and gaining access to the protected web server. It is possible for the server to indirectly identify the client by this IP address. Anonymous Kerberos tickets provide similar privacy guarantees to unauthenticated web browsing, where web servers can use clients' IP addresses, cookies and other attributes to indirectly identify clients.

Anonymous Kerberos ticket allow the client's KDC to cache the TGTs and reuse them if a different client from the same realm requests a service ticket for the same web server. Non-anonymous TGTs are bound to a known client principal and re-used by the client's KDC for other clients. However, the KDC can cache the anonymous TGT from K_{n-1}. If another client from the same realm requests a service ticket for the same web server, the KDC can use the cached TGT and obtain the service ticket without traversing the authentication path again. This reduces the number of messages exchanged and the load on the intermediate KDCs.

4.3 EXTENDED SENTINEL DESIGN

Verifying Kerberos tickets can incur additional computational overhead on a web server under a DDoS attack. The server must process bot-originated packets in its network protocol stack and attempt to verify the Kerberos credentials before dropping the packets. I address this problem by extending the Sentinel front-end device described in Chapter 3. Using a stand-alone Sentinel device deployed in front of the web server also allows using hardware acceleration without modifying the web server's hardware configuration. Hardware acceleration allows packet classification and early dropping of bot-originated packets at the hardware level.

Sentinel parses incoming HTTP and HTTPS requests to verify the Kerberos credentials in each request. A web browser encodes the Kerberos credentials in an HTTP request as an HTTP cookie. For authentication over HTTPS, I propose a new TLS Client Hello message extension that the browser can send the credentials in.

4.3.1 AUTHENTICATION OF HTTP REQUESTS

Sentinel parses every incoming HTTP request and checks it for valid Kerberos credentials. A Kerberos credential can be reused in multiple HTTP requests, but has a short expiration period, after which the user must re-authenticate and receive new credentials.

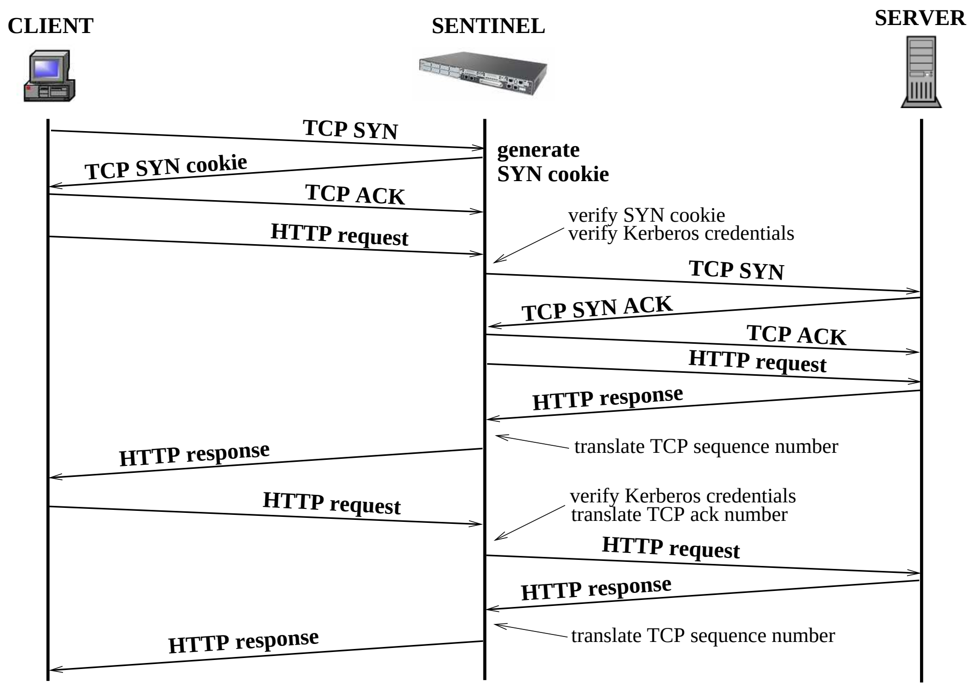


Figure 19: Splicing of HTTP messages over TCP

Fig. 19 shows the splicing for HTTP messages over TCP. Sentinel intercepts the client TCP SYN packet and replies with a SYN cookie [45]. Validating SYN cookies is stateless and prevents attackers from flooding Sentinel with spoofed IP packets. The client then completes the TCP handshake and send an HTTP request. Sentinel verifies the SYN cookie and the Kerberos credentials in the HTTP request before establishing a mirror TCP connection with the web server. The web server's initial sequence number does not match Sentinel's SYN cookie, so Sentinel must translate server-to-client sequence numbers and client-to-server acknowledgement numbers. Sentinel also parses each HTTP request and verifies its Kerberos credentials.

4.3.2 AUTHENTICATION OF HTTPS REQUESTS

Verifying Kerberos credentials in HTTP requests is impractical and insufficient for authenticating HTTPS requests over TLS because verifying Kerberos credentials over TLS requires decrypting and re-encrypting TLS records. This incurs additional overhead on Sentinel and requires additional configuration, because Sentinel must be able to obtain the TLS session key. This solution is also insufficient because web servers execute public-key encryption during the TLS handshake, i.e. before the transmission of any HTTP requests. An attacker can deplete the web server's resources by establishing new TLS connections at a high rate.

I address this problem by encoding the Kerberos credentials in a new TLS extension [46] of the Client Hello message. The server receives the TLS Client Hello message before dedicating any computational resources to the new TLS connection. Sentinel verifies the Kerberos credentials in the TLS Client Hello extension before allowing the TLS handshake to proceed. Sentinel parses only the first client-originated message on a TCP connection with HTTPS payload (the Client Hello). Sentinel currently does not support HTTP-to-TLS upgrades or resumable TLS sessions, where the Client Hello message may occur in the middle of a TCP connection.

Fig. 20 shows the splicing of HTTPS message over TLS. The mechanism is similar to the one described in Fig. 19, but Sentinel checks for Kerberos credentials in the TLS Client Hello message.

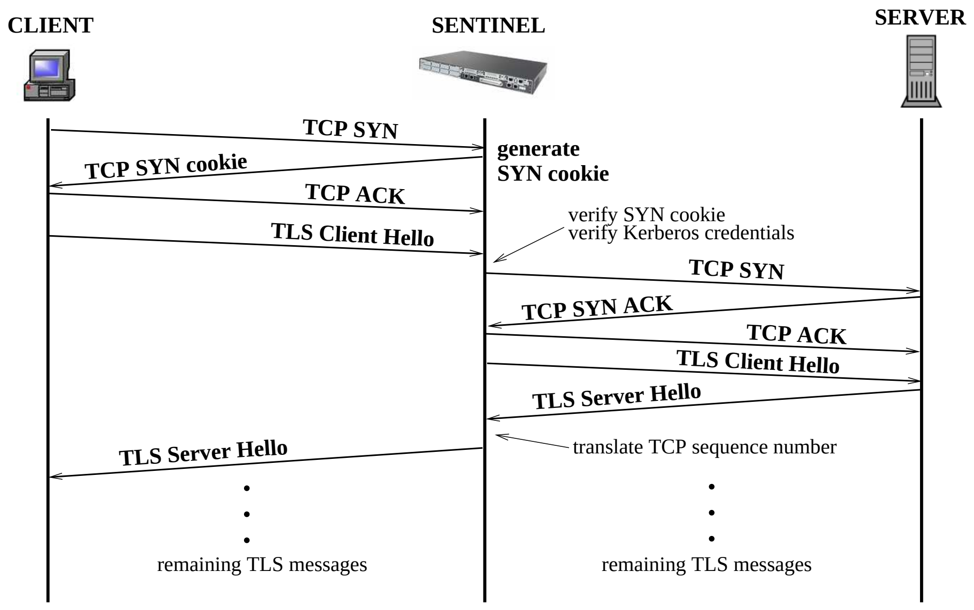


Figure 20: Splicing of HTTPS messages over TLS

4.4 CLIENT INTERFACE: IDENTITY METASYSTEM AND INFORMATION CARDS

To obtain the Kerberos proxy ticket required by Sentinel to allow access to a protected server, the client's web browser must prompt the user for a Kerberos password and an RSA SecurID token code. I address this problem by integrating the Identity Metasystem architecture (IdM) with the Kerberos V5 protocol.

4.4.1 IDENTITY METASYSTEM

In the Identity Metasystem (IdM) framework [53], users authenticate with relying parties using digital identities issued by identity providers. An identity provider issues the user an information card for each of the user's digital identities. The information card contains information about the identity provider and a number of identity claims that form the digital identity. For example, a user's digital identity for medical insurance may consist of claims about the user's insurance provider and policy number.

When authenticating, an IdM client obtains the security policy of the relying party (e.g., web server) that it wishes to connect to. The policy contains a list of the identity claims required by the relying party for successful authentication. The client queries the user for an information card that can provide all required claims. The user chooses an information card and enters the authentication credentials for the identity provider that issued the card. The client requests from the identity provider a security token with the required claims and sends the token to the relying party.

The Identity Metasystem information cards present the user with an intuitive interface for authentication. Many users already have multiple identities issued by government agencies, financial and medical institutions. People already often carry with them plastic cards that represent those identities – e.g., a driver's license for personal identification, debit/credit cards that identify their financial accounts, and health insurance cards identifying their medical insurance information. The Identity Metasystem digital information cards mimic the appearance and functionality of conventional plastic cards, making authentication more intuitive for non-expert users.

Digital information cards are not security-sensitive because they only store the descriptions of the claims that the respective identity provider can make and not any assertions about those claims. An attacker may steal or reproduce a legitimate user's information card, but they still need the user's authentication credentials, such as username and password, to authenticate with the identity provider and obtain assertions for the information card claims.

IdM is not a federated authentication scheme, but a framework for implementing such schemes. The security token contents are treated as opaque and depend on the authentication scheme used. Clients request security tokens by sending to the authentication provider a Request Security Token (RST) message that includes authentication credentials. The existing IdM specification suggests three types of authentication credentials for the RST message: username/password pair, X.509 client certificate, and Kerberos service ticket. Upon successful authentication, the provider returns the security token inside a Request Security Token Response (RSTR) message.

4.4.2 INTEGRATING IDENTITY METASYSTEM WITH KERBEROS

The existing IdM architecture allows Kerberos service tickets to be used as user credentials to authenticate when requesting a security token. I propose hardware-authenticated Kerberos proxy tickets as a new type of a security token. A user can requests the new security token type after obtaining an "RSA SecurID" information card from an identity provider. The authentication credentials required to obtain the new type of token are the user's Kerberos password and RSA SecurID token code. The authentication provider acts the the user's Kerberos KDC, verifies the user's password and token code, obtains a hardware-authentication Kerberos proxy ticket, and returns the encoded ticket as an IdM security token.

I extend IdM by adding hardware-authenticated Kerberos proxy tickets as a new security token type and pre-authenticated AS-REQ messages as a new Request Security Token (RST) credential type (Fig. 21). The client application queries the user for their Kerberos password and RSA SecurID token code. It uses the password to generate and encrypt a Kerberos AS-REQ request with the token code as pre-authentication data. The client sends the pre-authenticated AS-REQ message in a RST message to the Kerberos KDC, which also acts as an IdM identity provider. The KDC verifies the AS-REQ message with the user's Kerberos password, extracts the SecurID token code and contacts RSA authentication manager to verify it. Upon success, the Kerberos KDC traverses the authentication path to the desired web server to obtain an anonymous proxy ticket on behalf of the client, and returns the ticket as an IdM security token. The client includes the anonymous proxy ticket from the token with all subsequent requests to the web server.

The existing Identity Metasystem implementations are vulnerable to DoS attacks. Request Security Token (RST) and Request Security Token Response (RSTR) messages can be encrypted using either symmetric or asymmetric encryption. The specification, however, does not provide key exchange mechanisms required for symmetric encryption. Therefore,

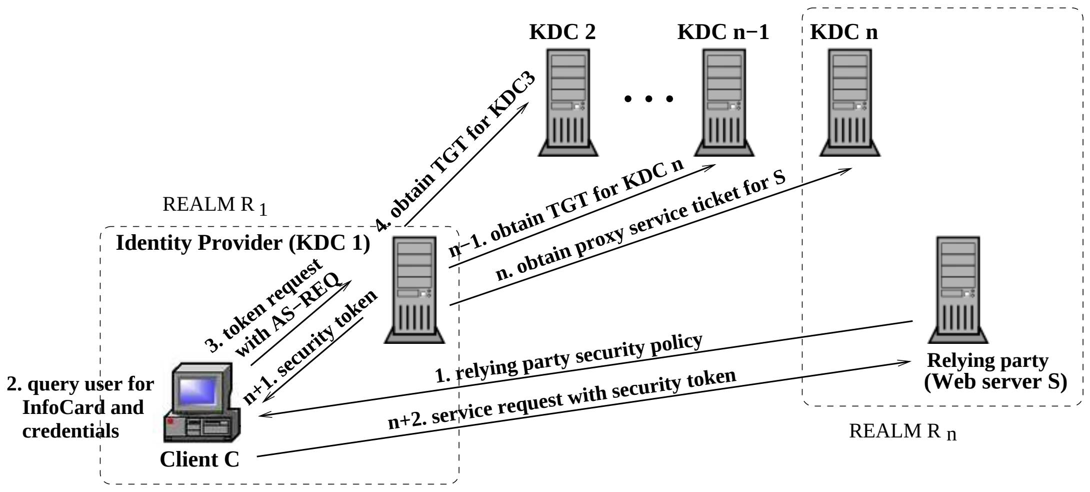

KDC 2
KDC n−1
KDC n
4. obtain TGT for KDC3
n−1. obtain TGT for KDC n
n. obtain proxy service ticket for S
REALM R 1
Identity Provider (KDC 1)
3. token request with AS−REQ
n+1. security token
2. query user for InfoCard and credentials
Client C
1. relying party security policy
n+2. service request with security token
Relying party (Web server S)
REALM R n

Figure 21: Identity Metasystem architecture integrated with Kerberos V5

most implementations use asymmetric encryption or transmit the RST and RSTR messages over TLS. Both approaches use public-key cryptography and make the identity provider vulnerable to DoS attacks. Integrating IdM with Kerberos improves IdM's resilience to denial-of-service. The Kerberos ticket in the security token is encrypted using a symmetric key shared between the client and the webserver's KDC. The RST and RSTR messages, therefore, do not need additional protection and the identity provider does not execute any public-key encryption operations.

5.0 SYSTEM IMPLEMENTATION

This chapter describes the implementation of the system architecture described in Chapter 4. I present the network protocol messages I extended to implement the system architecture, the software applications and libraries I used to integrate the system components, and snapshots of the authentication graphical user interface. Section 5.1 presents details about the integration of RSA SecurID authentication in the Kerberos V5 protocol. Section 5.2 discusses the implementation of offloaded Kerberos authentication requests. Section 5.3 describes the transmission and verification of Kerberos credentials in HTTP and HTTPS requests. Section 5.4 presents the integration of Kerberos and RSA SecurID authentication into the Identity Metasystem and shows snapshots of the user interface.

5.1 INTEGRATING KERBEROS AND HARDWARE TOKEN AUTHENTICATION

The Kerberos Single-use Authentication Mechanism (SAM) framework [80] extends the Kerberos V5 protocol with one-time password authentication mechanisms, such as RSA SecurID authentication. I ported the SAM framework implementation from the MIT Kerberos library [10] into Heimdal v1.2 [14]. If a client sends the Kerberos KDC an AS-REQ request without pre-authentication, the KDC checks a new *hwauth_required* flag I added to the KDC configuration file to decide whether hardware pre-authentication is required. If it is, the KDC responds with a SAM challenge encoded in the PADATA field of a KRB_ERROR message (Fig. 17), requesting the client to authenticate with a hardware token. Hardware token codes are an example of single-use authentication data (SAD). Listing 5.1 shows the ASN.1

format of the hardware authentication SAM challenge:

- *sam-type*: the type of SAM required, set to *PA_SAM_TYPE_SECURID*.
- *sam-flags*: bitmap of flags indicating whether the SAD is known by the KDC and should be used as a shared key, or it should be encrypted using a different shared key known by the KDC. RSA SecurID authentication uses the *send-encrypted-sad* flag, which indicates that the KDC does not know the SAD (hardware token code) and the client must send it encrypted with a shared key derived from the user's Kerberos password.
- *sam-nonce*: a random nonce to prevent attackers from replaying the SAM challenge or response.
- *sam-etype*: encryption algorithm to encrypt the SecurID token code with.

Listing 5.1: ASN.1 format of SAM challenge

```
PA-SAM-CHALLENGE-2-BODY ::= SEQUENCE {
   sam-type[0]        INTEGER (0..4294967295),
   sam-flags[1]       SAMFlags,
   ...
   sam-nonce[8]       INTEGER (0..4294967295),
   sam-etype[9]       INTEGER (0..4294967295),
}
```

Upon receipt of the SAM challenge, the client queries the user for the Kerberos password and SecurID token code. It derives the shared key from the user's password and encrypts the SecurID token code and SAM challenge nonce. The client sends a SAM response (Listing 5.2) encoded in the PADATA of a new AS-REQ message:

- *sam-type*: matches the SAM challenge type, set to *PA_SAM_TYPE_SECURID*.
- *sam-flags*: match the SAM challenge flags, set to *send-encrypted-sad*.
- *sam-enc-nonce-or-sad*: SAM SAD and nonce encrypted with the shared key derived from the user's password.

Listing 5.2: ASN.1 format of SAM response

```
PA-SAM-RESPONSE-2 ::= SEQUENCE {
  sam-type[0]         INTEGER (0..4294967295),
  sam-flags[1]        SAMFlags,
  ...
  sam-enc-nonce-or-sad[3]         EncryptedData,
                                  -- PA-ENC-SAM-RESPONSE-ENC
  ...
}

PA-ENC-SAM-RESPONSE-ENC ::= SEQUENCE {
  sam-nonce[0]        INTEGER (0..4294967295),
  sam-sad[1]          GeneralString OPTIONAL,
  ...
}
```

Upon receipt of the SAM response, the KDC derives the shared key from the user's Kerberos password, decrypts the encrypted data, and verifies the nonce. It then sends the SecurID token code to the RSA Authentication Manager for verification. For this implementation, I used the RSA Authentication Manager v7.0 and the RSA SecurID authentication agent. The authentication agent is implemented as a Linux Pluggable Authentication Module (PAM). I extended the Heimdal Kerberos KDC to interact with the authentication agent via the Linux *libpam* API. The authentication agent communicates with the authentication manager over the network using a proprietary UDP protocol.

5.2 OFFLOADED KERBEROS AUTHENTICATION REQUESTS

Kerberos client principals can obtain *proxy* tickets and delegate the authentication credentials to a different principal to access the service from their name. For example, a client may access a front-end server F, which requires service from a backend server B. The client obtains a proxy service ticket for B and passes it to F, which accesses B from the client's name. Kerberos proxy tickets are bound to a service and the client's proxy ticket only grants access to B. Therefore, F cannot use the ticket to access a different server from the client's name.

I use the Kerberos ticket proxy flag to implement offloaded Kerberos authentication requests in the Heimdal Kerberos v1.2 library [14]. The client requests an anonymous, proxy ticket by setting the *proxiable* and *anonymous* flags in the hardware pre-authenticated AS-

REQ request to the KDC. The KDC verifies the pre-authentication and passes the request to a new *kdcproxy* service that I implemented. *kdcproxy* extracts the authentication path from the KDC configuration file and obtains ticket-granting tickets from each intermediate KDC on the path using standard Kerberos V5 messages. *kdcproxy* then obtains from the web server's KDC an anonymous, proxy service ticket bound to the client's IP, and returns it to the client.

5.3 VERIFYING KERBEROS CREDENTIALS IN HTTP AND HTTPS TRAFFIC

The client sends the Base64 encoded Kerberos ticket inside the Cookie HTTP header line of each HTTP request (Listing 5.3). Sentinel parses the HTTP request and locates the Sentinel cookie. If a valid cookie is not found, Sentinel returns to the client an HTML page suggesting that the client pre-authenticates to access the web server.

Listing 5.3: HTTP request with Sentinel cookie

```
GET / HTTP/1.1
Host: www.ddosresilient.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.5)
Gecko/2013041720 Firefox/3.0.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: close
Cookie: SentinelCookie=boICEzCCAg+gAwIBBaEDAgEOogcDBQAAAAAAo4IBamGC
...<Base64 encoded cookie>...
P0Qw==
```

When using HTTPS, the client encodes the Kerberos ticket inside a novel TLS Client Hello "kerberos_credential". TLS v1.2 [62] defines the format of TLS Hello extensions (Listing 5.4). Each extension has a type and variable-length opaque data. Public extension type numbers are assigned by the Internet Assigned Numbers Authority (IANA).

Listing 5.4: TLS Hello message extension format

```
struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;
```

Listings 5.5 and 5.6 describe the format of the TLS “kerberos_credential” extension. I use non-public TLS extension type 36 for this implementation. The extension data containing the Base64 Kerberos ticket has variable length and can occupy up to 65535 bytes. The “kerberos_credential” extension is backwards-compatible with the TLS protocol because the TLS extension specification [46] requires servers to process and ignore unrecognized TLS extensions.

Listing 5.5: TLS “kerberos_credential” extension type

```
enum {
    kerberos_credential(36), (65535)
} ExtensionType;
```

Listing 5.6: TLS “kerberos_credential” extension data

```
struct {
    opaque kerb_credential<1..2^16-1>;
} KerberosCredentialExtData;
```

I implemented the “kerberos_credential” extension in the Network Security Services (NSS) 3.12 library that provides the TLS implementation used in the Mozilla Firefox 3.0.5 web browser. The Personal Security Manager (PSM) Firefox component acts as the interface between NSS and the other Firefox components. The NSS API allows PSM to register callback functions that NSS calls during the TLS handshake. I extended PSM and the NSS API with a Sentinel cookie lookup callback function. When sending a Client Hello message, NSS calls the registered function. PSM searches Firefox’s HTTP cookie cache and returns the result to NSS. If a Sentinel cookie was found, NSS adds the “kerberos_credential” extension with the encoded Kerberos ticket to the outgoing Client Hello message.

Sentinel intercepts the TLS Client Hello and checks for the “kerberos_credential” extension. If the extension is not found or does not contain a valid Kerberos ticket, Sentinel responds to the Client Hello with a fatal TLS alert. The DigitalMe Firefox extension intercepts the TLS alert and queries the user to pre-authenticate with an RSA SecurID token.

5.4 IDENTITY METASYSTEM AND INFORMATION CARD INTEGRATION

This implementation integrates Kerberos and RSA SecurID authentication into the Bandit-Projct identity provider and the DigitalMe identity selector [2]. The Bandit Project identity provider is a Java application running on the Apache Tomcat web server. DigitalMe is the Bandit Project's identity selector application. It is a standalone software application, but also provides a Firefox extension for launching the application within the web browser.

5.4.1 INFORMATION CARD EXTENSIONS

Identity Metasystem information cards contain metadata about the identify provider that issues them. The metadata includes lists of token types and identity claims supported by the provider. I implemented RSA SecurIDs as a new type of information card that uses hardware-authenticated Kerberos proxy tickets as security token and identity claim (Listings 5.7 and 5.8). To use this card, the IdM client queries the user for their Kerberos password and SecurID token code, generates a hardware pre-authenticated Kerberos AS-REQ message an sends the AS-REQ as an authentication credential in the security token request.

Listing 5.7: Supported token types in information card

```
<ic:SupportedTokenTypeList>
  <trust:TokenType
      xmlns:trust="http://schemas.xmlsoap.org/ws/2005/02/trust">
    http://www.cs.pitt.edu/Kerb5HWAuthProxyTicket
  </trust:TokenType>
</ic:SupportedTokenTypeList>
```

Listing 5.8: Supported claim types in information card

```
<ic:SupportedClaimTypeList>
  <ic:SupportedClaimType
      Uri="http://cs.pitt.edu/HWAuthKerbProxyTicket">
  <ic:DisplayTag>
    Kerberos Credentials
  </ic:DisplayTag>
  <ic:Description>
    Kerberos Credentials
  </ic:Description>
  </ic:SupportedClaimType>
</ic:SupportedClaimTypeList>
```

5.4.2 SECURITY TOKEN REQUEST AND RESPONSE EXTENSIONS

IdM clients send Request Security Token (RST) messages to authenticate with the identity provider and request a security token. The identity provider verifies the authentication credentials, and returns the security token with the desired claims inside a Request Security Token Response (RSTR) message. In the existing IdM protocol, the identity provider protects the integrity of the security token using a "proof key", which can be symmetric or asymmetric depending on the type of security token used. I extend the RSTR message to add a hardware-accelerated Kerberos proxy ticket as a new security token type. Kerberos tickets are encrypted and have a built-in integrity protection mechanism. Therefore, the new security token type I propose does not require a proof key for integrity protection and it is not used in the extended RSTR message.

Listing 5.9 shows a sample RST message requesting a Kerberos proxy ticket as a security token. The *KeyType* field indicates that no proof key is required.

Listing 5.9: Request Security Token message

```
<wst:RequestSecurityToken
    xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
    Context="ProcessRequestSecurityToken">
  <wst:RequestType>
    http://schemas.xmlsoap.org/ws/2005/02/trust/Issue
  </wst:RequestType>
  <wst:TokenType>
    http://www.cs.pitt.edu/Kerb5HWAuthProxyTicket
  </wst:TokenType>
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey
  </wst:KeyType>
  ...
  <wst:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
    <wsid:ClaimType
        xmlns:wsid="http://schemas.xmlsoap.org/ws/2005/05/identity"
      Uri="http://cs.pitt.edu/HWAuthKerbProxyTicket"/>
  </wst:Claims>
  ...
</wst:RequestSecurityToken>
```

The identity provider passes the AS-REQ message to the Kerberos KDC for verification. The KDC verifies the RSA SecurID pre-authentication, obtains a proxy Kerberos service ticket for the desired web server, and returns the ticket to the IdM identity provider. The provider encodes the ticket into a security token and returns it to the client inside the RSTR message (Listing 5.10).

Listing 5.10: Request Security Token Response message

```
<RequestSecurityTokenResponse
    xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust"
    Context="ProcessRequestSecurityToken">
  <wst:TokenType
      xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
    http://www.cs.pitt.edu/Kerb5HWAuthProxyTicket
  </wst:TokenType>
  <wst:RequestedSecurityToken
      xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
    <...encoded Kerberos proxy ticket...>
  </wst:RequestedSecurityToken>
</RequestSecurityTokenResponse>
```

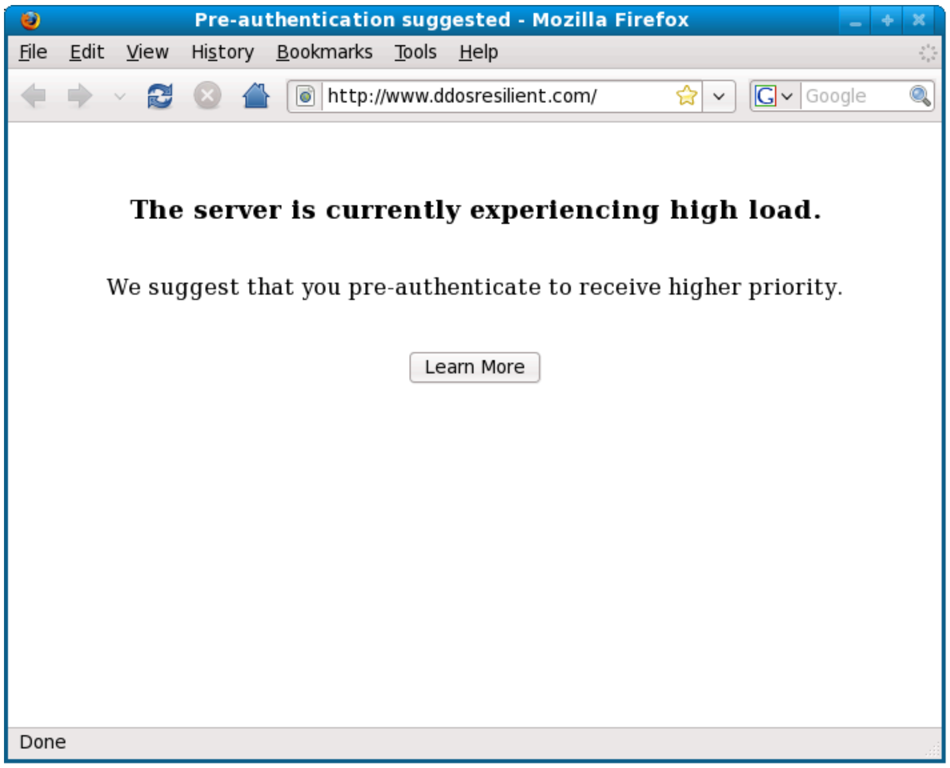
5.5 USER INTERFACE

This section presents the authentication user interface in the Mozilla Firefox browser and DigitalMe identity selector. The interface snapshots are organized in the order that the user sees them.

5.5.1 SENTINEL RESPONSE TO UNAUTHENTICATED HTTP REQUESTS

When a user requests a web page from a web server, e.g., http://www.ddosresilient.com, that is protected by a Sentinel front-end device currently under a DoS attack, Sentinel checks the HTTP request for a valid Kerberos service ticket. If a valid ticket is not found, Sentinel responds with a web page (Fig. 22), which alerts the user of the high server load and suggests pre-authentication before receiving service. The user can click on the "Learn More" button to obtain more information, or try to access the web server again later.

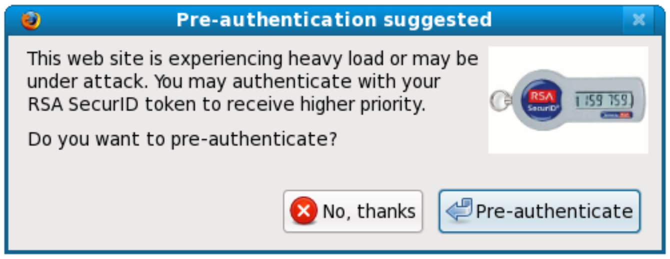
Sentinel responds with this web page to both client and bot HTTP requests without valid credentials. A botnet generating dummy HTTP requests at high rate can force Sentinel to


Pre-authentication suggested - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.ddosresilient.com/
Google
Learn More
Done

The server is currently experiencing high load.

We suggest that you pre-authenticate to receive higher priority.

Figure 22: HTTP response from Sentinel during a DDoS attack


Pre-authentication suggested

This web site is experiencing heavy load or may be under attack. You may authenticate with your RSA SecurID token to receive higher priority.

Do you want to pre-authenticate?

Figure 23: Pre-authentication query

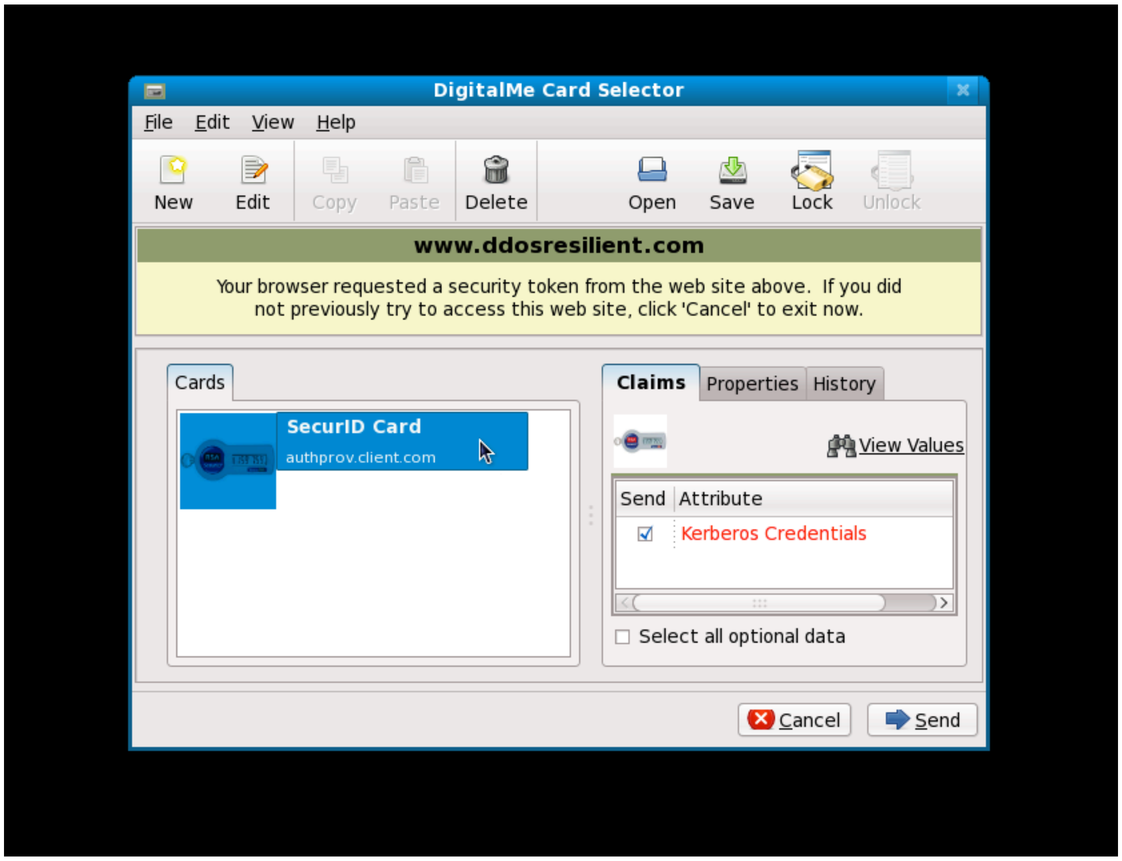
respond with web pages at the same rate. Therefore, the web page returned must be small so as to deplete only a minimum amount of Sentinel's network bandwidth. Sentinel's response should fit into a single Ethernet frame to reduce the resources Sentinel spends on responding to a bot request.

Sentinel responds with this web page only to HTTP requests. If a client initiates a TLS connection without valid credentials, Sentinel responds with a TLS alert message. Firefox detects and processes the TLS alert without displaying the pre-authentication notice page.

5.5.2 IDENTITY SELECTOR QUERY DIALOG WINDOW

The DigitalMe Firefox extension displays a dialog window with further information to the user (Fig. 23). The dialog explains that the desired web server is likely under attack and queries the user to pre-authenticate using an RSA SecurID token. The user can choose to pre-authenticate or cancel the request.

If the user chooses to pre-authenticate, the Firefox extension launches the DigitalMe identity selector (Fig. 24), which queries the user for an information card. The identity selector shows all information cards that can provide the identity claim required by Sentinel – a hardware-authenticated Kerberos proxy ticket. If the information card provides more claims than Sentinel requires, the user can uncheck the claims that should not be requested



www.ddosresilient.com

Your browser requested a security token from the web site above. If you did not previously try to access this web site, click 'Cancel' to exit now.

SecurID Card
authprov.client.com

Send	Attribute
✓	Kerberos Credentials

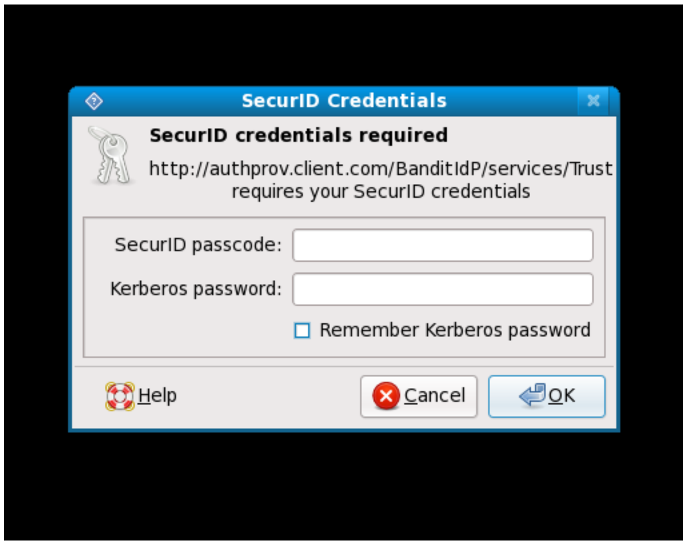
☐ Select all optional data

Figure 24: DigitalMe card selector

from the identity provider.

The DigitalMe window is resilient to interface spoofing attacks. The dark background in Fig. 24 shows the identity selector's *security desktop*. The security desktop is a modal window that covers the user's desktop space. This prevents malicious web scripts or applications from queries the user with spoofed windows that mimic the DigitalMe identity selector dialogs. Resilience to interface spoofing is important in the context of federated authentication because obtaining a single set of credentials gives an attacker access to multiple service providers.

When the user selects the RSA SecurID information card, the identity selector queries the user for their Kerberos password and SecurID token code (Fig. 25). DigitalMe then generates the hardware pre-authenticated Kerberos AS-REQ message and sends it with the



http://authprov.client.com/BanditIdP/services/Trust requires your SecurID credentials

SecurID passcode:

Kerberos password:

☐ Remember Kerberos password

Figure 25: Kerberos/SecurID authentication dialog

security token request to the identity provider. The identity provider authenticates the user and returns the Kerberos proxy ticket, which the DigitalMe Firefox extension saves in the Firefox HTTP cookie store for subsequent HTTP/HTTPS requests.

6.0 PERFORMANCE ANALYSIS

This chapter evaluates Sentinel's ability to mitigate distributed denial-of-service attacks against HTTP and HTTPS servers. I generate HTTP and TLS flood attacks at variable rates, and compare the resilience of Sentinel to that of a standard web server running on the same hardware platform as Sentinel. The performance experiments also evaluate the experience of legitimate human users accessing the web server during the attack. I compare the Sentinel performance and legitimate user experience using two different forms of authentication implemented in Sentinel: CAPTCHA puzzles and RSA SecurID authentication.

Section 6.1 describes the experimental setup for my performance tests. Section 6.2 presents the experimental design. Section 6.3 evaluates the resilience of a standard web server against HTTP request floods. Sections 6.4 and 6.5 compare the performance of Sentinel using CAPTCHA authentication with, respectively, dynamically generated and pre-generated puzzles. Sections 6.6 and 6.7 evaluate Sentinel using RSA SecurID authentication without and with hardware-accelerated blacklisting, respectively. Sections 6.8 and 6.9 compare the performance of a standard web server host and the Sentinel front-end device using RSA SecurID authentication against floods of TLS requests.

6.1 EXPERIMENTAL SETUP

Fig. 26 shows the experimental setup for the performance experiments in this chapter. Two 8-port HP ProCurve 1800G gigabit Ethernet switches S_1 and S_2 are connected to each other via a 1Gbps link. Six attacker hosts are connected via 1Gbps links to switch S_1. Switch S_2 connects the RSA Authentication Manager server and hosts belonging to two different

Kerberos realms – client.com and ddosresilient.com. The client.com Kerberos realm consists of the legitimate client host and the client's KDC, which are connected to S_2 via 100Mbps links. The ddosresilient.com domain contains Sentinel, the ddosresilient.com web server and the web server's KDC. The ddosresilient.com KDC is connected to S_2 via a 100Mbps link. Sentinel's NFE I-8000 card and a standard Ethernet network card are connected to S_2 via 1Gbps links. Sentinel's NFE I-8000 is also connected to the ddosresilient.com web server. The RSA Authentication Manger is connected to switch S_2 via a 100Mbps link.

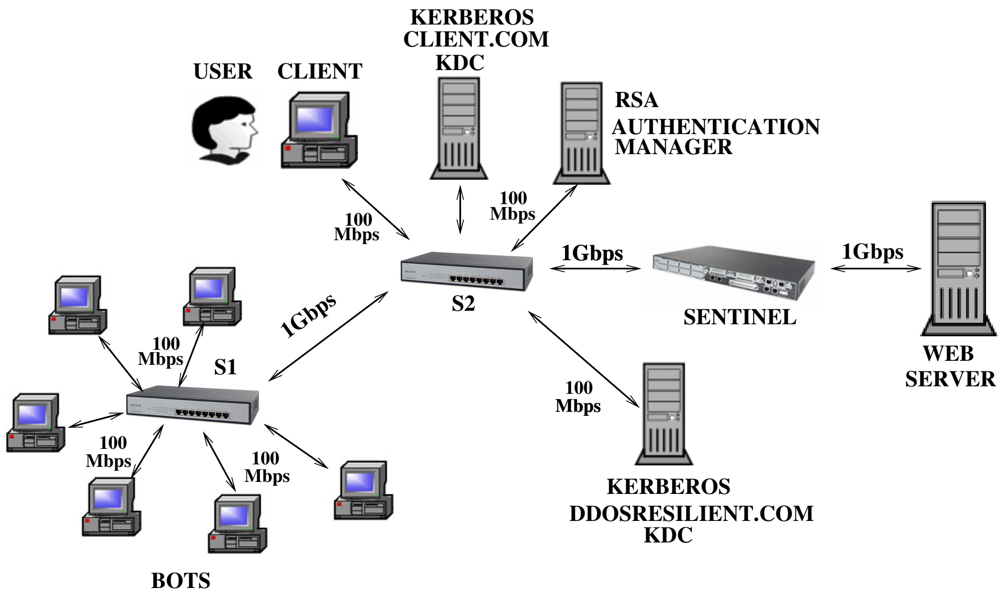


Figure 26: Experimental setup

The hosts' hardware and software configuration is as follows:

- Legitimate client: Firefox 3.05 with DigitalMe extension installed on an HP Pavillion zx5000 laptop with Intel Pentium 4 CPU at 3.06 GHz, 2GB RAM, and Fedora 10 Linux.
- Sentinel: Sentinel application, NFM 2.0 and NFD 1.2.3 installed on a Dell Dimension 9200 with dual-core Intel Pentium Core Duo at 2.4GHz, 2GB RAM, NFE-I8000 card, and Fedora Core 7 Linux. I turned off one of the CPU cores to facilitate performance measurements. In experiments where the DDoS attack target is a web server host, nginx 1.2.8 with OpenSSL 0.9.8i run on the Sentinel device.

- Web server: Apache 2.2.10 and OpenSSL 0.9.8g installed on a Dell Dimension 4550 with Intel Pentium 4 CPU at 2.4 GHz, 256MB RAM, and Fedora 10 Linux. This is the back-end web server behind Sentinel and it is only used in performance tests where Sentinel mitigates the DDoS attack.
- Kerberos client.com and ddosresilient.com KDCs: extended versions of Heimdal v1.2 installed on two Dell Dimension 4550 with Intel Pentium 4 at 2.4 GHz, 256MB RAM, and Fedora 10 Linux.
- RSA authentication server: RSA Authentication Manager v7.1.2 installed on Dell PowerEdge 1600SC with quad-core Intel Xeon CPU at 2.4GHz, 3GB RAM and RHEL 4 Linux.
- Attackers 1-4: modified version of BoNeSi 0.2.0 [4] installed on Dell Optiplex 380 with dual-core Intel Core2 Duo CPU at 3.06GHz, 4GB RAM and CentOS 6.4 Linux.
- Attackers 5-6: modified version of BoNeSi 0.2.0 installed on Dell PowerEdge T105 with quad-core AMD Opteron 1352 CPU, 4GB RAM, and Fedora 10 Linux.

6.1.1 DDOS ATTACK GENERATION WITH BONESI

The DDoS Botnet Simulator (BoNeSi) is a DDoS attack generation tool developed by the German Research Center for Artificial Intelligence [4]. BoNeSi can generate HTTP floods from a pre-configured set of IP addresses to a pre-configured set of URLs. It uses the libnet and libpcap Linux libraries to, respectively, inject IP packets into and receive IP packets from the IP layer of the Linux kernel network stack. During an HTTP flood attack, BoNesi establishes a TCP connection and sends an HTTP request to the target URL with a “Connection: close” header line (Fig. 27). The header line instructs the web server to close the TCP connection immediately after sending the HTTP response.

I extended BoNeSi to generate floods of TLS connections that use ephemeral Diffie-Hellman (eDH) key exchange. In such a TLS connection, BoNeSi sends a TLS Client Hello message to the server and advertises that it only supports TLS cipher suites using eDH key exchange (Fig. 28). The TLS server must pick one of the cipher suites advertised by the client. The server responds with Server Hello, Server Certificate and Server Key Exchange messages.

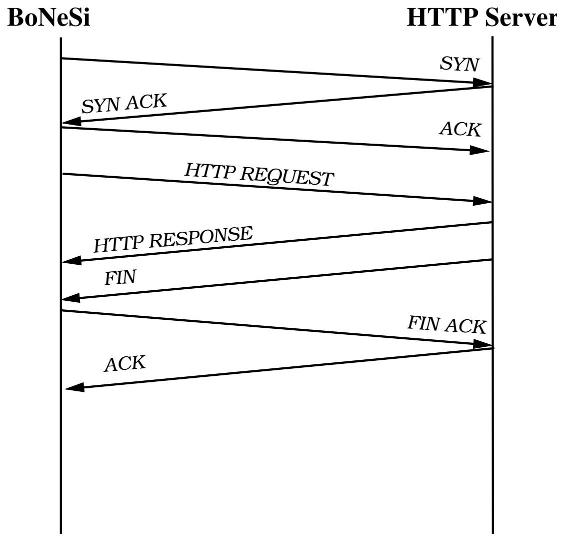


Figure 27: Bot HTTP connection

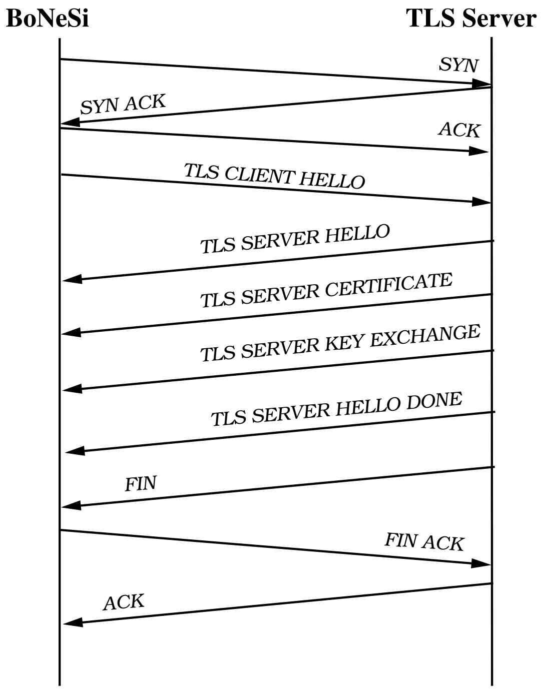


Figure 28: Bot TLS connection

The Server Certificate contains an X.509 certificate with the server's RSA public key. The Server Key Exchange message contains a randomly generated ephemeral Diffie-Hellman key signed by the server's RSA private key. Generating the ephemeral Diffie-Hellman key and the RSA signature are both computationally intensive operations that allow BoNeSi to deplete the TLS server's CPU capacity with a small number of requests. BoNeSi waits for the Server Key Exchange message and immediately terminates the TCP connection.

6.1.2 WEB SERVER PAGES

.

Fig. 29 and 30 show the web pages returned by, respectively, the Sentinel-protected and the standalone web server. The sizes of the HTML pages are 195 bytes and 205 bytes, respectively. Each HTML page contains two images – a 45507-byte University of Pittsburgh logo PNG image and a 5031-byte "PITT" text JPEG image. I disabled HTTP pipelining in

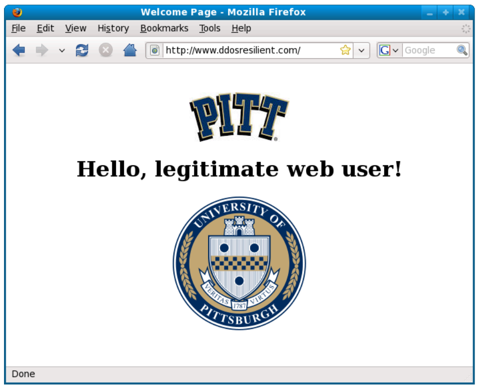

Welcome Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.ddosresilient.com/
Google
Hello, legitimate web user!
Done

Figure 29: Web page from web server behind Sentinel

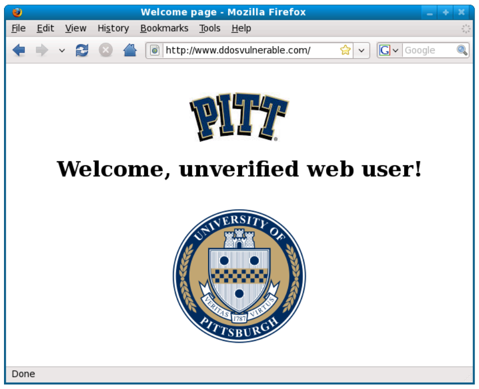

Welcome page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.ddosvulnerable.com/
Google
Welcome, unverified web user!
Done

Figure 30: Web page from standalone web server

Firefox on the legitimate client host to force it to use a new TCP connection to download each file. When accessing the web pages over TLS, the browser will also establish a new TLS connection for each request. Having multiple files to download from the web server and using different TCP and TLS connections for each file mimics the browser downloading a real-world web page consisting of multiple files possibly hosted on more than one web server.

The minimal size and number of objects of the web pages was chosen to facilitate the performance experiments. A higher number of larger objects retrieved by legitimate clients would increase the probability of dropped packets and should cause increased response times for legitimate client requests. The performance results in this chapter show that the web pages used are large enough to trigger packet loss and increased legitimate client response times under heave DDoS attack load.

6.2 PERFORMANCE EXPERIMENT DESIGN

I measure three performance metrics during all performance experiments to evaluate Sentinel's resilience to DDoS attacks and the attack's effect on legitimate client requests:

- *DDoS attack target CPU utilization*: I measure the utilization of the Intel Pentium Core Duo CPU on the Sentinel hardware platform, which runs the Sentinel application or a standard web server. During DDoS attacks that target the host CPU resources, CPU utilization indicates the ability of the target to scale to higher attack rates. I take CPU utilization measurements once every 15 seconds during the test.
- *Network bandwidth consumed by attack*: I measure the amount of bi-directional network traffic generated during the DDoS attack. For attacks where the network infrastructure of the target is the performance bottleneck, the network bandwidth indicates the ability of the target's network to scale to higher DDoS attack rates. I measure the consumed network bandwidth by querying the statistic counters for the port of switch S_2 that is connected to switch S_1 via SNMP every 15 seconds (Fig. 26). This guarantees that only traffic from and to the attacker hosts will be measured.
- *Response time of legitimate client requests*: I collect packet captures of legitimate client requests on the client host and measure the response time to download the web page from the web server. For experiments that use CAPTCHA and hardware token authentication, I also measure the response time to obtain, respectively, a CAPTCHA puzzle, or a pre-authentication notice HTML page. High response times while the target is under attack indicate that the legitimate client's packets get queued up and/or dropped before getting processed. The ideal scenario is one where the DDoS attack is mitigated sufficiently that it does not affect the response time of legitimate clients accessing the web server under attack. High response times to access web pages results in legitimate users giving up trying to access the web server under attack. A recent survey shows that 40% of consumers would abandon a web site if it does not load within 3 seconds [25].

An additional metric that I do not consider in this performance analysis is the false positive rate of Sentinel's Bloom filter. The Bloom filter parameters (number of buckets, number of bits per bucket, number of hash functions) were chosen to minimize the false positive rate. Section 3.4 presents the chosen parameter values and the approximate false positive rate computed based on these values.

For all performance metrics, I record 20 repetitions of each data point. I plot the performance data using box-and-whisker plots where the box plot shows the 25th, 50th and 75th

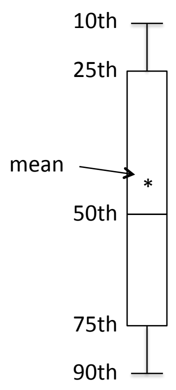


Figure 31: Sample box plot

percentiles, the whiskers show the 10th and 90th percentiles, and a separate star shows the mean of the 20 repetitions (Fig. 31).

Table 1 shows a list of variable and fixed parameters for this dissertation's performance experiments. The attack rate measured in requests/second is the main variable parameter. I compare the performance of Sentinel and a standard web server under both HTTP and TLS flood attacks. When testing against Sentinel, I compare the performance of CAPTCHA puzzles and hardware token authentication. I also discuss the performance of dynamically generating CAPTCHA puzzles during the attack and pre-generating them in advance. All performance experiments use a fixed number of 6000 simulated bot IP addresses – the maximum IP addresses I could simulate on the six attacker hosts without affecting negatively the performance of the BoNeSi attack generation tool.

Given the list of variable parameters above, we defined a set of performance experiments for this dissertation. The rest of this chapter chapter presents and analyzes the results from the following experiments:

- HTTP flood against standard web server (Section 6.3)
- HTTP flood against Sentinel with dynamic CAPTCHA authentication and no blacklisting (Section 6.4)

Attack rate	Fixed / variable	Possible values
Attack type	Variable	HTTP or TLS flood
Attack target	Variable	Standard web server or Sentinel
Authentication type	Variable	CAPTCHA puzzles or hardware tokens
CAPTCHA generation option	Variable	Dynamically generated or pre-generated
Number of bot IP addresses	Fixed	6000
Web pages for HTTP GET requests	Fixed	Described in Section 6.1

Table 1: Variable and fixed parameters for performance experiments

- HTTP flood against Sentinel with pre-generated CAPTCHA authentication and no blacklisting (Section 6.5)
- HTTP flood against Sentinel with hardware authentication and no blacklisting (Section 6.6)
- HTTP flood against Sentinel with hardware authentication and blacklisting (Section 6.7)
- TLS flood against standard web server (Section 6.8)
- TLS flood against Sentinel with hardware authentication and no blacklisting (Section 6.9)

I intentionally omit performance experiments with CAPTCHA authentication and blacklisting, as well as with HTTPS floods, hardware token authentication and blacklisting because blacklisting in independent of the authentication mechanism and the type of application-level flood. The experiment using HTTP floods, hardware token authentication, and blacklisting presented in Section 6.7 shows that the performance data before Sentinel has started blacklisting is similar to the data collected when blacklisting is not used at all (Section 6.6). At 19,000 requests/second during both experiments, the results show 193 Mbps of consumed network bandwidth and 98% Sentinel CPU utilization. Once Sentinel blacklists all bots' IP addresses, the DDoS attack is reduced to a SYN flood. The bots' SYN packets are dropped on Sentinel's network acceleration card and the bots never complete the TCP handshakes to send application-level requests or receive authentication challenges.

6.3 HTTP FLOOD ATTACKS AGAINST WEB SERVERS

Fig. 32 shows the CPU utilization of a standalone nginx web server running on Sentinel's hardware platform. The portion of CPU capacity consumed initially increases linearly with the DDoS attack rate. At 16,000 HTTP requests per second, BoNeSi depletes the web

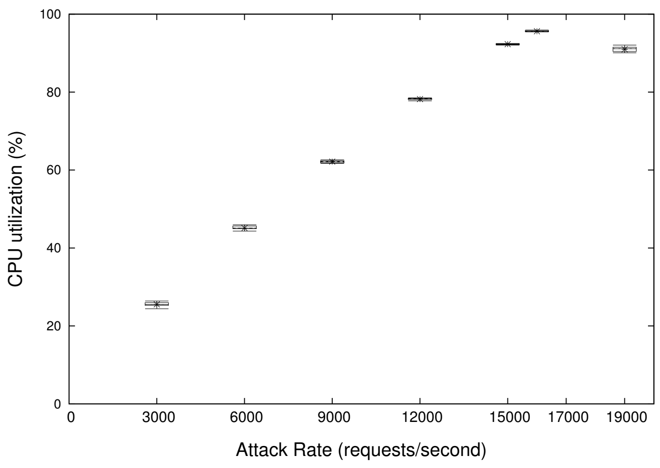
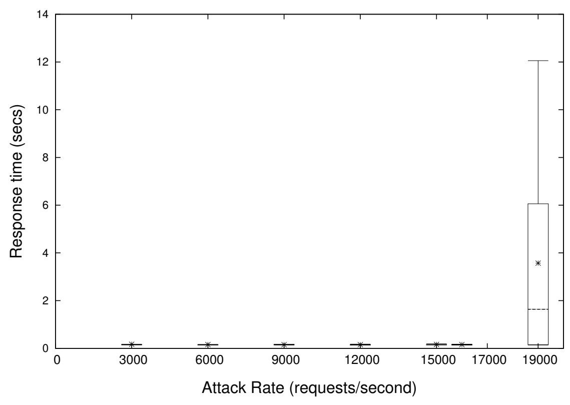


Figure 32: Experiment 6.3 – CPU utilization



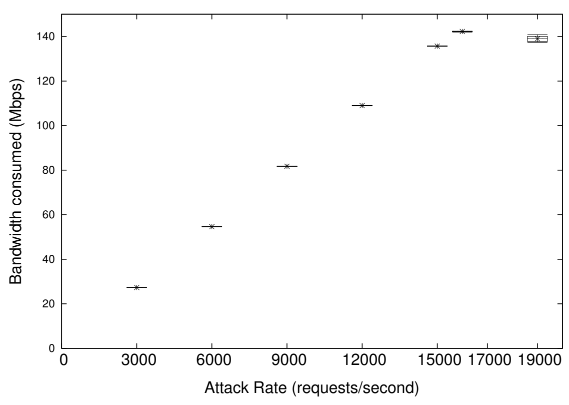


Figure 33: Experiment 6.3 – response time to get page

Figure 34: Experiment 6.3 – network bandwidth

server's CPU capacity and the CPU utilization reaches 95%. When the attack rate increases to 19,000 requests/second, nginx does not have sufficient CPU resources to service all HTTP requests at the rate they are received. The SYN packet backlog queue for nginx' listening socket in the kernel network stack fills up and the server starts discarding SYN packets from bots. This slightly reduces the load on the CPU and the CPU utilization decreases to 90%. However, the web server's kernel at this time also discards SYN packets from legitimate clients. Fig. 33 shows the legitimate client's response time to download the web page from the server during the DDoS attack. When the attack rate reaches 19,000 requests/second and nginx' SYN packet backlog queue fills up, the response times become higher and more variable. The Linux kernel on the client host uses the kernel's 3-second timeout before retransmitting a SYN packet, making most high client response times in multiples of 3 seconds. The 3-second timeout is a standard default parameter compiled into the Linux kernel and was originally recommended by RFC1122 as initial retransmission timeout for new TCP connections [48].

The DDoS attack depletes the web server's CPU capacity before it reaches the server's 1Gbps network connection capacity. Fig. 34 show the network bandwidth consumed by the attack. At 16,000 requests per second, the attack generates just 142Mbps of network traffic. When the web server discards bot-originated SYN packets at 19,000 requests/sec, the bots are unable to complete as many connections and the bandwidth decreases slightly to 139Mbps.

This experiment demonstrates the ability of HTTP floods to deplete a web server's CPU capacity with modest amount of network traffic. Web server administrators can adjust the Linux kernel network stack parameters, e.g., increase the TCP SYN packet backlog queue size, but the web server application is unable to keep up with the rate of incoming HTTP requests and a persistent DDoS attack will eventually cause the queue to overflow.

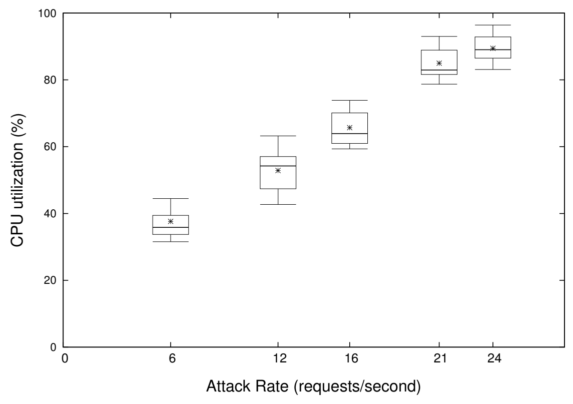


Figure 35: Experiment 6.4 – CPU utilization

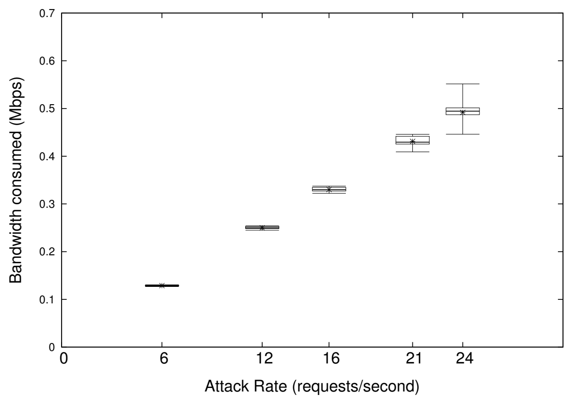


Figure 36: Experiment 6.4 – network bandwidth

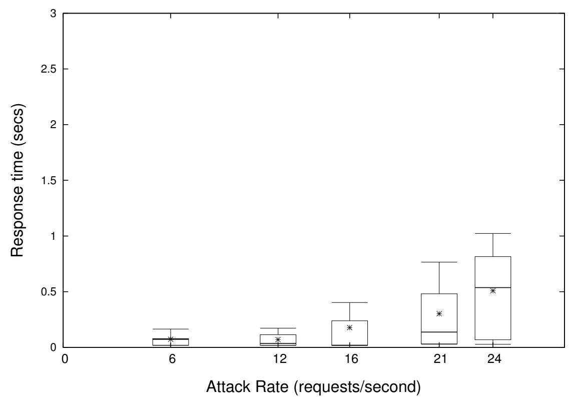


Figure 37: Experiment 6.4 – response time to get CAPTCHA

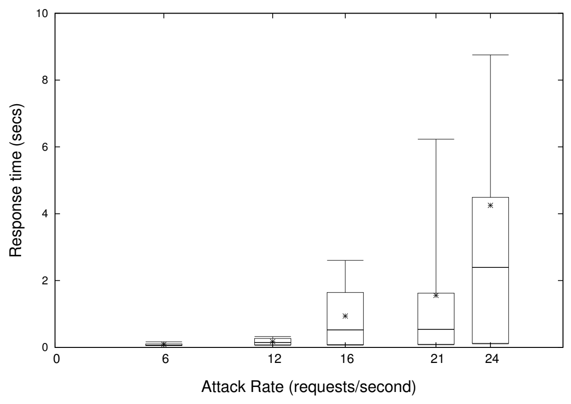


Figure 38: Experiment 6.4 – response time to get page

6.4 HTTP FLOOD ATTACKS AND CAPTCHA AUTHENTICATION WITH DYNAMICALLY GENERATED PUZZLES

CAPTCHA puzzle generation is a critical part of CAPTCHA authentication. If the target of a DDoS attack chooses to dynamically generate CAPTCHA puzzles, it must generate

Image format	Latency	Encoded image size
24-bit PNG	11.7 msecs	2190 bytes
8-bit PNG	45 msecs	1102 bytes
GIF	129 msecs	976 bytes
JPEG	11 msecs	2059 bytes

Table 2: CAPTCHA image format comparison

them at the same rate as the rate of the DDoS attack. This places additional computation load on a server already low on CPU resources. If the server chooses to pre-load a pool of pre-generated CAPTCHA puzzles, it risks having to re-use the puzzles if the DDoS attack rate is high enough to get all puzzles in the server's pool. Puzzle reuse is not as secure because the attacker can solve all of the server's puzzles out-of-band and then re-use the responses to launch a DDoS attack.

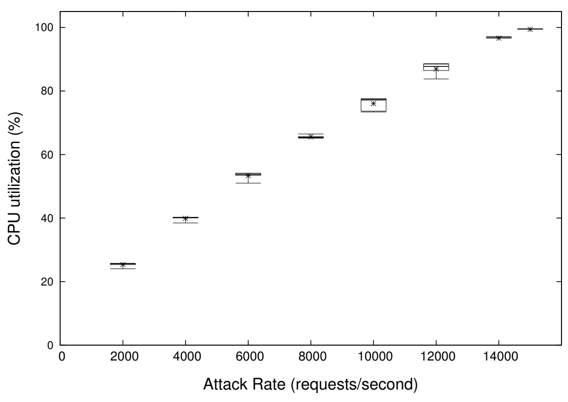
I implemented CAPTCHA authentication in Sentinel with both dynamically generated and pre-generated CAPTCHA puzzles. There are no standard or widely adopted algorithms for on-server CAPTCHA puzzle generation. Many web servers use the reCAPTCHA service [12] to provide clients with pre-generated reCAPTCHA puzzles, but reCAPTCHA's puzzle generation algorithms are not publicly known. I implemented in Sentinel a routine that generates 5-character puzzles using the ImageMagick API [8]. Through performance profiling, I determined that encoding the puzzle into the respective image format consumes nearly 90% of the latency for generating a single puzzle. I tested the encoding latency and the Base64 encoding size of the produced image for different image formats (Table 2).

None of the ImageMagic image encoding algorithms tested are fast enough for Sentinel to dynamically generate CAPTCHA puzzles for each intercepted bot request. For this experiment, I chose the 8-bit PNG image encoding. The current Sentinel implementation sends its HTTP response and CAPTCHA puzzle in two Ethernet frames. For performance reasons, the TCP segment in each Ethernet frame has a maximum size of 1000 bytes. The 24-bit PNG and JPEG encoding algorithms are faster than 8-bit PNG encoding, but not fast enough to generate puzzles for high-rate DDoS attacks and the produced images do not fit into the two 1000-byte TCP segments available. There is no publicly known explanation why 8-bit PNG encoding in ImageMagick is faster than 24-bit PNG encoding. A possible explanation may be the additional downsampling required to encode an image into 8-bit PNG format.

Fig. 35 shows the CPU utilization of Sentinel under attack while generating dynamic CAPTCHA puzzles. Sentinel's CPU is consumed when the attack rate reaches 24 connections/second, which is explained by the 45 milliseconds it takes to encode an 8-bit PNG image. The CPU utilization values measured for this experiment have higher variance because the legitimate client requests are not spread evenly during the test. A single legitimate client request can consume a noticeable amount of CPU cycles when Sentinel's CPU can process only 24 requests/second. Fig. 36 shows that the DDoS attack generates only up to 0.55Mbps of traffic. This is due to the low attack rate sufficient to consume Sentinel's CPU capacity. Each bot connection generates approximately 2,559 bytes of network traffic, so 24 requests/second should generate approximately 0.49Mbps of network traffic. Fig. 37 and 38 show the legitimate client response times, respectively, to get a CAPTCHA puzzle and to download the web page after correctly solving the CAPTCHA. As Sentinel's CPU utilization increases above 90% at 21 bot requests/second, the legitimate client response times increase and have higher variability.

6.5 HTTP FLOOD ATTACKS AND CAPTCHA AUTHENTICATION WITH PRE-GENERATED PUZZLES

Because there are no standard or widely-adopted mechanisms for CAPTCHA puzzle generation, it is infeasible to design a representative experiment for dynamic CAPTCHA generation. In a best-case performance scenario, Sentinel would respond to HTTP requests with CAPTCHA puzzles without spending CPU cycles for puzzle generation. I implemented a Sentinel mode, where Sentinel pre-generates 100 CAPTCHA images at start-up time and serves them in round-robin order. Pre-generated CAPTCHA puzzles provide less security



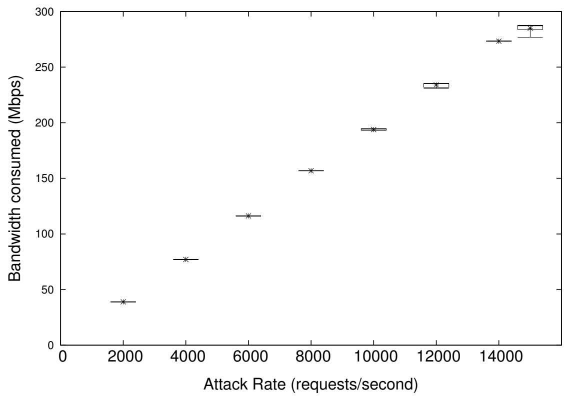


Figure 39: Experiment 6.5 – CPU utilization

Figure 40: Experiment 6.5 – network bandwidth

against DDoS attacks because the attacker can fetch all puzzles from Sentinel's pool, solve them out-of-band, and use the answers to stage a new DDoS attack that bypasses the CAPTCHA authentication defense mechanism. The attacker can store puzzle answers in hash tables and use constant-time lookup algorithm to fetch answers for puzzles issued by Sentinel.

I did not vary the size of the CAPTCHA table in this experiment because the algorithms to fetch the next puzzle and lookup a puzzle's expected answer execute in constant O(1) time. Sentinel keeps track of the index of the next available puzzle and encodes it into the HTML form that the client submits with the puzzle answer. The size of the CAPTCHA table also has limited effect on the ability of Sentinel to filter out bot-originated HTTP requests. A large botnet generating requests at high rate can force Sentinel to use all pre-generated CAPTCHAs in a short period of time. For example, a botnet generating a DDoS attack at 20,000 HTTP requests per second will force a Sentinel device with 1 million pre-generated CAPTCHAs to use all puzzles in just 50 seconds.

Fig. 39 shows the CPU utilization of Sentinel under attack when using pre-generated CAPTCHA puzzles. In this experiment, Sentinel scales better to higher DDoS attack rates – it takes approximately 15,000 requests to consume its CPU capacity. Sentinel's HTTP

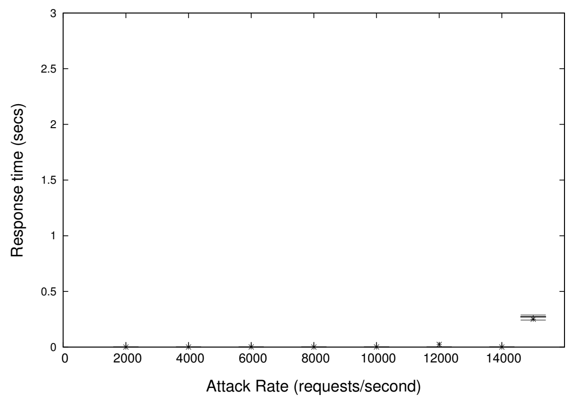


Figure 41: Experiment 6.5 – response time to get CAPTCHA

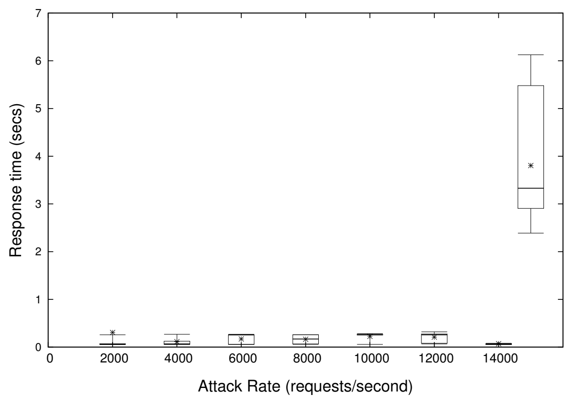


Figure 42: Experiment 6.5 – response time to get page

response to each bot request has a size of 1557 bytes. This explains why Sentinel with CAPTCHA authentication does not scale as well as the standard web server (Section 6.3), which responds with 195 bytes of HTTP data and can handle 16,000 bot requests/second. Fig. 40 shows the bandwidth consumed by the DDoS attack. Each bot connection on the average generated 2436 bytes of network traffic. This number is slightly lower than the 2,559 bytes per bot connection when Sentinel uses dynamically generated CAPTCHAs. In this experiment, BoNeSi generates a higher amount of traffic overall, which increases the probability of packet drops and incomplete bot requests.

6.6 HTTP FLOOD ATTACKS AND HARDWARE TOKEN AUTHENTICATION

This section evaluates the ability of Sentinel with hardware token authentication to mitigate DDoS attacks. Sentinel responds to HTTP requests with an HTML page suggesting hardware token pre-authentication. Fig. 43 shows the CPU utilization of Sentinel under attack. Sentinel scales better than when using CAPTCHA authentication – BoNeSi needs

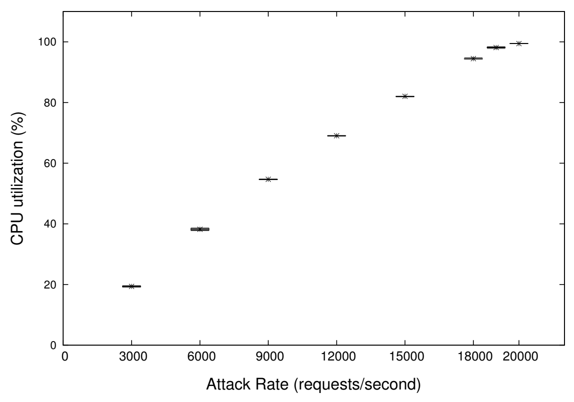


Figure 43: Experiment 6.6 – CPU utilization

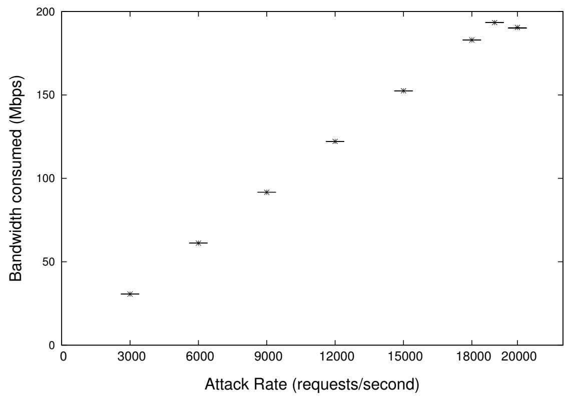


Figure 44: Experiment 6.6 – network bandwidth

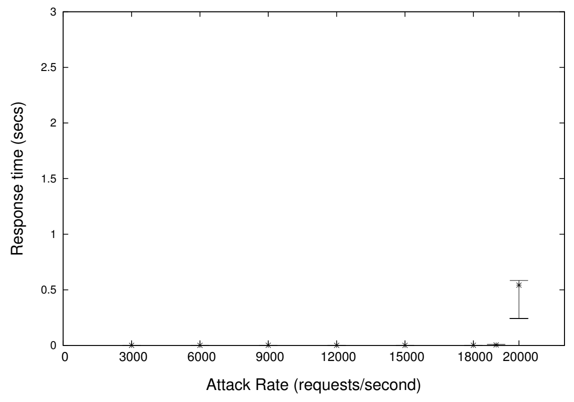


Figure 45: Experiment 6.6 – response time to get challenge

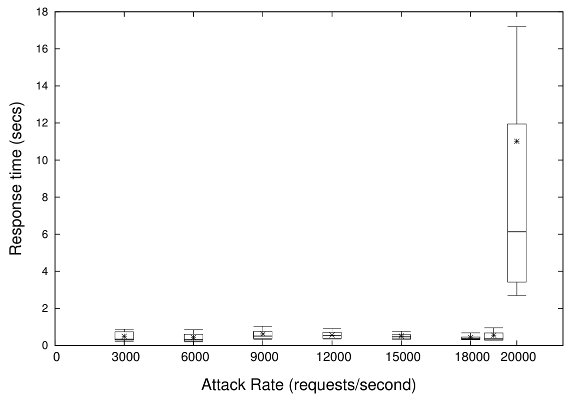


Figure 46: Experiment 6.6 – response time to get page

to generate 19,000 requests/second to deplete Sentinel's CPU capacity. Sentinel's response in this experiment contains 630 bytes of HTTP data, compared to the 1557 bytes of data returned when using CAPTCHA authentication. Fig. 44 shows the bandwidth consumed by the DDoS attack. At 19,000 requests/second, the attack generates 193 Mbps of bi-direction traffic. Fig. 45 and 46 present the response time of the legitimate client to, respectively, obtain the pre-authentication notice page and download the web page after authenticating. At 20,000 requests/second the response time to obtain the web page become high and very variable.

During this experiment, Sentinel does not use hardware-accelerated blacklisting. It performs some of the same tasks that a standard web server performs—receives packets, parses HTTP requests, sends HTTP responses—but also verifies hardware token authentication. The results show that Sentinel with hardware token authentication, but without hardware-accelerated blacklisting, can handle the same attack rate as a standard web server. Therefore, deploying Sentinel does not introduce performance degradation in the server farm and does not open an additional venue for DDS attacks. I show in Section 6.7 that hardware accelerated blacklisting provides Sentinel with a significant performance gain over the standard web server.

6.7 HTTP FLOOD ATTACKS AND BLACKLISTING

As shown in Section 6.6, DDoS attacks of 19,000 HTTP requests/second can deplete Sentinel's CPU resources. To improve its resilience to attacks, Sentinel can track the number of requests without authentication credentials per client and blacklist the client as a bot after a threshold of requests is exceeded. Sentinel drops packets from bot IP addresses on the NFE I-8000 network acceleration card. This prevents bot traffic from consuming Sentinel's host CPU resources. I modified the experiment showing a DDoS attack at 19,000 HTTP requests/second to show the advantages of hardware-accelerated blacklisting of bot-originated traffic. For the new experiment, the legitimate client uses the wget test HTTP client [5] to send HTTP requests once per second during the entire duration of the DDoS attack. To send authenticated HTTP requests at this rate, I encoded a pre-generated, static hardware-accelerated Kerberos proxy ticket as an HTTP cookie in all requests. Sentinel checks client requests for the static Kerberos ticket and, if found, does not attempt to validate it. The DDoS attack and the legitimate client requests last for 50 seconds. Sentinel blacklists client IP addresses as bots after sending 32 consecutive HTTP requests without valid authentica-

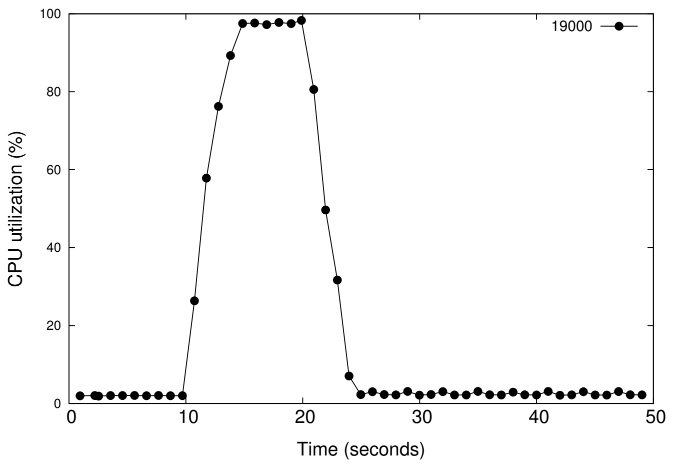
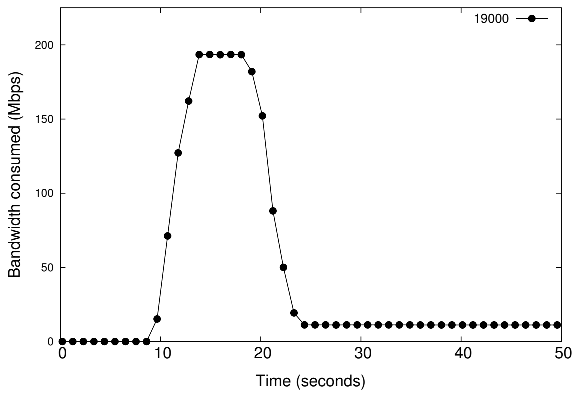


Figure 47: Experiment 6.7, 19000 cps – CPU utilization



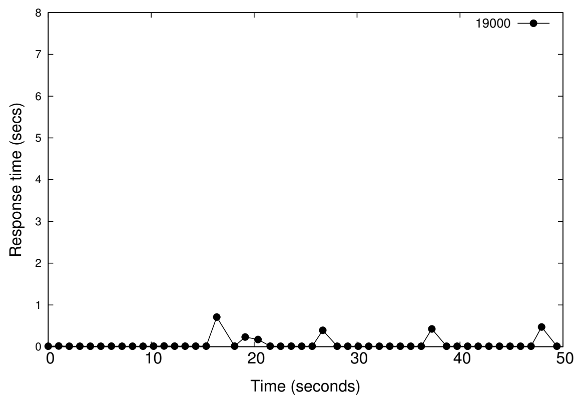


Figure 48: Experiment 6.7, 19000 cps – network bandwidth

Figure 49: Experiment 6.7, 19000 cps – response time to get page

tion credentials.

Fig. 47 shows Sentinel's CPU utilization during a DDoS attack of 19,000 HTTP requests/second before the attack, during the attack, and during blacklisting. The attack begins ten seconds into the test and Sentinel's CPU utilization spikes up to 97%. The attack uses

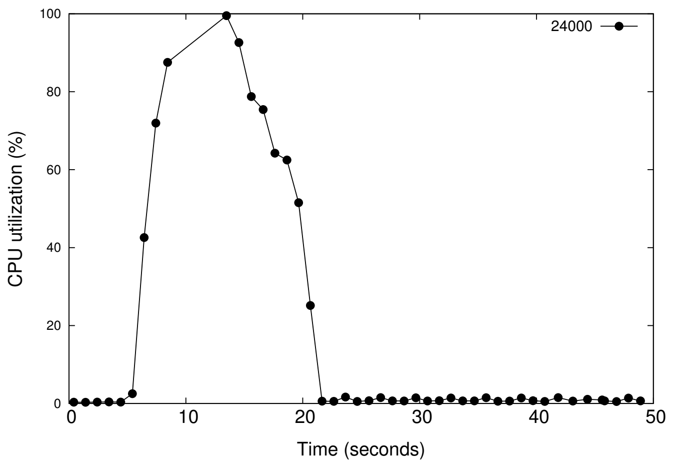
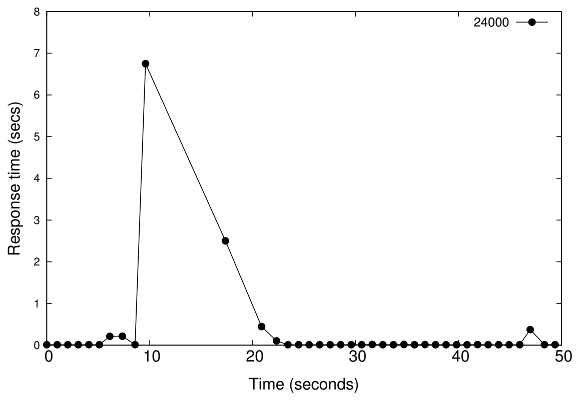


Figure 50: Experiment 6.7, 24000 cps – CPU utilization

6,000 distinct IP addresses and the the bot requests are spread evenly between them, so each request is used approximately 3.17 times per second. Therefore, it should take 10 seconds for BoNeSi to generate 32 HTTP requests from each IP address. The results show that approximately 10 seconds after the beginning of the attack, Sentinel starts blacklisting the bot IP addresses and dropping bot-originated packets on the network acceleration card. Sentinel's CPU utilization drops to 3% and remains at that level until the end of the test. Fig. 48 shows the network bandwidth consumed during the attack. The DDoS attack generates 193 Mbps of traffic (also seen in Fig. 44). Once blacklisting starts, the HTTP flood attack is mostly reduced to a SYN packet flood, because BoNeSi will never receive TCP SYN ACK packets from Sentinel. A SYN flood attack of 19,000 SYN packets per second, where each SYN packet Ethernet frame is 66 bytes, should generate 10.03 Mbps of traffic. The experiment shows that the bandwidth consumed after blacklisting has started is 11 Mbps. Fig 49 shows the response times of the legitimate client HTTP requests. The low response times match the response times observed at 19,000 requests/second in Fig. 44.

When Sentinel using hardware token authentication without blacklisting was flooded with more than 19,000 HTTP requests/second, the Sentinel application crashed and I was not able to collect reliable performance data. I repeated the same experiment with blacklisting



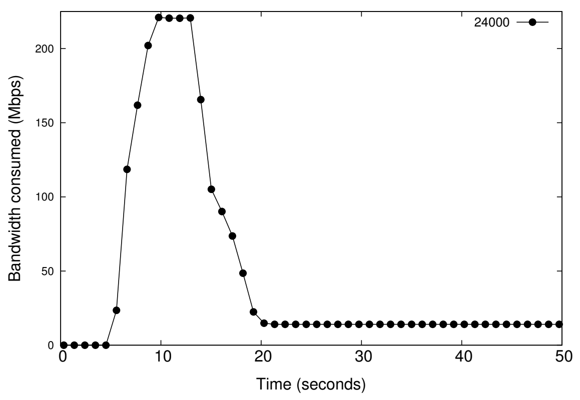


Figure 51: Experiment 6.7, 24000 cps – response time to get page

Figure 52: Experiment 6.7, 24000 cps – network bandwidth

at 24,000 HTTP requests/second. Fig. 50 shows the CPU utilization before the attack, during the peak of the attack and during blacklisting. The attack begins six seconds into the experiment and Sentinel's CPU utilization increases to nearly 100%. This can explain the crash of the Sentinel application I observed during the experiment without blacklisting. At 24,000 request/second, it should take eight seconds for BoNesi to generate 32 requests from each bot IP address. Because Sentinel could not keep up with the rate of the attack, it took four seconds for BoNeSi to ramp up the attack. Sentinel starts blacklisting bot-originated packets nine seconds after the beginning of the attack and Sentinel's CPU utilization drops to 1% until the end of the test. Fig. 52 shows the bandwidth consumed by the DDoS attack. The generated traffic peaks to 220 Mbps at the height of the attack. A SYN flood of 24,000 SYN packets/second, where each SYN packet Ethernet frame is 66 bytes, should generate 12.67 Mbps of network traffic. The experiment shows that the network bandwidth consumed after blacklisting has start is 14 Mbps. Fig. 51 shows the response times of the legitimate client's request during the experiment. The results show that the client response time increases when the DDoS attack ramps up to its peak and then gradually decreases when Sentinel starts blacklisting bot IP addresses and discarding bot-originated traffic.

Sentinel with hardware-accelerated blacklisting can scale to mitigate attacks of more than

24,000 requests/second, but the rate of the attacks I can generate is limited by the CPU resources of the attacker hosts in my performance setup. Once Sentinel has blacklisted all bots and has inserted "reject" rules for all bot IP addresses in the NFE-I8000 card's TCAM table, the HTTP flood attack is reduced to a flood of SYN packets, which are dropped on the NFE card. The NFE-I8000 is designed to handle up to 1Gbps of network traffic. If a SYN packet Ethernet frame has a size of 90 bytes (32 bytes TCP header, 20 bytes IP header, 14 bytes Ethernet header, and 24 bytes Ethernet CRC and inter-frame gap), it would take approximately 1.4 million SYN packets per second to overload the NFE-I8000 card. Overloading the NFE-I8000 will also result in exhausting the 1GigE network links of my performance setup. Sentinel is not designed to mitigate DDoS attacks where the attacker can deplete the target's network link capacity.

6.8 HTTPS FLOOD ATTACKS AGAINST WEB SERVERS

In this section, I measure the resilience of a standard web server against HTTPS flood attacks. BoNeSi generates TLS Client Hello messages that force the TLS server to pick a TLS cipher suite using ephemeral Diffie-Hellman (DH) key exchange. In response of the TLS Client Hello, the TLS server must generate a temporary (ephemeral) DH key pair, sign the DH parameters and public key using its RSA private key and send the DH parameters, DH public key and RSA signature inside the Server Key Exchange TLS message (Fig. 28). For this experiment, the nginx web server runs on Sentinel's hardware platform. Through performance benchmark tests, I established that the hardware platform can generate up to 178 1024-bit DH key pairs/second or up to 469 1024-bit RSA signatures/second. Taking the inverse of the rates, the host takes 5.618 milliseconds to generate a DH key pair and 2.132 milliseconds to produce an RSA signature. Because the web server has to do both for each TLS connection, it will have to spend 7.75 milliseconds per TLS connection only to execute public key cryptographic operations. This implies that the web server should not be able to handle more than 129 TLS connections/second. The web server could offload TLS handshake processing to cryptographic accelerators that can handle higher rates of TLS

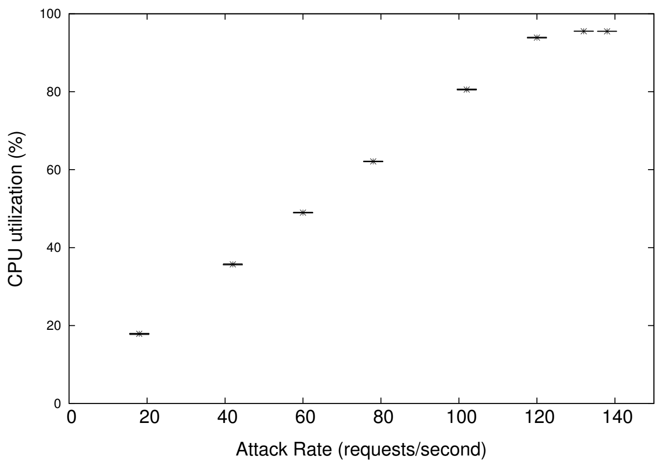


Figure 53: Experiment 6.8 – CPU utilization

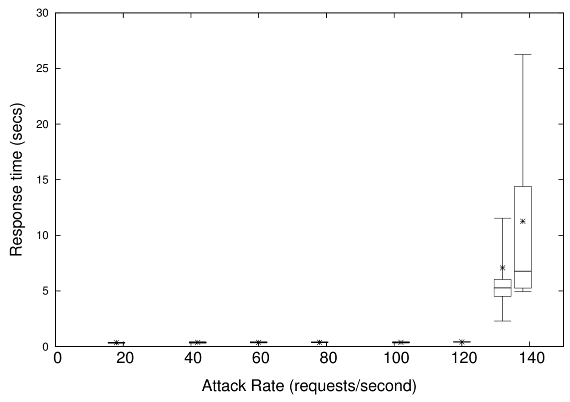


Figure 54: Experiment 6.8 – response time to get page

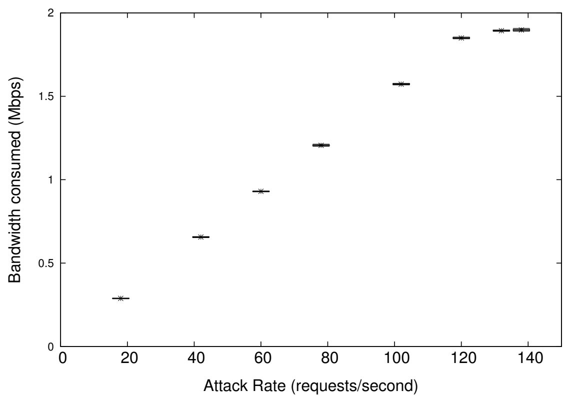


Figure 55: Experiment 6.8 – network bandwidth

connections. However, cryptographic accelerators do not verify client credentials in HTTP requests. A botnet can still flood the target with HTTP requests inside the established TLS tunnels.

Fig. 53 shows the web server's CPU utilization under attack. At 132 TLS requests/second, the web server's CPU utilization reaches 95%. Increasing the attack rate further to

140 TLS requests leaves the CPU utilization capped a 95%. The server is not able to keep up with the attack rate and can not respond to all incoming TLS requests. Fig. 55 presents the network bandwidth consumed by the attack. Due to the low attack rate, the amount of generated traffic is negligible. This shows that application-level TLS floods can overwhelm standard TLS servers with low volumes of custom TLS requests. Fig. 54 shows the response time of the legitimate client's requests during the attack. The results show high and very variable response times after the server's CPU utilization reaches 95%.

6.9 HTTPS FLOOD AND HARDWARE TOKEN AUTHENTICATION

This section tests Sentinel's resilience to floods of TLS requests. Sentinel looks for valid Kerberos credentials and in TLS Client Hello message and responds with a TLS fatal alert if the credentials are not found. Sentinel can scale to higher-rate floods of TLS requests because it does not execute any computationally intensive public key operations. If Sentinel finds valid credentials in the Client Hello message, it passes it to the back-end web server, which generates the ephemeral Diffie-Hellman key and the RSA signature. The web server performs these operations only for requests from clients that have demonstrated to be legitimate human users.

Fig. 56 shows the CPU utilization under attack. Sentinel can mitigate TLS request floods up to 36,000 requests/second. This is higher than the maximum of 19,000 HTTP requests/second that Sentinel can handle because TLS Client Hello messages and alerts are significantly smaller in size – 89 and 7 bytes, respectively. Fig. 57 presents the network bandwidth consumed by the DDoS attack. At 36,000 TLS requests/second, the attack generated 145Mbps of bi-directional network traffic. Fig. 58 and 59 show the legitimate client's response times to, respectively, get the pre-authentication challenge TLS alert and download the web page after authenticating. At 39,000 requests/second, the response times to download the web page peak at more than 10 seconds and become highly variable.

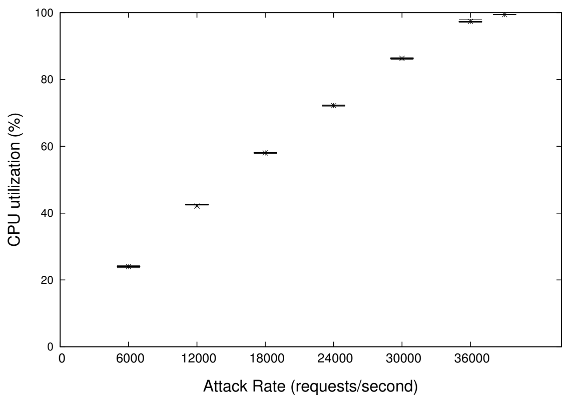


Figure 56: Experiment 6.9 – CPU Utilization

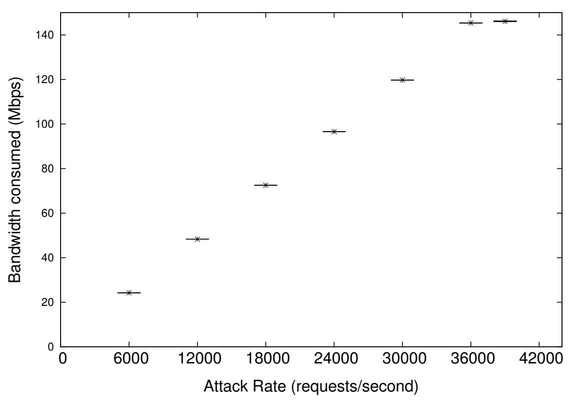


Figure 57: Experiment 6.9 – network bandwidth

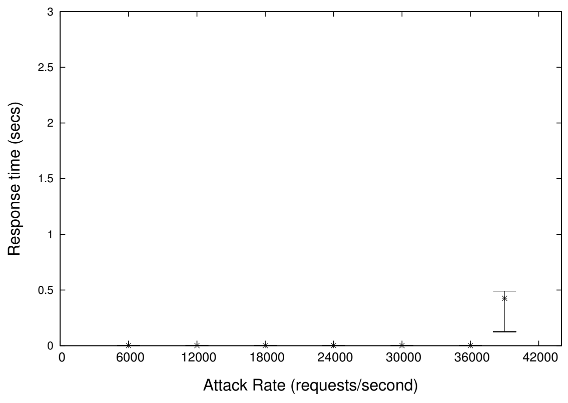


Figure 58: Experiment 6.9 – response time to get challenge

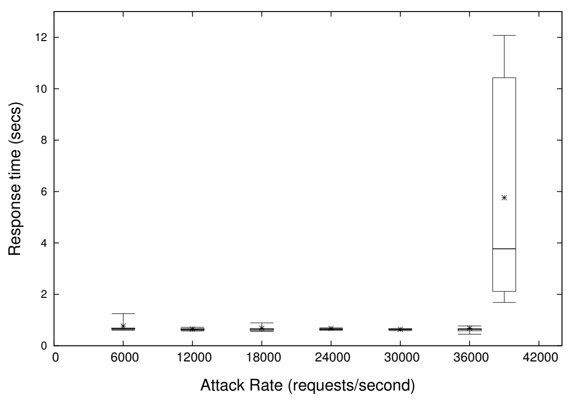


Figure 59: Experiment 6.9 – response time to get page

7.0 SECURITY ANALYSIS

This chapter defines a threat model for the system architecture in this dissertation and analyzes its security. I use the threat modeling techniques proposed in Myagmar et al. [108] because they are suitable for modeling network systems composed of multiple components with different roles. This threat modeling approach is related to the software application threat modeling approach described in Swiderski, et al. [125], but is tailored to complex network systems.

Section 7.1 characterizes the system's components, interconnections and dependencies in a network model. Section 7.2 identifies the system's valuable assets and the access points to those assets. Section 7.3 presents the threat profile of the system and the protection mechanisms in place to mitigate the high-priority threats. Section 7.4 concludes this chapter.

7.1 NETWORK MODEL

Fig. 60 presents the network model of this dissertation's system architecture. The model consists of the following components:

- Client host:
  - Web client application: requests web pages from a web server using the HTTP and optionally TLS protocols.
  - IdM client application: queries users for authentication credentials, makes function calls into the Kerberos client library to obtain Kerberos AS-REQ messages, and requests security tokens from an IdM identity provider using SOAP-based protocols

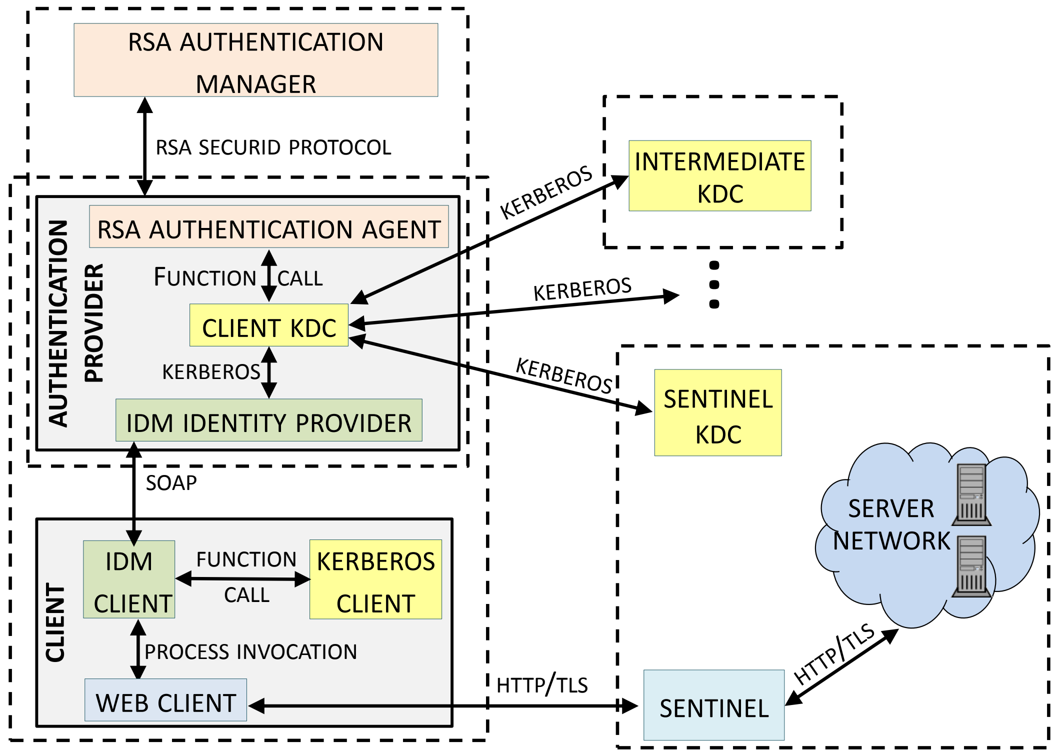


Figure 60: Network model – dashed lines define trust domains.

over HTTP.

- Kerberos client library: receives user authentication credentials from IdM client and generates AS-REQ requests for Kerberos proxy service tickets.

- Web server host:
  - Web server application: listens for client HTTP and TLS connections on, respectively, TCP ports 80 and 443, and serves requested web pages.
- Sentinel front-end device:
  - Sentinel application: intercepts HTTP and TLS packets destined for the web server's TCP ports 80 and 443 and verifies Kerberos service tickets. If necessary, Sentinel sends to the web client challenges for SecurID authentication using the HTTP or TLS protocol.
- Authentication provider host:

  - IdM identity provider application: listens on TCP port 80 for security token requests using SOAP-based protocols over HTTP and passes the AS-REQ from them to the Kerberos KDC using the Kerberos protocol over TCP.
  - Kerberos KDC application: listens on TCP port 88 for Kerberos AS-REQ requests from the IdM identity provider, verifies the requests and passes usernames and RSA SecurID token codes to the RSA SecurID authentication agent via library function calls. Upon successful SecurID authentication, the KDC requests ticket-granting and service tickets from the other Kerberos KDCs using the Kerberos protocol over TCP.
  - RSA SecurID authentication agent library: receives RSA SecurID usernames and SecurID token codes from Kerberos KDC and passes them to the RSA Authentication Manager for verification using an unknown, proprietary network protocol over UDP.
- Web server's Kerberos KDC host:
  - Kerberos KDC application: listens on TCP port 88 for Kerberos service ticket requests, verifies ticket-granting tickets and issues service tickets granting access to the web server.
- (optional) Intermediate Kerberos KDC hosts:
  - Kerberos KDC application: listen on TCP port 88 for Kerberos ticket-granting ticket requests, verify the ticket-granting tickets in the requests, and issue new ticket-granting tickets.
- RSA authentication manager host:
  - RSA Authentication Manager application: listens on UDP port 5500 for SecurID authentication requests from the SecurID authentication agent and checks the SecurID credentials against an internal database.

7.1.1 USAGE SCENARIO

In a typical usage scenario, the web client wishes to communicate over HTTP or TLS with the web server. The Sentinel transparent front-end device intercepts the client's HTTP or

TLS requests and challenges the client to authenticate using an RSA SecurID. The web client invokes the IdM client application. The IdM client queries the user for their RSA SecurID token code and Kerberos password, generates a Kerberos AS-REQ request for a hardware-authenticated Kerberos proxy ticket, encodes the AS-REQ into an IdM security token request and sends the request in a SOAP message over HTTP to the IdM identity provider. The identity provider extracts the AS-REQ message and passes it to the client's Kerberos KDC. The KDC extracts the SecurID token code from the AS-REQ and passes it to the RSA authentication agent library, which sends it to the RSA authentication manager for verification. Upon successful SecurID authentication, the client's KDC obtains ticket-granting tickets from all intermediate Kerberos KDCs, if any, obtains a service ticket for the web server from the server's Kerberos KDC and returns the service ticket to the IdM identity provider. The identity provider encodes the ticket into a security token and returns it to the IdM client, which extracts the Kerberos ticket and stores into the web client's HTTP cookie database. The web client sends the Kerberos ticket with subsequent HTTP and TLS requests to the web server.

7.1.2 TRUST DOMAINS AND DEPENDENCIES

The authentication path between the web client and Sentinel contains a series of trust domains and dependencies. Each Kerberos realm and its KDC are typically located in a separate trust domain. For example, the Kerberos client and its KDC form a trust domain. The client is registered as a principal on the client's KDC. The two share a cryptographic key derived from the client's Kerberos password and use the key to encrypt and authenticate each other's messages. System administrators can explicitly configure two KDCs to trust each other and thus bridge two trust domains. A shared key is exchanged out-of-band when the trust relationship is established. Sentinel, Sentinel's KDC and the web server also form a trust domain. Sentinel is registered as a server principal in the KDC and the two also share a cryptographic key exchanged out-of-band.

Authentication paths with multiple hops form a series of dependencies and trust links between Kerberos realms. Trust in inter-realm Kerberos authentication is not transitive because each KDC must be explicitly configured to trust both adjacent and non-adjacent KDCs on the authentication path. Non-transitive trust is important because each KDC relies on all upstream KDCs before it to have correctly authenticated the Kerberos client. A single malicious KDC can compromise the security of Kerberos by issuing a ticket-granting ticket without properly authenticating the client. The Kerberos protocol allows KDCs to be configured with explicit lists of trusted realm KDCs. All Kerberos tickets have a "transited-realms" field that records the list of realms traversed on the authentication path so far. If a list of explicitly trusted realms is configured, the KDC must check the "transited-realms" field against the list and reject the ticket-granting ticket request if any of the KDCs traversed are not explicitly trusted. If the KDC trusts all upstream KDCs, it copies the "transited-realms" field from the existing ticket-granting ticket into the newly issued ticket and appends the upstream KDC to the list.

The RSA authentication agent and manager also form a trust domain overseen by the client's authentication provider, which offers RSA SecurID authentication as a service. Each authentication agent is registered in the manager and the manager generates a binary configuration file for the new agent. The format of the binary file are proprietary and unknown, but includes the IP address and port number of the authentication manager. The binary configuration file is used during the authentication agent installation. When the agent contacts the manager for the first time, the manager passes to the agent a "node secret", which is a symmetric encryption key. The type and format of the node secret, as well as the mechanism for generating and passing the node secret onto an agent, is proprietary and not publicly known. The authentication manager and agent communicate over a proprietary UDP protocol. Future agent-manager communication is encrypted with the node secret.

The client KDC and, ultimately, Sentinel depend on the RSA Authentication Manager to authenticate clients. The authentication manager must verify that the client's token code is correct within an admissible time range and must check or token revocation. Failure to do so may make it more feasible for a bot to authenticate to Sentinel as a human user.

7.2 SYSTEM ASSETS AND ACCESS POINTS

I identified the following assets an attacker may target and their access points:

- Kerberos user passwords, obtained by key stroke capturing malware, bruce-force attacks or social engineering
- Kerberos AS-REQ requests, sniffed from the network
- Kerberos proxy service tickets, sniffed from the network, or read from web client HTTP cookie store
- Kerberos shared keys, accessed from the KDC or Sentinel filesystem
- RSA SecurID tokens, stolen from the client
- RSA SecurID token seeds, stolen from the client or a central database
- RSA SecurID token codes, stolen from the client as they become available
- RSA SecurID PRNG algorithm, deduced through reverse engineering
- RSA authentication manager CPU capacity, depleted by a large volume of requests
- Kerberos KDC CPU capacity, depleted by a large volume of requests
- Sentinel CPU capacity, depleted by a large volume of requests
- Sentinel TCAM memory capacity, depleted by having to blacklist a large number of bots

7.3 THREAT PROFILE

Below is the list of threats an attacker may use to attack the system assets listed in Section 7.2. For each threat, I discuss existing and possible mitigation mechanisms.

7.3.1 REPLAYING STOLEN KERBEROS PROXY SERVICE TICKETS

Web clients obtain Kerberos proxy service tickets after successful hardware token pre-authentication, store them in HTTP cookie databases, and pass them to web servers in subsequent HTTP requests and TLS Client Hello messages. Kerberos tickets are encrypted with a Kerberos shared key, but attackers may steal the encrypted ticket and bypass hardware token authentication to flood the web server with HTTP or TLS requests.

Attackers have multiple points of access for stealing Kerberos proxy service tickets. Major web clients, such as Mozilla Firefox, store the HTTP cookie table unprotected on the client's hard disk. An attacker with access to a client host can continuously monitor the HTTP cookie table and obtain the Kerberos credentials stored in it. The attacker can also use a malicious browser extension to intercept Kerberos credentials returned by the IdM client. If the attacker has access to the client's network interfaces, they can also sniff the proxy service tickets in the security token response from the IdM provider, or in the HTTP requests and TLS Client Hello messages to the web server.

This dissertation work mitigates the threat of replaying Kerberos credentials by binding Kerberos service ticket to a client's IP address, using SYN cookies on Sentinel to prevent IP spoofing, and limiting the number of HTTP requests a client can send with a Kerberos service ticket. The Kerberos protocol allows tickets to be bound to client IP addresses. The client KDC in the proposed architecture uses this feature to bind the proxy service ticket to the IP address of the client that requested it. An attacker may bypass this defense by stealing the Kerberos credentials and spoofing the source IP address for the HTTP/TLS request flood to be the client's IP address. To address this, the Sentinel front-end device intercepts client TCP SYN packets and uses SYN cookies to ensure that the client's IP address is not spoofed.

An attacker with access to the client's computer may bypass the defense mechanisms above by generating the HTTP/TLS flood from the client's computer. This attack mimics a legitimate flash crowd and Sentinel cannot distinguish between the legitimate client and the bot requests. Sentinel mitigates this attack by limiting the number of HTTP/TLS requests a client or bot can send with a valid Kerberos credential. The current Sentinel implementation sets the limit to four requests per credential, but the parameter can be adjusted higher to fit the needs of users accessing complex web sites with tens or even hundreds of objects on a single web page. Assuming that the legitimate client has a single SecurID token and given that the SecurID token outputs a new token code every 60 seconds, an attacker controlling a legitimate client's computer is rate-limited in the number of malicious HTTP/TLS requests they can generate per minute from a compromised client computer. Future work can extend the system architecture to temporarily revoke the SecurID tokens of clients, who obtain authenticate very often and obtain excessive number of Kerberos service tickets.

7.3.2 KERBEROS PASSWORD AND RSA SECURID TOKEN CODE THEFT

An attacker may compromise a user's Kerberos password as a first step towards obtaining a Kerberos proxy service ticket from the name of the client. Compromising the password may allow the attacker to obtain a single token code if they can intercept the Kerberos client's AS-REQ request to the client's KDC. The AS-REQ message is encrypted with a key derived from the user's password and contains a SecurID token code. If the attacker extracts the token code, they can generate a new AS-REQ message and send it to the client's KDC to obtain a new proxy service ticket. The attacker may also compromise the user's computer and install malware that steals Kerberos passwords and RSA SecurID token codes by capturing keyboard key strokes.

Both attacks allow the attacker to continuously obtain service tickets and use them to send floods of HTTP/TLS requests to the web server. The system architecture proposed mitigates this attack by limiting the number of HTTP/TLS requests and client can send with a Kerberos proxy service ticket. The rate at which the attacker can obtain service tickets is also limited to the rate at which the user's SecurID token generates new token codes (once a minute).

7.3.3 RSA SECURID TOKEN AND TOKEN SEED THEFT

An attacker may steal legitimate users' RSA SecurID tokens and use the token codes from them to obtain Kerberos proxy service tickets and send a flood of HTTP/TLS requests to the web server. This attack is impractical due to the number of hardware tokens registered with the RSA Authentication Manager that the attacker must steal to generate a sufficiently large flood of requests. The RSA Authentication Manager allows revoking lost or stolen hardware tokens, thus limiting the time window available for the attack. This dissertation's system architecture mitigates the attack by requiring the attacker to also know the Kerberos user's password. Without knowledge of the password, it is infeasible for the attacker to generate a valid AS-REQ request for the client's KDC.

The SecurID token seed is a secret AES key used to generate new token codes. In a recent attack against RSA, attackers may have stolen mappings between SecurID serial numbers and token codes [74]. The RSA SecurID token code generation algorithm is not publicly known, but is allegedly known to RSA partners and there have been attempts to reverse engineer and analyze it [84, 15, 107]. The algorithm allegedly combines the current time and the token's secret seed key to generate a new token code. If an attacker knows both the code generation algorithm and the secret token seed, they can likely predict future token codes. This attack is similar to the SecurID token theft attack and is mitigated in a similar way. Token codes become valid on at the time when a SecurID hardware token would generate them. If an attacker can guess multiple token codes prior to the time they will be displayed on the token's LCD screen, the attacker must still wait for the codes to become valid. This rate-limits the number of malicious HTTP/TLS requests the attacker can generate per compromised SecurID token seed. If the attacker compromises a large database of SecurID token seeds, they still need the respective users' Kerberos passwords to successfully execute the attack.

7.3.4 AVAILABILITY OF RSA AUTHENTICATION MANAGER

The ability of the client to access the web server depends on the availability of the RSA Authentication Manager application. The authentication manager is provided as a service by the user's identity provider and verifies token codes of RSA SecurID tokens issued by the provider. An attacker could make the manager unavailable by overloading it with authentication requests. The manager requires all authentication agents authorized to contact this server to be explicitly registered by agent IP address. Therefore, an attacker must use spoofed client IP addresses. The authentication manager is run on general purpose operating systems, such as Windows Server and Red Hat Enterprise Linux. Enabling TCP SYN cookies in the operating system kernel would prevent attackers from using IP spoofing. The internal architecture of the authentication manager and the network protocol used are not publicly known. Therefore, it is infeasible to analyze in detail the resilience of the authentication manager to DDoS attacks.

7.3.5 AVAILABILITY OF KERBEROS KDCS

The ability of the client to access the web server also depends on the availability of the Kerberos KDCs. AS-REQ messages and ticket-granting ticket requests contain encrypted portions that the KDC must decrypt before it verifies the authenticity of the request. An attacker could cause the Kerberos KDC to decrypt a flood of fake requests and deplete its CPU cycles. The Kerberos protocol provides a limited resilience to DDoS attacks because allows using efficient symmetric encryption algorithms, such as AES, to encrypt and decrypt Kerberos requests. Kerberos KDC have better ability than general purpose web servers to filter out DDoS traffic. General-purpose web servers often have large and constantly changing pools of users. KDCs typically only need to be accessible by Kerberos clients registered with the KDC as part of the Kerberos realm. Filtering rules configured in the KDC's operating system network stack, such as iptables rules, or in external firewalls can discard traffic from client IP addresses that are not registered in the Kerberos realm. This will not prevent new clients from registering in the realm because new user registration is typically done offline. I assume that KDCs verify users before registering them in the domain. It should be infeasible for a botmaster to register enough client IP addresses to be able to flood the KDC with Kerberos authentication requests. Filtering rules can optionally be inserted only when the Kerberos host's CPU utilization rises above a critical threshold.

7.3.6 DEPLETING SENTINEL'S TCAM MEMORY CAPACITY

Sentinel uses the TCAM memory on its network acceleration card to blacklist client IP addresses identified as bots. A large botnet may have more bots than Sentinel can blacklist given its TCAM memory capacity. The TCAM table of the NFE-i8000 network acceleration card currently used in Sentinel has space for up to 32,768 filtering rules for blacklisted IP addresses. As explained in the system model in Section 2.1, this dissertation considers botnets of up to 220,000 bots. In order to prevent botnets of this size from depleting

Sentinel's TCAM memory capacity, Sentinel can deploy the newer-generation NFP-32400 network acceleration card, which has expanded TCAM memory and can fit up to 262,144 filtering rules. Sentinel can also specify IP address subnets in the filtering rule keys. Such rules can match more than one IP address with a single rule and can reduce the number of filtering rules Sentinel must store in its TCAM memory. However, subnet blocking can result in false positives, where legitimate clients on the same subnet as bots are identified and blacklisted as bots. Sentinel alleviates this problem by periodically rehabilitating all blacklisted IP addresses. This is only a temporary resolution because legitimate clients that were once incorrectly identified as bots are likely to be blacklisted again as long as their IP subnet remains houses a large number of bots. Therefore, IP subnet blocking should only be used when the circumstances do not allow deploying a network acceleration card with large enough TCAM memory.

7.4 CONCLUSION

Instead of flooding Sentinel with HTTP and TLS requests to deplete its CPU resources, a botmaster may choose different means to attack Sentinel and the web server behind it. This chapter has explored multiple alternative attacks: stealing credentials or token codes to authenticate bot traffic to the web server, attacking the availability of the authentication provider, and depleting the capacity of Sentinel's table of blacklisted IP addresses. The analysis in this chapter has argued that the system architecture proposed in this dissertation can defend against the attacks considered, supporting the claim that the system can effectively mitigate botnet-based HTTP and TLS floods against web servers.

8.0 RELATED WORK AND ALTERNATIVE APPROACHES

Existing work in literature has proposed solutions to a subset of the problems addressed in this dissertation. Section 8.1 and 8.2 present CAPTCHA puzzles and trusted computing as mechanisms to distinguish legitimate client activity from bot traffic. Different proposals exist to mitigate application-level DDoS attacks by using indirection to obfuscate the location and identities of active servers. Section 8.3 discusses WebSOS, which routes web requests through a secure overlay of redundant network paths to prevent link congestion attacks. Section 8.4 discusses the usage of proactive roaming honeypots to obfuscate the location of backend servers. Extensive work exists on using client puzzles to redistribute the computational load of the TLS protocol from servers to clients and make it hard for attackers to cripple TLS servers with CPU-intensive requests. An overview of this work is provided in Section 8.5. Section 8.6 presents commercial solutions to detect and mitigate application level DDoS attacks. The design of the system proposed in this dissertation is a result of a number of design choices. Section 8.7 compares the choices made to alternative approaches to solve the problems addressed in the dissertation.

8.1 CAPTCHAS

Researchers have proposed client authentication with CAPTCHA (Completely Automated Public Turing Test To Tell Computers and Humans Apart) graphical puzzles [131] to distinguish between traffic from human clients and bots. Web sites use CAPTCHAs to prevent automated spam-related activity, such as e-mail account creation and online forum posts. The images are generated using distortion techniques (Fig. 61) to resist automated object recognition techniques [105]. Other CAPTCHAs present the clients with graphical image, audio, or video puzzles [117].

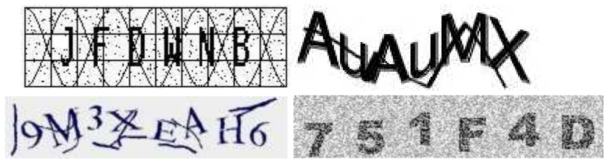


Figure 61: Sample CAPTCHA visual test designs

Existing CAPTCHAs are vulnerable to machine learning attacks [52, 41] and there are multiple commercially available CAPTCHA crackers on the market [123]. In 2002, researchers reported a mechanism with 92% success rate of breaking Yahoo CAPTCHAs [105]. Yahoo subsequently improved its CAPTCHAs, but more recent reports suggest successful attacks against Yahoo [58], Windows Live [90], Google [69] and other CAPTCHAs [52].

CAPTCHA authentication is also vulnerable to DDoS attacks. Servers cannot reuse CAPTCHAs because an attacker can solve a puzzle and replay the stored answer when the puzzle is reused. Assuming limited server storage space, single-use puzzles require servers to generate new CAPTCHA images at the same rate as client request rates. A bot master can exploit this using DDoS attacks with a high rate of CAPTCHA requests.

reCAPTCHA [12] allows web servers to offload CAPTCHA generation and verification to dedicated CAPTCHA servers (Fig. 62). The client requests a page and the web server responds. The response instructs the client to obtain a puzzle from a reCAPTCHA server. The user solves the puzzle and returns the answer to the web server, which asks the reCAPTCHA server to verify it.

reCAPTCHA addresses attacks where an attacker publishes reCAPTCHA puzzles on the attacker's own web sites. Legitimate users solve the puzzles and the attacker uses the responses to authenticate. reCAPTCHA generates a public/private key pair for each web server domain. The private key is not assumed to be secure and is only used to tie puzzle answers to the CAPTCHA puzzles they solve. The web server sends its reCAPTCHA public key to the client, which uses it to request a puzzle. The reCAPTCHA server matches the

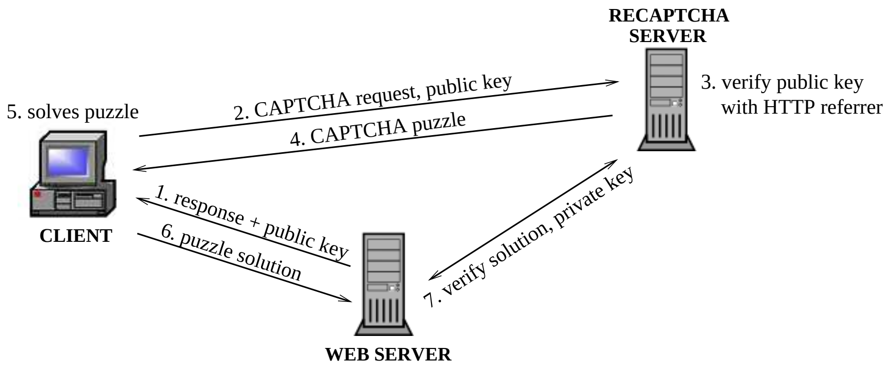


Figure 62: Message exchange using reCAPTCHA

domain name in the client request's HTTP Referrer field with the domain name for the public key. The client solves the puzzle and sends the solution to the web server. To verify the result, the web server sends it along with its reCAPTCHA private key to the reCAPTCHA server. The reCAPTCHA server verifies that the private key matches the web server's domain name and that the puzzle was issued for the corresponding public key. The client and web server may use a TLS connection to communicate with the reCAPTCHA server.

reCAPTCHA uses scanned book text, so its graphic puzzles represent common dictionary words and names. The lack of randomness makes reCAPTCHA puzzles vulnerable to attacks using packet shaping and artificial intelligence techniques [41]. reCAPTCHA is also vulnerable to DDoS attacks. It limits the puzzle request rate per client, but a large botnet can exhaust the reCAPTCHA server CPU resources using only a few CAPTCHA requests per bot.

Web servers cannot restrict the set of users that can solve a CAPTCHA. When a client host (or a bot) sends a web request and receives back a puzzle, the client can forward the puzzle to an unintended user on another client host to solve. Botmasters, who control adult web sites, lure human users to solve CAPTCHAs before accessing restricted content [96]. When the users want to access a video or a photo, they are asked to solve a CAPTCHA. The CAPTCHA displayed is from another website that the botmaster wants to attack. In a similar fashion, botmasters could also hijack other web browsing sessions to present the user with CAPTCHAs to solve [68]. A botmaster can also forward puzzles to commercial online CAPTCHA-solving services [42, 106]. Such services hire third party human solvers to solve hundreds of thousands of CAPTCHAs per day for as little as $1 per 1000 puzzles [82].

iCAPTCHAs aim to defend against third-party human solvers by presenting users with multiple puzzles with varying timeouts [126]. The iCAPTCHA solution uses a training period, during which it presents users with single character CAPTCHAs and measures the time users take to solve them. After the training period is over, iCAPTCHA uses the information gathered to set limits on the time users have to solve real CAPTCHAs. An attacker must receive the puzzle, forward it to a third-party human solver, receive the response and send it to iCAPTCHA before the puzzle timeout expires. The authors claim that the roundtrip communication latency makes it infeasible for the attacker to return the puzzle answer on time. iCAPTCHAs are vulnerable to attacks where attackers have human solvers standing by to solve CAPTCHAs in real time. They are also vulnerable to artificial intelligence attacks because they use the same format for single-character training puzzles and real puzzles.

8.1.1 KILL-BOTS

The Kill-Bots system [87] uses CAPTCHA authentication to mitigate bot-based DDoS attacks against web servers. It intercepts incoming HTTP requests and challenges clients to authenticate using CAPTCHA puzzles. If a client exceeds a threshold of issued puzzles without a correct answer, Kill-Bots blacklists it as a bot and blocks further traffic from its IP address.

Kill-Bots does not support HTTP 1.1 features, such as persistent connections and pipelining, because it implicitly assumes that each HTTP request uses a separate TCP connection. An attacker can pipeline multiple HTTP 1.1 requests in the same TCP connection to evade the CAPTCHA authentication requirement. Kill-Bots also does not protect secure servers using HTTPS.

Kill-Bots drops requests from blacklisted IP addresses even after malicious clients stopped using them. A different proposal by Mehra et al. [102] blocks the IP address for a period of time after three unsuccessful attempts. It also uses a variety of CAPTCHAs including image-based, text-based, vector-based and geometry-based CAPTCHAs. However, both Mehra et al. and Kill-Bots implicitly assume that attackers do not maliciously fragment packets or spoof fields other than the IP address. A single IP address can identify both legitimate clients and bots if they are behind the same HTTP proxy or NAT router.

8.2 TRUSTED COMPUTING

Not-a-Bot [76] and Jamshed et al. [83] use trusted computing and Trusted Platform Modules (TPMs) to distinguish legitimate client activity from bot traffic. TPMs are small cryptographic co-processors commonly found in contemporary general-purpose laptop and desktop computers that provide secure key storage and implement a limited set of cryptographic operations in hardware. In Not-a-Bot, the TPM is used to bootstrap a trusted human activity attester application that monitors the mouse and keyboard activity to detect human presence at the client host. The attester executes inside a Xen hypervisor, which protects the attester from the client's untrusted operating system. When a user requests a web page, the browser requests an attestation from the human activity attester. The attester deduces human presence at the client host by verifying the existence of recent keyboard and mouse activity. If it verifies human presence, the attester generates an *attestation* containing the secure hashes of the attester executable, the Xen hypervisor, the system BIOS and the web URL requested by the client. The attester signs the attestation with a private *attestation signing key* sealed into the client's TPM and only released to the trusted attester application. The attestation signing public/private key pair is derived by an *attestation identify key* permanently stored in the TPM. The TPM uses the attestation identity private key to issue a certificate for the attester's attestation signing public key.

The web client sends the signed attestation and the attestation signing public key certificate along with the HTTP request. The web server receives the request, verifies the attestation public key certificate and uses the public key to verify the attestation signature. The web server verifies that the web URL in the attestation matches the URL for the resource requested by the client and scans the list of software hashes in the attestation to verify that it trusts the attester application, Xen hypervisor and BIOS firmware running on the client host.

The Not-a-Bot authors admit that their system is vulnerable to smart attacks, where a bot executable can trigger human activity on the client host and fool the attester into providing the bot with a human presence attestation. Not-a-Bot does not defend from the more general scenario where a human user is using the keyboard and mouse while the bot is sending malicious requests to the web server. Jamshed et al. [83] addresses this weakness in Not-a-Bot by tightly binding the keyboard activity to the content requested from the web server.

Verifying attestation places a significant burden on the web server. The server must obtain and verify a certificate for the client TPM's attestation identity public key before it can use the public key to verify the attestation signing public key certificate. The attestation identify public key certificate must be issued by a well-known certifying authority trusted by the web server. Only then the server can trust the attestation signing public key to verify the attestation signature. The web server then must examine the list of hashes in the attestation and verify that the client host is running trusted versions of the attester application, Xen hypervisor and BIOS firmware. Verifying the trustworthiness of the software running on a remote host is a challenging problem in trusted computing because the server, or a third party, must maintain lists of known software vulnerabilities and new releases.

The system proposed in this dissertation does not require the modification of web servers because the web server farm behind Sentinel is unaware of Sentinel and the Kerberos federated authentication architecture. Not-a-Bot could also use unmodified web servers by offloading the verification of client attestation to a dedicated front-end device, but the problem with verifying the trustworthiness of the client's software remains. A malicious or compromised attester application can generate attestations without verifying human presence and send malicious requests to the web server. In comparison, Chapter 7 shows that Sentinel assumes untrusted software running on the client host. An attacker can continuously steal SecurID token codes or Kerberos service tickets from the client host, but would not be able to use them to generate DDoS attacks against the web server.

8.3 SECURE OVERLAYS

WebSOS [104] addresses link congestion DDoS attacks against web servers. It authenticates clients with CAPTCHA puzzles and eliminates “pinch points” by routing authenticated requests through a secure overlay of redundant network paths. Overlay nodes may have the following roles:

- Each web server periodically picks a number of *secret servlets* authorized to forward traffic to it. The web server notifies firewalls deployed deep in the wide area network to drop all packets that did not pass through one of the newly chosen secret servlets. The identities of the servlets are secret to clients.
- When assigned to a server, each servlet selects a small number of *beacon nodes* and notifies them of its identity and the web server it protects. The beacon nodes then route traffic destined for the server through the servlet. The dynamically changing set of beacons and secret servlets provides redundant paths to the web server and makes it harder for an attacker to identify and congest critical links.
- Any node in the overlay can act as a *Secure Overlay Access Point (SOAP)*. Access points authenticate client requests CAPTCHA puzzles and route them through the overlay to a beacon node. Authenticated clients receive short-lived X.509 client certificates that allow them to access the web server multiple times without re-authenticating.

WebSOS requires significant network modifications to deploy. The secure overlay is spread over a geographically distributed network, such as the Internet, that spans multiple administrative domains. The firewalls that discard server-bound traffic that has not passed through a secret servlet must be deployed deep into the wide area network, typically inside the web server’s ISP, to keep malicious traffic away from the server’s network. The secure overlay cannot easily be deployed incrementally because WebSOS assumes a low ratio between secret servlets and other overlay nodes to make it hard for an attacker to guess the identities of the current secret servlets. WebSOS assumes that overlay nodes communicate with each other using SSL tunnels to protect the confidentiality of the server-to-servlet and servlet-to-beacon messages that can reveal the servlet’s identities. To further defend against guessing attacks, WebSOS assumes that the secret servlets can modify packet header fields to mark legitimate traffic.

WebSOS does not handle attacks against CAPTCHA authentication using third-party solvers. A botmaster can use hired solvers to solve CAPTCHA puzzles and pass them to the bots, which authenticate with WebSOS access points, receive short-lived X.509 client certificates and flood the web server with requests until the certificate expires.

WebSOS aims to solve the problem of link congestion DDoS attacks, which are outside the intended scope of this dissertation. To eliminate pinch points and keep malicious traffic away from the web server's network, WebSOS requires significant network modifications to deploy a large enough secure overlay. In comparison, the system proposed in this dissertation aims to mitigate application-level DDoS attacks that deplete the target's CPU resources and takes a more end-to-end approach [120] that does not require significant network modifications. This dissertation's system model assumes that malicious traffic can reach the server network, where it is identified and discarded by the Sentinel front end device.

WebSOS verifies the human identify of clients using CAPTCHA authentication, but does not address attacks that use third-party human CAPTCHA solvers. A botnet can use hired solvers to solve CAPTCHA puzzles, obtain short-lived client certificates from WebSOS and flood the web server with requests until the client certificates expire. WebSOS cannot use hardware token authentication instead of CAPTCHA puzzles because it assumes that all overlay nodes, potentially located in different administrative domains, can act as access points and authenticate clients. WebSOS does not use federated authentication to delegate the authentication responsibility to independent authentication providers.

WebSOS assumes that client-server communication is encrypted end-to-end inside SSL tunnels through a series of overlay nodes that act as web proxies. The mandatory use of SSL introduces performance and administrative overhead on web servers. Web clients must be pre-configured to proxy web requests through one or more WebSOS access points. WebSOS also requires a Java applet to be installed in client browsers. In comparison, the system in this dissertation only requires a client browser extension that acts as a Kerberos client. Client requests need not be proxied and need not be tunneled inside SSL.

8.4 ROAMING HONEYPOTS

Khattab et al. [91] introduces the usage of *roaming honeypots*, an architecture where the backend servers in a server farm proactively switch between being active servers and honeypots. The new set of active servers is determined at pre-generated, pseudo-random intervals of time known to both the server farm and legitimate clients, but not to attackers. Servers not actively providing service act as honeypots and track and blacklists attackers trying to guess the current set of active servers.

Legitimate clients obtain the schedule of active servers by subscribing to receive service from an overlay of *access gateways*, which act as server front-ends. They authenticate clients, authorize their requests, and route the requests to the pool of active backend servers. The authors do not specify how access gateways authenticate and authorize client requests. In comparison, the system proposed in this dissertation authenticates the human identities of clients using hardware token authentication in a federated setting.

The overlay of access gateways is assumed to be large, geographically dispersed and continuously adapting to attacks. The system model places the gateways in close proximity to clients to avoid extra latency for client requests. The solution described in this dissertation does not require the addition of network hosts outside the server farm network. In this proposal, the Sentinel front-end device is transparent to both clients and servers. Web servers can be used without any software modifications and need not act as honeypots.

8.5 CLIENT PUZZLES

Researchers have proposed challenging clients with network-based or computationally hard client puzzles. Network-based puzzles, such as proposed in Abliz et al. [37], introduces network latency to throttle the rate of DDoS attacks. Computational puzzles, also known as proof-of-work, to discourage clients from sending frivolous requests. Client puzzles are based on different intractable or moderately hard problems [119, 67, 88, 85] and require clients to perform computational work before sending a request. Stebila et al. [124] increase the complexity of the client puzzles to decrease the frequency of requests from a single attacker. Morein et al. [103] uses client puzzles to redistribute the load of TLS connections from servers to clients. While client puzzles make harder DDoS attacks from a small number of attacking nodes, botnets have the computational resources to solve puzzles at high rates.

Juels and Brainard [86] propose thwarting TCP SYN flooding attacks [57] with client puzzles composed of independent sub-puzzles. A botnet can defeat this scheme by solving subpuzzles on different bots in parallel. Rivest et al. [119] and Dean at. al. [61] propose computational puzzles with dependent subparts, such as repeated modular exponentiation where the output of step i is the input for step $i+1$. A botmaster, however, can command sufficient number of attacking nodes and generate a large attack using only a few connection requests per node.

A web server must generate or acquire new client puzzles at sufficient rates to meet client demand, but puzzle generation can become a target of DDoS attacks. Lakshminarayanan et al. [97] and Waters et al. [134] propose to offload the computation to a dedicated external node. However, offloading the computation can make the dedicated external nodes targets of DDoS attacks.

Bocan [47] proposes threshold puzzles, which enforce lower and upper bounds on the latency of puzzles solutions. This work, however, does not specify ways to reliably establish such bounds in the presence of variable network round-trip times or client hardware configurations with different amount of computational power. Moreover, a botmaster can artificially delay puzzle responses to avoid detection.

Wang and Reiter [133] introduce puzzle auctions, where legitimate clients and attackers can choose the difficulty of the puzzles they solve. Hosts that solved more difficult puzzles are serviced with higher priority when the server is under an attack. Legitimate clients can outbid attackers by gradually increasing the difficulty of the puzzles they solve until receiving service. However, clients may have to solve a possibly unbounded number of puzzles before receiving service, and thus may experience high delays. This approach may also engage clients in “bidding wars” against botnets with greater hardware resources.

8.6 COMMERCIAL APPLICATION LEVEL DDOS SOLUTIONS

With the rise of application level DDoS attacks, a number of commercial systems to detect and mitigate such attacks have been offered commercially. The information presented here is compiled from marketing whitepapers and other publicly available documentation.

The BIG-IP Application Security Manager (ASM) by F5 networks [3] is a security appliance that detects and mitigates Layer 7 DDoS attacks, including HTTP floods. This solution has a learning period, during which it gathers information about the normal usage of the protected web application. The information collected may include access rate, response latency, rate of HTTP error page responses and geographical location of legitimate users. It may also include information about legitimate traffic peaks, such as Monday morning or payday peaks in traffic. After the learning phase, the ASM treats irregularly high traffic patterns as suspected DDoS attacks. Once a DDoS attack is detected, the ASM attempts to distinguish legitimate clients from automated bots by inserting a custom JavaScript snippet into application responses. Legitimate clients using web browsers execute the JavaScript snippet and pass the tests, but bots using automated traffic generation tools do not. The system then mitigates the DDoS attack by introducing access delays for or dropping the connections of suspicious clients.

The JavaScript snippets used by the ASM solution identify automated DDoS generation tools instead of verifying the human nature of web users. The information publicly available from F5 does not reveal details about the snippet. It may be effective in detecting automated bots running basic tools, but will likely fail against more sophisticated tools that use a JavaScript engine to respond to challenges. The snippets may be computationally intensive and resemble client puzzles, but a large botnet has the computational resources to respond to challenges at high rates. In comparison, the system proposed in this dissertation filters out automated bots by verifying the human nature of legitimate users. It is infeasible for an attacker to use automated bots to respond to authentication challenges.

The Peakflow SP Threat Management System (TMS) [6] from Arbor Networks also provides mitigation of application-level DDoS attacks. The TMS appliance is based on the Peakflow SP [1], which is a security-oriented, network-wide traffic monitoring system.

The Peakflow SP detects application-level DDoS attacks and routes the requests through a TMS appliance where they are analyzed in greater depth. The TMS appliance can be deployed inline and intercept incoming traffic, or it can advertise a BGP route to cause the attack traffic to be rerouted through the appliance. A network operator can mitigate the DDoS attack by blocking or rate-limitting HTTP requests based on destination URL, source geolocation, or another traffic attribute.

The Arbor Networks PeakFlow solution is deployed throughout a server or ISP network and detects DDoS attacks by monitoring network traffic. Once it has identified an attack, it does not attempt to distinguish between legitimate clients and attackers, but rate limits or blocks traffic based on traffic attributes. The solution I propose in this dissertation is more fine-grained and less intrusive than PeakFlow. The system only requires deploying the transparent Sentinel front end device in the server farm network and does not require any other network infrastructure. I propose to distinguish between legitimate clients and attackers using hardware token authentication and discard only the attackers' traffic without affecting the performance of legitimate client requests.

8.7 ALTERNATIVE APPROACHES

This section describes some of the design choices made when designing Sentinel and compares them to some of the alternative approaches that could have been taken. The Sentinel front-end device is a vital component of the system. Sentinel performs some of the same functionality as existing stateful application-level firewalls [3]: packet scrubbing, stateful packet filtering, deep packet inspection, and discarding packets from malicious hosts. However, Sentinel cannot be replaced with an application-level firewall because Sentinel checks for hardware-accelerated Kerberos credentials and responds with authentication challenges to unauthenticated requests. Sentinel can be integrated into a future firewall product, but existing firewall products currently do not provide this functionality. Without Sentinel in place to send authentication challenges, web clients would have to know to explicitly authenticate in advance and obtain hardware-authenticated Kerberos credentials before accessing the web server.

Continuously checking for authentication credentials also enables Sentinel to identify clients as bots, blacklist them, and discard bot traffic in hardware. Application-level firewalls typically blacklist clients based on anomalies in client traffic, such as zero-length TCP windows [29]. Sentinel monitors the load on the web servers behind it to decide when to require client authentication and when it can pass unauthenticated requests through to the web server farm. Sentinel also periodically rehabilitates blacklisted IP addresses.

Another alternative approach to the Sentinel front-end device is to replace it with a number of web server hosts. The web server farm can either attempt to withstand the DDoS attack by having a large enough number of servers, or it can implement Sentinel's functionality on the web servers. In the former scenario, web servers must check HTTP and TLS requests for authentication. The performance results in Section 6.6 show that when running on identical hardware, Sentinel without blacklisting and a standard nginx web server have similar performance under a flood of HTTP requests. In this experiment, the Sentinel software checks for and verifies hardware-authenticated Kerberos credentials, while nginx does not. Using hardware-accelerated blacklisting provides Sentinel with significant performance boost, as shown in Section 6.7. It is harder to implement hardware-accelerated blacklisting on the web server platform because this requires adding the network acceleration card to the server's hardware configuration and developing a different version of the web server application to communicate with the network via the acceleration card APIs instead of standard network sockets. Therefore, because the Sentinel without blacklisting and the nginx web server have identical performance on the same hardware and because it is easier to implement hardware-accelerated blacklisting on a dedicated platform than on the web server host, it is more beneficial to use Sentinel front-end devices than to add more web servers to the web server farm to perform Sentinel's responsibilities. This analysis assumes that the nginx web server performance under an HTTP flood is representative for the performance of other popular web servers. It is possible to implement the Sentinel functionality on the web servers, but that would also require very disruptive hardware and software modifications.

As described in Section 5.2, a client KDC traverses the Kerberos authentication path, obtains a service ticket from Sentinel's KDC for accessing the web server, and passes it back to the client. The client then encodes the ticket into extensions to the HTTP and TLS protocols. To avoid extending HTTP and TLS, Sentinel's KDC could notify Sentinel, or a firewall, that the client has been authenticated. Sentinel can maintain a whitelist of clients currently authorized to access the web server and rate limit or discard traffic from other clients during a DDoS attack. The drawback of this approach is that whitelisted clients can only be identified by their IP addresses. This is a security problem because clients behind middleboxes that translate between public and private IP addresses, such as routers and web proxies, can share a public IP address with bots. Identifying legitimate clients based on authentication tokens is more robust, but requires web clients to store tokens and send them to the server inside a new type of HTTP cookies or a TLS extension. Both extensions are acceptable because HTTP and TLS are designed to be extensible. HTTP cookies are generated by servers and opaque to clients. There have been numerous extensions to TLS [46] and the TLS specification requires endpoints to ignore extensions that they do not support.

The system described in this dissertation uses RSA SecurID hardware token authentication to verify that a web request was generated by a human user. Passwords and X.509 client certificates are alternative ways for the client to authenticate with their authentication provider. However, passwords client certificates do not verify the human nature of web users and could be stolen by bots and used to send malicious requests to the web server. Sentinel uses hardware token authentication in a federated authentication environment to reduce the number of tokens a user must have to one or a few. There are cheaper hardware token alternatives to RSA SecurIDs that have similar features. An example is the Vasco Digipass [7], which Ebay and PayPal offer to their customers for two-factor authentication.

8.8 CONCLUSION

This chapter discussed related work and alternative approaches to solving the problem of mitigating botnet-based DDoS attacks against web servers. In this section, I summarize how Sentinel differs from the approaches described. Hardware token authentication can distinguish more effectively distinguish between human user and bots compared to CAPTCHA puzzles, JavaScript code snippets, passwords and certificate-based authentication. It can be verified more easily than TPM-based attestation, which requires the verifier to maintain lists of trusted and untrusted applications, and compare the client software against them. The system proposed in this dissertation does not require changes to network infrastructure besides deploying the transparent Sentinel front-end device. However, our system assumes the existence of a Kerberos federated authentication environment. In comparison, deploying WebSOS or roaming honeypots requires changes to the wide area network between the clients and servers. Clients in the Sentinel architecture do not perform additional computation and their requests do not see the extra latency incurred by client puzzles.

9.0 CONCLUSION

This chapter concludes this dissertation. Section 9.1 summarizes the dissertation contributions and argues how they satisfy the initial thesis statement. Section 9.2 discusses future work.

9.1 THESIS STATEMENT AND CONTRIBUTIONS

This dissertation provides evidence satisfying the following thesis statement:

I claim that federated client authentication using hardware tokens can effectively protect HTTP and HTTPS servers against botnet-based DDoS attacks, and that hardware-accelerated server front-ends can transparently and scalably facilitate such authentication.

Hardware token authentication (Section 4.1) can be effectively used to distinguish between human users and bots. Unlike CAPTCHA puzzles, which hired human workers can solve in arbitrarily large numbers, it is infeasible to obtain sufficiently many RSA SecurID token codes because they become available at fixed intervals of one minute.

Hardware token authentication is impractical for the Web unless used in a federated authentication setting because a user must possess a token for every web server that requires one. In Sections 4.2 and 5.1, I integrate SecurID authentication with the Kerberos V5 protocol to reduce the number of hardware tokens a user must possess. Kerberos is resilient to both man-in-the-middle and DDoS attacks, but is not well-suited for light-weight clients authenticating with arbitrary web servers. Kerberos places the majority of the authentication burden on the client, requires substantial client authentication, and prevents clients from anonymously browsing the Web. Section 4.2.2 describes a novel offloaded Kerberos authentication mechanism to push the authentication burden from the client to the client's KDC and reduce the required client configuration. Section 4.2.4 describes the application of anonymous Kerberos proxy tickets to allow web users to prove their humaneness to web servers without revealing their Kerberos identities.

Section 5.3 describes mechanisms to efficiently encode and verify hardware-authenticated Kerberos credentials in HTTP and HTTPS requests. Chapter 3 and Section 4.3 present the design and implementation of Sentinel, a hardware-accelerated server front-end device that verifies Kerberos credentials. Sentinel is transparent to both clients and servers, and does not require web server modification or re-configuration. It performs packet scrubbing, stateful packet filtering, TCP connection splicing and deep packet inspection to robustly check HTTP and HTTPS requests for authentication credentials. Sentinel blacklists clients with excessive numbers of unauthenticated requests as bots and uses a network acceleration card to efficiently discard bot-originated traffic in hardware.

The performance analysis in Sections 6.3 and 6.7 shows that Sentinel using blacklisting is more resilient to HTTP request floods than a standard web server. Sentinel with hardware-accelerated blacklisting can mitigate attacks of at least 24,000 HTTP requests/second. The network acceleration card that implements blacklisting is commercially specified to handle up to 1Gbps of attack traffic, or more than 100,000 HTTP requests/second, without affecting non-blacklisted legitimate client traffic. A standard nginx web server with the same hardware configuration can handle floods of only up to 19,000 HTTP requests/second. Sentinel without blacklisting has identical performance to the nginx web server under HTTP request floods.

Sentinel offers significant performance benefits when mitigating floods of HTTPS requests. Section 6.8 shows that an attacker can deplete the nginx web server CPU capacity with floods of just 130 requests/second by forcing the server to execute two private key cryptographic operations per HTTPS request. Verifying authentication credentials in Client Hello extensions prevents Sentinel from executing private key operations and allows it to mitigate attacks of up to 36,000 HTTPS requests/second, or an improvement of 275 times over the nginx web server.

For hardware token authentication on the Web to be effective, web browsers need an intuitive user interface, where users can enter their Kerberos and hardware token authentication credentials. Sections 4.4 and 5.4 present the integration of Kerberos and hardware token authentication into the Identiy Metasystem (IdM) framework and information cards. Section 5.5 describes the interface presented to the user before and during hardware token authentication.

This dissertation satisfies the claims presented in the thesis statement. It shows that hardware token authentication integrated with Kerberos federated authentication can be used to effectively distinguish between human users and bots. It presents mechanisms to efficiently encode and verify hardware-authenticated Kerberos credentials in HTTP and HTTPS requests. This dissertation also describes the design and implementation of Sentinel, a hardware-accelerated, front-end device that efficiently verifies Kerberos credentials and mitigates HTTP and HTTPS flood attacks against web servers.

9.2 FUTURE WORK

This section discusses three possible areas of future work for this dissertation: enabling legitimate automated processes to access web servers under a DDoS attack, researching alternatives to hardware tokens as forms of authentication, and designing cross-realm Kerberos routing algorithms for authentication path discovery.

9.2.1 LEGITIMATE AUTOMATED PROCESS ACCESS TO WEB SERVERS UNDER DDOS ATTACKS

The system architecture described in this dissertation allows human users to pre-authenticate with a hardware token in order to access a Sentinel-protected web server under a DDoS attack. However, not all legitimate HTTP and HTTPS requests are initiated by human users using a web browser. The approach taken in this dissertation can be generalized and modified to allow legitimate automated processes to access a web server under a DDoS attack. These automated processes may be started by a human user, or by another automated process. For example, the former process may be a stock quote aggregator started on-demand by a broker who needs to automate the analysis of up-to-date stock quote information. The latter may be a travel portal that periodically pulls airline and hotel reservation data from different web sites and displays the information to the portal's customers. Both applications are time-sensitive and the unavailability of the respective online resources can cause significant losses of revenue. Both tasks are also too complex for a human user to perform from a web browser.

9.2.2 ALTERNATIVE FORMS OF HUMAN USER VERIFICATION

The work in this dissertation can also be expanded by researching alternatives to hardware tokens for authentication. In some cases, hardware tokens can be replaced by *software tokens* – software applications running on desktop computers or mobile devices that mimic the functionality of hardware tokens. Software tokens are easier to obtain and manage, but they are also more vulnerable to reverse engineering. Attackers have greater opportunity to reverse engineer software tokens and clone them to obtain arbitrary numbers of software tokens. Software tokens codes are also more easily read by automated bots.

9.2.3 KERBEROS CROSS-REALM ROUTING PROTOCOLS FOR AUTHENTICATION PATH DISCOVERY

This dissertation proposes offloaded Kerberos authentication, which transfers the majority of the authentication burden from a web client to its Kerberos KDC. The client's KDC traverses the cross-realm Kerberos authentication path and obtains a service ticket for the desired web server from the name of the client. The client's KDC can use server name canonicalization with the web server's DNS name to discover the identity and address of the server's Kerberos KDC. However, the client's KDC must be able to discover the authentication path between itself and the server's KDC in order to traverse the path and obtain the desired service ticket. A possible extension to this dissertation is to propose cross-realm routing protocols that allow KDCs to discover authentication paths to other KDCs.

BIBLIOGRAPHY

[1] Arbor Networks peakflow SP. http://www.arbornetworks.com/en/peakflow-sp.html.

[2] Bandit project's cross-platform card selector gives users control of their internet identities. http://www.novell.com/news/press/2007/6/bandit-projects-cross-platform-card-selector-gives-users-control-of-their-internet-identities.html.

[3] BIG-IP application security manager. http://www.f5.com/pdf/products/big-ip-application-security-manager-ds.pdf.

[4] BoNeSi, the DDoS botnet simulator. https://code.google.com/p/bonesi/.

[5] GNU Wget. http://www.gnu.org/software/wget/.

[6] The growing threat of application-layer DDoS attacks. http://www.arbornetworks.com/component/docman/doc_download/467-the-growing-threat-of-application-layer-ddos-attacks?Itemid=442.

[7] How RSA security token quality compares to Vasco. http://www.signify.net/upload/Documents/How_RSA_Tokens_Compare_to_Vasco.pdf.

[8] ImageMagick: Convert, edit and compose images. http://www.imagemagick.org/.

[9] Log in with PayPal (PayPal Access). https://developer.paypal.com/webapps/developer/docs/integration/direct/log-in-with-paypal/#getStart.

[10] MIT Kerberos distribution page. http://web.mit.edu/kerberos/dist/.

[11] PayPal security key. https://www.paypal.com/securitykey. Viewed on July 29, 2009.

[12] reCAPTCHA: Stop spam, read books. http://www.recaptcha.net/.

[13] SBL – IP Spam Filter – The Spamhaus Project. http://www.spamhaus.org/sbl/.

[14] The Heimdal Kerberos 5, PKIX, CMS, GSS-API, SPNEGO, NTLM, Digest-MD5 and SASL implementation. `http://www.h5l.org/`.

[15] Weaknesses in SecurID. `ftp://ftp.cerias.purdue.edu/pub/doc/access_control/securid.ps.gz`.

[16] What is OpenID? `http://openid.net/what/`.

[17] Windows Live ID. `http://www.passport.net/`.

[18] Windows Live ID web authentication SDK. `http://msdn.microsoft.com/en-us/library/bb676633.aspx`.

[19] PKCS #1 v2.1: RSA cryptography standard. `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`, June 2002.

[20] Breaking CAPTCHA without using OCR - a new technique. `http://www.puremango.co.uk/cm_breaking_captcha_115.php`, December 2007.

[21] OpenID authentication 2.0 - final. `http://openid.net/specs/openid-authentication-2_0.html`, December 2007.

[22] All certificates show not trusted - get error code (MITM in-the-wild). `https://bugzilla.mozilla.org/show_bug.cgi?id=460374`, October 2008.

[23] Know your enemy: Fast-flux service networks. `http://www.honeynet.org/papers/ff/`, August 2008.

[24] Two Europeans accused of cyberattacks. `http://www.upi.com/Top_News/2008/10/03/Two-Europeans-accused-of-cyberattacks/UPI-21181223054984/`, October 2008.

[25] eCommerce web site performance today. An updated look at consumer reaction to a poor online shopping experience. `http://www.damcogroup.com/white-papers/ecommerce_website_perf_wp.pdf`, August 2009.

[26] Developer roadmap update: Moving to OAuth 2.0 + HTTPS. `https://developers.facebook.com/blog/post/497/`, May 2011.

[27] Arbor special report: Worldwide infrastructure security report. `http://www.arbornetworks.com/research/infrastructure-security-report`, 2012.

[28] DDoS survey: Q1 2012 when businesses go dark. `http://www.neustar.biz/enterprise/docs/whitepapers/ddos-protection/neustar-insights-ddos-attack-survey-q1-2012.pdf`, 2012.

[29] DoS attacks: Response planning and mitigation. `http://www.checkpoint.com/downloads/campaigns/whitepapers/check-point-dos-whitepaper.pdf`, August 2012.

[30] Global application and nework security report. , 2012.

[31] The OAuth 2.0 authorization framework. http://tools.ietf.org/html/rfc6749.html, October 2012.

[32] 2013 industry requirements: Ending support for 1024-bit keys. http://www.symantec.com/page.jsp?id=1024-bit-certificate-support, 2013.

[33] Federated login for Google account users. https://developers.google.com/accounts/docs/OpenID#oauth, April 2013.

[34] Important: All 1024-bit certificates must migrate to 2048-bit encryption by Dec 31, 2013! https://community.thawte.com/forums/important-all-1024-bit-certificates-must-migrate-2048-bit-encryption-dec-31-2013, March 2013.

[35] RSA authentication manager: The strength of RSA SecurID authentication combined with the convenience and flexibility of risk-based authentication. http://www.emc.com/collateral/data-sheet/h11403-RSA-AM8-ds.pdf, 2013.

[36] SSL/TLS acceleration and offload. http://www.a10networks.com/products/axseries-ssl_acceleration.php, 2013.

[37] Mehmud Abliz, Taieb Znati, and Adam Lee. Mitigating distributed service flooding attacks with guided tour puzzles. *International Journal in Advances to Security*, 5(3&4):121–133, December 2012.

[38] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Proc. 22th USENIX Security Symposium*, 2013.

[39] Kazumaro Aoki, Jens Franke, Thorsten Kleinjung, Arjen Lenstra, and Dag Arne Osvik. A kilobit special number field sieve factorization. Cryptology ePrint Archive, Report 2007/205, 2007. http://eprint.iacr.org/.

[40] Alan Arnold. Assessing the financial impact of downtime. http://www.businesscomputingworld.co.uk/assessing-the-financial-impact-of-downtime, April 2010.

[41] Paul Baecher, Niklas Büscher, Marc Fischlin, and Benjamin Milde. Breaking reCAPTCHA: A holistic approach via shape recognition. In *Future Challenges in Security and Privacy for Academia and Industry*, pages 56–67. Springer, 2011.

[42] Vikas Bajajs. Spammers pay others to answer security tests. http://www.nytimes.com/2010/04/26/technology/26captcha.html?_r=0, April 2010.

[43] Elaine Barker and Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf, January 2011.

[44] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. `http://tools.ietf.org/html/rfc1945`, May 1996.

[45] Daniel Bernstein. SYN cookies. `http://cr.yp.to/syncookies.html`.

[46] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. `http://tools.ietf.org/html/rfc4366`, April 2006.

[47] Valer Bocan. Threshold puzzles: The evolution of DOS-resistant authentication. *Transactions on Automatic Control and Computer Science*, 49(63), 2004.

[48] R. Braden. Requirements for Internet hosts – communication layers. `http://tools.ietf.org/html/rfc1122`, October 1989.

[49] Andrei Broder and Michael Mitzenmacher. Network application of Bloom filters: A survey. In *Allerton*, 2002.

[50] John E. Bryson and Patrick D. Gallagher. Electronic authentication guideline. `http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf`, December 2011.

[51] Elie Bursztein, Steven Bethard, Celine Fabry, John C Mitchell, and Dan Jurafsky. How good are humans at solving CAPTCHAs? a large scale evaluation. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 399–413. IEEE, 2010.

[52] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 125–138, New York, NY, USA, 2011. ACM.

[53] K. Cameron and M. Jones. *Design Rationale behind the Identity Metasystem Architecture*. December 2007.

[54] Kim Cameron and Michael Jones. Design rationale behind the Identity Metasystem architecture. *ISSE/SECURE 2007 Securing Electronic Business Processes*, pages 117 – 129, December 2007.

[55] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and protocols for the OASIS Security Assertion Markup Language (SAML) v2.0. `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`, March 2005.

[56] Martin Casado and Michael Freedman. Peering through the shroud: The effect of edge opacity on IP-based client identification. In *Proc. 4th Symposium on Network Systems Design and Implementation (NSDI)*, Cambridge, MA, 2007.

[57] CERT. Advisory CA-96.21: TCP SYN flooding and IP spoofing attacks, September 1996.

[58] Thomas Claburn. Yahoo's CAPTCHA security reportedly broken. `http://www.informationweek.com/news/showArticle.jhtml?articleID=205900620`, January 2008.

[59] Cristian Coarfa, Peter Druschel, and Dan Wallach. Performance analysis of TLS web servers. In *In Proc. of the Network and Distributed System Security (NDSS '02)*, 2002.

[60] J. Davis. Hackers take down the most wired country in Europe. `http://www.wired.com/politics/security/magazine/15-09/ff_estonia`, August 2007.

[61] Drew Dean and Adam Stubblefield. Using client puzzles to protect TLS. In *In Proc. 10th Conference on USENIX Security Symposium*, 2001.

[62] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol. Version 1.2. `http://www.ietf.org/rfc/rfc5246.txt`, August 2008.

[63] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[64] Peter Djalaliev, Muhammad Jamshed, Nicholas Farhan, and José Brustoloni. Sentinel: Hardware-accelerated mitigation of bot-based DDoS attacks. In *In Proc. of the International Conference on Computer Communications and Networks (ICCCN '08)*, August 2008.

[65] Michael Drewniak. Michigan man gets 30 months for conspiracy to order destructive computer attacks on business competitors. `http://www.usdoj.gov/criminal/cybercrime/araboSent.htm`, August 2006.

[66] Geoff Duncan. Feds to remotely delete Coreflood from infected PCs. `http://www.digitaltrends.com/computing/feds-to-remotely-delete-coreflood-from-infected-pcs/`, April 2011.

[67] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147. Springer-Verlag, 1993.

[68] Manuel Egele, Leyla Bilge, Engin Kirda, and Christopher Kruegel. CAPTCHA smuggling: hijacking web browsing sessions to create CAPTCHA farms. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1865–1870, New York, NY, USA, 2010. ACM.

[69] Ahmad S El Ahmad, Jeff Yan, and Mohamad Tayara. *The robustness of Google CAPTCHA's.* Computing Science, Newcastle University, 2011.

[70] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. `http://tools.ietf.org/html/rfc2616`, June 1999.

[71] Jim Finkle. Exclusive-Microsoft, FBI take aim at global cyber crime ring. `http://in.reuters.com/article/2013/06/05/idINL1N0EG16N20130605`, June 2013.

[72] D. Florencio and C. Herley. A large-scale study of web password habits. In *In Proc. of 16 Intl Conference on World Wide Web (WWW'2007)*, pages 657–666, 2007.

[73] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol version 3.0. `http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00`, November 1996.

[74] Dan Goodin. RSA won't talk? Assume SecurID is broken. `http://www.theregister.co.uk/2011/03/24/rsa_securid_news_blackout/`, March 2011.

[75] Thomas Groß. Security analysis of the SAML single sign-on browser/artifact profile. In *In Proc. of 19th Annual Computer Security Applications Conference (ACSAC '03)*, page 298, 2003.

[76] Ramakrishna Gummadi, Hari Balakrishnan, Petros Maniatis, and Sylvia Ratnasamy. Not-a-Bot (NAB): Improving service availability in the face of botnet attacks. In *NSDI 2009*, Boston, MA, April 2009.

[77] Eran Hammer. OAuth 2.0 and the road to hell. `http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/`, July 2012.

[78] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *Proc. 10th USENIX Security Symposium*, page 9, 2001.

[79] Andrew Hendry. Rent-a-botnet makes cyber crime a breeze. `http://www.itworld.com/rent-a-botnet-080515`, May 2008.

[80] K. Hornstein, K. Renard, C. Newman, and G. Zorn. Integrating single-use authentication mechanisms with Kerberos. `http://tools.ietf.org/html/draft-ietf-krb-wg-kerberos-sam-03`, July 2004.

[81] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. `http://tools.ietf.org/html/rfc2459`, January 1999.

[82] Markus Jakobsson. *The death of the Internet*. Wiley-IEEE Computer Society Press, 2012.

[83] Muhammad Asim Jamshed, Wonho Kim, and KyoungSoo Park. Suppressing bot traffic with accurate human attestation. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*, pages 43–48. ACM, 2010.

[84] Jeff Jarmoc. RSA compromise: impacts on SecurID. `http://www.secureworks.com/research/threats/rsacompromise`, March 2011.

[85] Yves Igor Jerschow and Martin Mauve. Non-parallelizable and non-interactive client puzzles from modular square roots. In *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security*, ARES '11, pages 135–142. IEEE Computer Society, 2011.

[86] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. Network and Distributed Security Systems (NDSS)*, pages 151–165, 1999.

[87] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz-4-Sale: Surviving organized DDoS attacks that mimic flash crowds. In *Proc. 2nd Symposium on Network Systems Design and Implementation (NSDI)*, Boston, MA, 2005.

[88] Ghassan O. Karame and Srdjan Čapkun. Low-cost client puzzles based on modular exponentiation. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, pages 679–697. Springer-Verlag, 2010.

[89] G. Keizer. Hacked GOP site infects visitors with malware. `http://www.pcworld.com/article/137226/hacked_gop_site_infects_visitors_with_malware.html`, September 2007.

[90] Gregg Keizer. Spammers' bot cracks Microsoft's CAPTCHA. `http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9061558`, February 2008.

[91] Sherif M. Khattab, Chatree Sangpachatanaruk, Daniel Mossé, Rami Melhem, and Taieb Znati. Roaming honeypots for mitigating service-level denial-of-service attacks. In *In Proceedings of the 24 th International Conference on Distributed Computing Systems (ICDCS*, pages 238–337, 2004.

[92] Tracy Kitten. FBI: DDoS botnet has been modified - new attack methods attempt to circumvent mitigation. `http://www.bankinfosecurity.com/fbi-ddos-botnet-has-been-modified-a-5719/op-1`, April 2013.

[93] Donald Knuth, James H. Morris, and Vaughan Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1973.

[94] Neal Koblitz. *A Course in Number Theory and Cryptography.* Springer, 2nd edition, 1992.

[95] David P. Kormann and Aviel D. Rubin. Risks of the Passport single signon protocol. *Computer Networks: The International Jounal of Computer and Telecommunications Networking*, 33:51 – 58, June 2000.

[96] Munir Kotadia. Porn gets spammers past Hotmail, Yahoo barriers. `http://www.news.com/2100-1023-5207290.html`.

[97] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming IP packet flooding attacks. In *In 2nd ACM Workshop on Hot Topics in Networks*, November 2003.

[98] J. Leyden. Conficker botnet growth slows at 10m infections. `http://www.theregister.co.uk/2009/01/26/conficker_botnet/`, January 2009.

[99] J. Leyden. Techwatch weathers DDoS extortion attack. `http://www.theregister.co.uk/2009/01/30/techwatch_ddos`, January 2009.

[100] Flora Liu. MDK: The largest mobile botnet in china. `http://www.symantec.com/connect/blogs/mdk-largest-mobile-botnet-china`, January 2013.

[101] R. McMillan. Major web sites hit with growing web attack. `http://www.infoworld.com/t/data-management/major-web-sites-hit-growing-web-attack-739`, March 2008.

[102] M Mehra, M Agarwal, R Pawar, and D Shah. Mitigating denial of service attack using CAPTCHA mechanism. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pages 284–287. ACM, 2011.

[103] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, April 1978.

[104] William Morein, Angelos Stavrou, Debra Cook, Angelos Keromytis, Vishal Misra, and Dan Rubenstein. Using graphic Turing tests to counter automated DDoS attacks against web servers. In *Proc. 10th ACM International Conference on Computer and Communications Security (CSS)*, pages 8–19, October 2003.

[105] Greg Mori and Jitendra Malik. Breaking a visual CAPTCHA. `http://www.cs.sfu.ca/~mori/research/gimpy/`.

[106] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Re: CAPTCHAs–understanding CAPTCHA - solving services in an economic context. In *USENIX Security Symposium*, volume 10, 2010.

[107] Mudge and Kingpin. Initial cryptanalysis of the RSA SecurID algorithm, 2001.

[108] Suvda Myagmar, Adam Lee, and William Yurcik. Threat modelling as a basis for security requirements. In *Requirements Engineering for Information Security (SREIS), 2005 IEEE Symposium on*. IEEE, August 2005.

[109] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos network authentication service (v5). `http://tools.ietf.org/html/rfc4120`, July 1993.

[110] Gunter Ollmann. The opt-in botnet generation: Social networks, hacktivism and centrally-controlled protesting. `https://www.damballa.com/downloads/r_pubs/WP_Opt-In_Botnet.pdf`, 2010.

[111] Brian Prince. WikiLeaks supporters' attacks show power of opt-in botnets. http://www.eweek.com/c/a/Security/WikiLeaks-Supporters-Attacks-Show-Power-of-Optin-Botnets-810005, December 2010.

[112] N. Provos, P. Mavrommatis, M. Abu Rajab, and F. Monrose. All your iFRAMEs point to us. In *In Proc. of 17th USENIX Security Symposium*, July 2008.

[113] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser: Analysis of web-based malware. In *In Proc. of 1st Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, April 2007.

[114] K. Raeburn and L. Zhu. Kerberos principal name canonicalization and KDC-generated cross-realm referrals. http://tools.ietf.org/html/draft-ietf-krb-wg-kerberos-referrals-11, July 2008.

[115] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging. In *In Proc. of 1st Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, page 5, April 2007.

[116] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proc. 6th ACM SIGCOMM Conference on Internet Measurement*, pages 41–52, Rio de Janeiro, Brazil, 2006.

[117] M Kameswara Rao and K Kiran Kumar. Using human cognitive abilities to distinguish computers and humans for preventing bot attacks. *International Journal of Computer Science and Engineering*, 2013.

[118] G. Richards. OTP pre-authentication. draft-ietf-krb-wg-otp-preauth-08. http://tools.ietf.org/html/draft-ietf-krb-wg-otp-preauth-08, September 2008.

[119] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, 1996.

[120] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984.

[121] Molly Sauter. Distributed denial of service actions and the challenge of civil disobedience on the Internet, May 2013.

[122] Eytan Seidman. Microsoft CRM integration. http://www.consultcrm.co.uk/documents/mscrm4/ms-crm3-application-integration.pdf, June 2003.

[123] Merrill Serrao, Shanu Salunke, and Amrita Mathur. Cracking CAPTCHAs for cash: A review of CAPTCHA crackers. *International Journal of Engineering*, 2(1), 2013.

[124] Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Gonzalez Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In *Topics in Cryptology–CT-RSA 2011*, pages 284–301. Springer, 2011.

[125] Frank Swiderski and Window Snyder. *Threat Modelling*. Microsoft Press, June 2004.

[126] H.D. Truong, C.F. Turner, and C.C. Zou. iCAPTCHA: The next generation of CAPTCHA designed to defend against 3rd party human attacks. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–6, 2011.

[127] E. Tsyrklevich and V. Tsyrklevich. OpenID single sign-on for the Internet: A security story. `https://www.blackhat.com/presentations/bh-usa-07/Tsyrklevich/Whitepaper/bh-usa-07-tsyrklevich-WP.pdf`, 2007.

[128] Liam Tung. Google upgrading all SSL certificates to 2048-bit keys by end of 2013. `http://www.zdnet.com/google-upgrading-all-ssl-certificates-to-2048-bit-keys-by-end-of-2013-7000015863/`, May 2013.

[129] D. Utter. Storm botnet triples in size. `http://www.securitypronews.com/insiderreports/insider/spn-49-20080104StormBotnetTriplesInSize.html`, January 2008.

[130] Guido van Rooij. Real stateful TCP packet filtering in IP Filter. Invited talk in the 10th USENIX Security Symposium, August 2001.

[131] Lius von Ahn, Manuel Blum, Nicholas Hopper, and John Langforg. CAPTCHA: Using hard AI problems for security. In *Proc. Eurocrypt*, pages 294–311, 2003.

[132] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing me onto your accounts through Facebook and Google: a traffic-guided security study of commercially deployed single-sign-on web services. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 365–379. IEEE, 2012.

[133] XiaoFeng Wang and Michael Reiter. Defending against denial-of-service attacks with puzzle auctions. In *In Proc. 2003 Symposium on Security and Privacy*, pages 78–92, May 2003.

[134] Brent Waters, Ari Juels, J. Halderman, and Edward Felten. New client puzzle outsourcing techniques for DoS resistance. In *In Proc. of 11th Conference on Computer and Communications Security (CCS2004)*, 2004.

[135] L. Zhu and P. Leach. Anonymity support for Kerberos. `http://tools.ietf.org/html/draft-ietf-krb-wg-anon-10`, October 2008.